TÉCNICO LISBOA

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

A Comparison of Different Time Series Prediction Techniques

Gonçalo Leote Cardoso Mestre

Thesis to obtain the Master of Science Degree in

Data Science and Engineering

Supervisors: Luís Miguel Teixeira d'Ávila Pinto da Silveira
Ruxandra Georgeta Barbulescu

Examination Committee

Chairperson: TBD
Supervisor: Luís Miguel Teixeira d'Ávila Pinto da Silveira
Members of the Committee: Prof. Alexandra Sofia Martins de Carvalho
Dr. Ruxandra Georgeta Barbulescu
TBD

October 2021

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Para os meus avós,

Acknowledgments

First of all, I would like to thank my supervisors Professor Luís Miguel Silveira and Researcher Ruxandra Barbulescu, for all their support, advises and extensive reviews provided during the development of this thesis. Their availability was immense, meeting as regularly as any student could ask for, sometimes even several times a week, always with a good mood and jokes which made this work much more enjoyable.

A thank you is also deserved for all the professors who I crossed my path with during these five years on IST and that stimulated me to keep my desire to study and learn, since without this, the following work would not be possible. A special mention is also deserved to professor Carlos Bispo, who worked closely with me in the last year in the AED course, being an example of what a professor should be, while being able to make me laugh with funny emails all the time.

I would also like to thank all my colleagues that made my time at IST even more enjoyable, through barbecues, Queimas, afternoon's at arco, days and nights closed in Espaço 24 "working" and all other enjoyable moments. A FRA for all of you.

A special thanks is deserved for my parents who supported me during all years of my existence providing me love even and giving me the tools to learn and question what the world presents me. A thank you also to my siblings Miguel and Sara, for putting up with me and helping me when needed. None of this would also be possible without the help of my girlfriend throughout the last three years, whenever I had any issue, she was there for me.

This work was partially supported by Portuguese National funds through FCT, Fundação para a Ciência e a Tecnologia, under the project NeuronReduce - PTDC/EEI-EEE/31140/2017.

Abstract

Given its inner complexity and the potential for human advancement resulting in 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 of the worm. 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 structures, prove successful in modeling 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, *Caenorhabditis Elegans*, Time Series Prediction, Machine Learning, Artificial Neural Networks

Resumo

Devido à complexidade interna e potenciais avanços científicos resultantes de uma maior compreensão do funcionamento do cérebro e sistema nervoso humano, este estudo é um dos maiores desafios em neurociência computacional. Para entender as dinâmicas deste sistema nervoso, pode ser obtido conhecimento através do estudo de organismos mais simples como a *Caenorhabditis Elegans* (*C. elegans*).

Este nematóide possui um pequeno sistema nervoso que permitiu a sua caracterização quase completa, levando à criação de modelos baseados no mapa de ligações completo. Escalar estes modelos para outros organismos é uma tarefa exigente devido à quantidade de informação utilizada para descrever um sistema nervoso, gerando modelos complicados com elevados custos computacionais, quando em muitos casos apenas existe um interesse em saber a resposta do sistema a estímulos variados.

Como objectivo de modelar apenas comportamento periférico de entrada e saída do sistema nervoso da *Caenorhabditis Elegans*, este trabalho testa a capacidade de diferentes técnicas de caixa negra de reproduzir o comportamento deste sistema nervoso usando apenas informação periférica observável do sistema. Com este objectivo, um modelo detalhado do sistema nervoso deste nematóide é utilizado no simulador NEURON para gerar conjuntos de dados correspondentes a dois comportamentos da minhoca. Estes dados são utilizados para testar a capacidade de cinco arquitecturas diferentes de redes neuronais de modelar estes comportamentos. Duas destas arquitecturas, LSTM e GRU, provaram o seu sucesso em replicar estes comportamentos com rigor, também sendo capazes de criar modelos que conseguem replicar a resposta do sistema em dois comportamentos diferentes simultaneamente.

Palavras Chave

Sistemas Nervosos, *Caenorhabditis Elegans*, Previsão de Séries Temporais, Aprendizagem Automática, Redes Neuronais Artificiais

Contents

List of Figures

List of Tables

Acronyms

ARIMA	Autoregressive Integrated Moving Average
BPTT	Backpropagation Through Time
BiLSTM	Bidirectional Long Short-Term Memory Unit
CNS	Central Nervous System
CNN	Convolutional Neural Network
DeepESN	Deep Echo State Network
GPR	Gaussian Process Regression
GRU	Gated Recurrent Unit
HNS	Human Nervous System
LMA	Levenberg-Marquardt Algorithm
KNNR	K-Nearest Neibors Regression
LSTM	Long Short-Term Memory Unit
MAE	Mean Absolute Error
MA	Moving Average
MSE	Mean Squared Error
MLP	Multilayer Perceptron
NARX	Neural Autoregressive Exogenous Model
PTRS	Posterior Touch Response Stimulation
RNN	Recurrent Neural Network

RTRL Real Time Recurrent Learning

RMSE Root Mean Squared Error

SVR Support Vector Regression

TDNN Time Delay Neural Network

1

Introduction

Contents

This chapter introduces the main goals of the work carried out in this thesis, providing motivation on the study that was performed. It starts by describing the importance of understanding complex nervous systems like the human brain and why the analysis of smaller and simpler organisms can provide insight into the fundamentals of neuronal dynamics. Then, the relevance of using different data-driven black box models to reproduce the nervous system behavior is described. The objectives of this work and its main contributions are also briefly summarized and motivated. Additionally, the organization of the thesis is described to better guide the reader throughout the document.

1.1 Motivation

The study of the human brain and nervous system is a very challenging and stimulating task and definitely one of the greatest challenges in computational neuroscience. One of the reasons why this is such a compelling and difficult task is 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 [3], making it one of the largest complex systems available to study.

The comparison between the number of neurons in the human nervous system and other living species' nervous systems confirms the immense complexity of the human brain: 177 neurons (*Ciona Intestinalis*, Sea Squirt), 302 neurons (hermaphrodite *Caenorhabditis Elegans*, Roundworm), 1.35×10^5 neurons (*Drosophila Melanogaster*, Fruit Fly), 6.79×10^7 neurons (*Mus Musculus*, House Mouse), 8.61×10^{10} neurons (*Homo Sapiens*, Human) and 2.57×10^{11} neurons (*Loxodonta*, African Elephant) [4], [5], [6], [7], [3], [8]

For some of these simpler and smaller nervous systems the underlying principles of network organization and information processing are easier to postulate due to their simplicity and availability of recordings. Therefore these organisms can become useful to gain insight into the fundamentals of neuronal dynamics and whole brain organization. One of the organisms that belongs to this category is the *Caenorhabditis Elegans* (*C. elegans*), a small nematode (roundworm) of about 1 mm in length, for which synthetic data of the nervous system is used in this thesis 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 [9]. The complete connectome of geometrically distributed neurons and synapses for the *C. elegans* is composed of 302 neurons for the adult hermaphrodite [10], and of 385 neurons for the adult male [9].

The connectome 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.

The connectome models mentioned above are usually white-box models, based on direct knowledge and access to the internal structure and parameters' values of the modeled system. White-box models are often created from the ground up, starting from a set of equations that describe phenomena or derived from observations from which an understanding of the system behavior is sought. They seek to understand and explore the structure and internal functionality of a system. These models enable highly accurate simulation of the dynamic behavior of the corresponding organisms, but this comes at the cost of more complex models with detailed structural and functional information of the system. These models are also not transferable among organisms, involving really complex state of the art research to build detailed descriptions and reconstructions of any nervous system.

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. Additionally, often the excessive detail of internal structure common to white-box models is avoided by black-box models that concentrate on peripheral responses, 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 thesis 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 without having knowledge of all the inputs, and an experiment on the performance of the models replicating different behaviors simultaneously was also performed.

1.2 Contributions

The main goal of this thesis is to test the ability of different data-driven black box techniques to model peripheral input-output behavior of nervous systems, with all tests being conducted on adult hermaphrodite *C. elegans* nervous system synthetic data. More specifically we will concentrate on machine learning techniques as a means to develop such black-box models. To this end the main contributions offered by this thesis are:

- A comprehensive and broad review of the available works on modeling the *C. elegans* nervous system's behavior;

- A clear review and explanation of the machine learning techniques usually employed to model time series data;
- A working implementation of all these models for the *C. elegans* data;
- Experiments concerning the impact of the sample rate of the data used on the different models;
- A comparison on the performance of these models with different sizes, including the number of parameters used, seeking to determine the simpler, but still accurate techniques, available;
- An experiment concerning the performance of the models when not all input stimuli given to the system are available and a test on the performance of the models on a dataset concerning two different behaviors.

1.3 Organization of the Document

This document is structured as follows:

Chapter 1 provides the introduction, which gives context and motivation for this work, stating also the main contributions and describing the organization of the document.

Chapter 2 contains an introduction and some background on the human nervous system and a description of the *C. elegans* and its nervous system, reviewing the available work on modeling this complex system. This chapter also describes which datasets were used in this thesis and how these were generated using the NEURON simulator.

Chapter 3 reviews the available work on modeling time series data using machine learning techniques, also explaining each of the machine learning techniques used in this thesis along with their advantages and disadvantages for the case study.

Chapter 4 discusses the methodology used in this work, explaining how the different models were implemented and how each of these models was evaluated. The tuning of the hyperparameters of the different models is explained and the corresponding results are reported. At the end of this chapter tests on the sampling rate used for the data on the different models are performed.

Chapter 5 explains the different experiments made, starting with a comparison on the performance of the different models. Then, the ability of the better performing models to predict the nematode's behavior when deprived of some inputs and also on more complex data. Finally, the last experiment performed in this chapter tests the ability of the best performing models to reproduce the behavior of the system, with data concerning two different behaviors of the *C. elegans*.

Chapter 6 concludes the thesis with a discussion on the results and directions for future work.

2

Modeling Neuronal Systems

Contents

This chapter provides general background and context on the human nervous system as well as the *C. elegans* nervous system used in this work, starting by a discussion on the human nervous system structure and reporting related work already done in order to model the it, as well as its behavior. This is followed by a description of the *C. elegans* and its nervous system, discussing how this system is organized and functions. Related work on modeling the *C. elegans* nervous system on different behaviors is also discussed.

The rest of the chapter focuses on the data used in this thesis, how it was generated and how it can be used in this work, so that it is clear which type of models can be used in the rest of the thesis.

2.1 Nervous System Background

The Human Nervous System (HNS) is one of the most complex systems available to study in biology, composed of trillions of cells distributed in a network throughout the human body [11]. This nervous system is composed of many structures which are intimately interconnected, but for convenience it is usually divided into two parts: the Central Nervous System (CNS) composed by the brain and the spinal cord and the Peripheral Nervous System consisting of the neurons that connect the CNS with the body's muscles, glands, sense organs and other tissues.

There are two types of cells composing the HNS, the Neurons and the Glial Cells. The Neurons are the functional unit of the nervous system and operate by generating electrical signals that move from one part of the cell to another part of the cell or to neighboring cells. They can also assume many forms, and are often divided into three classes, the Sensory Neurons, Motor Neurons and Interneurons. Sensory Neurons are the ones responsible for bringing information from the tissues and organs of the body towards the CNS, while Motor Neurons bring the information from the CNS to effector cells like muscles, gland cells or other cell types. Interneurons are the ones that connect neurons within the CNS and account for 99% of all neurons in the HNS.

As referred, neurons appear in a wide variety of sizes and shapes, but all share features that allow cell-to-cell communication, which is their main goal. An example neuron can be found on Figure 2.1, where three main components emerge, the cell body of the neuron, the dendrites and axons. The dendrites are a series of highly branched outgrowths of the cell which receive incoming information from other neurons. The axon is slightly more complex, extending from the cell body and carrying outgoing signals to target cells. An axon is usually divided into three main parts, the axon hillock, the axon collaterals and the axon terminal. The axon hillock is the part of the axon that starts at the cell body and is where in most neurons, the electrical signals are generated, which then propagate along the axon. Axons may also have branches which are called collaterals that increase the neuron's sphere of influence branching away from the main axon branch. Each branch also ends in an axon terminal

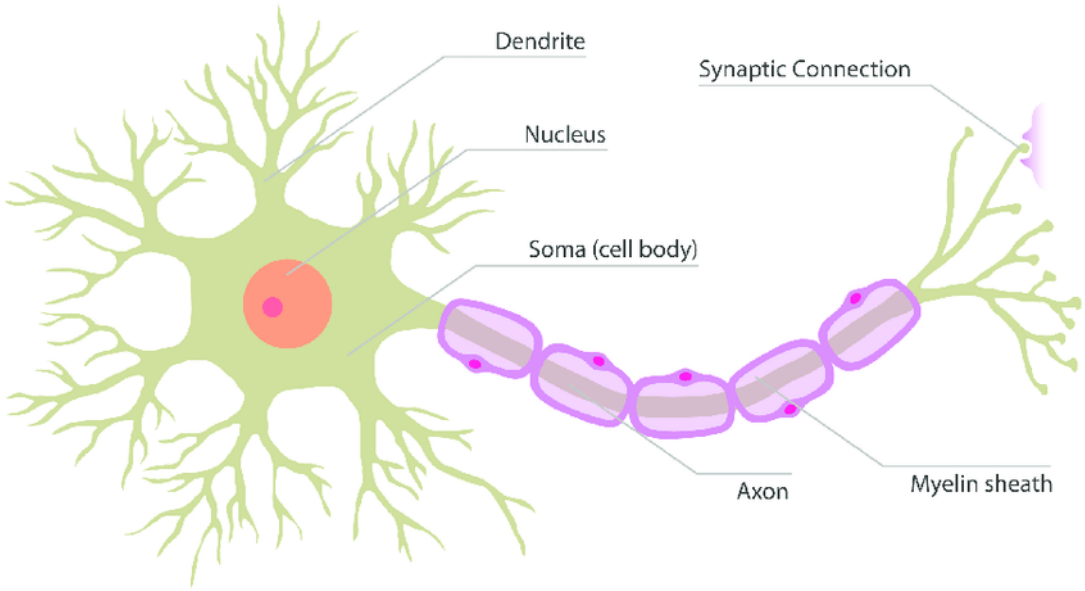

Figure 2.1: Example structure of a realistic neuron [1]

that is responsible for the connections to the exterior of the neuron. It is on the axon terminals that the neurotransmitters, which are generated by the electrical signals produced in the axon hillock, are sent through synapses.

Regarding axons, myelination is also an essential process in the formation of the nervous system, which begins before birth until adolescence, and consists of wrapping axons with an electrically insulating layer, built by a series of Glial Cells. This layer is represented on Figure 2.1 by the Myelin sheaths that cover the axon speeding up conduction of electrical signals along the axon.

These Glial cells that form the Myelin sheats do not participate directly in any electrical communication from cell to cell but serve to provide the axon with physical and metabolic support. The spaces between adjacent sections of Myelin sheaths that can be seen on Figure 2.1 are called Ranvier nodes. During the transmission of signals along the myelinated axons, the signals seem to jump from node to node, through a phenomenon called saltatory conduction [12].

One of the most important models obtained in the twentieth century is the Hodgkin-Huxley mathematical model for the transmission of electric current through the surface membrane of the giant nerve fibre, [13]. This is a model that was improved and adapted along the years on works like [14], where the author claims that one could simulate tens of thousands of cortical neurons in real time in a desktop PC. Nevertheless, this model is too complex to model large scale systems in the brain [15]. Other works try to use models as reduced as possible to simulate smaller parts of the neuron like dendrites [16] or myelinated axons [17]. Another important task besides modeling the functional unit of the HNS and its components is the one on modeling the relations between these components, the synapses [18], [19], [20]
.

Other research focuses on modeling the function of parts of the HNS, like [21], which provides a model of the CNS for patients with coronary arterial obstruction of at least 70% or [22] that models the simulation of human movement on tasks like jumping, pedaling, and walking.

This huge diversity of approaches just indicates the complexity of the task of modeling the HNS and its behavior, explaining why in order to surpass the complexity of modeling of the human brain, some works also focus on modeling the nervous system of smaller species like the *C. elegans* [23], the *Drosophila Melanogaster* [24], the *Carausius Morosus* [25], or the mouse [26]. This is the reason why the *C. elegans* is used in this work, since it is a simple organism commonly used as a benchmark in computational neuroscience, for which the connectome is completely mapped. The availability of the model used in [27] also allows for an easier generation of responses of the *C. elegans* nervous system to different types of stimuli.

2.2 Caenorhabditis Elegans

C. elegans is a small transparent non-parasitic soil nematode (roundworm), shown in Figure 2.2, with a cylindrical body of approximately 1.0mm in length and a weight of around $5\mu\text{g}$. The body of this worm contains the fundamental parts for its survivability such as a nervous system, muscles, a pharynx, a hypodermis, an alimentary canal and genitals. Despite the rather small size of its nervous system, it is able to process various stimuli from the environment, allowing to solve basic but essential problems for its survivability like feeding, toxin detection, mate-finding, predator avoidance or even forward and backward locomotion [28].

Figure 2.2: An adult hermaphrodite Caenorhabditis Elegans, image from [2]

The nervous system of this organism is really small when compared with the nervous system of others species like the *Homo Sapiens Sapiens*, consisting of less than 1000 cells across both sexes and around 15000 connections [9]. Nevertheless, as mentioned before, the ability of this simple organism to

solve basic problems together with the relative simplicity of its nervous system created the conditions for it to become a common benchmark in whole brain organization studies.

Another detail that makes this organism a good benchmark for techniques employed in computational neuroscience is the fact that its cell-lineage and anatomy are invariant. This means that every individual of this species possesses the same number of neurons and that they occupy fixed positions in the organism. While this happens for the cell-lineage and for the anatomy of the worm, the invariance of the synaptic connections between neurons is still under debate [29].

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 [30], [31], [32], online books [33], [34], atlases of neurobiology, structural and behavioral anatomy [35] and electron micrographs and associated data [36].

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 [37], [38], [39] and computational models [40], [41].

These models are based on the connectome, which is the map of the neuronal connections in the nervous system, meaning that these are white-box models based on direct knowledge and access to the internal structure and parameters' values of the *C. elegans* nervous system. The connectome can be seen as a graph where the neurons and synapses are represented by the nodes and the edges in it, respectively. For the *C. elegans*, the complete connectome differs from the hermaphrodite to the male, containing 302 neurons for the first [42] and 385 for the latter [43]. For the male *C. elegans* the complete 3D reconstructions of the neurons are not published yet [44], with only the the posterior nervous system 3D recordings of 144 neurons having been published [45].

Similarly to the human nervous system, the *C. elegans* nervous system has its neurons divided into three categories, sensory neurons, interneurons and motor neurons. The sensory neurons have their axons stretch out to sensory receptors in different parts of the *C. elegans* body like the tip of the head, the anterior part or the posterior part of the body, so it can sense chemical, mechanical, temperature and many other stimuli from the environment the worm is in. The interneurons receive the information perceived by the sensory neurons so that it is processed and passed on to other interneurons in the network or to motor neurons. Motor neurons are responsible for sending control signals to contract and relax muscle cells based on the information received, also sending feedback signals to interneurons [35].

In order to model this complex system or parts of it many different approaches have been suggested, using as a resource many different techniques. Some works try to model just part of the system, while

others try to model the entire system, showing how diverse is the research on modeling the *C. elegans* nervous system and its behavior.

The first type of approach discussed is the one in which only part of the system's behavior is modeled, where the majority of the works focus on two different tasks, chemotaxis and touch induced movement. Other works also try to model the *C. elegans* nervous system and its behavior on different tasks like nictation or temperature and smell based movement.

Chemotaxis is an essential task for the survival of any organism, usually defined as a fundamental biological process in which a cell, a group of cells or the entire organism moves in a direction that corresponds to a gradient of an increasing or decreasing concentration of a particular substance. This is a common task for the *C. elegans* when trying to detect toxins or find food.

Different works that try to model the *C. elegans* nervous system during chemotaxis were found, like [46], which is a work from Ferrée et al. where the authors use previous studies on the *C. elegans* locomotion to generate a reduced model of the movement of the worm during chemotaxis, based on the angle between the head and the rest of the body. Then they obtain a model for the control circuit based on already studied neuroanatomy and neurophysiology of the *C. elegans*, in order to replace the function of the nervous system during chemotaxis. This model is then optimized using simulated annealing, an optimization algorithm, so that the sets of synaptic weights that control chemotaxis can be obtained, and the results obtained with the control circuit and the movement model are then compared with success with the behavior of real nematodes in laboratory.

Another work from Ferrée and Lockery [47] follows a different approach, but still using the same reasoning for the nematode locomotion model as the one from the previous work [46]. A neural network is then used to model the *C. elegans* chemotaxis circuit in the nervous system, being composed of one chemosensory neuron, three interneurons and two motor neurons, one representing the dorsal and the other representing the ventral motor neurons used during chemotaxis based locomotion of the worm. This model is then linearized, optimized using simulated annealing and subsequently simulated and compared with the behavior of real nematodes in the laboratory, being able to produce similar behavior to the real nematodes. The authors then use the model to obtain computational rules that describe how the network controls chemotaxis.

Two other works from Lockery et al. that use simulated annealing to optimize a neural network model of chemotaxis based locomotion are [48] and [49]. The first of these, [48] uses a model composed of a sensory neuron, representing all the chemosensory neurons in the *C. elegans*, eight interneurons, representing all chemosensory interneurons in the worm and one motor neuron representing the four motor neurons that regulate turning behavior. They modeled this network 50 different times for 10 different worms, generating 500 different networks that were studied. By studying these networks they found that most of them could be pruned to have only one, two or three interneurons. The obtained models

were also studied in order to understand the functions of the different synaptic connections. On the other work from Lockery et al. [49], the authors optimized 50 different neural networks to model chemotaxis tasks. Each of these networks is composed of two sensory neurons, one interneuron and one output neuron for which the output would be 1 if the worm would run and 0 if it would turn. A new algorithm was also developed, Neural Dynamic Clustering, to identify neural dynamic structural motifs (i.e. common connectivity patterns) in the obtained neural networks.

A different type of approach, also used to model the *C. elegans* nervous system and its behavior during chemotaxis, are the ones that use learning algorithms to train a neural network so it can reproduce the behavior of the part of the nervous system responsible for these tasks. One work using this strategy is [23], where a neural network with seven neurons is trained using Real Time Recurrent Learning (RTRL) to model different chemotaxis behaviors. Two different strategies are tried, one that uses one neuron to replicate the sensory neuron as input and one to replicate the behavior of a specific interneuron and another strategy that uses two sensory neurons as input. Both of these strategies use two neurons in the network to output a voltage of the left and right neurons that control the movement of the worm. The model in this work is simulated with success, but the results are not compared to the ones of a real worm, instead testing the ability of the model to succeed in tasks as finding food, avoiding a toxin, or finding food and avoiding a toxin at the same time.

Another work that uses a learning algorithm to train a neural network is [50], where instead of RTRL, it is used Backpropagation Through Time (BPTT) [51] which was also used in this thesis. The network trained with BPTT is designed in the same way as the one reported in literature for the part of the *C. elegans* nervous system that handles the worm's attraction to sodium chloride. The model was then tested, proving to be able to calculate the temporal and spatial gradients of sodium chloride concentration. The ability of the model to perform with ablation of one of the two sensory neurons was also tested in this work, proving this to be a good model to study the behavior of *C. elegans* nervous system during a specific chemotaxis task.

C. elegans is able to sense touch stimuli in its body, which is another behavior that many works in the literature try to model, where gently touching the worm in the anterior or posterior part of the body often generates backward or forward locomotion, respectively. This is an important behavior to model since this allows the worm to avoid harmful external conditions and to provide the adequate escape response to this stimuli. Although certainly there may be more approaches to model this behavior, three main approaches to model touch induced locomotion on the *C. elegans* where found and are discussed next.

The first of these approaches consists on creating both a virtual body model for the worm and a model for the part of the nervous system responsible for touch induced locomotion. One work that follows this is from Suzuki et al. [52], that starts by developing a neural network model based on the already studied part of the *C. elegans* nervous system that handles this task. The used network is composed of three

sensory neurons for anterior touch and three sensory neurons for posterior touch, ten interneurons and two motor neurons, one for the anterior movement and another for posterior movement, representing the fifty motor neurons responsible for this behavior in the real *C. elegans* nervous system. Then a virtual body model composed of twelve joints is developed and integrated with the developed network model that controls the body model. The integrated model behaved well in simulations, proving that it is able to reproduce the behavior of the nematode in this task.

Also from Suzuki et al., [53] builds a twelve joints dynamic body model where the waving movement of the *C. elegans* during forward and backward locomotion is reproduced, being controlled based on stimulation information. To reproduce the rhythmic behavior of this movement was used a rhythm generator that controls the first joint of the model. This first joint corresponds to the pharyngeal muscle of the worm. The other eleven joints of the body model correspond to the body-wall muscles, which were controlled using a neural network similar to the one used in [52], but adding another input to the interneurons layer, a wave generated based on the output of the rhythm generator. The complete model was subsequently simulated proving to be capable of realizing motor control similar to the actual worm, but no comparison was made regarding the similarity between the movement in space produced by the model and by a real nematode in laboratory.

A second type of approach to model this behavior focuses on the fact that most approaches ignore, the influence of the rest of the nervous system not being modeled on the one being modeled. Iwasaki et al. [54] presents a stochastic formulation for a framework that can be used to study large biological neural circuits. This framework models just part of the nervous system, but not ignoring the rest of the network, treating it as noise. The authors use this framework to obtain a model for the touch sensitivity circuit of the *C. elegans*, effectively reducing the size of the original system. The produced model is also used to study the sign of the synapses in this part of the worm's nervous system. Another work from Iwasaki [55] obtains a way to model a fraction of a nervous system, without ignoring the remaining neurons in the network, using dynamical mean-field approximation. The obtained model has a restriction, assuming that any neuron connected with a neuron in the part being modeled cannot be connected with a different neuron belonging to that part of the network. The author tests with success the ability of the model to reproduce the behavior of the same touch sensitivity circuit of the *C. elegans* used on [54].

Another way to model touch induced locomotion of the *C. elegans* uses CMOS transistors to build circuits that replicate some basic functions of the nervous system [56], [57]. The basic blocks of these models are circuits that model excitatory and inhibitory synapses, the axon hillock and on [57] is also used a circuit for a rhythm generator. The first of these two works, [56] uses these circuits to build two different models, using two types of synapses, one implemented with circuits that replicate the behavior of spike-responsive synapses and another with circuits that replicate the behavior of graded-potential synapses. These models are used to replicate the touch induced locomotion behavior studied

in literature for anterior and posterior touch induced movement, proving to model well the behavior of the organism. On [57] a similar approach was followed, but modeling undulating locomotion using a rhythm generator at the start and end of the worm's body model used.

Other works also try to model the behavior and structure of the *C. elegans* nervous system on different tasks besides chemotaxis or touch induced movement. One of this works [58], presents a neuronal model for directional control of a body model based on environment stimuli for movements as turning and steering. When feeling a comfortable temperature or a good smell, the worm tends to move in that direction, also tending to steer or turn during any movement if a bad smell or not comfortable temperature is detected. This work handles this issue by modeling the part of the *C. elegans* nervous system responsible for this behavior, using a model composed of eight sensory neurons, two for temperature, four for good smell and two for bad smell, six interneurons and four motor neurons. The four motor neurons represent the four groups in which the fourteen motor neurons of the pharynx are divided, left and right dorsal, left and right ventral. The model is then integrated with a twelve joints body model, which proves to be controlled with success in the computer simulations performed, replicating the sinusoidal movements of the worm.

Another relevant work, [59] obtains a neural network model for the part of the *C. elegans* that produces responses to aversive stimuli. This is a big part of the network, which is composed by 126 neurons and modeled by a network described using the McCulloch-Pits model to reproduce the dynamics of biological neural systems. The network is optimized using a Monte Carlo optmization process and is then used to study the synaptic polarity of the connections, finding which ones should be excitatory and which ones should be inhibitory.

An approach that was tested on different worm behaviors was also developed [27], with the main goal of obtaining reduced models without this implying a loss of accuracy. On this article the authors start by developing a model of the *C. elegans* using the NEURON simulator [60] and test the behavior of the model in four different scenarios. The results produced were in concordance with the ones from literature about the behavior of real *C. elegans* on all the four scenarios, one referring to a nictation task and the other three referring to forward crawling motion (one with a normal worm and two others with specific sensory neurons or interneurons ablated). It is used a black box model reduction technique, more specifically Proper Orthogonal Decomposition, to replicate the behavior of the *C. elegans* original high fidelity model, producing a reduced model of the worm's behavior on these tasks with good prediction capabilities.

There exist also more general and ambitious approaches where the goal is to model the entire *C. elegans* nervous system [61]. In this work, the authors provide a detailed demonstration of a framework they created for simulation and visualization of the worm's neuromuscular system in a 3D physical environment. This framework includes a realistic flexible body model, with a muscular system and a small

part of the nervous system, having the goal to include a complete model of the worm's nervous system in their framework in the future. These components of the nematode interact together, also interacting with a simulated physical environment which allows the simulated nervous system in the framework to receive updated sensory data during simulation.

Throughout this chapter, different approaches that try to model either the entire *C. elegans* nervous system or just part of it were discussed, targeting different behaviors as touch induced locomotion, chemotaxis, temperature and smell based movement, response to aversive stimuli or nictation. Some of these approaches build just a model of parts of the nervous system, while others try to integrate this first type of models with models of the worm's body. Some are based on domain knowledge of the nematode at study, while others are completely black box and data driven using no domain knowledge. This diversity of approaches already employed to model the *C. elegans* nervous system structure and the also diverse set of behaviors is immense, proving how even modeling a simple and compact nervous system like this one is a challenging task.

There is one detail that is common to many of the different works seen: the attempt to also model the structure of the nervous system. This is something that is not as feasible for large scale systems as it is for this one, since it is harder to have as much domain knowledge on large scale systems as there is available for *C. elegans*. Hence, the structure of the nervous system is not modeled in this work, since the goal is to test the ability of different scalable machine learning techniques to obtain models as reduced as possible for the behavior of the *C. elegans* nervous system in different tasks.

The next sub-chapter focuses on the data used in this work, discussing the model used to generate it, that was the same model used in [27], while also discussing the different behaviors of the nematode that are modeled in this thesis. It also provides a discussion on the parts of the nervous system involved in the behaviors of forward crawling motion and nictation being modeled.

2.3 Data

In order to train and test the machine learning models discussed in the next chapter, were required datasets containing information regarding the peripheral input-output behavior of the *C. elegans*. The availability of 3D neuron reconstructions and computational models of this nematode allowed for the use of synthetic simulated data. This is an important detail, since the simulator and the model used allow for an easier generation of peripheral input-output data of this nematode, which would not be such an easy task using real worms in a laboratory.

The synthetic data used in this work was generated using the NEURON simulator [60] and the same model used in [27], with the help of the authors of this last work. With the goal of testing the model's performance replicating different behaviors of the *C. elegans*, there were generated datasets regarding

two different scenarios.

The two different scenarios, which refer to different behaviors, are a Posterior Touch Response Stimulation (PTRS) scenario and a Nictation scenario. For each of these behaviors different datasets were generated in order to perform the tests on the impact of the sampling rate in subchapter 4.4 and the experiments described in Chapter 5. At the end of the chapter, a dataset combining data from the two different scenarios is also created.

Each dataset was divided into three parts, the training, validation and test sets. The first one is used to train the models discussed in Chapter 3, while the second is used to validate the models and tune the hyperparameters and test the effect of the time step on the models, as explained in Chapter 4. The last one, the test set, is used to test and compare the performance of the different models in Chapter 5.

Next, the two scenarios are discussed with an additional description of the generated datasets.

2.3.1 Posterior Touch Response Stimulation

In this scenario described on [62], the response of the *C. elegans* when stimulated in the posterior part of its body by touch is represented, which is known to produce a forward crawling response of the worm. This behavior of the nematode's nervous system usually involves stimulation of two types of neurons, PLM sensory neurons, PLML and PLMR, and AVB interneurons, AVBL and AVBR. The PLM sensory neurons are known as posterior mechanoreceptors, which excite the motor neurons associated with forward crawling motion when stimulated by tail touch. The AVB interneurons are 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 [27]. Since this work has the purpose of testing different machine learning techniques, it was decided to use only a subset of these neurons for the study due to the computational costs that would be added by measuring the output on all the affected neurons. For this case, a sample known to be large enough, composed of 16 neurons was chosen, since these specific neurons are known to show strong responses during the behavior being studied. The neurons used are AS1, AS7, DA1, DA9, DB1, DB5, DD1, DVA, LUAL, PHCL, PLML, PVCL, PVR, SIBVL, VB1, VD1.

In order to tune the hyperparameters of the different models as described in subchapter 4.3, a base dataset concerning this behavior of the nematode was designed, **PTRS-20-0.5** with 50% of the data for the training set and 25% for the validation and test sets. This dataset has a total of 40 different sequences each with a timestep of 0.5ms between samples, generated based on different stimuli. The division of the sequences in each of these datasets was done randomly.

To generate this dataset, various different stimuli were chosen in a random way, combining different pulses and ramps as current inputs to the PLM and AVB neurons of the *C. elegans* nervous system. The

inputs used had a maximum current of 1.4nA for the PLM neurons and of 2.3nA for the AVB neurons. To also keep the computational costs as low as possible due to the amount of tests run, in this first dataset, was used only data of a subset of the output neurons measured. The output data used for Chapter 4 concerns only the data of the neurons DB1, LUAL, PVR and VB1. An example of three of the sequences generated can be seen in Figure 2.3, with the corresponding inputs and outputs.

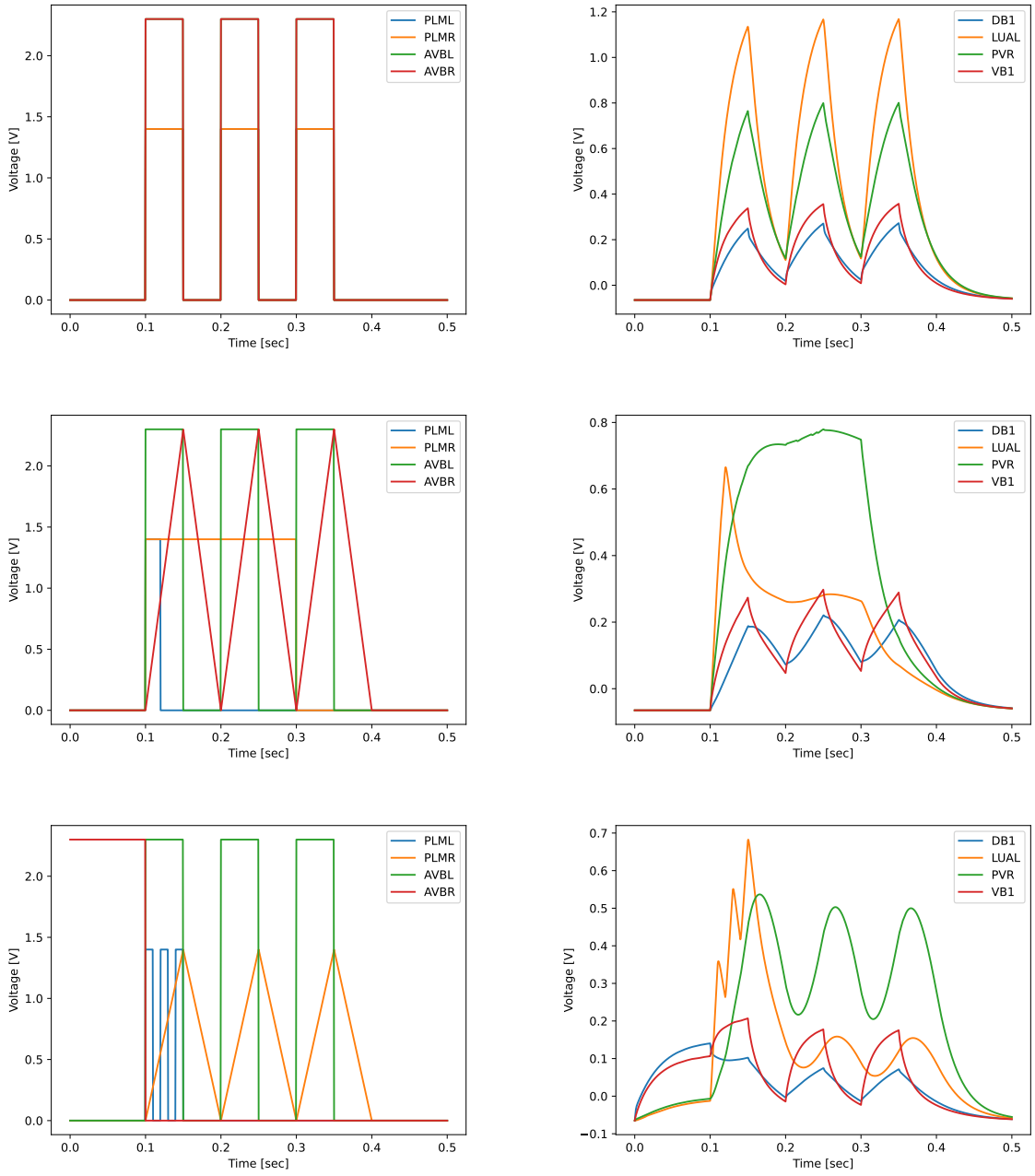

Figure 2.3: Example inputs(left) and outputs(right) for three different sequences of **PTRS-20-0.5**, respectively.

After the hyperparameters optimization discussed in subchapter 4.3, two of the machine learning techniques were found to have difficulties modeling the response of the *C. elegans* nervous system. This inspired an attempt to test how the time step between samples of the generated data affects the performance of the models used, which was performed in subchapter 4.4.

So, in order to test the effect of the time step between samples, five new datasets were designed based on the one used in subchapter 4.4, **PTRS-20-0.5**. The only difference between these datasets is the time step used. The six datasets that were used in subchapter 4.4 are described below:

- **PTRS-20-10** - The time step used is of 10ms
- **PTRS-20-5** - The time step used is of 5ms
- **PTRS-20-1** - The time step used is of 1ms
- **PTRS-20-0.5** - The time step used is of 0.5ms
- **PTRS-20-0.1** - The time step used is of 0.1ms
- **PTRS-20-0.05** - The time step used is of 0.05ms

After the tests performed in subchapter 4.4, was chosen to use a time step between samples of 0.5ms, so the dataset **PTRS-20-0.5** was used on the comparisons of the following subchapters: 5.1 and 5.2.

With the goal of finally testing the ability of the best performing models on behavior more similar to the one of the *C. elegans*, a final dataset is generated for this task: **PTRS-Final**, used in subchapter 5.3. This dataset has a time step of 0.5ms, but instead of just having the measured output on 4 different neurons, it has the measured output on 16 different neurons.

2.3.2 Nictation

The Nictation behavior 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 [63]. 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 data related to this behavior, which is discussed next, is generated using different combinations of pulses and ramps to the IL2 neurons in a random way, using a maximum current as input on these neurons of 4.8nA.

Since the data on this behavior was generated later in the study, not being used on the hyperparameter optimization in subchapter 4.3, the six different datasets were generated simultaneously for the tests performed in subchapter 4.4. An example of three sequences generated, which were included in the **Nictation-20-0.5** dataset, can be found in Figure 2.4, with the corresponding inputs and outputs. In these datasets only 6 of the measured outputs were included in order to reduce the computational costs, similarly to what was done for the PTRS scenario: RMED, RMEL, RMER, RMEV, RMGL and RMGR.

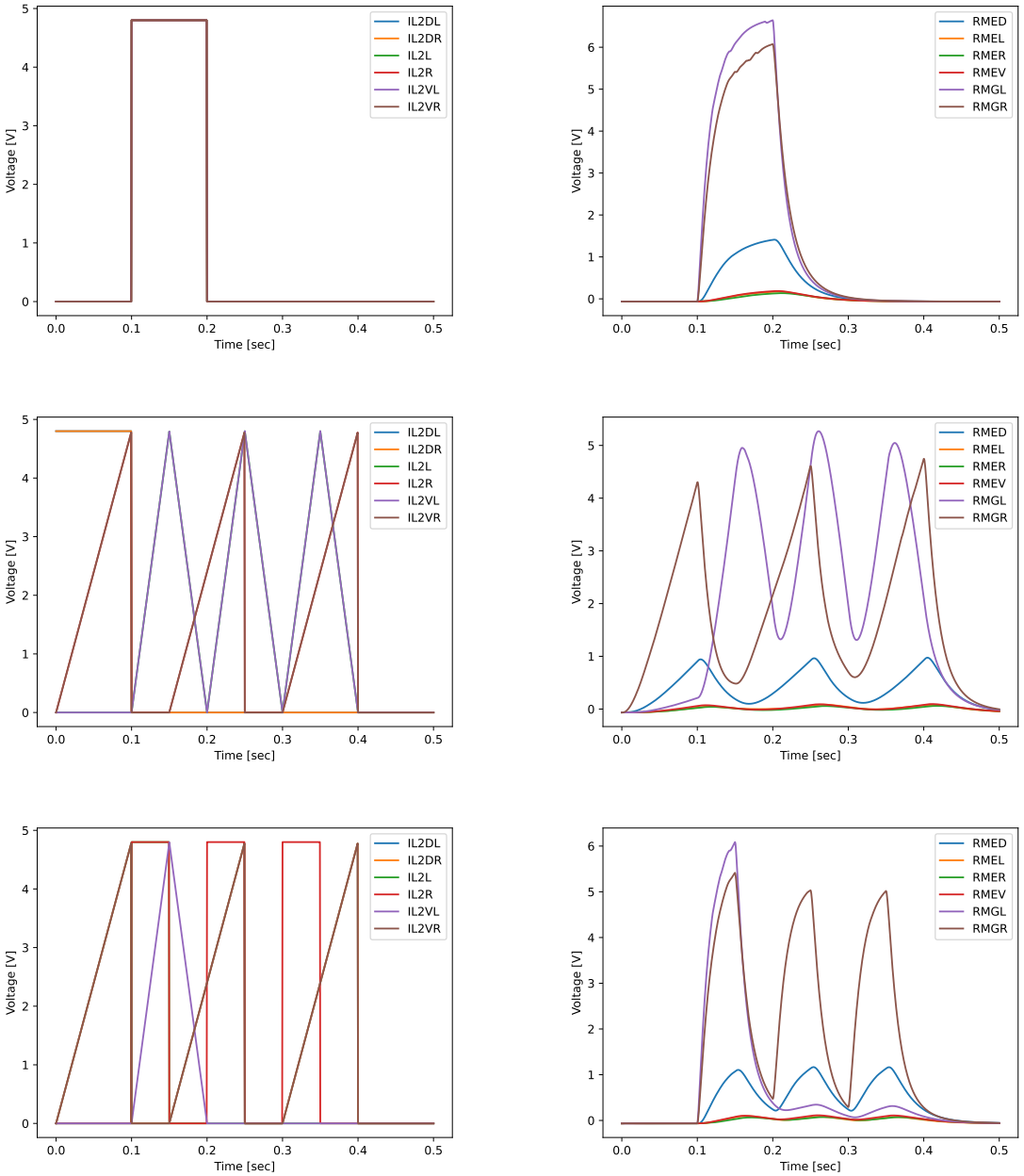

Figure 2.4: Example inputs(left) and outputs(right) for three different sequences of **Nictation-20-0.5**, respectively.

These six datasets were also generated in order to understand how the use of different time steps affects the performance of the different models in the tests performed in subchapter 4.4, similarly to what was done for the previous behavior. Also similarly to what was done for the previous behavior, the six different datasets referred were generated using the same examples, just changing the time step used as it is described below:

- **Nictation-20-10** - The time step used is of 10ms
- **Nictation-20-5** - The time step used is of 5ms
- **Nictation-20-1** - The time step used is of 1ms
- **Nictation-20-0.5** - The time step used is of 0.5ms
- **Nictation-20-0.1** - The time step used is of 0.1ms
- **Nictation-20-0.05** - The time step used is of 0.05ms

After the tests performed in subchapter 4.4, was chosen to use the dataset with a time step between samples of 0.5ms, **Nictation-20-0.5** in subchapters 5.1 and 5.2. Similarly to the previous behavior, with the goal of testing the ability of the models to perform with more outputs, a final dataset is generated: **Nictation-Final**, used in subchapter 5.3. This dataset has the output measured for the 12 neurons referred before, using a time step of 0.5ms.

With the goal of testing the ability of the models to perform on a dataset composed of data of two completely different behaviors, it is created a final dataset: **Two-Behaviors-Data**. This dataset combines both **PTRS-Final** and **Nictation-Final**, using the sequences present in both datasets, but now having $4 + 6 = 10$ inputs and $16 + 12 = 28$ outputs.

3

Machine Learning Techniques

Contents

This chapter focuses on the different machine learning techniques that will be applied to time series data. First there is a discussion on the modeling problem being solved using time series data, followed by a review of available work on machine learning for similar tasks. Next, a review of available work on forecasting time series is done, along with a conclusion on which machine learning methods should be used for this case study. The different machine learning techniques chosen to model the generated *C. Elegans* data are then discussed, with a corresponding subchapter for each of the techniques applied. Each of these subchapters presents and discusses the given technique in detail, along with its advantages and disadvantages for the work at stake.

3.1 Machine Learning for Time Series Data

One of the most important parts of modeling a system's behavior is choosing which techniques to use, so that the adequate ones are chosen, avoiding approaches that are clearly not suited for the task at stake, not even with some adaptation. In order to do this, a correct problem definition must be formulated to better guide the selection of correct techniques.

To correctly formulate the problem being solved, one must look at the generated data discussed in subchapter 2.3. This data takes the form of a multivariate time series composed of n variables, $\mathbf{X} = (X_1, X_2, ..., X_n)$, where $\mathbf{X}$ is the multivariate time series and X_1, X_2, ... X_n are the different univariate time series that compose $\mathbf{X}$. These univariate time series are all composed of N data points.

From the different time series that compose $\mathbf{X}$, n_i are input time series that will be used by the different machine learning techniques to predict the other $n_o = n - n_i$ time series, which are the output ones. This means that the type of models discussed next must be models that are able to learn how to use n_i different time series to obtain n_o corresponding time series, all of them composed of N data points.

This is clearly a sequence to sequence modeling task, that can be used to explain many dynamical systems found in nature. However, the vast majority of related work on modeling time series using machine learning techniques is not focused on solving this problem. Instead, most research on modeling time series data using different machine learning techniques corresponds to forecasting problems, where researchers try to predict one or more points ahead on a given time series.

Forecasting is usually employed in many different fields. One of these fields is meteorology, predicting temperature, pressure or other physical quantities values for a future time period, given its historical values. Another field where forecasting is common is finance, with researchers predicting stock price values based on historical values. Besides these examples, there exist other fields where forecasting is common such as energy demand prediction, disease spreading prediction, traffic prediction, among others.

There are two angles one can use to solve this problem using machine learning, either using techniques that already have been proven successful in solving sequence to sequence problems or adapting other successful techniques applied to forecasting time series. Based on this, the current subchapter was separated into three parts, one concerning a review of techniques that are usually employed or developed for sequence to sequence problems, another for reviewing works on forecasting time series and a third one on conclusions of the available work and techniques that should be used.

3.1.1 Sequence to Sequence Modeling

This review concerns two different types of works, ones that develop or discuss techniques that may be used in sequence to sequence modeling and others that apply one or more techniques to a given task in order to find which is best suited for their specific application. All the works reviewed in this section discuss specific neural network architectures used to tackle this type of problems, most them focusing exclusively on recurrent neural network architectures.

Starting with the works that don't have a specific task for application in sight, there is an article where the authors give an overview on different dynamic neural networks that employ feedback connections between the neurons of a given layer, and/or between the layers of the network [64]. The techniques reviewed are deemed appropriate for tasks such as system modeling, identification, control or filtering applications. The authors divide these techniques into two different groups. One of these groups corresponds to network architectures that have the single neuron as its basic unit: Brain-State-in-a-Box, Dynamic Neuron Unit, or Recurrent Neural Networks (RNNs) and Time Delay Neural Networks (TDNNs) which will be later discussed in depth and used in this thesis. The other group corresponds to techniques that are based on the interaction of different neural sub-populations, such as the Positive-Negative Processor or the Dynamic Neural Processor.

A neural network based approach for modeling time-dependent data and learning approximations to non-linear dynamical systems using ordinary and partial differential equations was proposed in [65]. The authors test their approach on data extracted from various dynamical systems based on differential equations. The approach is also tested on different image classification tasks, proving that it is able to maintain accuracy while reducing the number of parameters used.

Another work [66] proposes a Multi-layered Echo State Machine architecture and algorithm in which multiple RNNs have their parameters generated randomly as the core of the architecture. The authors test the ability of this model to perform with different types of data, some based on differential equations, while also testing in time series classification problems, among other different tasks.

With a major focus on data driven Model Order Reduction for time dependent problems, the authors of [67] develop a technique based on artificial neural networks that may be applicable to different dynamical systems based on ordinary and partial differential equations. The developed network is later

adapted, having a similar structure to RNNs after the time discretization. It is also trained with the Levenberg-Marquardt Algorithm (LMA) that was also later used in this thesis to train TDNN and Neural Autoregressive Exogenous Model (NARX). The model proves to perform well on modeling three different test cases: the nonlinear pendulum, a nonlinear transmission line circuit and the heat equation.

Another developed model for time series prediction uses a strategy that is becoming common when developing these techniques, which is creating adaptations to the RNN based on the Long Short-Term Memory Unit (LSTM), that will be discussed later in this chapter and used along the rest of this thesis. One work that follows this strategy is the one from Jin et. al. [68] where another technique based on neural networks is proposed, combining a Convolutional Neural Network (CNN) with a Bidirectional Long Short-Term Memory Unit (BiLSTM). The CNN is used to extract the relation between different univariate time series that compose the multivariate time series dataset. The BiLSTM is a LSTM based RNN composed of two layers, one processing the data in a forward direction and the other in the opposite direction. This BiLSTM is used to extract the temporal relationships between the different data points. The model developed is experimented upon using Beijing meteorological data, obtaining high accuracy in predicting wind speed and temperature data.

In order to study the human genome, another work [69] also proposes the use of a BiLSTM on a genome dataset to perform sequence alignment. The author decomposes the reference human genome into multiple sequences that are used to train a BiLSTM, obtaining fixed length vectors that can be used for sequence alignment.

One article that also uses LSTM layers for a sequence to sequence modeling task, similar to the one performed in this thesis is [70]. In this paper it is used a Seq2Seq model, which is a common state of the art model in machine translation. This model is composed of two LSTM layers, the first as an encoder and the other as a decoder. This model is then applied to predict the temperature in a firing furnace process, where the gas temperature, voltage, current and resistance data from the heating furnace is used as input to predict the temperature in the firing furnace, which ends up being a similar time series problem to the one being solved in this thesis.

3.1.2 Forecasting Time Series

Looking at the more abundant applications of machine learning techniques to forecasting problems, the articles studied can be divided into three main categories. The first category is composed of articles that provide comparisons on different machine learning techniques, while the second one is composed of works that just apply an already studied technique with some adaptation or not to solve an existing problem. The third and last group is composed of articles that propose new techniques to improve the results obtained on forecasting problems.

Starting with the works that compare different machine learning techniques, [71] performs a compar-

ison of a Multilayer Perceptron (MLP) and a NARX with classic time series modeling techniques, such as the well known Moving Average (MA) or Autoregressive Integrated Moving Average (ARIMA). The authors compare the ability of these models to predict the inventory levels of refrigeration compressors, where the NARX shows a superior performance compared to the other three methods.

A comparison between a large set of machine learning techniques was also performed using the M3 competition dataset, composed of 3003 time series, from which only 1045 monthly time series were used [72]. The set of machine learning techniques used in this comparison is composed of eight different techniques, namely: a MLP, a Bayesian Neural Network, a Radial Basis Function Neural Network, a Generalized Regression Neural Network, K-Nearest Neibors Regression (KNNR), Support Vector Regression (SVR) and Gaussian Process Regression (GPR). This work also evaluates the influence of three different preprocessing techniques previously applied to the data, predicting the last 18 points of each of the 1045 different time series, where the MLP and the GPR show the best performance.

Another work from Bontempi et. al. [73] discusses different machine learning strategies for forecasting time series, such as the supervised learning setting, but also discussing specific algorithms such as the k-nearest neighbors and a lazy learning algorithm. The article also discusses different strategies for multi-step forecasting, but does not perform any comparison of the different strategies on any type of data.

Since snowmelt is one of the major sources of fresh water in the world, the authors of [74] had the goal to predict snowmelt-driven streamflow using data from Central Himalayas in Nepal. For this matter, the authors compare different machine learning techniques, LSTM, NARX, GPR, SVR. The developed models use as input the snow covered area, the temperature of the air, precipitation and ascendant discharge time series, being trained with data from 2002 to 2010 in this area. The models were also tested with data from 2011 and 2012, where GPR and SVR do not seem to show a big difference in performance. NARX and LSTM showed a superior performance, with the latter performing slightly better.

A comparison was also performed between Deep Echo State Networks (DeepESNs), RNNs, LSTM and the Gated Recurrent Unit (GRU), in terms of both prediction accuracy and efficiency [75]. The comparison was made on four different polyphonic music datasets composed of high dimensional time series. The results indicate that the DeepESN performed better than the other techniques, not only efficiency-wise, always taking the least computation time, but also yielding the higher accuracy on three of the four datasets. On the other dataset, the GRU is the model that has the better test accuracy, but still taking much more computation time than the DeepESN.

There are also applications to groundwater level forecasting, where [76] compares NARX, LSTM and CNN on groundwater level forecasting, using data from 17 different groundwater wells in the Upper Rhine Graben area. The models are compared using only meteorological inputs, but also adding past groundwater levels as input, where the NARX tends to perform slightly better than the other two models.

Other research articles just apply one already existing technique, sometimes adapting it, to solve an existing problem in some area of research, such as [77]. In this work, the authors apply a two layer LSTM model with the goal of detecting faults in industrial cyber-physical systems using as an example a gasoil plant heating loop. The LSTM model forecasts the values of the reservoir tank level and temperature and the heating tank temperature, which are then compared with the values measured in real time. Whenever the error of the forecasted values exceeds a certain threshold, the model will flag it as a detected fault in the system.

With the main goal of assisting policy makers during the spread of Covid-19 in Qatar, [78] proposes a new approach to forecast the total of cases of this disease. A k-means algorithm to group countries with similar demographic and socioeconomic factors is used at first. Then the data of countries in the same cluster as Qatar is used to train a BiLSTM. This approach is subsequently tested on Covid-19 and related restrictions data, being compared with other forecasting approaches, where it proved to be successful.

A Seq2Seq model that is suited for time series was adapted in [79], where the authors propose a Seq2seq model that integrates two attention mechanisms. This model introduces an input attention layer that uses the previous encoder state as input, followed by an activation function layer and the encoder. Between the encoder and the decoder the model also has a temporal attention layer, which uses the previous decoder state as input, and an activation function layer. This model was tested on a temperature and on a stock price forecasting datasets, proving to produce better results than other state of the art techniques.

An adaptation to a Seq2Seq model was also employed in [80] to detect unexpected events that disrupt an ecosystem in near real time. The proposed model uses a BiLSTM layer as an encoder, followed by an attention mechanism layer and a LSTM layer as a decoder. The forecasted values predicted by this Seq2Seq model are then compared with the real ones, flagging them as a disturbance when there exists a noticeable difference. This model was then tested on data from two different forest areas in China, which had large burned areas, proving to perform better than other state of the art models.

Looking at models that were specifically proposed in order to better model time series data with no specific application in mind, [81] proposes an adaptation to the LSTM unit that also takes into account the correlation between different univariate time series that compose a multivariate one. The authors duplicate the existing set of gates in the LSTM unit, so that one set of gates helps modeling the relation between univariate time series and the other set models the temporal relation in each time series. To evaluate this model, two real world datasets were used, one regarding passenger demand prediction for aviation and another regarding air quality prediction, showing that the model performs better when compared to other state of the art models.

In order to create a superior approach, other works combine different models already known to perform well [82], where a NARX is combined with a LSTM layer, creating a model for photovoltaic power forecasting. It uses temperature, relative humidity, wind speed and solar radiation as inputs for the NARX, which produces a preliminary forecast of the photovoltaic power. This value is then used as input, along with the other four inputs for the LSTM layer, that will obtain the final forecast of the model. The authors train the model with two years of data on this matter from Australia and test it on one year of data, proving to perform better than a tree regressor, a KNNR model and a simple LSTM.

With the goal of improving the accuracy of multivariate time series forecasting, a Spectral Temporal Graph Neural Network is proposed in [83]. This model is based on Graph Fourier Transforms and Discrete Fourier Transforms, taking into account both the temporal correlations and the correlations between time series. It is also tested on ten real world datasets with topics ranging from traffic, energy, electrocardiogram or epidemiology domains. The comparison is made with other state of the art models and the model proves to outperform existing approaches.

Another work that proposes a state of the art model for this task is [84], which creates the LSTNet architecture that combines a convolutional layer, a recurrent layer and a recurrent skip layer. The authors also propose an adaptation to their LSTNet architecture using an attention mechanism, comparing these two models with other state of the art models. The comparison is made on four different datasets regarding solar energy, traffic, electricity and exchange rate of different countries, proving the efficacy of the proposed models.

3.1.3 Conclusions on Available Work

This review shows how dominant neural network models are on research for predicting time series, due to their versatility and ability to map non-linear functions given a sufficient amount of data. It is also clear that the most common approaches involve using RNNs, usually in the form of its LSTM unit. Other techniques used that do not involve neural networks are different regressors, such as SVR, KNNR, tree regressors or GPR.

Based on this review, it was decided to only test different neural networks in this work, since they show a clear dominance in time series prediction tasks when compared with the remaining techniques. In order to better model the time dependencies in the data, only neural network architectures that take this into account were used, with the goal of choosing the simplest architectures that fulfill this condition.

The first architecture chosen was the TDNN due to its similarity to the simple and common MLP. Following this, the NARX was also chosen since it is an adaptation to the TDNN already proven to perform better than the first, but slightly more complex with the use of a recurrent connection.

Due to their dominance in current research applications, the simple RNN and two of its variations, LSTM and GRU, were also used. Even though Seq2Seq models and more complicated architectures

involving more than one recurrent layer have already been proven very successful, these were not tested. This choice was made so that only simple models with one hidden layer are chosen to allow for an easier future scalability of the models, starting from the simplest possible models.

3.2 Time Delay Neural Network

The TDNN, which was first proposed for speech processing data on [85], is an adaptation of the classical MLP, [86], where time is converted into spatial representation. So, to better understand the TDNN, one must first understand its parent, the MLP, for which an example graph can be found in Figure 3.1.

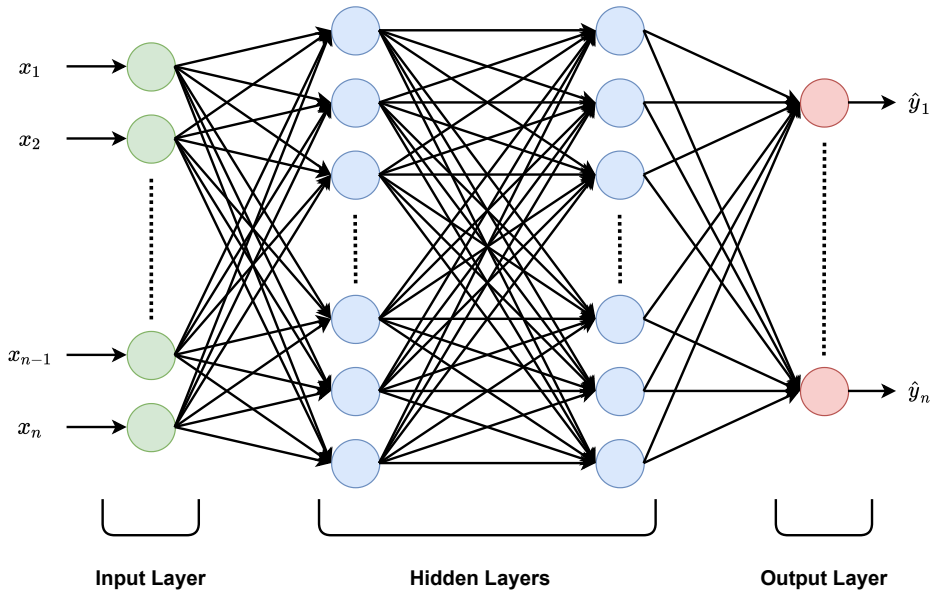

Figure 3.1: Multilayer Perceptron Graph with 2 Hidden Layers

The MLP is a well known model inspired by the human brain, which was developed over the years, starting in 1943 with the very first mathematical model of neurons [87], followed by the perceptron model that appeared in 1958 [88]. Subsequent research over the years led to the MLP, which is composed of an input layer, an output layer and one or more hidden layers as can be seen in Figure 3.1, where a MLP with two hidden layers is represented. Each layer is composed of a fixed number of neurons, with no connections among themselves, having only connections to the neurons in the next layers. Each of these connections has an associated weight and bias, so that the output of each neuron in a previous layer is multiplied by the weight and added the bias. Then, the subsequent layer adds all the neurons contributions and passes the result through an activation function, which will give the output of the neuron.

Equations (3.1), (3.2) and (3.3) define the example MLP shown in Figure 3.1, where $\mathbf{x}$ is the input of the network, $\mathbf{h}_1$ and $\mathbf{h}_2$ are the outputs of each of the hidden layers and $\mathbf{y}$ is the output of the network.

The parameters that the network has to learn during the training procedure are the weights $\mathbf{W}_1$, $\mathbf{W}_2$ and $\mathbf{V}$ and the biases $\mathbf{b}_1$, $\mathbf{b}_2$ and $\mathbf{c}$.

$$\mathbf{h}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \tag{3.1}$$

$$\mathbf{h}_2 = \mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2 \tag{3.2}$$

$$\mathbf{y} = \mathbf{V}\mathbf{h}_2 + \mathbf{c} \tag{3.3}$$

Usual activation functions used for the neuron unit that were used in this work are shown in equations (3.4), for the sigmoid, and (3.5) for the hyperbolic tangent.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.4}$$

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.5}$$

The MLP is trained by backpropagating errors through the network, a method that first appeared in [89], being improved throughout the years, and popularized after its improvement in 1988 [90]. This model is particularly useful due to its ability to model non-linear processes, finding applications in many areas, but it presents an issue when modeling multivariate time series data.

The model 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, therefore multiplying the total of input variables by $p + 1$, as can be seen in Figure 3.2.

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 all used as inputs. For the work done in this thesis, since the goal is to 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 [90].

This model, as the MLP, has its advantages and disadvantages, only the ones related to this work being highlighted here. So, for the work at stake, it is clear that this model benefits from maintaining the capacity of the MLP to model non-linear processes, while adding the possibility of looking at past values of the input. The downside of this improvement is that with higher delays, the model will become more complex, increasing the computation resources needed, both in space and time. Another issue that might appear with this model when compared to some of the other models used in this thesis 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.

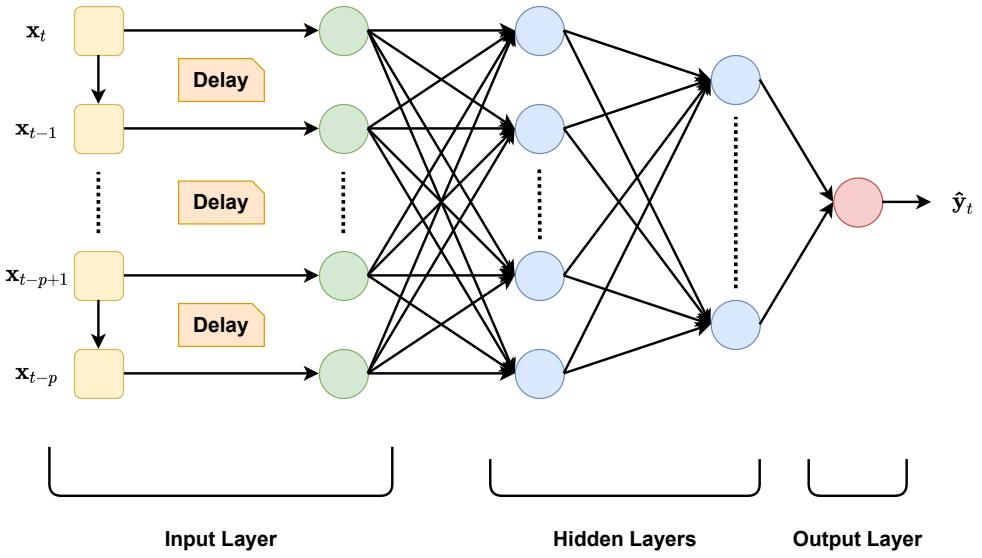

Figure 3.2: Time Delay Neural Network Graph with 2 hidden layers

3.3 Neural Autoregressive Exogenous Model

The NARX is a type of recurrent neural network [91], very similar to the TDNN, with the main difference consisting on the fact that it also uses previous predicted values of the output time series to predict the current value of the output, as can be seen in Figure 3.3.

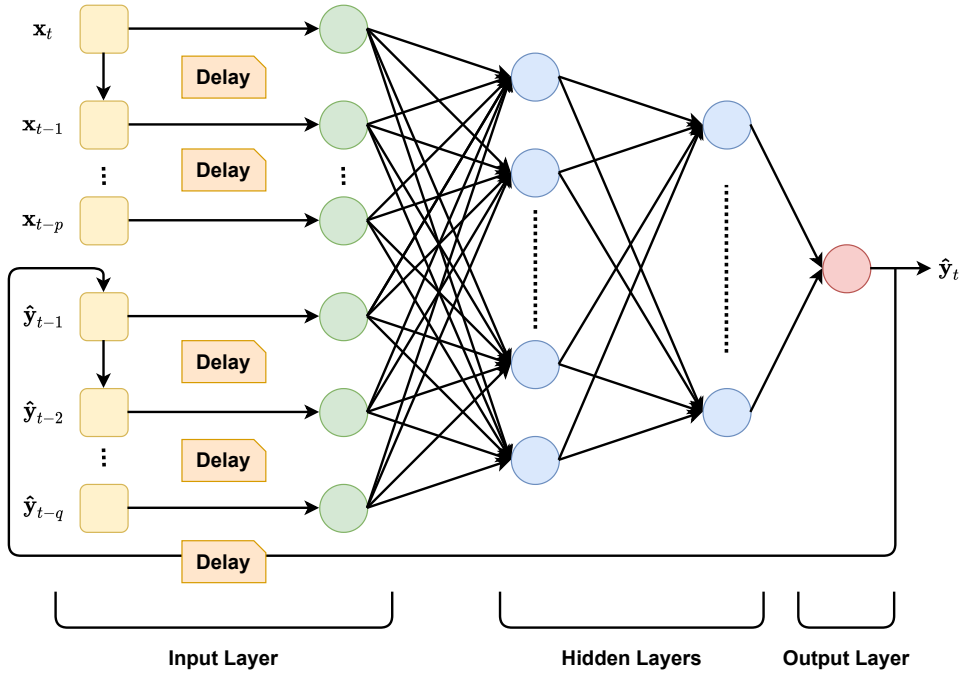

Figure 3.3: Neural Autoregressive Exogenous Model Graph for a Network with 2 hidden layers

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. These added variables give the NARX information on the system state, which is more useful for the task at stake, since the output of the nonlinear system not only depends on the input variables, but also on previous states of the system, that can be inferred from predicted outputs.

Differently from the TDNN, 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 [90], but with an important detail in mind, the recurrent connection is used open, as shown in Figure 3.4. This is done so that the model converges, 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. Therefore, by using the real output values instead of predicted ones, the training process is assured to converge, usually converging even faster.

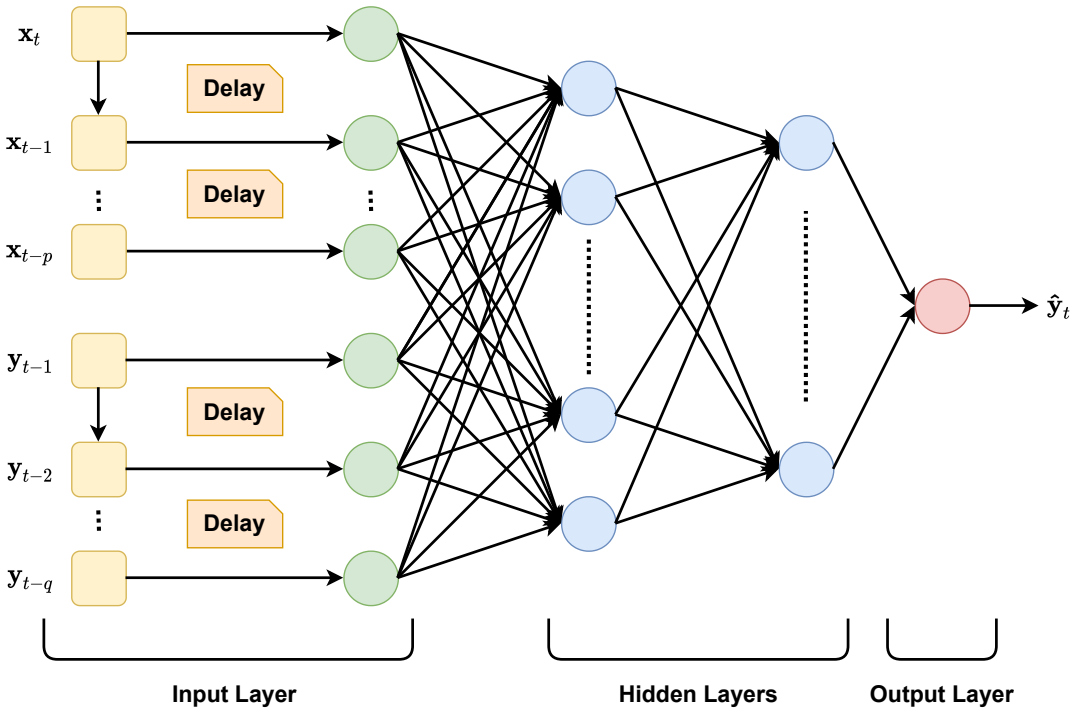

Figure 3.4: Neural Autoregressive Exogeneous Model Graph for the training procedure of a Network with 2 hidden layers

So, to sum up, the model is trained with a configuration as the one in Figure 3.4, then the loop is closed after the training procedure, and all the predictions are made with the loop closed as in Figure 3.3.

As far as its advantages and disadvantages go, the NARX by using predicted output values already

has knowledge of the system state, fixing a potential issue of the TDNN. An issue that the NARX still holds from its parent, the TDNN, is the scaling problem 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.

3.4 Recurrent Neural Network

RNNs are a family of neural networks that appeared in the 1980's to process sequential data [92], [93]. 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 3.5, where compact schemes of the MLP and the 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.

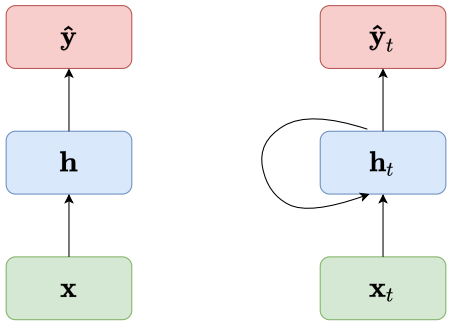

Figure 3.5: A compact scheme of the MLP graph vs a rolled RNN graph.

Regarding Figure 3.5, equations (3.6) and (3.7) describe the MLP, while equations (3.8) and (3.9) describe the RNN:

$$\mathbf{h} = \phi(\mathbf{V}\mathbf{x} + \mathbf{c}) \tag{3.6}$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b} \tag{3.7}$$

$$\mathbf{h}_t = \phi(\mathbf{V}\mathbf{x_t} + \mathbf{U}\mathbf{h_{t-1}} + \mathbf{c}) \tag{3.8}$$

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}. \tag{3.9}$$

In these equations $\mathbf{x}$ denotes the input vector, while $\mathbf{x_t}$ refers to the input vector values at time t, $\mathbf{V}$, $\mathbf{W}$ and $\mathbf{U}$ are weights learned by the model and $\mathbf{b}$ and $\mathbf{c}$ are biases, also learned by the model. 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 3.5 shows how similar the two structures are, which can also be seen by the similarities between equations (3.7) and (3.9) that have a really similar manner to compute the output. The main difference between equations (3.6) and (3.8) is clearly the inclusion of values from previous units in the same layer for the RNN. This difference, also apparent in Figure 3.5, is what makes the RNN suitable for time series data like the data used in this thesis, 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 BPTT [51], suffer from two main problems during the training procedure: vanishing and exploding gradients [94]. Exploding gradients refer to a large increase in the norm of the gradient during training, appearing due to the explosion of long term components. Even though this is the less common problem, there are known solutions to handle it such as the gradient clipping technique [95].

The vanishing gradient problem [94] happens when long term components go exponentially fast to norm 0, implying that distant events stop impacting the predictions of the model. This vanishing gradient problem is what creates the main problems when using RNNs, 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. Therefore, the RNN may experience difficulties in modeling the dynamical system's response with the desired accuracy.

3.5 Long Short-Term Memory

The LSTM is a gated unit used in RNN as a replacement for the simple one which is only composed of a hyperbolic tangent function. This unit was proposed in 1997 [96], containing only two gates, the input and output gate, and was later improved with the inclusion of the forget gate [97]. 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:

$$\mathbf{i_t} = \sigma(\mathbf{W_i x_t} + \mathbf{U_i h_{t-1}} + \mathbf{b_i}) \tag{3.10}$$

$$\mathbf{f_t} = \sigma(\mathbf{W_f x_t} + \mathbf{U_f h_{t-1}} + \mathbf{b_f}) \tag{3.11}$$

$$\mathbf{o_t} = \sigma(\mathbf{W_o x_t} + \mathbf{U_o h_{t-1}} + \mathbf{b_o}) \tag{3.12}$$

Each of these three gates holds a different function. The input gate controls whether the cell state is updated or not, equation (3.10), the forget gate defines how the previous memory cells affect the current

one, equation (3.11) and the output gate controls how the hidden state is updated, equation (3.12).

Contrarily from 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:

$$\tilde{\mathbf{c}}_\mathbf{t} = \phi(\mathbf{W_c x_t} + \mathbf{U_c h_{t-1}} + \mathbf{b_c}) \tag{3.13}$$

$$\mathbf{c_t} = \mathbf{f_t} \circ \mathbf{c_{t-1}} + \mathbf{i_t} \circ \tilde{\mathbf{c}}_\mathbf{t} \tag{3.14}$$

$$\mathbf{h_t} = \mathbf{o_t} \circ \phi(\mathbf{c}_t), \tag{3.15}$$

Equation (3.13) computes the candidate cell state which is then used along with the previous cell state to obtain the cell state on equation (3.14), by using the input and forget gates. The hidden state is computed from the cell sate and the output gate using equation (3.15).

In equations (3.10) to (3.15) 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 defined on Section 3.2, respectively.

Like the RNN, the LSTM is also trained using BPTT, [51]. A scheme of the LSTM unit can be found in Figure 3.6 alongside the simple RNN unit and the GRU discussed next in Section 3.6.

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.

3.6 Gated Recurrent Unit

In 2014 a new type of unit for RNN was proposed, the GRU [98]. 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 because the most used LSTM unit is really complex, having many parameters to train that increase both the size of the model and the time used to train it.

In order to reduce the size of the model, the GRU is only composed of two gates defined by the following equations:

$$\mathbf{z}_t = \sigma(\mathbf{W_z x}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z) \tag{3.16}$$

$$\mathbf{r}_t = \sigma(\mathbf{W_r x_t} + \mathbf{U_r h_{t-1}} + \mathbf{b_r}) \tag{3.17}$$

Equation (3.16) 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 (3.17) 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:

$$\hat{\mathbf{h}}_\mathbf{t} = \phi(\mathbf{W_h x_t} + \mathbf{U_h}(\mathbf{r_t} \circ \mathbf{h_{t-1}} + \mathbf{b_h})) \tag{3.18}$$

$$\mathbf{h}_t = (\mathbf{1} - \mathbf{z_t}) \circ \mathbf{h_{t-1}} + \mathbf{z_t} \circ \hat{\mathbf{h}}_\mathbf{t}. \tag{3.19}$$

Equation (3.18) 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 (3.19).

In equations (3.16) to (3.19) 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 $\mathbf{U}_h$, while the biases are $\mathbf{b}_x$, $\mathbf{b}_r$ and $\mathbf{b}_h$. $\sigma(\cdots)$ and $\phi(\cdots)$ correspond again to the logistic sigmoid and the hyperbolic tangent activation functions, respectively.

The RNN using the GRU is also trained with BPTT [51] and its scheme can be seen in Figure 3.6. 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.

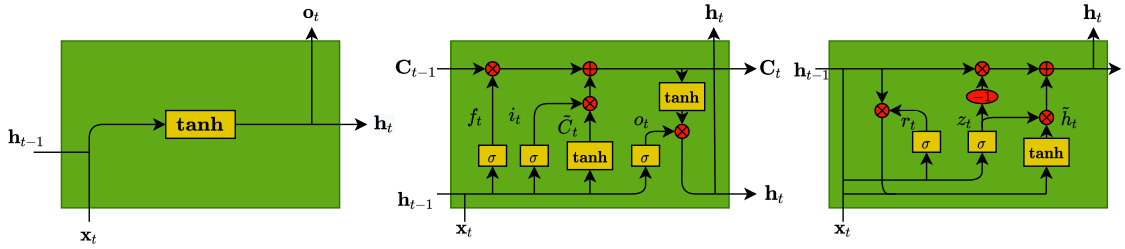

Figure 3.6: Comparison between the three different units, RNN, LSTM and GRU, respectively.

4

Methodology

Contents

This chapter discusses the methodology used to model the *C. elegans* data described in subchapter 2.3 with the architectures introduced in Chapter 3. It starts with a discussion on the implementation setting, describing how each of these models was implemented. This is followed by a discussion on the evaluation criteria for these models, debating error metrics appropriate for the task and describing how to compute the number of parameters of each model. Then, it is described how the hyperparameter optimization was done for each of the models used in this thesis and the rationale behind how the parameters were chosen. The final part of the chapter reports on tests performed to assess the effect of the time step between samples on the predictions obtained by the used techniques.

4.1 Implementation Setting

The five different machine learning techniques discussed in detail in the previous chapter lead to five different implementations reviewed in the current chapter and tested in the next one. The five different models implemented will be referred in this document with a name based on the machine learning technique used at its core: TDNN, NARX, RNN, LSTM and GRU.

The five models were not all implemented in the same environment, since some programming languages have an easier framework to implement some models, while others are more adequate to others. Therefore, in order to implement the TDNN and NARX models, it was used the MATLAB language [99] and the 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, since these models are composed of two different layers, the main recurrent layer, which the models are named after and a small dense layer at the end, which converts the output of the recurrent layer to the output of the model. Since python [100] and its keras [101] and tensorflow [102] libraries provide an easier framework for creating models with multiple layers of different kinds, these were used to build the RNN, LSTM and GRU models.

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 [103], [104]. The RNN, LSTM and GRU models were optimized using the Adam algorithm [105] 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`.

4.2 Model Evaluation

Since one of the goals of this thesis is to present a comparison between different machine learning techniques, 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 should be used. Common regression metrics used to evaluate the performance of this type of models are: the Mean Absolute Error (MAE), the Mean Squared Error (MSE) and the Root Mean Squared Error (RMSE).

The MAE obtains the mean of the absolute difference between each pair of corresponding data points from the two sets using equation (4.1). A common issue with the MAE when it is used as a metric to optimize a model is that it gives the same importance to all differences between pairs of corresponding data points, just averaging the error. This might not be good for some regression models, since predicting all data points with some small error is better then obtaining some points of the predicted sequence with a huge error and the remaining points with an even smaller error.

Instead of doing this, the MSE computes the mean of the squared difference between each pair of corresponding data points from the two sets, using equation (4.2). Therefore, the difference between a pair of corresponding data points is inflated when this difference is larger and it is lowered even more when the difference is already small. This detail allows the models that use the MSE as the loss function during the training procedure to improve on data points where the error is large, while also giving a more exact information on the correctness of the results, since larger errors get more punished and lower errors less punished.

The MSE does a good job on inflating higher errors, but this comes at the cost of not being in the same units as the values being predicted, since the results come in the square of that unit. For this work, that would mean that the results come in squared Volt. To allow for an easier comparison of the magnitude of the errors, one can use the RMSE that is computed as the square root of the mean of the squared difference between each pair of corresponding data points from the two sets, using equation (4.3).

In equations (4.1), (4.2) and (4.3), 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:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{4.1}$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{4.2}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{4.3}$$

When choosing the error metrics to use, one must choose them according to the task at hand, which for this case means that the appropriate error metric is considered to be the MSE for the training process of the models and the RMSE for the evaluation of the results in the end. The MSE is chosen for the training process as the loss function, since the goal is to model the behavior of the *C. elegans* nervous system, meaning that the MAE is not suitable because the error should be as low as possible for every data point, instead of having a just a low average error. The error values are not analyzed in terms of their absolute value, only being compared in terms of their differences, during the training phase, so the RMSE is not needed for this case, also explaining the choice of the MSE. For the evaluation of the results after the models are trained it the RMSE is used, since the errors are compared also in terms of their magnitude and the usage of values with the same units as the ones of the data being predicted allows for an easier comparison.

Regarding the evaluation of the models' complexity, there are two main characteristics that can be evaluated, the time and the space required by the models to run. In this thesis the comparison between the time taken by each model to run is neglected, since the models were implemented on 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. Next, a formula to compute the number of parameters of each model is introduced, according to the hyperparameters of each architecture.

One aspect common to all the models used is the existence of two layers, a hidden layer, which diverges between models, and an output layer that is of the same type and structure for all the models used. This layer is a linear layer that can be defined by equation (4.4). There are also hyperparameters that are common to all the used models: the number model inputs i, the number of model outputs o and

the size of the hidden layer s.

$$\mathbf{y_t} = \mathbf{x_t}\mathbf{A^T} + \mathbf{b} \tag{4.4}$$

The output linear layer common to all models has a matrix of weights $\mathbf{A}$ and an array of biases $\mathbf{b}$, both having o lines and the matrix having s columns. The number of parameters of this layer for any model used are equal to $o \times (s + 1)$.

The TDNN models implemented have a hidden linear layer, which means that this layer is also defined by equation (4.4) having a matrix of weights $\mathbf{A}$ and an array of biases $\mathbf{b}$. It is important to note that the number of parameters for this layer is not the same as for the output layer, since the input and output sizes of the respective layers are different. The input size of this layer depends not only on the number of model inputs i, but also on the delay used for the inputs d_x. This means that both the matrix and the array have s lines and the matrix has $i \times (d_x + 1)$ columns, which indicates that the hidden layer has $s \times [i \times (d_x + 1) + 1]$ parameters. Adding the number of parameters of the two layers leads to equation (4.5) that computes the total number of parameters of the TDNN models:

$$\#TDNN = s \times [i \times (d_x + 1) + 1] + o \times (s + 1) \tag{4.5}$$

NARX models also possess a hidden linear layer, but these models have a main difference when compared to the TDNN models as discussed on Chapter 3, which is the recurrent connection from the output to the input. This connection also has a tapped delay line, meaning that the total number of parameters of this hidden layer will also depend on the delay used for these outputs, d_y, which leads to the number of inputs of this layer to be $i \times (d_x + 1) + o \times d_y$. Therefore, the matrix of weights $\mathbf{A}$ and the array of biases $\mathbf{b}$ will still have s lines, while the matrix has a number of columns corresponding to the number of inputs of the hidden layer. Based on this, the total number of parameters of this layer will be defined as $s \times [i \times (d_x + 1) + o \times d_y + 1]$, implying that the total number of parameters for the NARX models can be computed as in equation (4.6):

$$\#NARX = s \times [i \times (d_x + 1) + o \times d_y + 1] + o \times (s + 1) \tag{4.6}$$

The hidden layer of the RNN models is a recurrent layer that has two matrices $\mathbf{V}$ and $\mathbf{U}$ and one vector $\mathbf{c}$, as discussed in Chapter 3, all of them having s lines. Matrix $\mathbf{V}$ has i columns and matrix $\mathbf{U}$ has s columns, meaning that this layer has $s \times (i + s + 1)$ parameters. Based on this and on the number of parameters of the output layer, the total of parameters of the RNN models can be computed using

equation (4.7):

$$\#RNN = s \times (i + s + 1) + o \times (s + 1) \tag{4.7}$$

Although the LSTM models still use a recurrent layer, this layer is composed of more complicated units that have three different gates. Each of these gates translates to a set of two additional matrices of weights and an array of biases when compared with the RNN models. This sums up to a total of eight matrices and four arrays, all of them having s lines, while four of the matrices have i columns and the other four have s columns. This leads to a total of $4 \times s \times (i + s + 1)$ parameters of the LSTM recurrent layer, that when added up with the number of parameters of the output layer leads to the total number of parameters of the LSTM models, obtained with equation (4.8):

$$\#LSTM = 4 \times s \times (i + s + 1) + o \times (s + 1) \tag{4.8}$$

The hidden layer of the GRU models is also a recurrent layer, but only composed of two gates, which also leads to two additional matrices of weights and an array of biases per gate when compared with the RNN models. These also hold s lines for each matrix or array and i columns for half of the matrices and s columns for the other half, leading to a total of $3 \times s \times (i + s + 1)$ parameters defining the hidden layer. When these are added up to the number of parameters of the output layer, the total number of parameters for the GRU models is obtained by equation (4.9):

$$\#GRU = 3 \times s \times (i + s + 1) + o \times (s + 1) \tag{4.9}$$

This set of five equations will be used throughout the remainder of this thesis to help computing the total number of parameters of the models so that models with similar size are compared.

4.3 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 subchapter handles this, explaining how some of these hyperparameters were optimized and indicating the results for each of the tests made, so that the best set of parameters are chosen for each model.

The optimization was made using the first dataset described in subchapter 2.3, **PTRS-20-0.5**, which refers to the PTRS of *C. Elegans* and where four input neurons are stimulated, while the output voltage of other four neurons is measured.

All of these models have at least one hyperparameter, the size of the hidden layer. Some of them also having two hyperparameters for the training procedure, while the others have one or two hyperparameters for the network structure. In order to optimize these in an efficient way, all hyperparameters are fixed, except one. This hyperparameter that was not fixed takes different values and the corresponding results are compared. Then, the best experimented value according to its RMSE and added complexity to the model is chosen and the procedure is repeated for the remaining hyperparameters until all have been optimized.

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 used to predict on the validation data multiple times. In this work it was chosen to do it ten times. The RMSE values shown in the tables of this subchapter are averages of ten different simulations made for each model with the specific hyperparameter values. To better tune the hyperparameters and understand the behavior of the models, the results are shown for the training and validation sets, while the test set is only used in the experiments of the next Chapter so that the hyperparameters are not biased to the test set.

4.3.1 TDNN Model

The TDNN models only have two different hyperparameters to be optimized, the delay used for the input of the network and the size of the hidden layer. For this part of the thesis, the goal regarding the size of the hidden layer is only to understand what are the appropriate sizes that can and should be used. Another goal regarding this hyperparameter is also to understand what is the smaller size that can be used, without increasing the error of the model to unacceptable levels. Therefore, the first hyperparameter being optimized is the delay used for the input of the network, using a fixed value of 16 units for the size of the hidden layer.

The different delays tested for the input were of 1, 2, 4, 8 and 16 samples, having in mind that these last values might become a problem if chosen for datasets with many inputs, since it would increase the size of the network to unreasonable levels. The average of the RMSE for the ten simulations on the different values of the delay on the input can be found in Table 4.1.

Table 4.1: The average RMSE obtained for the TDNN model for ten simulations, comparing different delays used on the input.

	1	**2**	**4**	**8**	**16**
Training	1.2453e-01	1.1444e-01	1.0944-01	1.0172e-01	8.7260e-02
Validation	1.7522e-01	1.75329e-01	1.6682e-01	1.5374e-01	1.5087e-01

From Table 4.1, it can be concluded not only that the RMSE is clearly large, but also that it does not

change significantly when the delay on the input is increased, just decreasing slightly. Also, looking at the plots obtained for each simulation run, it is clear that the TDNN models have a hard time reproducing the behavior of the system, as expected, since these models have no knowledge of the state of the system. With the increase of the delay used leading to an increase in the complexity of the model and also based on the results from Table 4.1, it was chosen a delay of 1 sample for the input.

Therefore, the network is then tested for different sizes of the hidden layer, using a delay of 1 sample for the input. The different values tested for the network sizes were of 2, 4, 8, 16, 32 and 64 units. The average of the RMSE for the ten simulations of the different sizes of the hidden layer can be found in Table 4.2.

Table 4.2: The average RMSE obtained for the TDNN model for ten simulations, comparing different sizes of the hidden layer.

	2	**4**	**8**	**16**	**32**	**64**
Training	1.4100e-01	1.3318e-01	1.2813e-01	1.2472e-01	1.2259e-01	1.2140e-01
Validation	1.5520e-01	1.5864e-01	1.6655e-01	1.8109e-01	2.5269e-01	8.2442e-01

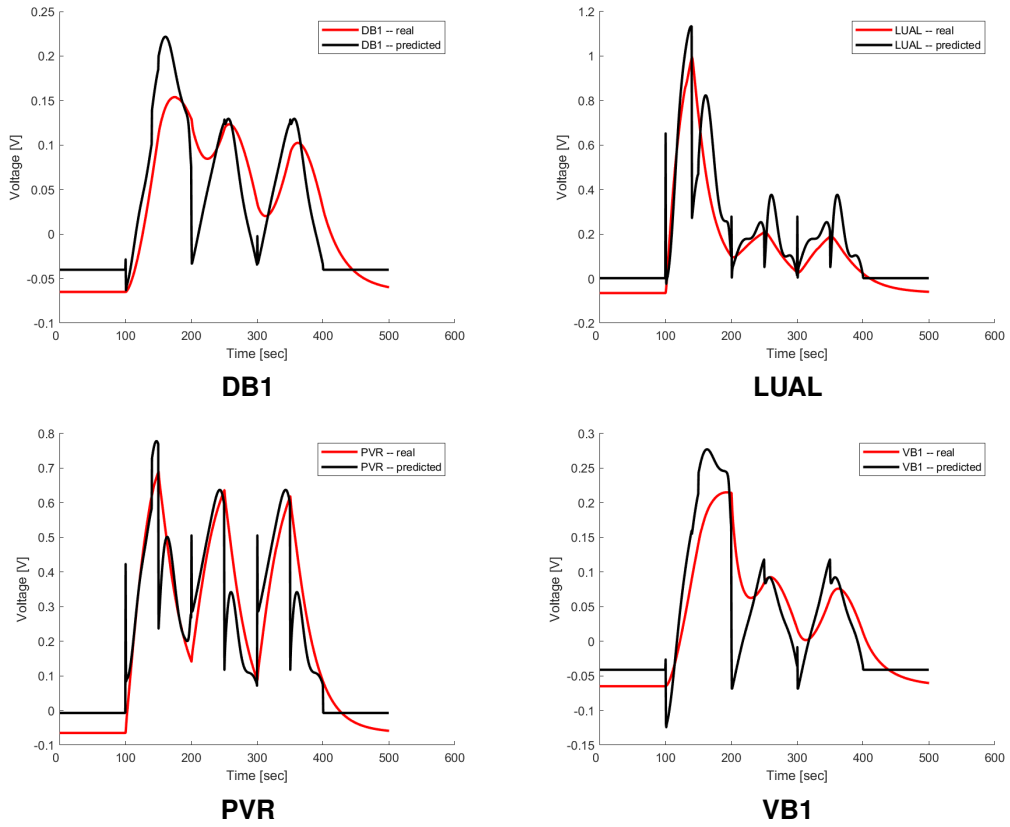

Figure 4.1: Predicted outputs of the TDNN model for one of the examples of the validation set.

The previous table clearly indicates that with an increase of the size of the network, the model tends

to overfit the training data, the RMSE decreasing slightly on this set. Nevertheless, this improvement on the training set comes with a slight increase of the RMSE on the validation set for the first sizes experimented, increasing a lot when the size of the hidden layer is increased from 32 to 64 units, overfitting the training data completely.

The results obtained for the TDNN models are clearly unacceptable as can be seen in Figure 4.1, which comes as expected since the model predicts only based on the stimuli given to the system, having no knowledge of the system state. Nevertheless, the plots of the obtained results suggest that the models are still able to reproduce behavior which is similar to that of the system during some moments.

Based on this, TDNN models seem to be inappropriate for modeling the *C. elegans* nervous system, but were still used to understand how the time step between samples affects the performance of the model in subchapter 4.4. These models were also used in subchapter 5.1 as a lower ground for the comparison among the different techniques used in this thesis.

4.3.2 NARX Model

The NARX models have three different hyperparameters to be optimized, the delay used for the input of the network and the size of the hidden layer, as the TDNN models, and the delay used for the output. Since the optimization of the size of the hidden layer in this part of the thesis has only the goal of understanding how this hyperparameter affects the performance of the different models obtained, this will be the last hyperparameter being optimized. It was chosen to optimize first the delay for the output of the network and then the delay for the input. For this first optimization was used a hidden layer with a size of 16 units and a delay for the input of 1 sample.

For the output delay, the values tested were of 1, 2, 4, 8 and 16 samples, but it is worth noticing that these last few values might be an issue when handling data with more inputs, increasing the size of the models considerably. The results in the form of the average of the RMSE for the ten simulations of the different values of the output delay can be found in Table 4.3.

Table 4.3: The average RMSE obtained for the NARX model for ten simulations, comparing different delays used on the output.

	1	**2**	**4**	**8**	**16**
Training	4.4827e-03	3.7847e-03	3.1061e-01	6.8091e-01	2.0196e-01
Validation	9.0621e-03	7.8523e-03	3.8289e-01	6.0374e-01	1.9119e-01

The results presented in the previous table indicate that when the number of delays increases for values larger than 2 samples, the network tends to have more difficulty learning the parameters that better model the system's response. This may be caused the increased number of parameters that the model has to learn, since when the delay used increases, the number of parameters increases, as

stated by equation (4.6). Based on the results from the previous table, an output delay of 2 samples is chosen, since from the experimented values, this is the one that leads to a superior performance of the model.

For the next hyperparameter, the input delay, the size of the hidden layer will still remain of 16 units and the delay on the output is the one chosen previously, of 2 samples. The different values tested for the delay of the input were the same as for the output in the previous optimization: 1, 2, 4, 8 and 16 samples. The results obtained can be found on Table 4.4.

Table 4.4: The average RMSE obtained for the NARX model for ten simulations, comparing different delays used on the input.

	1	**2**	**4**	**8**	**16**
Training	3.7604e-03	4.2377e-03	2.3714e-03	2.7143e-02	8.1431e-02
Validation	7.9686e-03	8.0784e-03	4.9271e-03	2.4240e-02	8.7613e-02

The previous table indicates that the models performs better when a delay of 4 samples for the input is used, having a higher RMSE for lower delays on the input indicating that with lower delays there might not be available enough information for the model to predict accurately. For delays on the input higher than 4 samples, the model tends to perform poorly, having values of the RMSE an order of magnitude higher, which indicates some difficulty of the network on converging and learning the increased amount of parameters. Therefore, it is chosen to fix the delay on the input at 4 samples.

The last hyperparameter to be optimized for this model is the size of the hidden layer, for which different values were tried: 2, 4, 8, 16, 32 and 64 units. The results obtained for this optimization can be found in Table 4.5

Table 4.5: The average RMSE obtained for the NARX model for ten simulations, comparing different sizes of the hidden layer.

	2	**4**	**8**	**16**	**32**	**64**
Training	8.2140e-02	5.3333e-03	6.2724e-03	2.3272e-03	1.5925e-03	9.8046e-04
Validation	9.1681e-02	9.8454e-03	1.0215e-02	7.8945e-03	6.3531e-03	1.0753e-02

Table 4.5 indicates that the model tends to perform better as the size of the recurrent layer increases, lowering the RMSE on both the training and validation sets, except in two cases. The first one is when changing from 4 to 8 units, where there is a slight increase on the RMSE, which is not significant. The second case is when the model increases from 32 to 64 units, clearly overfitting the network to the training data, having difficulties generalizing for the validation set. Based on these results it is deemed optimal to use a number of units between 4 and 32, for which the obtained results have a low RMSE on both training and validation sets. An example of the outputs obtained for the NARX model can be found in Figure 4.2.

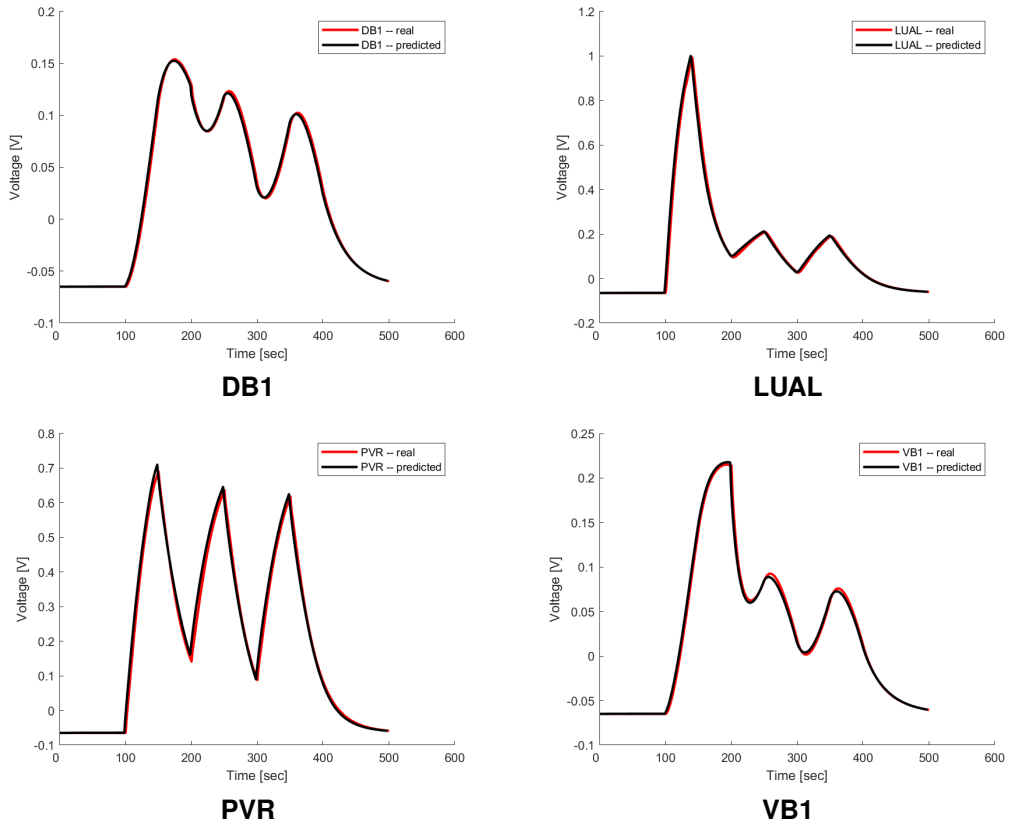

Figure 4.2: Predicted outputs of the NARX model for one of the examples of the validation set.

4.3.3 RNN Model

For the RNN models there are three hyperparameters to be optimized, the batch size that controls the number of sequences the model forward propagates before updating the parameters, the learning rate that defines how much the network changes its parameters on each update and the size of the hidden layer. The size of the hidden layer is the last of these hyperparameters being optimized, with the goal of understanding how the RMSE changes with the different sizes experimented for the recurrent layer and also how small of a model can be used without loosing prediction capability. From the two other parameters, it is chosen to optimize first the learning rate, fixing a size for the hidden layer of 16, while fixing the batch size as 32.

In order to find the learning rate that best suits the model and data being used, different values were experimented: 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001. The results obtained can be found in Table 4.6.

The results on the previous table clearly demonstrate that the learning rate doesn't have a high impact on the ability of the model to learn the data, since the order of magnitude of the RMSE is equal in all test cases. The lowest value experimented with, 0.0001, proves to be a small one, indicating that the model is not able to learn as well, perhaps needing more epochs for the training procedure. The

Table 4.6: The average RMSE obtained for the RNN model for ten simulations, comparing different values for the learning rate.

	0.05	**0.01**	**0.005**	**0.001**	**0.0005**	**0.0001**
Training	1.3179e-01	1.1477e-01	1.0093e-01	1.0964e-01	1.2359e-01	1.7491e-01
Validation	1.4792e-01	1.3849e-01	1.2053e-01	1.3388e-01	1.4976e-01	1.9297e-01

values of the learning rate for which the model performs better are 0.005 and 0.001, having really small differences in the RMSE, which should not be taken into account. So, since there is not a relevant difference among these values, the default value of 0.001 for the Adam optimizer is kept as the learning rate for this model.

The batch size values tested were of 2, 4, 8, 16 and 32, having in mind the fact that smaller batch sizes imply a slower training phase for the model. The results obtained with the learning rate fixed at 0.001 and the size of the hidden layer still fixed at 16 units are available in Table 4.7.

Table 4.7: The average RMSE obtained for the RNN model for ten simulations, comparing different values for the batch size.

	2	**4**	**8**	**16**	**32**
Training	7.7420e-02	8.1851e-02	8.2691e-02	1.0180e-01	1.1105e-01
Validation	9.5989e-02	9.9151e-02	1.0134e-01	1.2157e-01	1.3752e-01

The results obtained for the batch size indicate that this hyperparameter does not have a big effect on the model performance, since the RMSE increases slightly with an increase of the batch size. Nevertheless, this hyperparameter has a significant impact on the computational time that the model needs for training, which increases exponentially with the decrease of the batch size. Due to this cost, it is chosen a batch size of 32 in order to maintain a low computational cost, since the differences in the RMSE are small.

The last hyperparameter tested for this model is the size of the hidden layer, for which the different values tested are of 2, 4, 8, 16, 32 and 64 units. Table 4.8 shows the average RMSE obtained for the different simulations.

Table 4.8: The average RMSE obtained for the RNN model for ten simulations, comparing different sizes of the hidden layer.

	2	**4**	**8**	**16**	**32**	**64**
Training	1.8026e-01	1.4389e-01	1.3015e-01	1.0345e-01	8.2856e-02	6.9216e-02
Validation	1.9459e-01	1.6380e-01	1.5456e-01	1.2644e-01	1.0672e-01	9.0953e-02

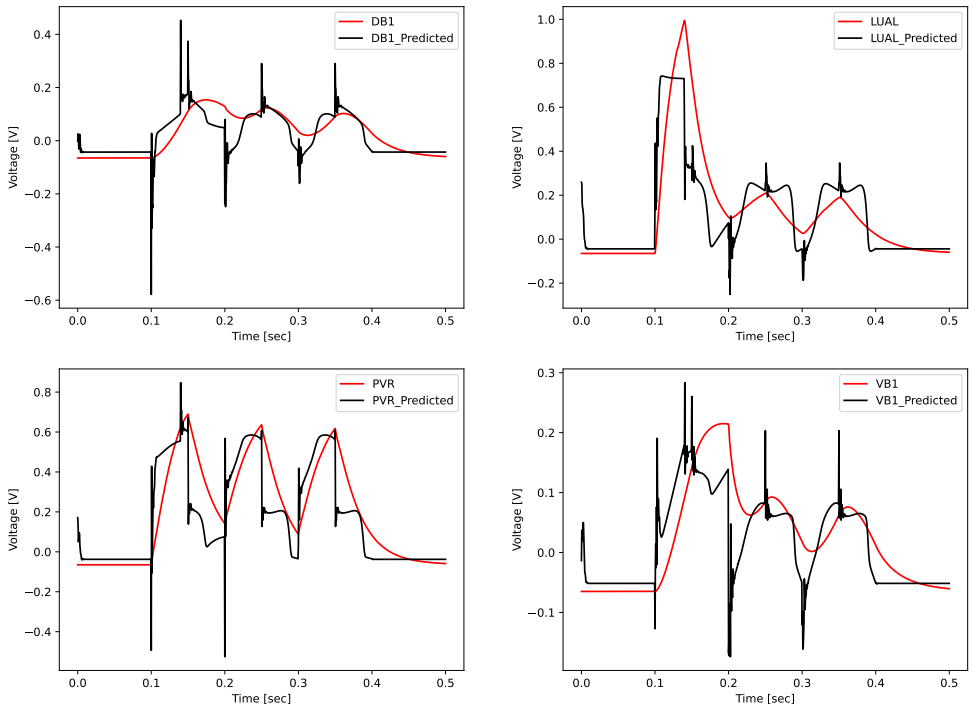

Figure 4.3: Predicted outputs of the RNN model for one of the examples of the validation set.

The results in the previous table clearly demonstrate that the model tends to perform better when the size of the recurrent layer is increased, lowering the average RMSE. This indicates that the RNN models might need larger networks with more parameters to train, which is not optimal for this case, since one of the goals is to have models as reduced as possible.

As can be seen for all hyperparameters optimized for the RNN models, the RMSE is too high, since the outputs are of the order of magnitude of 1V and an average RMSE of just one order of magnitude lower is not accurate enough. Also looking at the obtained sequences predicted by the model, it is clear that the RNN models have some ability to learn and replicate the system's behavior, but have high oscillations in the predictions after some time steps, as seen in the example from Figure 4.3. This is coherent with the long term dependency problems that were expected of the simple RNN, which is able to learn the system's behavior, but fails when the dependencies are distant in time.

Despite, the poor results provided by this model, it is kept for testing in subchapter 4.4, in order to evaluate how the data used affects the performance of these models. It is also tested in subchapter 5.1, where it is compared with the other models used in this thesis.

4.3.4 LSTM Model

The LSTM models have the same hyperparameters as the RNN ones, since the network architecture is the same and only the inner architectures of the hidden layer units differ from one type of model to the

other. Therefore, the same procedure will be followed, optimizing the learning rate, the batch size and then the size of the hidden layer.

So, at first the values for the batch size and the number of units on the hidden layer were fixed at 32 and 16, respectively. The values tested are also the same for each of the three hyperparameters. The results obtained for the learning rate optimization are on Table 4.9.

Table 4.9: The average RMSE obtained for the LSTM model for ten simulations, comparing different values for the learning rate.

	0.05	**0.01**	**0.005**	**0.001**	**0.0005**	**0.0001**
Training	6.3150e-03	9.4984e-03	1.1084e-02	2.3348e-02	3.7055e-02	1.0877e-01
Validation	1.1974e-02	1.7834e-02	2.0112e-02	3.5754e-02	5.1243e-02	1.3207e-01

The results indicate that this model benefits from using larger learning rates, obtaining a higher error as the learning rate decreases, the RMSE being an order of magnitude larger for the learning rate of 0.0001 than for the rest of the values experimented. Since the RMSE lowers with the increase of the learning rate, a value of 0.1 was also tested for this hyperparameter, obtaining an RMSE of 8.3522e-03 for the training set and of 1.4024e-02 for the validation set.

Since with the value of 0.1 for the learning rate, the model doesn't tend to perform better, a value of 0.05 was chosen for this hyperparameter. Then, with the size of the hidden layer fixed at 16 units, the batch size is optimized generating the results in Table 4.10.

Table 4.10: The average RMSE obtained for the LSTM model for ten simulations, comparing different values for the batch size.

	2	**4**	**8**	**16**	**32**
Training	4.6965e-03	4.2512e-03	4.6874e-03	5.3438e-03	5.8704e-03
Validation	6.4522e-03	6.9625e-03	7.9463e-03	9.5933e-03	1.2578e-02

The results indicate that the batch size has an impact on the RMSE for this model, increasing the RMSE with an increase of the batch size. Nevertheless, this increase in the RMSE is not even linear, while the increase in computational time that comes with the decrease of the batch size is exponential.

Based on this, since the RMSE values are of a similar order of magnitude for all batch size values experimented, it is decided to sacrifice a small amount of accuracy in order to maintain the computational cost as small as possible. So, the value for the batch size is fixed at 32 and the size of the hidden layer is then optimized, obtaining Table 4.11.

Table 4.11: The average RMSE obtained for the LSTM model for ten simulations, comparing different sizes of the hidden layer.

	2	4	8	16	32	64
Training	4.3768e-02	1.4651e-02	7.0741e-03	6.0565e-03	5.6000e-03	5.3207e-02
Validation	4.7067e-02	1.9899e-02	1.1479e-02	1.1604e-02	1.3097e-02	6.5767e-02

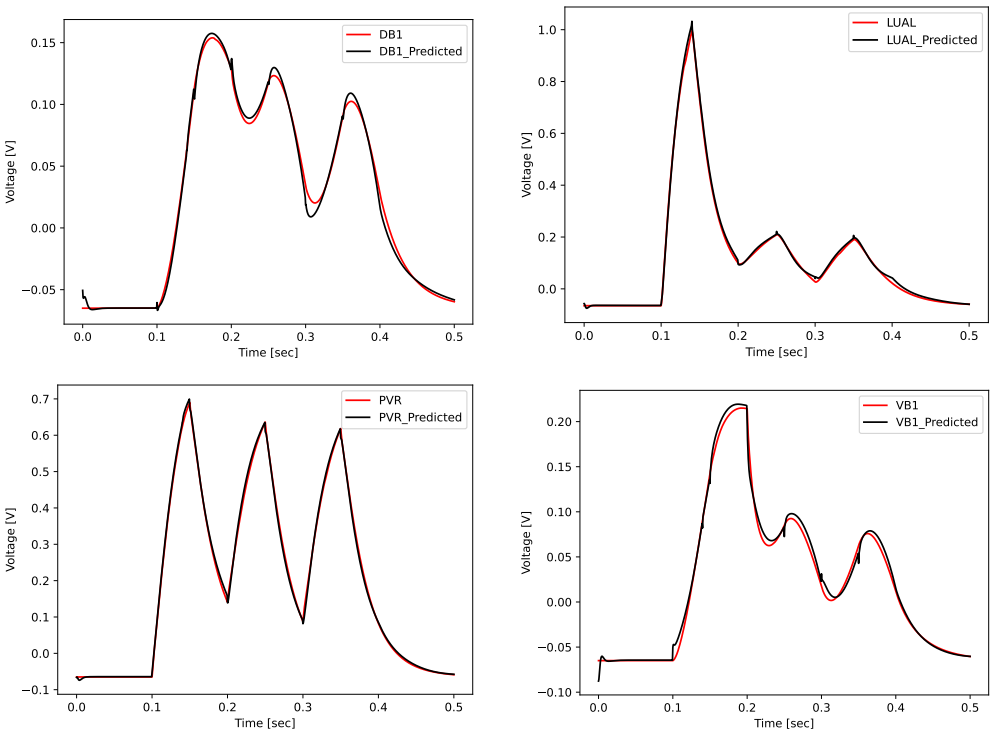

Figure 4.4: Predicted outputs of the LSTM model for one of the examples of the validation set.

Table 4.11 indicates that the model performs better on the training set as the size of the recurrent layer increases, lowering the RMSE as the number of units increases. Regarding the results on the validation set, there is also a tendency to decrease the RMSE as the number of units increase in the first values, but then increasing for the last values experimented, clearly overfitting the training data for the model with 64 units. Based on these results it is deemed optimal to use a number of units between 8 and 32, for which the obtained results are better on both training and validation sets. An example of the outputs obtained for the LSTM model can be found in Figure 4.4.

4.3.5 GRU Model

Since GRU also holds the same network structure as the RNN and the LSTM models, the hyperparameters that need to be optimized will also be the same, the learning rate, the batch size, and the size

of the recurrent layer. The procedure is repeated, optimizing the parameters in the order previously mentioned, testing the same values and fixing the values of the batch size and the size of the recurrent layer at the beginning as 32 and 16, respectively. The results obtained for the learning rate can be found in Table 4.12.

Table 4.12: The average RMSE obtained for the GRU model for ten simulations, comparing different values for the learning rate.

	0.05	**0.01**	**0.005**	**0.001**	**0.0005**	**0.0001**
Training	2.9427e-03	4.3568e-03	6.2120e-03	1.9496e-02	4.0387e-02	1.3576e-01
Validation	6.4908e-03	8.2566e-03	1.0647e-02	2.6548e-02	5.2263e-02	1.4455e-01

From this table it can be concluded that the behavior of this model is again similar to the one of the LSTM, decreasing the RMSE as the learning rate value increases. Similarly to the LSTM model, here was also tested a learning rate of 0.1, for which was obtained an average RMSE of 4.2231e-03 for the training set and of 9.0623e-03 for the validation set.

Since the value of 0.05 for the learning rate provides the best results, this hyperparameter is fixed as 0.05 and the number of units of the hidden layer is optimized next, for which the results can be found in Table 4.13.

Table 4.13: The average RMSE obtained for the GRU model for ten simulations, comparing different values for the batch size.

	2	**4**	**8**	**16**	**32**
Training	7.0473e-03	4.3307e-03	2.9801e-03	3.4886e-03	3.0034e-03
Validation	9.1723e-03	5.9723e-03	5.0231e-03	5.8706e-03	5.4331e-03

The optimization of the GRU for this hyperparameter indicates that the RMSE has its lowest values for a batch size of 8. Regarding the remaining values, only a value of 2 for the batch size has a significant difference when comparing to the RMSE obtained when using a batch size of 8.

Since the difference in the RMSE between using a batch size of 8 or 32 is not significant, the latter is chosen in order to keep a low computational time. The size of the hidden layer is then optimized, obtaining Table 4.11.

Table 4.14: The average RMSE obtained for the GRU model for ten simulations, comparing different sizes of the hidden layer.

	2	**4**	**8**	**16**	**32**	**64**
Training	4.2692e-02	4.1482e-03	4.5130e-03	3.4511e-03	2.7011e-03	2.5744e-03
Validation	4.4990e-02	6.9469e-03	7.0234e-03	5.6767e-03	7.1177e-03	9.6214e-03

The previous table indicates that the model does not perform as well when using a size of the recur-

rent layer of 2 having a RMSE an order of magnitude higher when compared to the RMSE obtained for the remaining values. There seems to also be a tendency for the model to start overfitting the training data as the size of the recurrent layer increases for values higher than 16 units. All values between 4 to 64 units lead to a good performance of the GRU model, to accurately reproduce the behavior of the *C. elegans* nervous system. An example of the outputs obtained for the GRU model can be found in Figure 4.4.

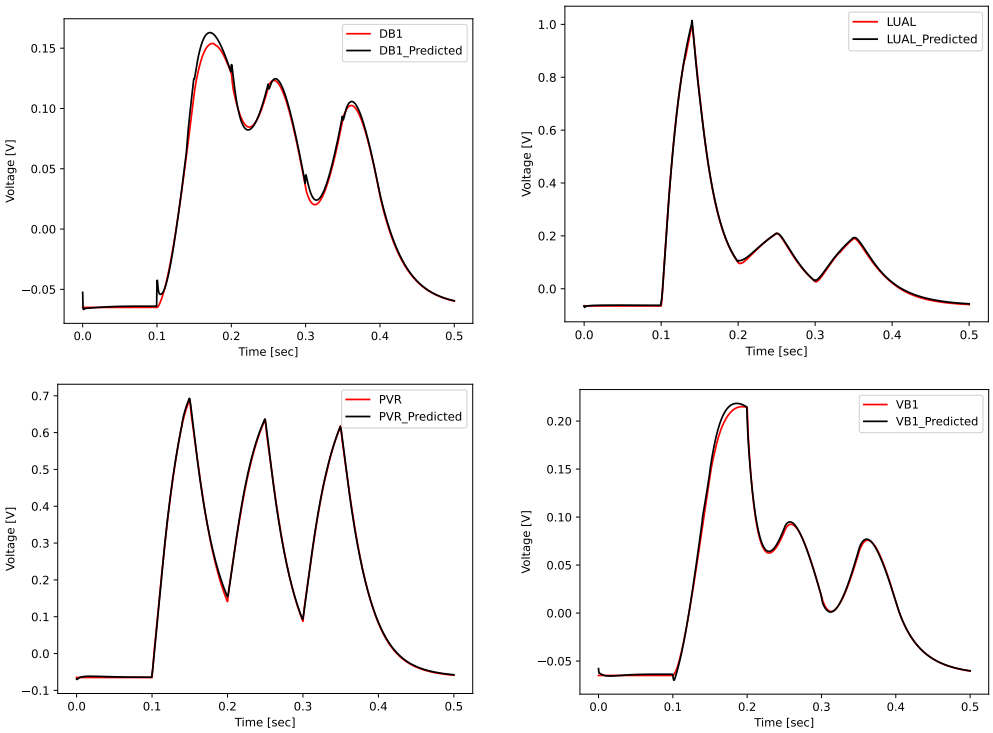

Figure 4.5: Predicted outputs of the GRU model for one of the examples of the validation set.

4.4 Sample Rate Tests

One important part of this thesis described previously in subchapter 2.3 is the generation of the data on the *C. elegans* behavior. So, understanding 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. According to this need, this part of the thesis concerns the study of how many samples are needed for each example in order to accurately model the available data.

Even though there are two models that in the previous subchapter proved to 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 in this subchapter, using a hidden layer of 16 units for the five models, since this value was in the set of

optimal values experimented previously for all the models hidden layer sizes.

These techniques are tested on two different base datasets, one concerning PTRS and another concerning the Nictation behavior. Each of these datasets is generated in six different versions, using a time step between samples of 10ms, 5ms, 1ms, 0.5ms, 0.1ms and 0.05ms.

The results obtained for each of the techniques on the different datasets can be found in Table 4.15 for the TDNN, Table 4.16 for the NARX, Table 4.18 for the RNN, Table 4.19 for the LSTM and Table 4.20 for the GRU.

Table 4.15: The average RMSE obtained for the TDNN model for ten simulations, comparing different time steps on both the PTRS and the Nictation scenarios.

	$\mathbf{10}$ms	$\mathbf{5}$ms	$\mathbf{1}$ms	$\mathbf{0.5}$ms	$\mathbf{0.1}$ms	$\mathbf{0.05}$ms
Train - PTRS	1.0697e-01	1.2078e-01	1.3197e-01	1.3331e-01	1.3450e-01	1.3475e-01
Valid - PTRS	1.2767e-01	1.4395e-01	1.5552e-01	1.5810e-01	1.6166e-01	1.5968e-01
Train - Nictation	3.9651e-01	4.6775e-01	5.2534e-01	5.3384e-01	5.3777e-01	5.4184e-01
Valid - Nictation	6.7436e-01	8.0270e-01	8.0447e-01	8.4049e-01	7.9135e-01	9.4291e-01

The average RMSE obtained using TDNN models on datasets with different time steps is not significantly different among the datasets, with these models still performing poorly, having no significant improvement. For the PTRS scenario, the RMSE decreases slightly as the time step between samples increases, but this does not happen in a significant way with the TDNN models maintaining the same problems discussed in the previous subchapter. The TDNN models have a higher RMSE when modeling the data generated for the Nictation behavior, which comes as expected since the hyperparameters of all models used were optimized using only data from the PTRS scenario. The data from the Nictation scenario is also more complex, with 6 inputs and outputs instead of 4 like for the PTRS scenario and the outputs on the Nictation scenario have a wider range of values, naturally creating larger RMSE values for the Nictation scenario, since the error metric used is not relative, providing an error in Volt. Therefore, it is expected that the models used in this thesis obtain a higher RMSE on the Nictation scenario when compared to the PTRS one. In this scenario, there is also a slight decrease of the RMSE as the time step between samples increases.

Table 4.16: The average RMSE obtained for the NARX model for ten simulations, comparing different time steps on both the forward crawling motion and the nictation scenarios.

	$\mathbf{10}$ms	$\mathbf{5}$ms	$\mathbf{1}$ms	$\mathbf{0.5}$ms	$\mathbf{0.1}$ms	$\mathbf{0.05}$ms
Train - PTRS	2.2339e+00	4.5385e-01	7.0188e-02	2.1096e-03	3.4426e-02	4.6336e-01
Valid - PTRS	3.3455e+00	7.5693e-01	6.6491e-02	4.7522e-03	4.9205e-02	4.6505e-01
Train - Nictation	3.4822e+01	1.4717e+01	4.1617e+00	1.3084e+00	2.1575e+00	2.2755e+00
Valid - Nictation	3.6198e+01	1.5117e+01	4.0853e+00	1.1187e+00	2.1446e+00	2.2683e+00

The performance of this model on the PTRS datasets indicates a reduced ability of the NARX to also perform on datasets with different time steps than the ones with which its hyperparameter optimization was performed. It only produces an accurate response of the dynamical system for the dataset with a time step of 0.5ms, also performing accurately in some of the simulations on the dataset with a time step of 1ms, but not converging in the rest of the simulations.

For the datasets on the Nictation behavior, this model shows unacceptable performance, not being able to converge in the simulations performed. This indicates a poor ability of the architecture to generalize for the data of this different behavior. A similar problem can be found in the results from Table 4.4 where the model shows a poor ability to converge when larger delays are used, since this implies a more complex model, which would need more data and training time to learn. Therefore, due to the limitations on the available data and on the computational resources that can be used, it was decided to do an additional test, of a NARX model with a delay on the input of 1 sample, thus creating a simpler model. This was done as an alternative to using more data or increasing the number of epochs of the training process, since these are harder limitations to overcome with the available computational resources and the difficulty of generating more data. The results obtained for the NARX model using an input delay of 1 sample can be found in Table 4.17.

Table 4.17: The average RMSE obtained for the NARX model an input delay of 1 sample for ten simulations, comparing different time steps on both the forward crawling motion and the nictation scenarios.

	$\mathbf{10}$ms	$\mathbf{5}$ms	$\mathbf{1}$ms	$\mathbf{0.5}$ms	$\mathbf{0.1}$ms	$\mathbf{0.05}$ms
Train - PTRS	2.4714e-02	1.8844e-01	1.0731e-01	3.6855e-03	6.7505e-02	2.3148e-01
Valid - PTRS	3.8851e-02	2.0263e-01	1.4759e-01	2.1629e-02	7.3484e-02	2.7504e-01
Train - Nictation	2.3239e+01	8.9975e+00	5.6151e+00	2.0160e+00	2.1082e+00	3.3565e+00
Valid - Nictation	2.3236e+01	8.8657e+00	5.6343e+00	2.5622e+00	2.0756e+00	3.3420e+00

These results indicate that the NARX with a reduced delay on the input has more flexibility performing for time steps different than 0.5ms on the PTRS datasets, but still having a good performance for this case. Nevertheless, the results with a delay of 0.5ms are worse than the ones obtained by the NARX with a delay on the input of 4 samples.

For the Nictation scenario, contrary to what would be the goal, the architecture with a lower delay on the input still has problems converging, generating predictions with exceedingly high error in this scenario. This indicates that this architecture might need more data for the training process, or more diversity in the training data. Based on the fact that the results obtained for the Nictation scenario show an even larger RMSE than previously, the architecture with a delay on the input of 4 samples is the one which was used throughout the rest of this thesis, for comparison purposes with the remaining models.

Table 4.18: The average RMSE obtained for the RNN model for ten simulations, comparing different time steps on both the forward crawling motion and the nictation scenarios.

	10ms	5ms	1ms	0.5ms	0.1ms	0.05ms
Train - PTRS	4.4860e-02	4.8402e-02	8.1645e-02	1.0980e-01	1.3590e-01	1.3627e-01
Valid - PTRS	5.8225e-02	6.0357e-02	1.0169e-01	1.3715e-01	1.5526e-01	1.5548e-01
Train - Nictation	1.8595e-01	2.1117e-01	4.4366e-01	5.5678e-01	6.1945e-01	6.2102e-01
Valid - Nictation	2.9086e-01	3.2581e-01	5.4223e-01	6.1008e-01	6.3956e-01	6.4026e-01

The results from Table 4.18 indicate that the RNN models still have problems modeling the behavior of the *C. elegans*, with an average RMSE higher than what is acceptable, for all the time steps used, although the RNN models are still able to model some of the behavior of the *C. elegans*. Nevertheless, it is noteworthy a decrease of the RMSE as the time step between samples increases, which is expected since with this increase, the dependencies in time being modeled are not as distant in the sequence of points received by the RNN. The results obtained for the Nictation scenario are also worse than from the PTRS one, which is comprehensible for the same reasons mentioned before for the TDNN. The previous statements hold for all the results in Table 4.18, except when comparing the use of a time step of 0.1ms with one of 0.05ms, for which there is no noticeable difference.

Table 4.19: The average RMSE obtained for the LSTM model for ten simulations, comparing different time steps on both the forward crawling motion and the nictation scenarios.

	10ms	5ms	1ms	0.5ms	0.1ms	0.05ms
Train - PTRS	7.3592e-03	5.7398e-03	5.3817e-03	5.9000e-03	1.1091e-02	2.0203e-02
Valid - PTRS	1.5158e-02	1.1973e-02	1.0620e-02	1.1188e-02	2.0747e-02	3.3295e-02
Train - Nictation	2.5451e-02	2.0825e-02	2.6286e-02	3.0815e-02	5.8267e-02	8.1306e-02
Valid - Nictation	1.0186e-01	6.8994e-02	7.6390e-02	9.8411e-01	1.6343e-01	1.9265e-01

Similarly to what happens with the TDNN and RNN models, the LSTM tends to decrease the average RMSE, as the time step between samples increase, but in this case, only until this value reaches 1ms and 5ms for the PTRS and Nictation behaviors, respectively. As the time step increases for values beyond these, the average RMSE increases slightly, not indicating a big difference in performance.

Based on these results, it is clear that the LSTM models are able to reproduce the *C. Elegans* peripheral input-output behavior on different types of data without the time step having a significant impact. Nevertheless, for an optimal performance, an intermediate value for the time step should be used, also depending on the behavior being modeled.

Table 4.20: The average RMSE obtained for the GRU model for ten simulations, comparing different time steps on both the forward crawling motion and the nictation scenarios.

	$\mathbf{10}$ms	$\mathbf{5}$ms	$\mathbf{1}$ms	$\mathbf{0.5}$ms	$\mathbf{0.1}$ms	$\mathbf{0.05}$ms
Train - PTRS	6.7589e-03	5.4144e-03	3.0509e-03	3.1004e-03	3.1026e-03	3.4269e-03
Valid - PTRS	1.3830e-02	8.4230e-03	5.6968e-03	5.7324e-03	6.8707e-03	7.1296e-03
Train - Nictation	3.3736e-02	2.6283e-02	2.2079e-02	2.4167e-02	2.4583e-02	2.2301e-02
Valid - Nictation	1.1239e-01	8.5702e-02	5.6453e-02	5.4355e-02	5.3394e-02	4.9202e-02

The GRU models tend to perform similarly with data sampled with different time steps between 1ms and 0.05ms, only showing a worse performance for the datasets with a time step of 10ms and 5ms. Nevertheless, this model is able to accurately reproduce the behavior of the dynamic system using data with different time steps, even though for some of these time step values the performance is not as good as for the others.

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 latter producing significantly more accurate predictions than the first. 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 not able 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 than for the others. 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 thesis in order for the NARX to still be comparable with the other models.

5

Experiments and Results

Contents

This chapter reports on the different experiments and corresponding results for the models discussed in Chapter 3 and implemented in Chapter 4 using the different datasets discussed in subchapter 2.3. As in the previous chapter, all the results presented in each table herein, correspond to ten different simulations of the corresponding model with the chosen dataset.

The first experiment performs a comparison of the ability of the different architectures to model the data, also studying the impact of the model sizes on the results, in order to understand what models should be used for the remainder of this thesis, avoiding models which perform poorly. The second experiment tests the ability of the models to replicate the *C. elegans* behaviors using less time series as input than the ones available. The last two experiments concern more demanding tasks, like studying the ability of the models to predict the full set of outputs instead of just a smaller one and testing the models' performance when fed with only one dataset that encapsulates the two behaviours.

5.1 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 in Chapter 4, 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 in Chapter 4, obtaining a lower RMSE for the predictions is the GRU, hence was chosen as the reference model for this experiment.

Three different sizes are tested for the GRU, 8, 16 and 32 units on the hidden layer, using dataset **PTRS-20-0.5** for the PTRS scenario and dataset **Nictation-20-0.5** for the Nictation scenario.

For the GRU with 8 units in the hidden layer, the corresponding model has 348 parameters. The results obtained for this first comparison are available in Table 5.1 and the sizes for the remaining four models that lead to a similar number of parameters are:

- TDNN with 26 units in the hidden layer - 342 parameters
- NARX with 10 units in the hidden layer - 334 parameters
- RNN with 15 units in the hidden layer - 364 parameters
- LSTM with 7 units in the hidden layer - 368 parameters

Table 5.1: 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

The next comparison has as a reference the model using a GRU as a recurrent layer with 16 units, which has 1076 parameters, according to equation (4.9). The results obtained for this comparison are available in Table 5.2 and the sizes for the remaining four models that lead to a similar number of parameters are:

- TDNN with 82 units in the hidden layer - 1070 parameters
- NARX with 32 units in the hidden layer - 1060 parameters
- RNN with 29 units in the hidden layer - 1106 parameters
- LSTM with 14 units in the hidden layer - 1124 parameters

Table 5.2: 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

The last comparison uses as a reference the model with a GRU recurrent layer with 32 units and 3684 parameters, according to equation (4.9). The results obtained for this comparison are available in Table 5.3 and the sizes for the remaining four models that lead to a similar number of parameters are:

- TDNN with 283 units in the hidden layer - 3683 parameters
- NARX with 112 units in the hidden layer - 3700 parameters
- RNN with 56 units in the hidden layer - 3644 parameters
- LSTM with 27 units in the hidden layer - 3568 parameters

Table 5.3: 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

Throughout the comparisons in Tables 5.1, 5.2 and 5.3, it is possible to see that the GRU models are the ones that perform better, always reproducing accurately the behavior of the dynamical system, as discussed previously. Nevertheless, the LSTM models still accurately reproduce the behavior of the system, just not as accurately as the GRU models.

When comparing the rest of the models to these two, there is a clear difference, starting with the NARX, which is able to perform accurately on the PTRS scenario with 10 and 32 units in the hidden layer, but not being able to perform on the Nictation scenario, always producing an unacceptable RMSE value. The poor performance of this model when using a hidden layer of 112 units on the PTRS dataset indicates that the model tends to overfit when the number of parameters of the network increases, not being able to generalize for new data, as already discussed based on the results of Table 4.5.

From the two remaining models, the TDNN models prove their inability to reproduce the system behavior, as also shown earlier, having a large RMSE in both scenarios. The RNN models are able to reproduce some of the tendencies of the system behavior, but present high oscillations in the predictions and have a large RMSE in general.

This comparison indicates that the most suited models from the ones studied in this thesis to model the *C. elegans* behavior are the ones with a GRU recurrent layer. The LSTM models were also used throughout the rest of the thesis, since these are able to reproduce accurately enough the response of the system. 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.

5.2 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 given.

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 the information of the input on the AVB neurons is not available, while for the second one the information of the input on the PLM neurons is missing. The results of these tests concerning PTRS are available in Table 5.4.

Table 5.4: 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

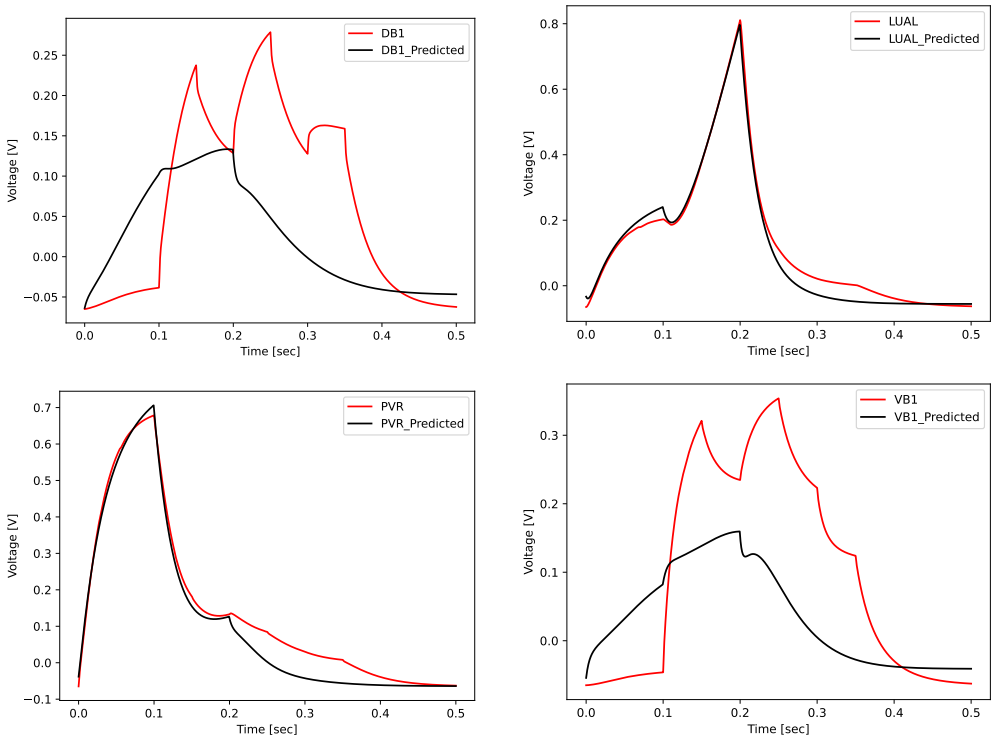

Figure 5.1: Predicted outputs for one simulation of the GRU model with 8 hidden units trained without the input information of the AVB neurons for one of the examples of the test set.

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. A comparison between the expected outputs and the predicted ones for one of the test examples, in which the model fails to produce an accurate response for the DB1 and VB1 neurons can be found in Figure 5.1.

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 the RMSE obtained is 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 increased magnitude of the output voltage of the LUAL and PVR neurons compared to the DB1 and VB1 neurons also justifies the increased RMSE obtained. An example of one of these cases is provided in Figure 5.2.

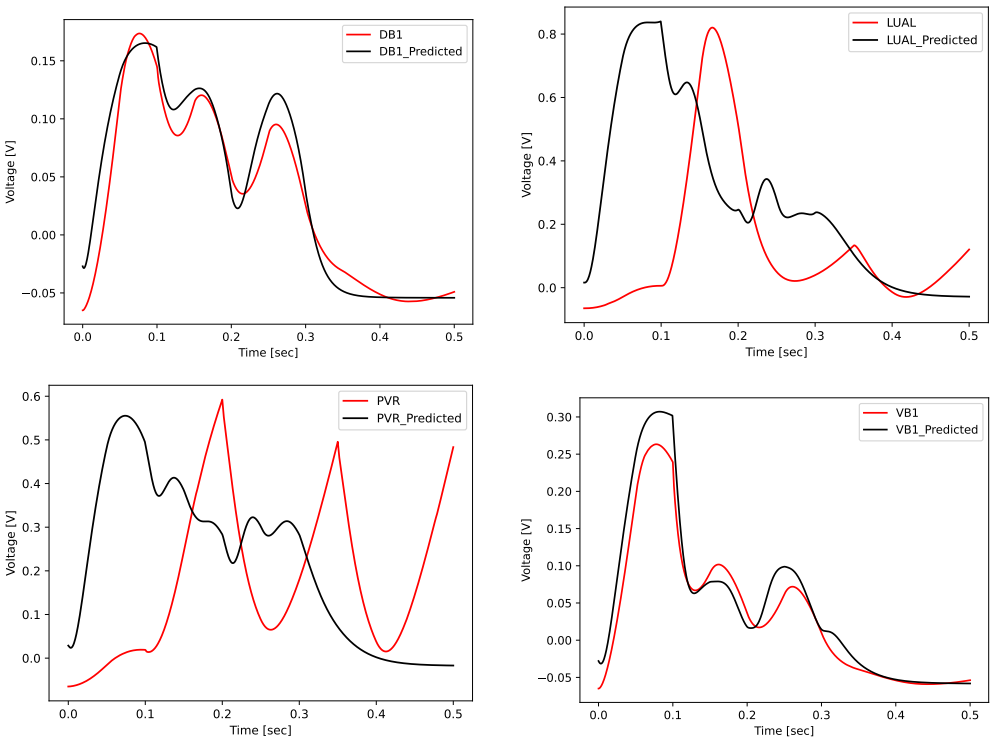

Figure 5.2: Predicted outputs for one simulation of the GRU model with 8 hidden units trained without the input information of the PLM neurons for one of the examples of the test set.

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 5.5.

Table 5.5: 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

The average RMSE of the obtained simulations for the cases with no information on the input pro-

vided to the IL2D and IL2V neurons is of similar 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, as it can be seen in the example of Figure 5.3 for the case with no information on the input given to the IL2D neurons.

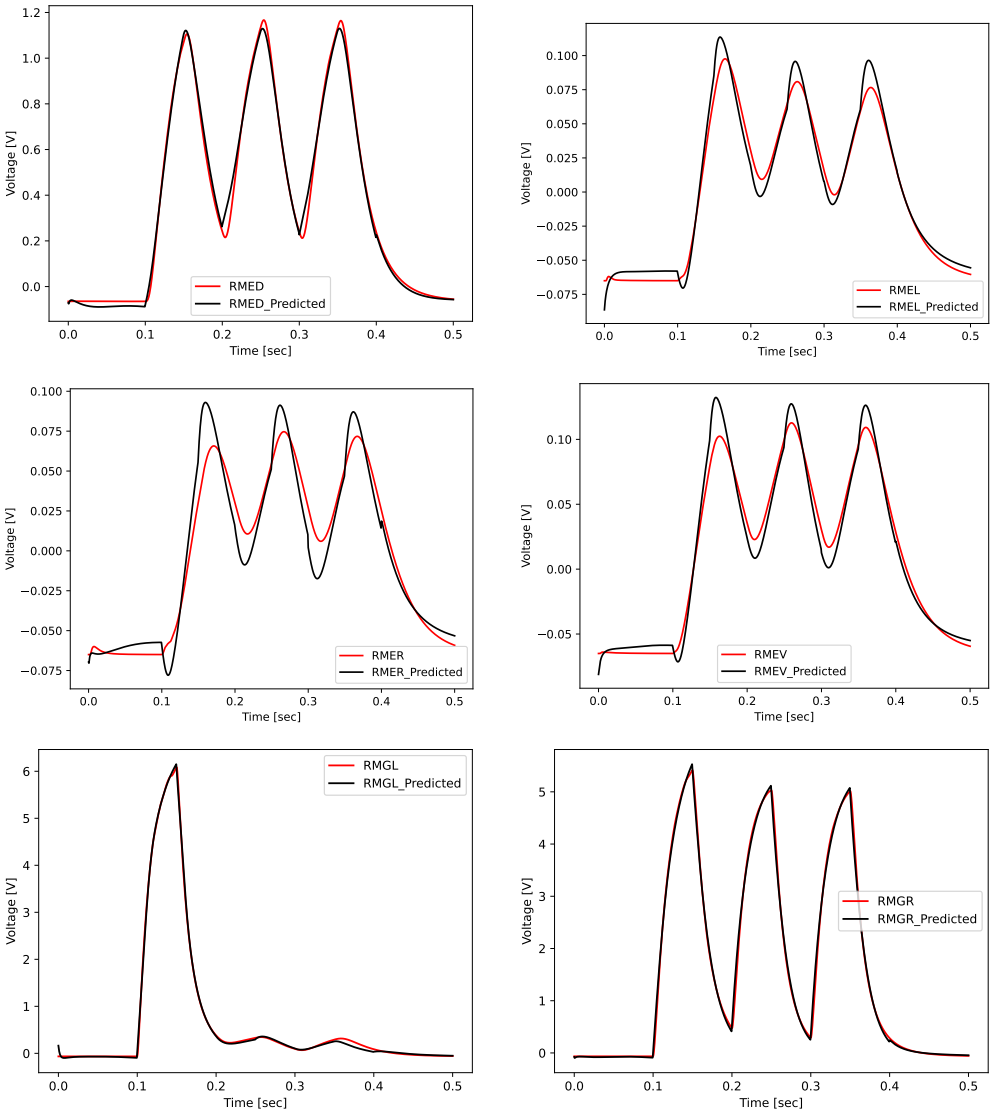

Figure 5.3: Predicted outputs for one simulation of the GRU model with 8 hidden units trained without the input information of the IL2D neurons for one of the examples of the test set.

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. An example of the predicted outputs for one of the examples on the test set, which demonstrates the poor prediction capabilities of these models when being deprived of the IL2 inputs are

shown in Figure 5.4.

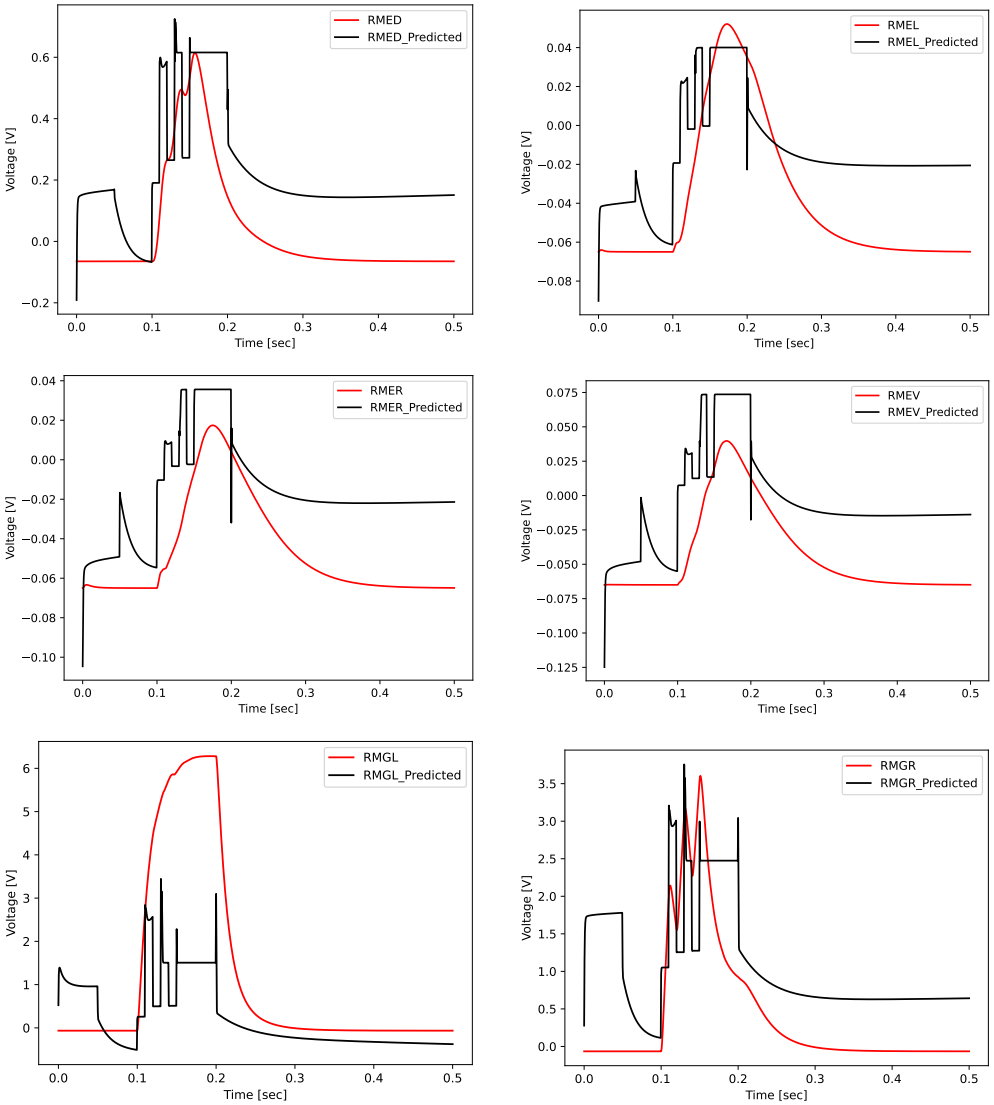

Figure 5.4: Predicted outputs for one simulation of the GRU model with 8 hidden units trained without the input information of the IL2 neurons for one of the examples of the test set.

To sum up, 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 experiment shows 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*.

5.3 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, as described in subchapter 2.3. This was a choice made in order to facilitate the simulations, since when having more output variables the models grow, increasing the complexity of the training procedure, making the models take considerable computational resources and encumber the study.

After all hyperparameters and the time step value are chosen in Chapter 4 and the models are compared in one of the previous experiments, subchapter 5.1, this experiment tests the models prediction capabilities for the full set of output variables measured and listed in subchapter 2.3. 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.

For this experiment, the two datasets used correspond to the PTRS and the Nictation scenarios, **PTRS-Final** and **Nictation-Final**, respectively. These datasets are similar to the ones used previously in subchapters 5.1 and 5.2 but with the full set of outputs instead of just the smaller set used previously. 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. The results obtained can be found in Table 5.6.

Table 5.6: 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

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 simulations performed with the **PTRS-20-0.5** dataset. This indicates that the models perform well in this scenario, as can be observed in Figure 5.5, 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 for dataset **Nictation-20-0.5**, 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, as can be observed in the example of Figure 5.6. 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 and the loss function used was not relative to the magnitude of the predicted values. 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.

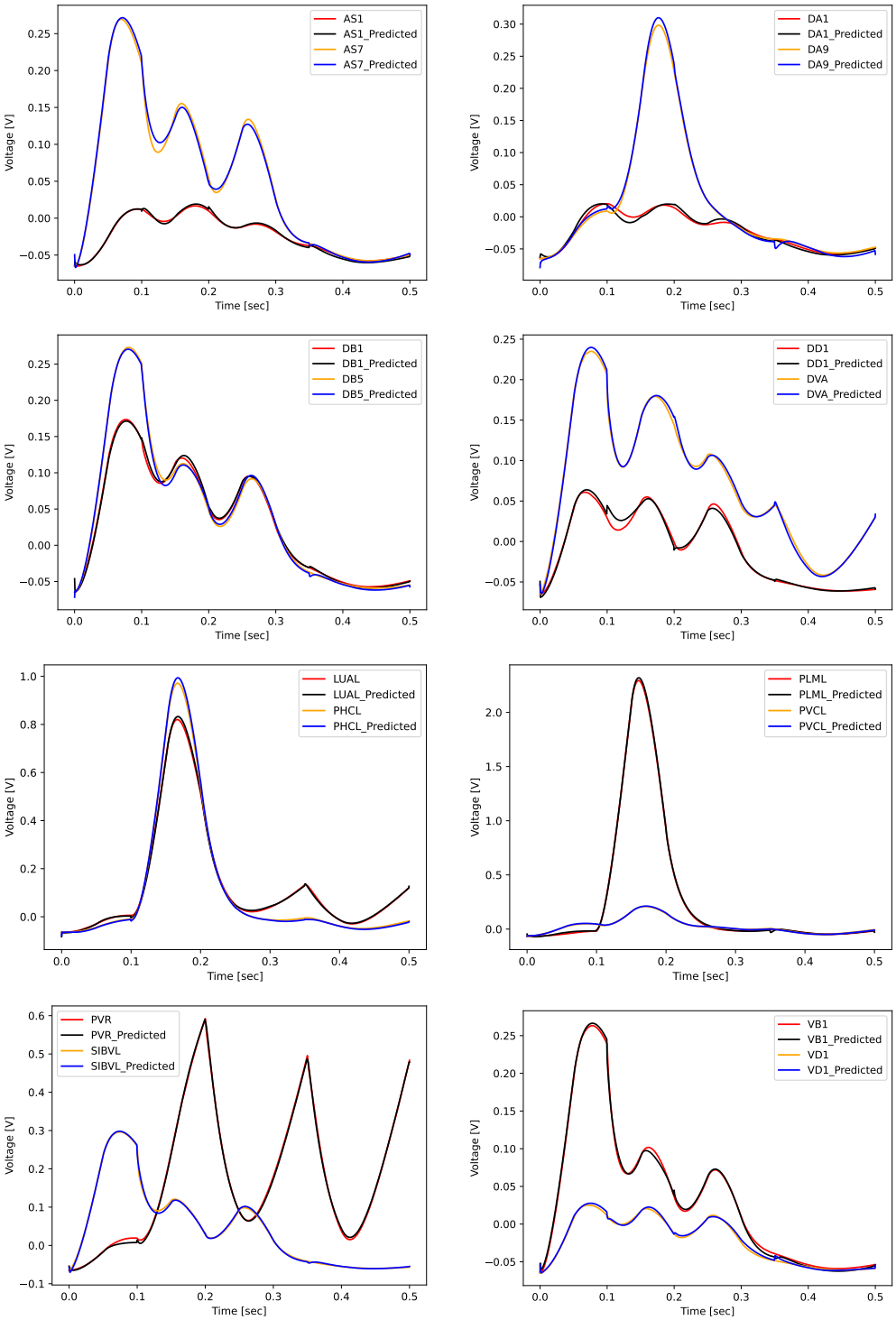

Figure 5.5: Predicted outputs for one simulation of the GRU model with 16 hidden units for one of the examples of the test set using a dataset with the 16 outputs measured on the PTRS scenario.

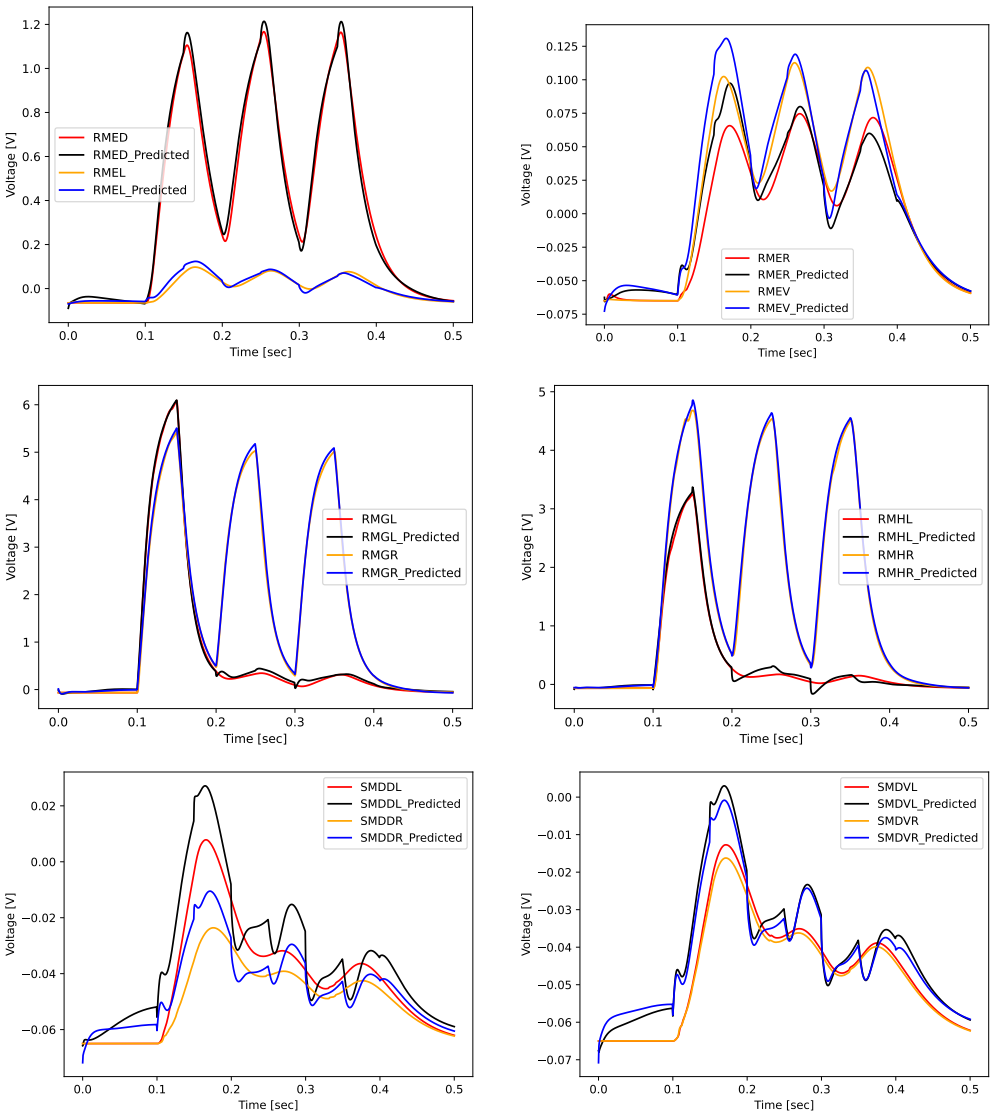

Figure 5.6: Predicted outputs for one simulation of the GRU model with 16 hidden units for one of the examples of the test set using a dataset with the 12 outputs measured on the Nictation scenario.

5.4 Predicting Two Behaviors Simultaneously

This last experiment tests the performance of the models on only one dataset, as opposed to the previous experiments. This dataset is the one discussed at the end of subchapter 2.3, **Two-Behaviors-Data**, where the two scenarios are combined. 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 models tested are the same as for the two previous experiments, LSTM and GRU models with

a recurrent layer size of 8, 16 and 32. The results obtained can be found in Table 5.7.

Table 5.7: 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

The results obtained indicate a slightly superior performance of the RNNs using a GRU as the unit for the recurrent layer, 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.

An example of the predictions provided by the GRU model with 16 units on the recurrent layer is represented in Figures 5.7 and 5.8 for an example of PTRS behavior, while another example for the Nictation behavior is available in Figures 5.9 and 5.10.

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. As it can be seen in Figures 5.8 and 5.9, that 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.

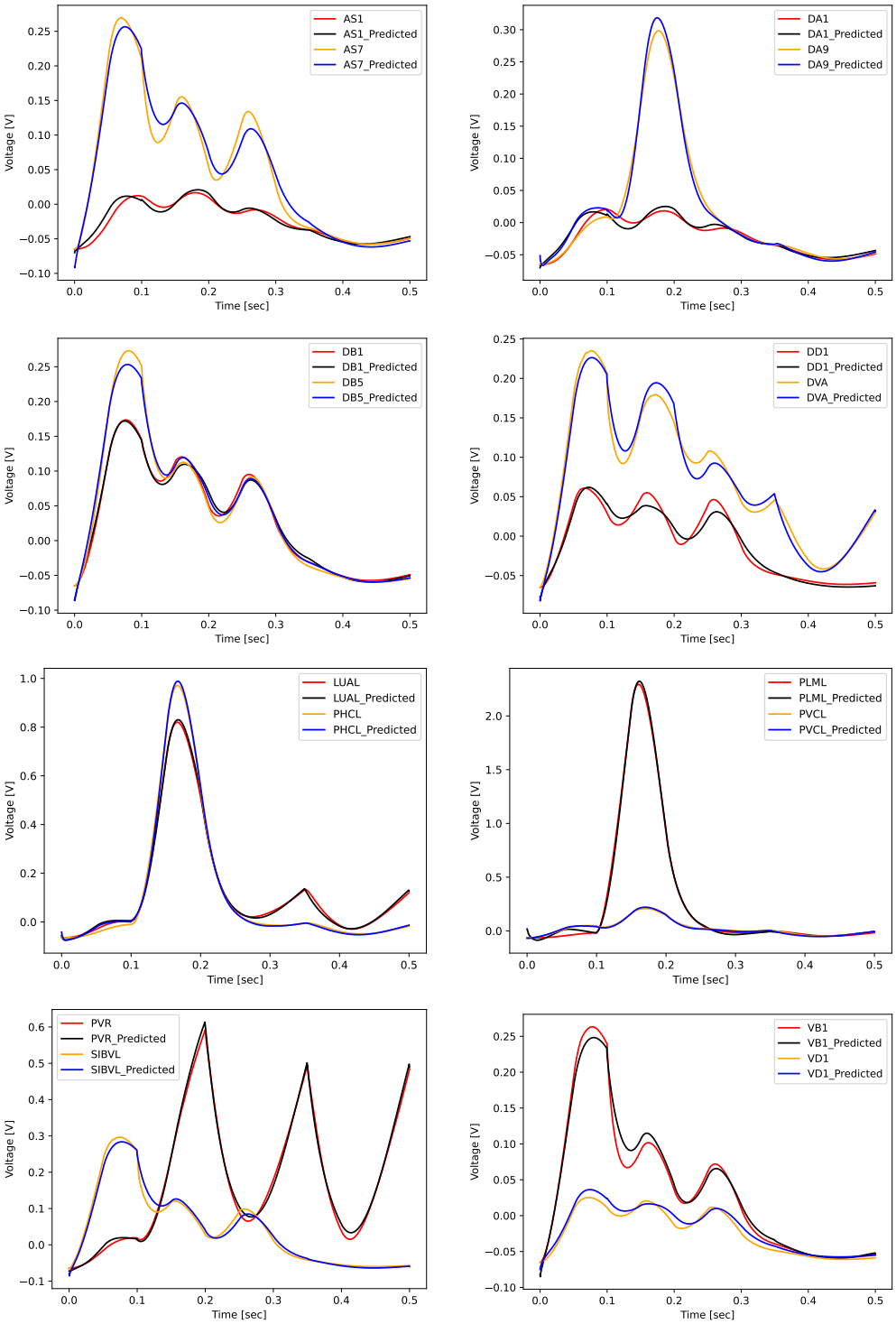

Figure 5.7: Predicted outputs related to the PTRS scenario for one simulation of the GRU model with 16 hidden units for one of the examples of the test set of a PTRS behavior using a dataset with data from both PTRS and Nictation behaviors

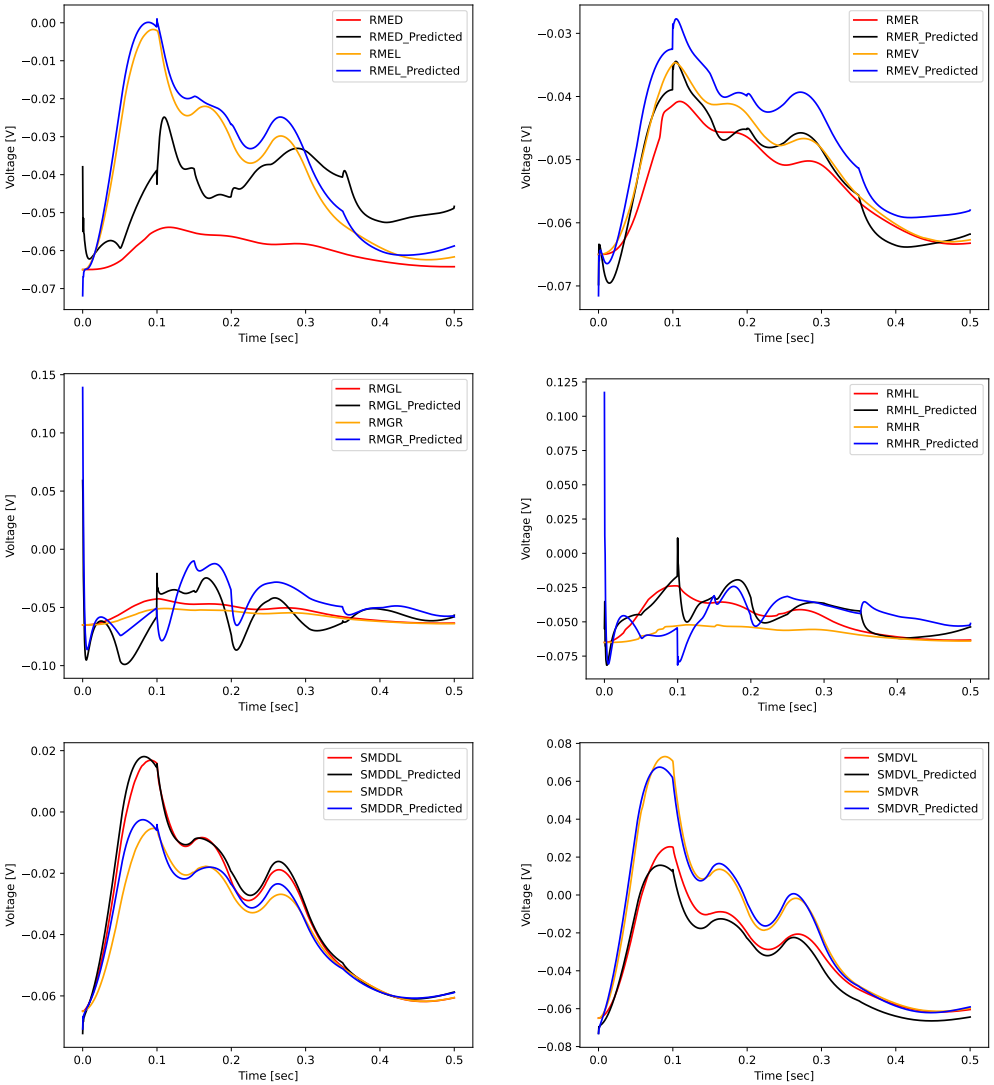

Figure 5.8: Predicted outputs related to the Nictation scenario for one simulation of the GRU model with 16 hidden units for one of the examples of the test set of a PTRS behavior using a dataset with data from both PTRS and Nictation behaviors

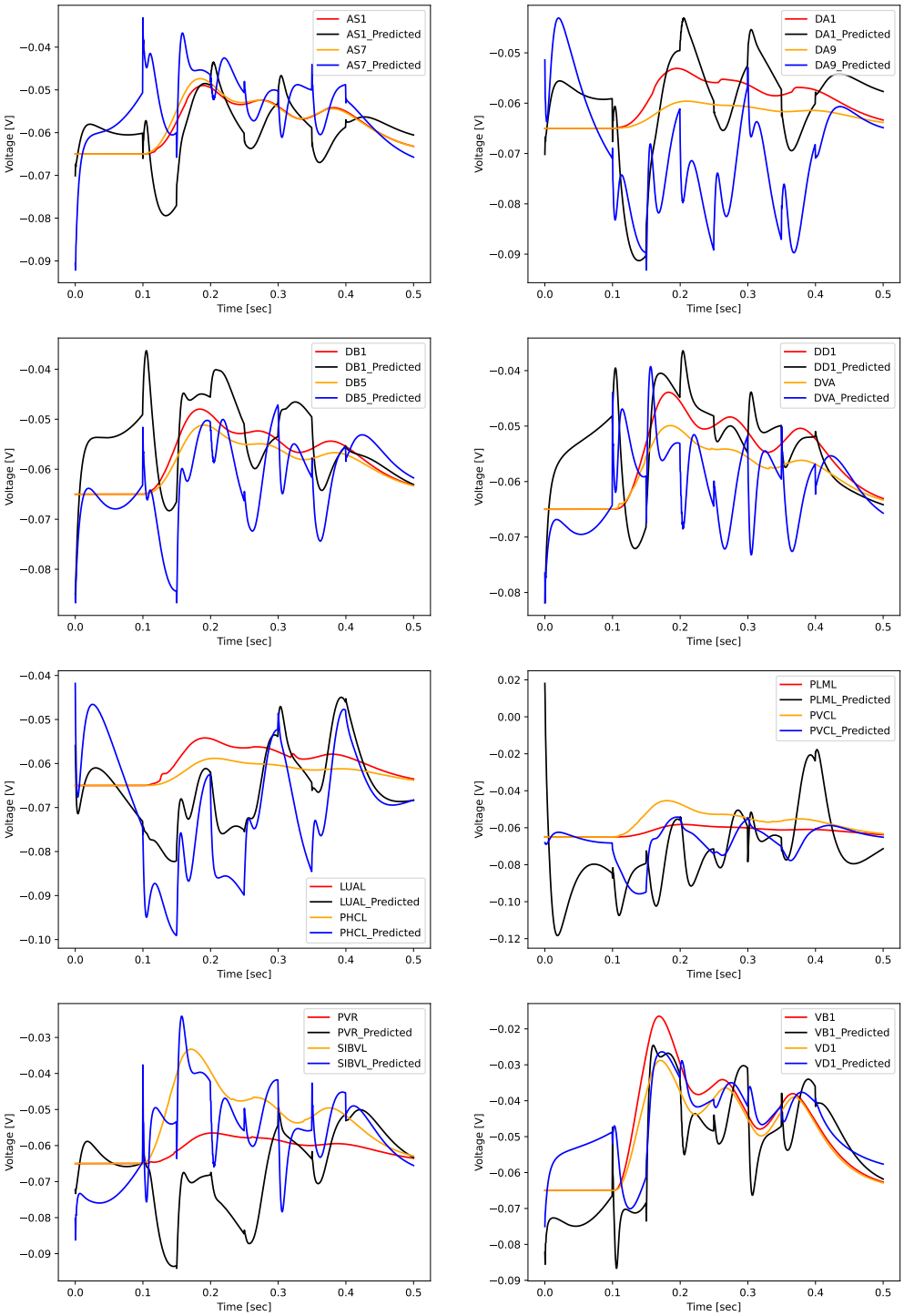

Figure 5.9: Predicted outputs related to the Nictation scenario for one simulation of the GRU model with 16 hidden units for one of the examples of the test set of a PTRS behavior using a dataset with data from both PTRS and Nictation behaviors

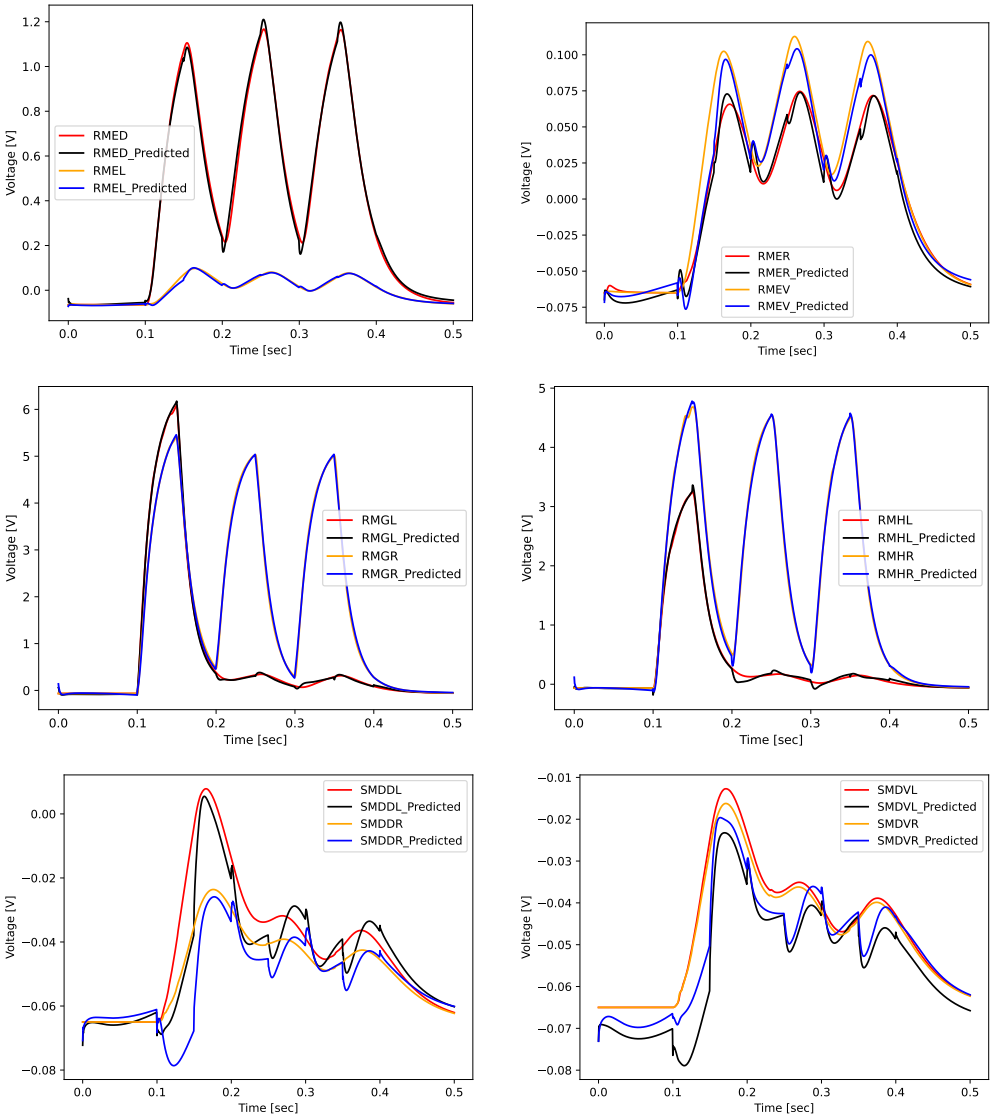

Figure 5.10: Predicted outputs related to the Nictation scenario for one simulation of the GRU model with 16 hidden units for one of the examples of the test set of a Nictation behavior using a dataset with data from both PTRS and Nictation behaviors

6

Conclusions

Contents

6.1 Discussion

To address the main goal of this dissertation, different datasets on two behavioral scenarios of the *C. elegans* nervous system were generated using the NEURON simulator. These datasets contain information regarding the behavior of the worm from a peripheral input-output perspective given that they include information about the stimuli used and the response observed at designated output neurons. The datasets are therefore composed of the time series of currents used to stimulate certain neurons and the voltage time series observed for other set of neurons, considered outputs.

The machine learning techniques that can be applied to perform the task at hand, predicting a set of output time series having the information of the input time series given to the system, were then reviewed. Based on the review of available work, different neural network architectures were chosen to model the behavior of the *C. elegans* on two different tasks, PTRS and Nictation.

The methodology used for the implementation of the different models was motivated and explained, along with the evaluation criteria used for the implemented models. Machine learning models in general are supported on a set of hyperparameters which need to be set before parameter fitting is conducted. Therefore some of the most relevant hyperparameters were optimized using data from the PTRS scenario, and tests on the effect of the sample rate used for the data generation were made using data from both the PTRS and the Nictation behaviors.

The different models were then compared, and the two best performing models, LSTM and GRU, were then used to conduct three different experiments. These two models 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, without a need to increase the size of the network used, proving their usefulness and scalability capacity to predict larger sets of outputs.

The last experiment made concerns the prediction of both the PTRS and the Nictation behaviors using a single model, in which the LSTM and GRU models proved their success again. During this process, several small issues were identified, which need to be considered, and some questions were raised, that need to be addressed, all together leading to a set of suggestions on how to handle them.

The GRU proved to be the best model for the tests made, almost always having the lowest average RMSE and also requiring a smaller amount of parameters to compute, when compared to the other model which performed well, the LSTM.

6.2 Future Work

As one would expect when working in such a complex topic where two different fields of science converge, future work may take many directions, for which different suggestions are provided next.

This set of suggestions starts by commenting on future work that can be done in the neuroscience field based on this work and the results provided. With the reported success of two of the experimented models in terms of their ability to accurately replicate the *C. elegans* behavior, these may be used with more targeted datasets, specifically designed by neuroscience researchers, which may be helpful in order to gain insight into the dynamics of this nervous system. The models developed can be applied to synthetic data on different behaviors than the ones used in this thesis, using real *C. elegans* data or even testing if the different artificial neural networks 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.

Regarding future work in the machine learning field, there are also many techniques that were not applied on this thesis, which may be explored in order to model this data in subsequent future works, such as KNNR, SVR, GPR, DeepESN, among others. Two of the three models which didn't prove successful may also be further looked upon, since they have shown to be able to learn how to replicate part of the *C. elegans* behavior. Not only additional studies on the capability of the RNN to learn the dynamical system and its costs may be of further interest, but also a study on why the NARX model fails to provide good predictions for the Nictation behavior and how this can be improved.

The training process of the used models can be improved, since the majority of the settings were fixed to default values, because of the high computational costs and the amount of models experimented which did not allow for a deeper study on the tuning of each setting. The default optimizer and the number of epochs of the training process were also fixed, which are parameters of interest to experiment with in the future with the goal of improving the predictions of the learned models. 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. This is indeed a particularly fruitful area in which to find new research topics.

Bibliography

[1] F. Zafeirios, "Spiking neural networks for human-like avatar control in a simulated environment," *M.S.c. thesis, Dept. Comput., Imperial College London*, 08 2011.

[2] J. Lightner, "Developmental system plasticity - a brief initial assessment of extent, design, and purpose within the creation model." *Journal of Creation*, vol. 28, pp. 67–72, 01 2014.

[3] 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*, vol. 513, no. 5, pp. 532–541, 2009. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cne.21974

[4] K. Ryan, Z. Lu, and I. A. Meinertzhagen, "The CNS connectome of a tadpole larva of *Ciona intestinalis* (l.) highlights sidedness in the brain of a chordate sibling," *eLife*, vol. 5, p. e16962, dec 2016. [Online]. Available: https://doi.org/10.7554/eLife.16962

[5] J. G. White, E. Southgate, J. N. Thomson, and S. Brenner, "The structure of the nervous system of the nematode caenorhabditis elegans," *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, vol. 314, no. 1165, pp. 1–340, 1986. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rstb.1986.0056

[6] A. P. Alivisatos, M. Chun, G. M. Church, R. J. Greenspan, M. L. Roukes, and R. Yuste, "The brain activity map project and the challenge of functional connectomics," *Neuron*, vol. 74, no. 6, pp. 970–974, Jun. 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0896627312005181

[7] S. Herculano-Houzel, B. Mota, and R. Lent, "Cellular scaling rules for rodent brains," *Proceedings of the National Academy of Sciences*, vol. 103, no. 32, pp. 12 138–12 143, 2006. [Online]. Available: https://www.pnas.org/content/103/32/12138

[8] S. Herculano-Houzel, K. A. de Souza, K. Neves, J. Porfirio, D. J. Messeder, L. M. Feijó, J. Maldonado, and P. Manger, "The elephant brain in numbers," *Frontiers in Neuroanatomy*, vol. 8, 2014.

[9] 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*, vol. 571, no. 7763, pp. 63–71, 2019.

[10] 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*, vol. 7, no. 2, p. e1001066, 2011.

[11] E. Widmaier, H. Raff, and K. Strang, *ISE Vander's Human Physiology*. McGraw-Hill Education, 2018. [Online]. Available: https://books.google.pt/books?id=IPneswEACAAJ

[12] A. Huxley and R. Stämpeli, "Evidence for saltatory conduction in peripheral myelinated nerve fibres," *The Journal of Physiology*, vol. 108, 1949.

[13] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of Physiology*, vol. 117, no. 4, pp. 500–544, 1952. [Online]. Available: https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1952.sp004764

[14] E. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[15] E. M. Izhikevich, "Which model to use for cortical spiking neurons?" *IEEE transactions on neural networks*, vol. 15, no. 5, pp. 1063–1070, 2004.

[16] B. Yan and P. Li, "Reduced order modeling of passive and quasi-active dendrites for nervous system simulation," *Journal of computational neuroscience*, vol. 31, no. 2, pp. 247–271, 2011.

[17] D. Ioan, R. Bărbulescu, L. M. Silveira, and G. Ciuprina, "Reduced order models of myelinated axonal compartments," *Journal of computational neuroscience*, vol. 47, no. 2, pp. 141–166, 2019.

[18] J. Stapmanns, J. Hahne, M. Helias, M. Bolten, M. Diesmann, and D. Dahmen, "Event-based update of synapses in voltage-based learning rules," *Frontiers in neuroinformatics*, vol. 15, 2021.

[19] A. Morrison, M. Diesmann, and W. Gerstner, "Phenomenological models of synaptic plasticity based on spike timing," *Biological cybernetics*, vol. 98, no. 6, pp. 459–478, 2008.

[20] M. Mikaitis, G. Pineda García, J. C. Knight, and S. B. Furber, "Neuromodulated synaptic plasticity on the spinnaker neuromorphic system," *Frontiers in neuroscience*, vol. 12, p. 105, 2018.

[21] A. Nebot, F. Mugica, F. E. Cellier, and M. Vallverdú, "Modeling and simulation of the central nervous system control with generic fuzzy models," *SIMULATION*, vol. 79, no. 11, pp. 648–669, 2003. [Online]. Available: https://doi.org/10.1177/0037549703038883

[22] M. G. Pandy, “Computer modeling and simulation of human movement,” *Annual Review of Biomedical Engineering*, vol. 3, no. 1, pp. 245–273, 2001, pMID: 11447064. [Online]. Available: https://doi.org/10.1146/annurev.bioeng.3.1.245

[23] 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*, 2010, pp. 1–6.

[24] P. Arena, L. Patané, and P. S. Termini, “An insect brain computational model inspired by drosophila melanogaster: Simulation results,” in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, 2010, pp. 1–8.

[25] Örjan Ekeberg, M. Blümel, and A. Büschges, “Dynamic simulation of insect walking,” *Arthropod Structure & Development*, vol. 33, no. 3, pp. 287–300, 2004, arthropod Locomotion Systems: from Biological Materials and Systems to Robotics. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1467803904000283

[26] F. Melozzi, M. M. Woodman, V. K. Jirsa, and C. Bernard, “The virtual mouse brain: A computational neuroinformatics platform to study whole mouse brain dynamics,” *Eneuro*, vol. 4, no. 3, 2017.

[27] 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.

[28] 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*, vol. 314, no. 1165, pp. 1–340, 1986.

[29] C. A. Brittin, S. J. Cook, D. H. Hall, S. W. Emmons, and N. Cohen, “Beyond the connectome: A map of a brain architecture derived from whole-brain volumetric reconstructions,” *bioRxiv*, 2020.

[30] “Wormbase, ws280,” https://www.wormbase.org/, accessed: 2021-09-02.

[31] “Gene expression database,” https://www.gfpworm.org/, accessed: 2021-09-03.

[32] 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*, vol. 5, no. 9, p. e237, 2007.

[33] “Wormbook, the online review of c. elegans biology,” http://www.wormbook.org/, accessed: 2021-09-03.

[34] 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*, vol. 4, no. 4, pp. 733–747, 2014.

[35] 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.

[36] A. E. C. o. M. Department of Neuroscience, “The WormImage Database,” https://www.wormimage.org/, accessed: 2021-09-03.

[37] “Neuromorph.org,” https://neuromorph.org/, accessed: 2021-09-04.

[38] “Open Source Brain,” https://www.opensourcebrain.org/, accessed: 2021-09-04.

[39] 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*, vol. 103, no. 3, pp. 395–411, 2019.

[40] “OpenWorm,” http://openworm.org/index.html, accessed: 2021-09-04.

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

[42] 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*, vol. 7, no. 2, p. e1001066, 2011.

[43] 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*, vol. 571, no. 7763, pp. 63–71, 2019.

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

[45] 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*, vol. 337, no. 6093, pp. 437–444, 2012.

[46] T. Ferrée, B. Marcotte, and S. Lockery, "Neural network models of chemotaxis in the nematode caenorhabditis elegans," in *Advances in Neural Information Processing Systems*, M. C. Mozer, M. Jordan, and T. Petsche, Eds., vol. 9. MIT Press, 1997. [Online]. Available: https://proceedings.neurips.cc/paper/1996/file/4d2e7bd33c475784381a64e43e50922f-Paper.pdf

[47] T. C. Ferree and S. R. Lockery, "Computational rules for chemotaxis in the nematode c. elegans," *Journal of computational neuroscience*, vol. 6, no. 3, pp. 263–277, 1999.

[48] N. A. Dunn, S. R. Lockery, J. T. Pierce-Shimomura, and J. S. Conery, "A neural network model of chemotaxis predicts functions of synaptic connections in the nematode caenorhabditis elegans," *Journal of computational neuroscience*, vol. 17, no. 2, pp. 137–147, 2004.

[49] N. A. Dunn, J. T. Pierce-Shimomura, J. S. Conery, and S. R. Lockery, "Clustered neural dynamics identify motifs for chemotaxis in caenorhabditis elegans," in *The 2006 IEEE International Joint Conference on Neural Network Proceedings*. IEEE, 2006, pp. 547–554.

[50] 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)*, 2015, pp. 668–673.

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

[52] M. Suzuki, T. Tsuji, and H. Ohtake, "A model of motor control of the nematode C. elegans with neuronal circuits," *Artificial Intelligence in Medicine*, vol. 35, no. 1-2, pp. 75–86, 2005.

[53] M. Suzuki, H. Ohtake, and T. Tsuji, "A dynamic body model of the nematode C. elegans with a touch-response circuit," in *2005 IEEE International Conference on Robotics and Biomimetics-ROBIO*. IEEE, 2005, pp. 538–543.

[54] Y. Iwasaki and S. Gomi, "Stochastic formulation for a partial neural circuit of c. elegans," *Bulletin of mathematical biology*, vol. 66, no. 4, pp. 727–743, 2004.

[55] Y. Iwasaki, "Effective modeling of a partial neural network of c. elegans: Theory and application," in *International Congress Series*, vol. 1291. Elsevier, 2006, pp. 125–128.

[56] 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)*. IEEE, 2015, pp. 1–8.

[57] N. Agarwal, N. Mehta, A. C. Parker, and K. Ashouri, "C. elegans neuromorphic neural network exhibiting undulating locomotion," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 3912–3921.

[58] 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*, 2005, pp. 167–172.

[59] E. Lanza, S. Di Angelantonio, G. Gosti, G. Ruocco, and V. Folli, “A recurrent neural network model of c. elegans responses to aversive stimuli,” *Neurocomputing*, vol. 430, pp. 1–13, 2021.

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

[61] 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*, vol. 11, no. 3, 4, pp. 137–147, 2012.

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

[63] 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*, vol. 15, no. 1, pp. 107–112, 2012.

[64] N. Sinha, M. Gupta, and D. Rao, “Dynamic neural networks: an overview,” in *Proceedings of IEEE International Conference on Industrial Technology 2000 (IEEE Cat. No. 00TH8482)*, vol. 1. IEEE, 2000, pp. 491–496.

[65] Y. Sun, L. Zhang, and H. Schaeffer, “Neupde: Neural network based ordinary and partial differential equations for modeling time-dependent data,” in *Mathematical and Scientific Machine Learning*. PMLR, 2020, pp. 352–372.

[66] Z. K. Malik, A. Hussain, and Q. J. Wu, “Multilayered echo state machine: A novel architecture and algorithm,” *IEEE Transactions on cybernetics*, vol. 47, no. 4, pp. 946–959, 2016.

[67] F. Regazzoni, L. Dede, and A. Quarteroni, “Machine learning for fast and reliable solution of time-dependent differential equations,” *Journal of Computational physics*, vol. 397, p. 108852, 2019.

[68] X. Jin, X. Yu, X. Wang, Y. Bai, T. Su, and J. Kong, “Prediction for time series with CNN and LSTM,” in *Proceedings of the 11th International Conference on Modelling, Identification and Control (ICMIC2019)*. Springer, 2020, pp. 631–641.

[69] N. Tavakoli, “Modeling genome data using bidirectional LSTM,” in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2. IEEE, 2019, pp. 183–188.

[70] S. Hwang, G. Jeon, J. Jeong, and J. Lee, “A novel time series based seq2seq model for temperature prediction in firing furnace process,” *Procedia Computer Science*, vol. 155, pp. 19–26, 2019.

[71] C. Mitrea, C. Lee, and Z. Wu, “A comparison between neural networks and traditional forecasting methods: A case study,” *International journal of engineering business management*, vol. 1, p. 11, 2009.

[72] N. K. Ahmed, A. F. Atiya, N. E. Gayar, and H. El-Shishiny, “An empirical comparison of machine learning models for time series forecasting,” *Econometric Reviews*, vol. 29, no. 5-6, pp. 594–621, 2010.

[73] G. Bontempi, S. B. Taieb, and Y.-A. Le Borgne, “Machine learning strategies for time series forecasting,” in *European business intelligence summer school*. Springer, 2012, pp. 62–77.

[74] S. Thapa, Z. Zhao, B. Li, L. Lu, D. Fu, X. Shi, B. Tang, and H. Qi, “Snowmelt-driven streamflow prediction using machine learning techniques (lstm, narx, gpr, and svr),” *Water*, vol. 12, no. 6, p. 1734, 2020.

[75] C. Gallicchio, A. Micheli, and L. Pedrelli, “Comparison between DeepESNs and gated RNNs on multivariate time-series prediction,” *arXiv preprint arXiv:1812.11527*, 2018.

[76] A. Wunsch, T. Liesch, and S. Broda, “Groundwater level forecasting with artificial neural networks: A comparison of LSTM, CNN and NARX,” *Hydrology and Earth System Sciences Discussions*, vol. 2020, pp. 1–23, 2020.

[77] P. Filonov, A. Lavrentyev, and A. Vorontsov, “Multivariate industrial time series with cyber-attack simulation: Fault detection using an LSTM-based predictive data model,” *arXiv preprint arXiv:1612.06676*, 2016.

[78] A. Ben Said, A. Erradi, H. Aly, and A. Mohamed, “Predicting COVID-19 cases using bidirectional LSTM on multivariate time series,” *arXiv e-prints*, pp. arXiv–2009, 2020.

[79] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. Cottrell, “A dual-stage attention-based recurrent neural network for time series prediction,” *arXiv preprint arXiv:1704.02971*, 2017.

[80] Y. Yuan, L. Lin, L.-Z. Huo, Y.-L. Kong, Z.-G. Zhou, B. Wu, and Y. Jia, “Using an attention-based LSTM encoder–decoder network for near real-time disturbance detection,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 13, pp. 1819–1832, 2020.

[81] H. Wan, S. Guo, K. Yin, X. Liang, and Y. Lin, “CTS-LSTM: LSTM-based neural networks for correlatedtime series prediction,” *Knowledge-Based Systems*, vol. 191, p. 105239, 2020.

[82] M. Massaoudi, I. Chihi, L. Sidhom, M. Trabelsi, S. S. Refaat, and F. S. Oueslati, “A novel approach based deep RNN using hybrid NARX-LSTM model for solar power forecasting,” *arXiv preprint arXiv:1910.10064*, 2019.

[83] D. Cao, Y. Wang, J. Duan, C. Zhang, X. Zhu, C. Huang, Y. Tong, B. Xu, J. Bai, J. Tong *et al.*, "Spectral temporal graph neural network for multivariate time-series forecasting," *arXiv preprint arXiv:2103.07719*, 2021.

[84] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, "Modeling long-and short-term temporal patterns with deep neural networks," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2018, pp. 95–104.

[85] 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*, vol. 37, no. 3, pp. 328–339, 1989.

[86] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, ser. Springer series in statistics. Springer, 2009. [Online]. Available: https://books.google.pt/books?id=eBSgoAEACAAJ

[87] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biology*, vol. 5, no. 4, pp. 115–133, Dec. 1943. [Online]. Available: http://dx.doi.org/10.1007/bf02478259

[88] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958. [Online]. Available: http://dx.doi.org/10.1037/h0042519

[89] H. J. Kelley, "Gradient theory of optimal flight paths," *Ars Journal*, vol. 30, no. 10, pp. 947–954, 1960.

[90] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-propagating Errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. [Online]. Available: http://www.nature.com/articles/323533a0

[91] T. Lin, B. Horne, P. Tino, and C. Giles, "Learning long-term dependencies in NARX recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 7, no. 6, pp. 1329–1338, 1996.

[92] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Networks*, vol. 1, no. 4, pp. 339–356, 1988. [Online]. Available: https://www.sciencedirect.com/science/article/pii/089360808890007X

[93] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1

[94] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[95] R. Pascanu, T. Mikolov, and Y. Bengio, “Understanding the exploding gradient problem,” *CoRR*, vol. abs/1211.5063, 2012. [Online]. Available: http://arxiv.org/abs/1211.5063

[96] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[97] 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)*, vol. 2, 1999, pp. 850–855 vol.2.

[98] 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*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

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

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

[101] F. Chollet *et al.*, “Keras,” https://keras.io, 2015.

[102] 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. [Online]. Available: https://www.tensorflow.org/

[103] K. Levenberg, “A method for the solution of certain non-linear problems in least squares,” *Quarterly of applied mathematics*, vol. 2, no. 2, pp. 164–168, 1944.

[104] D. W. Marquardt, “An algorithm for least-squares estimation of nonlinear parameters,” *Journal of the society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[105] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980