Politecnico di Torino

MEDIZINISCHE UNIVERSITÄT WIEN

POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Biomedica

Tesi di Laurea Magistrale

Model reduction and linearisation of a physiological dynamical system using deep learning-based Koopman analysis

Supervisors:

Prof. Filippo Molinari

Assoc. Prof. Francesco Moscato

Dipl.-Ing Max Haberbusch

Candidate:

Martina Aprile

AY 2022-2023

Abstract

Computational models give us insights into physiological processes and help to develop novel treatment strategies in the biomedical field. However, their complexity and non-linearity make them computationally expensive and challenging to analyse. Reduced or linearised representations of these models can help overcome these issues.

A potential application of such models addresses the design of physiological control algorithms for vagus nerve stimulation (VNS). Here models will be essential to control stimulation and regulate the cardiovascular system function improving clinical outcomes.

In this thesis steps to address this problem have been investigated using a deep neural network approach to Koopman analysis (DNN-KA). DNN-KA uses deep learning to globally linearise dynamics on a low-dimensional manifold by identifying non-linear coordinates with a modified autoencoder. The training optimised a combined cost function comprising a reconstruction-, prediction-, and linearity term. The general procedure involves data gathering, normalization, splitting, network training, and quantification of prediction accuracy. The practical work consisted of three main parts. First, to ensure validity of the implementation, the results reported in literature for a simple non-linear toy model were reproduced. Second, the approach was applied to the Fitz-Hugh-Nagumo (FHN) model. Random search was performed on hyperparameters to optimise prediction accuracy which included, but were not limited to, autoencoder width (w), number of layers in the autoencoder (d). Forty-eight DNN were trained on a dataset generated by numerically solving the FHN model for 1000 initial conditions and 30 timesteps. Finally, a preliminary exploration was conducted on applying the DNN-KA to identify the linear embeddings of selected hemodynamic and electrophysiological signals using a model architecture with predetermined parameters. Therefore, a dataset was generated by numerically solving a 0-D cardiovascular model for 5000 initial conditions, extracting a segment of 71 timesteps centred at the onset of VNS.

For the toy model, the training- and validation errors were in the magnitude of 10^{-7} which is in accordance with the results reported in literature, and the average root mean square error (ARMSE) was 0.2. For the FHN model, the training- and validation errors and ARMSE, for the best DNN architecture ($w = 128, d = 3$), were $8.8 \cdot 10^{-5}$, $1.1 \cdot 10^{-4}$, and 14.1, respectively. For the cardiovascular model, the training and validation errors and the ARMSE for the best combination of input parameters were $1.9 \cdot 10^{-5}$, $5 \cdot 10^{-5}$, and 7.2. For the FHN model, the prediction accuracy and lin-

earity were satisfying. However, the linearisation of the model came at a cost of computational time subjected to about a 5-fold increase, suggesting that model order reduction and linearisation may have competing effects on computational cost. Furthermore, it cannot be excluded that there might be better-performing DNN architectures, as the number of architectures tested was limited. The results for the cardiovascular system model were also promising, however, further work is needed, which includes an hyperparameter optimisation and expansion of the number of used hemodynamic signals.

Overall, the results are encouraging and add to the evidence supporting the usefulness of DNN-KA and its application to identify linear embeddings for hemodynamic signals is novel. This approach can be of particular interest for the development of physiological control algorithms in the future.

Keywords: dynamical systems, deep learning, model reduction, model linearisation, Koopman analysis, cardiovascular system, vagus nerve stimulation.

Acknowledgement

This master's thesis was carried out at the Center for Medical Physics and Biomedical Engineering of the Medical University of Vienna / Vienna General Hospital (AKH) under the supervision of Professor Francesco Moscato and Max Haberbusch. The thesis was partially funded by the European Project H2020-EU.1.2.2, "A Neuroprosthesis to Restore the Vagal-Cardiac Closed-Loop Connection after Heart Transplantation, NeuHeart" (Grant Agreement ID: 824071) and the Ludwig Boltzmann Institute for Cardiovascular Research. Parts of the computational results presented in this thesis were processed using the Vienna Scientific Cluster supercomputer.

Firstly, I would like to express my gratitude to Professor Moscato for giving me the opportunity to undertake this project in his department. He was always available to address any concerns I had and taught me a lot, from theory to how to conduct myself in a research setting to the concept of enjoying life.
Secondly, Max, who sat only two seats away from me, was extremely into in my project. He provided me with support from the outset and offered advice on a daily basis. I am truly thankful to him for his support.
Furthermore, I would like to sincerely thank Professor Molinari for agreeing to guide me in my thesis and for providing me with his support even from a distance.
My parents and sisters have always cheered me on. It sounds trivial, but I feel so lucky to have them as family. When I was feeling down, all I had to do was make a video call with them to get back to a positive mood.
And then, there was Milan, who accompanied me throughout the entire path and witnessed my worst moments. However, whenever I felt like giving up, he was always there to encourage me and push me to finish the work. I am immensely grateful to him.
I also thank my grandparents (even those far away from this world), aunts, uncles, cousins and little cousins who, even from afar, have always made me feel their warmth.
Special thanks go to Giusy and Jessica, my wonderful cousins, who have always supported me.
I thank Silvia for introducing me to this wonderful world and for sharing the first part of this experience with me. It was exciting doing the journeys together, meeting in the bathroom for rest breaks and supporting each other. In the second part, she made me feel her closeness by always wanting to be part of my life.

My lifelong friends Giusy, Marta, Cristiana, Margaret and Martina were a pillar for me, and I felt them so close during this time, more than ever.
I would like to thank my friends from Poli: Ginevra, Claudia, Laura, Gianluca, Catello and Matteo. They have transmitted their love to me from all over the world. And my roommates in Turin, Chiara and Manuela, who have always been there for me, even if only for advices or a talk.
I would also like to express my recognition to the guys of Room 18 - Lorenzo, Marta, Laurenz, Tina, Daniel, Nando, and Maria - as well as to those from the baby shark Room - Bruna, Lorenzo, and Ádam - and not just those in Room 18 - Daria, Massi, and Giulia. Without their sense of humor, coffee breaks with Scopa games, and Mensa at 11:59, the entire journey wouldn't have been that enjoyable.
I would like to gratefulness Ana and Cristina, my besties from Erasmus in Vienna. They made my life easier and brought laughter to every silly thing.
I also keep Alessia, Ilknur, and Erwan - my lovely friends from Erasmus in Mondragón - in my heart.

Ringraziamenti

Questa tesi di laurea magistrale è stata svolta presso il Center for Medical Physics and Biomedical Engineering della Medical University of Vienna / Vienna General Hospital (AKH) sotto la supervisione del professore Francesco Moscato e di Max Haberbusch. La tesi è stata parzialmente finanziata dal progetto europeo H2020-EU.1.2.2, "A Neuroprosthesis to Restore the Vagal-Cardiac Closed-Loop Connection after Heart Transplantation, NeuHeart" (Grant Agreement ID: 824071) e dal Ludwig Boltzmann Institute for Cardiovascular Research. Parte dei risultati di calcolo presentati in questa tesi sono stati elaborati con il supercomputer Vienna Scientific Cluster.

Innanzitutto, vorrei esprimere la mia gratitudine al professor Moscato per avermi dato l'opportunità di intraprendere questo progetto nel suo dipartimento. È sempre stato disponibile a sciogliere qualsiasi dubbio avessi e mi ha insegnato tanto, dalla teoria a come comportarmi in un contesto di ricerca, al concetto di godersi la vita.
In secondo luogo, Max, che sedeva a soli due posti da me, è stato molto coinvolto nel mio progetto. Mi ha sostenuto fin dall'inizio e mi ha dato consigli ogni giorno. Gli sono veramente grata per il suo sostegno.
Inoltre, desidero esprimere un sincero ringraziamento al Professore Molinari per aver accettato di guidarmi nella mia tesi e per avermi fornito il suo supporto anche a distanza.
I miei genitori e le mie sorelle che hanno da sempre fatto il tifo per me. Sembra banale, ma mi ritengo tanto fortunata ad avere loro come famiglia. Quando mi sentivo giù di morale, bastava fare una videochiamata con loro per tornare ad avere un mood positivo.
E poi c'era Milan, che mi ha accompagnato per tutto il viaggio e ha assistito ai miei momenti peggiori. Tuttavia, ogni volta che avevo voglia di arrendermi, lui era sempre lì a incoraggiarmi e a spronarmi a finire il lavoro. Gli sono immensamente grata.
Ringrazio anche i miei nonni (anche quelli distanti da questo mondo), zii, cugini e cuginetti che, pur da lontano, mi hanno sempre fatto sentire il loro calore.
Un ringraziamento speciale va a Giusy e Jessica, le mie meravigliose cugine, che mi hanno sostenuto in tutto e per tutto.
Ringrazio Silvia per avermi introdotto in questo meraviglioso mondo e per aver condiviso con me la prima parte di questa esperienza. È stato entusiasmante fare i tragitti insieme, incontrarsi in bagno per le pause e sostenersi a vicenda. Nella seconda parte, mi ha fatto sentire la sua vicinanza grazie

alla sua voglia di far sempre parte della mia vita.
Le mie amiche di sempre Giusy, Marta, Cristiana, Margaret e Martina, sono state un pilastro per me e le ho sentite tanto vicine in questo periodo, più che mai.
Ringrazio i miei amici del Poli Ginevra, Claudia, Laura, Gianluca, Catello e Matteo, che mi hanno sempre trasmesso il loro affetto da ogni parte del mondo, e le mie coinquiline di Torino, Chiara e Manuela, che sono sempre state lì per me, anche solo per un consiglio o un confronto.
Vorrei anche esprimere il mio apprezzamento ai ragazzi dell'ufficio 18 - Lorenzo, Marta, Laurenz, Tina, Daniel, Nando e Maria - così come a quelli della stanza dei baby shark - Bruna, Lorenzo e Ádam - e non solo a quelli dell'ufficio 18 - Daria, Massi e Giulia. Senza il loro senso dell'umorismo, le pause caffè con le partite a Scopa e la Mensa alle 11:59, l'intero percorso non sarebbe stato così piacevole.
Inoltre, vorrei ringraziare Ana e Cristina, le mie amiche dell'Erasmus a Vienna. Mi hanno reso la vita più facile e mi hanno fatto ridere per ogni sciocchezza.
Porto nel cuore anche Alessia, İlknur ed Erwan, i miei adorabili amici dell'Erasmus a Mondragón.

Table of Contents

List of Figures

List of Tables

Listings

Abbreviations

CO_2 Carbon dioxide.

O_2 Oxygen.

2D Two dimensional.

3D Three dimensional.

ACh Acetylcholin.

ANNs Artificial neural networks.

ANS Autonomic nervous system.

API Application programming interface.

AV Atrioventricular.

bpm Beats per minute.

C Current amplitude.

CNS Central nervous system.

CO Cardiac output.

Cvs Venous systemic compliance.

DBP Diastolic blood pressure.

DMD Dynamic mode decomposition.

DR Dimensionality reduction.

F Frequency of stimulation.

FHN Fitz-Hugh-Nagumo.

GANs Generative adversarial networks.

GPU Graphics processing unit.

GSA Global sensitivity analysis.

HF Heart failure.

HH Hodgkin-Huxley.

HR Heart rate.

LSTM Long short-term memory.

LVAD Left ventricular assist device.

LVP Left ventricular pressure.

MAP Mean arterial pressure.

MBD Model-based design.

ML Machine learning.

MPC Model predictive control.

NP Number of pulses per burst).

ODE Ordinary differential equation.

OVAT One-variable-at-time.

PI Proportional-integral.

PNS Peripheral nervous system.

PW Pulse width.

Q Charge.

Ras Arterial systemic resistance.

RMSE Root mean squared error.

RNNs Recurrent neural networks.

RRMSE Relative root mean squared.

SA Sensitivity analysis.

SBP Systolic blood pressure.

SLE Systemic lupus erythematosus.

SNS Somatic nervous system.

SV Stroke volume.

SVD Singular value decomposition.

TA-VNS Transcutaneous auricular.

TC-VNS Transcutaneous cervical.

VAD Ventricular assist device.

VN Vagus nerve.

VNS Vagus nerve stimulation.

1 Introduction

1.1 Model reduction techniques

Dynamical systems are used for describing the rich and complex evolution of real processes, and modelling these systems is of great importance for their analysis, design and control [4]. Some dynamical systems are described in [5], in particular the one and two-dimensional flows and the chaos systems.

However, the discretised dynamical systems for complex problems often have a large number of degrees of freedom, thus the computational cost can be very high. Model reduction methods are used to obtain simplified models (in terms of number of equations and number of inputs) of dynamical systems in order to reduce the computational complexity and cost while maintaining the main characteristics of the original model [4].

In recent years, the rise of machine learning and big data strategies has resulted in a change in the way complex space-time systems are modeled [6]. Some examples of implemented techniques to reduce the order of complex models are artificial neural networks, with the emerging generative adversarial networks, Koopman analysis, dynamic mode decomposition, and sensitivity analysis. They are analysed in more detail in the next sections.

1.1.1 Artificial neural networks

Artificial neural networks (ANNs) are a type of machine learning (ML) model that have become popular and useful in many disciplines in recent years due to their high processing speed, excellent self-learning properties, adaptability, fault tolerance, non-linearity and advancement in mapping inputs to outputs [7]. ANNs are inspired by biological neural networks of the human brain and are composed of a set of artificial neurons. Artificial neurons, like human neurons, receive input from other neurons, and depending on the total weighted input, are either activated or inactivated. A single-neuron model, often also called single-layer perceptron (Figure 1.1), may be considered a threshold unit. It receives inputs from several other units for which it calculates a weighted sum, to which a bias can also be added. If a certain threshold value is exceeded, the output is one, otherwise, it remains zero [8]. This is only a simple form of such a neuron model, it can have more complex activation functions to determine the output.

A multi-layer perceptron (Figure 1.2) consists of three consecutive layers: an input, a hidden, and an output layer. The input layer has input neurons that transmit data to the hidden layer via synapses, and likewise, the hidden layer transmits this data to the output layer via further synapses. The synapses store values, so-called weights, with the help of which they manipulate the input and output to the different layers [9].
Long short-term memory (LSTM), represented in Figure 1.3, is a recurrent neural network architecture developed by S. Hochreiter to overcome the backflow problem, in which error signals that flow backwards in time tend to inflate or disappear. Recurrent neural networks (RNNs) are networks with loops that allow information to persist. After the hidden state at time step t is calculated, it is returned to the recurrent unit and combined with the input at time step t+1 to calculate the new hidden state at time step t+1. This process is repeated for t+n until the predefined number (n) of time steps is reached. The LSTM instead uses different gates to decide which information to keep or discard. It also adds a cell state, which is like a long-term memory of the LSTM. The principal blocks of a LSTM network are described following [10].

- **Hidden state and new inputs:** The hidden state from a previous timestep (h_{t-1}) and the input in a current timestep (x_t) are combined before copies of them are passed through different gates.
- **Forget gate:** controls which information is to be forgotten, remembered or partially remembered.
- **Input gate:** helps identify important items to add to the cell state.
- **Update cell state:** first, the previous state of the cell (c_{t-1}) is multiplied by the results of the forget gate. Then, new information is added from [input gate $\times$ cell state candidate] to obtain the latest cell state (c_t).
- **Update hidden state:** to update the hidden state: the last cell state (c_{t-1}) is passed through the tanh activation function and multiplied by the results of the output gate.

LSTM can learn to bridge time intervals of more than 1000 steps without losing the ability to bridge short time intervals. This is achieved by a gradient-based algorithm that enforces a constant error flow through the internal states of the special units [11].

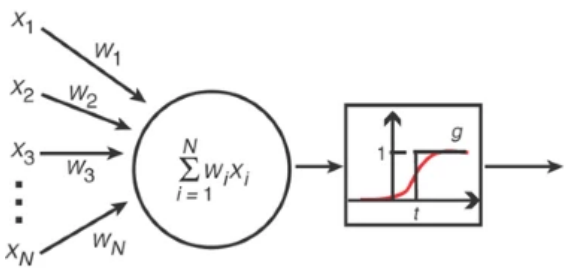

Figure 1.1: Graphical representation of the model neuron. Modified from [8, p. 196].

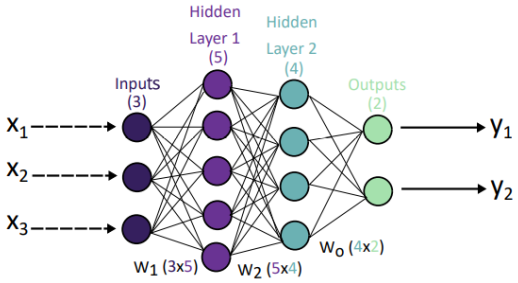

Figure 1.2: General structure of a neural network with two hidden layers. Reproduced from [14, p. 12].

A convolutional autoencoder, on the other hand, is a kind of ANN, which learns how to encode data efficiently. It then learns how to reconstruct the data from the reduced encoded representation into a representation that is as close as possible to the original input. It consists of an encoder, in which the model learns how to reduce and compress the input dimensions; a bottleneck, the compressed representation of the input data; and a decoder, in which the model learns how to reconstruct the data from the encoded representation [12].

F. J. Gonzalez et al. [13] developed a model reduction technique based on a deep convolutional encoder and an LSTM network. In this technique a deep convolutional autoencoder is used to learn a low-dimensional representation of the system, a LSTM to acquire the dynamics on the manifold and to model the possibly non-linear evolution of the model. The advantages of this approach are several: rather of applying an autoencoder to the high dimensional input data, they apply it to a reduced dimensional model obtained with a convolutional encoder; they uses LSTM network to avoid costly state reconstructions at every step; it can be applied to arbitrary high-dimensional spatio-temporal data. Nevertheless, despite that, there are also some disadvantages: the full capabilities of the method are yet unknown; filtering strategies are not that efficient, therefore there could be errors of high-frequency nature during the decoding phase; moreover, they are very data-hungry (they only work for large quantities of data), they tend to over-fit, no generalise, and they are often not explainable [13].

1.1.1.1 Generative adversarial networks

In generative adversarial networks (GANs), two models are trained (see Figure 1.4): a generative model G that captures the distribution of data and a discriminative model D that estimates the probability that a sample comes from the training data and not from G. The training procedure for

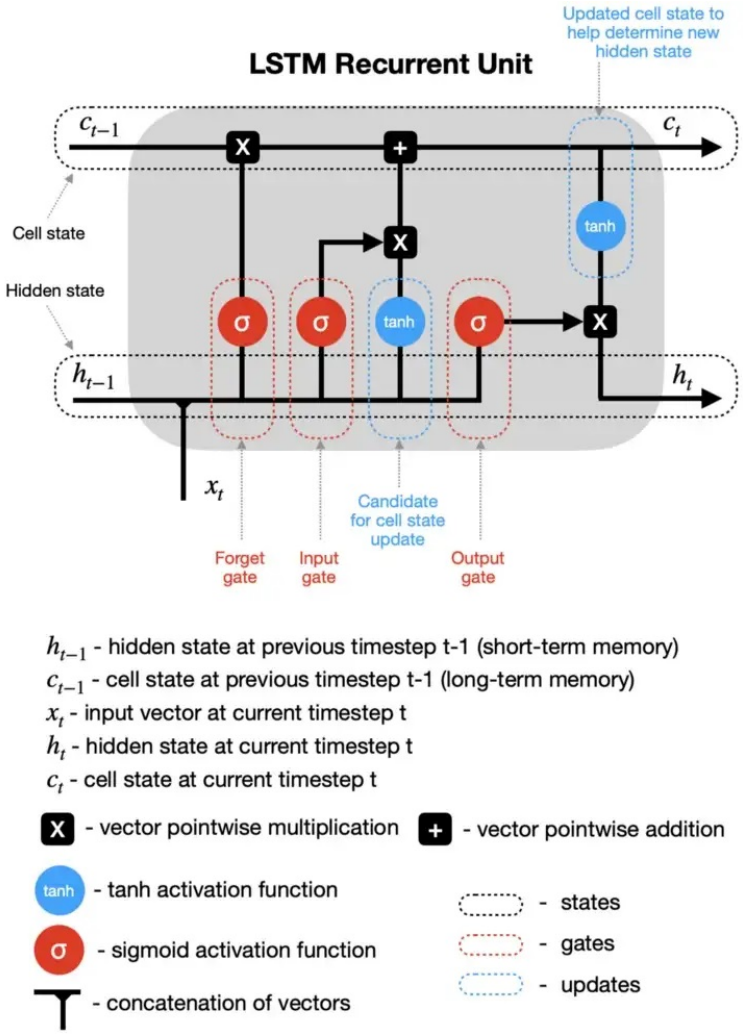

Figure 1.3: Long short-term memory (LSTM) recurrent unit. Reproduced from [10].

G is to maximise the probability that D makes an error. In the space of arbitrary functions G and D, there is a unique solution where G recovers the distribution of the training data and D is equal to 0.5 everywhere [15].

M. Farajzadeh-Zanjani et al. [16] proposed two novel GAN-based dimensionality reduction (DR) techniques that could enhance the accuracy and stability of the diagnostic system, including a supervised DR method, and an unsupervised DR method. The innovation of the proposed DR techniques is in the use of two feed-forward neural networks (the information moves in only one direction, forward from the input nodes, through the hidden and to the output nodes) in an adversarial network context.

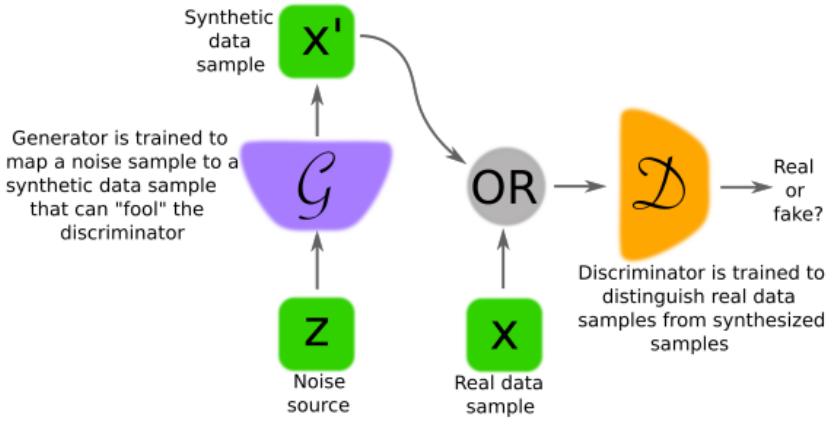

Figure 1.4: Schematic representation of the GAN. The two models that are learned during the training process for a GAN, discriminator (D) and generator (G), can be seen. Reproduced from [17, p. 2].

1.1.2 Koopman analysis

Non-linear dynamical systems are still poorly understood, and an emerging technique for dealing with them is the Koopman theory. The Koopman analysis has arisen as an important data-driven approach, as the eigenfunctions of this operator generate intrinsic coordinates that globally linearise the dynamics of any non-linear system [3].

As described in Figure 1.5, the scientist postulates a variety of reasonable dictionary functions to be included in the base set. At this point, the observables $\Psi(x)$ (measurements of the state of the system) are mapped at each instant by the dictionary into a higher-dimensional space where a finite approximation of the Koopman operator should be found [18].

The Koopman operator, $\mathbf{K}$, is an infinite-dimensional linear matrix that moves forward in time the

measurement functions, even if the underlying dynamical system is non-linear:

$$Kg(x_k) = g(x_{k+1}). \tag{1}$$

Expressing non-linear dynamics in a linear framework is useful because of the great amount of optimal control techniques available for linear systems [3].
We may define the eigenvalues λ_i and eigenfunctions φ_i of the Koopman operator as follows:

$$K\varphi_i(x) = \lambda_i\varphi_i(x). \tag{2}$$

For any observable g_i, that belongs to the span of the eigenfunctions φ_i, we can reconstruct the observable using Koopman eigenvalues (λ_i), eigenfunctions (φ_i), and modes (v_i):

$$g(x) = \sum_{j=1}^{\infty} v_j\varphi_j(x). \tag{3}$$

The application of the Koopman operator K to the observable g leads to the result:

$$(Kg)(x) = \sum_{j=1}^{N_K} v_j(K\varphi_j)(x) = \sum_{j=1}^{N_K} \lambda_i v_j\varphi_j(x). \tag{4}$$

Koopman modes represent the projection of the observables of the system onto the eigenfunctions of the Koopman operator and can be used to describe and decompose the dynamics of the time evolution of the observables of the original system [19].
In a few words: if one thinks of dynamical systems in terms of the Hilbert space of all possible measurements that could be made on that system, the non-linear dynamics in that measurement space look like a big infinite-dimensional linear operator that evolves those measurements in time. It is extremely difficult to work with this infinite-dimensional linear operator. Therefore, finite-dimensional approximations of the Koopman operator must be found, an example of which could be the dynamic mode decomposition (DMD), described more in detail in the next paragraph.
An application of the Koopman analysis on a two-dimensional (2D) non-linear dynamical system is

described below. A simple non-linear system of two first-order differential equations is considered:

$$\begin{cases} \dot{x_1} = \mu x_1 \\ \dot{x_2} = \lambda(x_2 - x_1^2) \end{cases} \tag{5}$$

Koopman theory states that if we measure x_1, x_2 and x_1^2 (a non-linear measurement), it is possible to rewrite the non-linear system in (5) into a linear dynamic 3 by 3 system using the eigenvalues. The Koopman linear system can be created as follows:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \end{bmatrix} \Rightarrow \frac{d}{dt} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \underbrace{\begin{bmatrix} \mu & 0 & 0 \\ 0 & \lambda & -\lambda \\ 0 & 0 & 2\mu \end{bmatrix}}_{\mathbf{K}} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \tag{6}$$

This non-linear system is analysed more in detail in the methods.

A biomedical application of the DMD and Koopman operator proposed by Parmar et al. [19] involves the study of a biological regulatory mechanism known as bistable system switching. In a bistable genetic system, two mutually repressive genes coexist, with only one gene being actively expressed at any given time. Koopman eigenfunctions can be used to study the temporal pulse mechanisms by which gene switching between the two stable states is achieved, both in determining the time required for such a system to reach a stable state and in calculating the set of drive impulses that force the system to reach such a state.

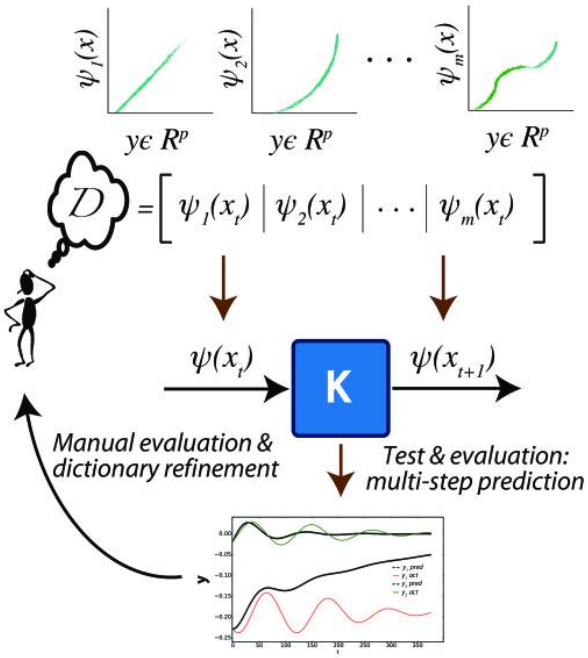

Figure 1.5: Schematic of the Koopman analysis technique. Reproduced from [18, p. 4].

1.1.3 Dynamic mode decomposition

Dynamic mode decomposition (DMD) is a robust data-driven method for analyzing complex systems, it is an equation-free architecture that rebuilds the dynamics of the system, through measurement data from numerical simulations or laboratory experiments. It is also used as a method for systems with non-linear dynamics, due to the strong connection with the Koopman operator [20]. The measurements, x_k and x_{k+1}, where k indicates the temporal iteration from a discrete dynamical system, are assumed to be approximately related by a linear operator $\mathbf{A}$:

$$x_{k+1} \approx Ax_k. \tag{7}$$

The goal is to find a best-fit solution of the operator $\mathbf{A}$ for all pairs of measurements $x(t)$, that can be collected at regular time intervals Δt, denoted by $x_k = x(k\Delta t)$. The relationship between pairs of measurement and the combined data snapshots can be described in the following matrix form:

$$X' \approx AX, \tag{8}$$

where **X** is the sequence of snapshots matrix and **X'** is the time-shifted snapshot matrix of **X**. The primary objective of DMD is finding an approximation of the matrix **A** for the measurement matrix **X** and **X'** [20].
In Figure 1.6 is shown a schematic overview of the DMD technique. The DMD algorithm consists of several steps, explained below [21].

1. First, the SVD (singular value decomposition) of **X** should be estimated to find the pseudoinverse matrix:

$$X \approx U\Sigma^{-1}V^{*}. \tag{9}$$

   where * denotes the conjugate transpose, $U \in \mathbb{C}^{n\times r}$, $\Sigma \in \mathbb{C}^{r\times r}$, and $V \in \mathbb{C}^{m\times r}$.

2. Therefore, the following approximation of the matrix **A** can be computed:

$$A \approx \bar{A} = X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}^{*}. \tag{10}$$

3. The reduced-order approximation $\tilde{A}$ is:

$$\tilde{A} = \tilde{U}X'\tilde{V}\tilde{\Sigma}^{-1}. \tag{11}$$

4. Then,the eigendecomposition of $\tilde{A}$ is computed:

$$\tilde{A}W = W\Lambda \tag{12}$$

   where columns of W are eigenvectors and Λ is a diagonal matrix containing the corresponding eigenvalues λ_k.

5. Finally, we may reconstruct the eigendecomposition of A from W and Λ. In particular, the eigenvalues of A are given by Λ and the eigenvectors of A (DMD modes) are given by columns of Φ:

$$\Phi = X'V\Sigma^{-1}W \tag{13}$$

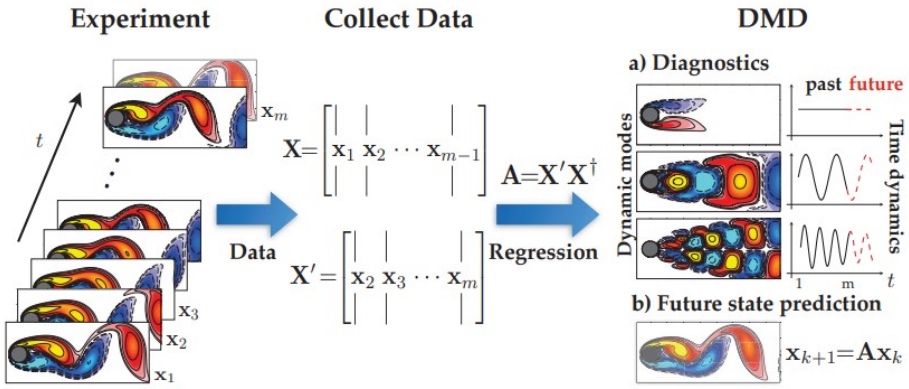

Figure 1.6: Schematic overview of DMD technique. Note that the regression step does not typically construct A but instead constructs $\tilde{A}$. The eigendecomposition of $\tilde{A}$ is then used to approximate the eigendecomposition of the high-dimensional matrix A. Reproduced from reference [21, p. 6].

1.1.4 Sensitivity analysis

Sensitivity analysis (SA) is often used as a technique to reduce the order of complex models. In particular, it is employed to identify the most influential parameters of a system, determine the interaction between these parameters and remove the redundant ones in order to reduce computation time and errors [22]. Through SA essential information about the behavior of the model, its structure, and its response to changes in model inputs are obtained [23]. Several SA methods have been developed, which may be divided into local and global sensitivity analyses.

The local sensitivity analysis is performed if one is interested in the analysis around a point of interest in the model input space. One approach is the one-variable-at-time (OVAT) method, which consists of studying the model output when one input at a time is varied. It is suggested in studies where spatial variations are more important than temporal variations [22]. Tornado charts are used to show the results of OVAT; for each input parameter, the difference in the model output for the minimum, base and maximum values of the input parameters is plotted as bars. The width of the bars is an indicator of the importance of a certain input parameter [24].

Global sensitivity analysis (GSA) intends to calculate sensitivity measures that represent the system's performance over the entire parameter space [24], to quantify the connection between the variance of the model output, knowing the variability of its inputs [25]. One of the most commonly used techniques in GSA is the variance-based method, in particular, Sobol's sensitivity analysis [25], described more in detail in the methods.

In this approach, a quantity of interest $\mathbf{Y}$ of a model is considered as a function of an input random vector $\mathbf{x}$ of dimension n: $Y = f(x) = f(x_1, x_2, ..., x_n)$.
A random variable x_i is considered to be influential to the model f, with output $\mathbf{Y}$, if the conditional variance $\mathbb{V}[\mathbb{E}[Y|x_i]]$ is larger than the variance of the quantity of interest $\mathbb{V}[Y]$, where $\mathbb{V}[\cdot]$ and $\mathbb{E}[\cdot]$ stand for the variance and the mean operators.
Different strategies are used to generate the input samples, utilized for the sensitivity analysis, with which the model is iteratively evaluated, some of them are described below.

- **Random sampling**: random values in the interval [0,1] are generated and associated with the input value. The previously assigned values do not influence the newly generated value, with the risk that the parameter space is not uniformly covered [26].
- **Sobol sampling**: quasi-random values are generated; with this strategy, the sample values are chosen considering the previously sampled points and thus avoiding the occurrence of clusters and gaps [26].
- **Latin hypercube sampling**: a sample of size nS is generated from the distributions. The range of each x_j is exhaustively divided into nS disjoint intervals of equal probability and one value x_{ij} is randomly selected from each interval. The nS values for x_1 are randomly paired without replacement with the nS value for x_2 to produce nS pairs. This process is continued until a set of nS nX-tuples $x_i = [x_{i1}, x_{i2}, ..., x_{i,nX}]$, i = 1, 2, ..., nS, is obtained [27].

For low-dimensional systems, local sensitivity analysis approaches, like OVAT, are preferred. However, as the number of inputs increases, the SA suffers from the curse of dimensionality [22].
For complex systems, where the interaction between parameters plays an important role, variance-based methods are adopted because they facilitate spatiotemporal dependence of inputs. The only limitation of the variance-based methods is the high cost of analysis; however, this can be reduced by a proper selection of the samples [22].

1.2 Fitz-Hugh-Nagumo model

In this section, first an overview of the anatomy of the nervous system is given, then stimulation of the vagus nerve is described and some models of vagus nerve stimulation for cardiovascular

applications are also listed and described. Finally, numerical models of the nerve cell are described, such as the Fitz-Hugh-Nagumo model.

1.2.1 Anatomy and physiology of the nervous system

The nervous system consists of two main parts: the *central nervous system* (CNS) and the *peripheral nervous system* (PNS).

The CNS is the site of learning, memory, emotions, thoughts, and language; it receives and processes information from sensory organs and viscera to determine the state of the external and internal environment, then it integrates this information and sends instructions to organs regarding tasks they should perform. The anatomy of the CNS includes the *brain* and the *spinal cord* [28].

The PNS establishes a communication between the CNS and the organs of the body using its functional units, the *neurons*, which are excitable cells that communicate by transmitting action potentials. The PNS includes two sections: *afferent*, which transmits information from the organs to the CNS, and *efferent*, which transmits information from the CNS to the organs. The latter section can be divided into two more branches: the *somatic nervous system* (SNS) and the *autonomic nervous system* (ANS) [28].

The SNS regulates skeletal muscle contractions by means of *motor neurons*. On the other hand, the ANS manages the function of internal organs and structures that are not under voluntary control. The ANS includes the *parasympathetic nervous system* and *sympathetic nervous system*, they are typically active under different conditions and they affect the organs in an almost antagonist way [28].

1.2.1.1 Autonomic nervous system

The autonomic nervous system is the portion of the nervous system that controls the visceral functions of the body and maintains homeostasis: it regulates the blood pressure and the contraction of the urinary bladder, and it controls thermoregulation. It is also called *involuntary nervous system* because it innervates organs whose functions are not under voluntary control, such as the heart, smooth muscles, and glands [29].

Both branches of the ANS generally innervate the same organs, how it is shown in Figure 1.7, but their effects are opposed to each other. Sympathetic and parasympathetic stimulation can cause excitatory effects in some organs and inhibitory effects in others; they can also act reciprocally to the

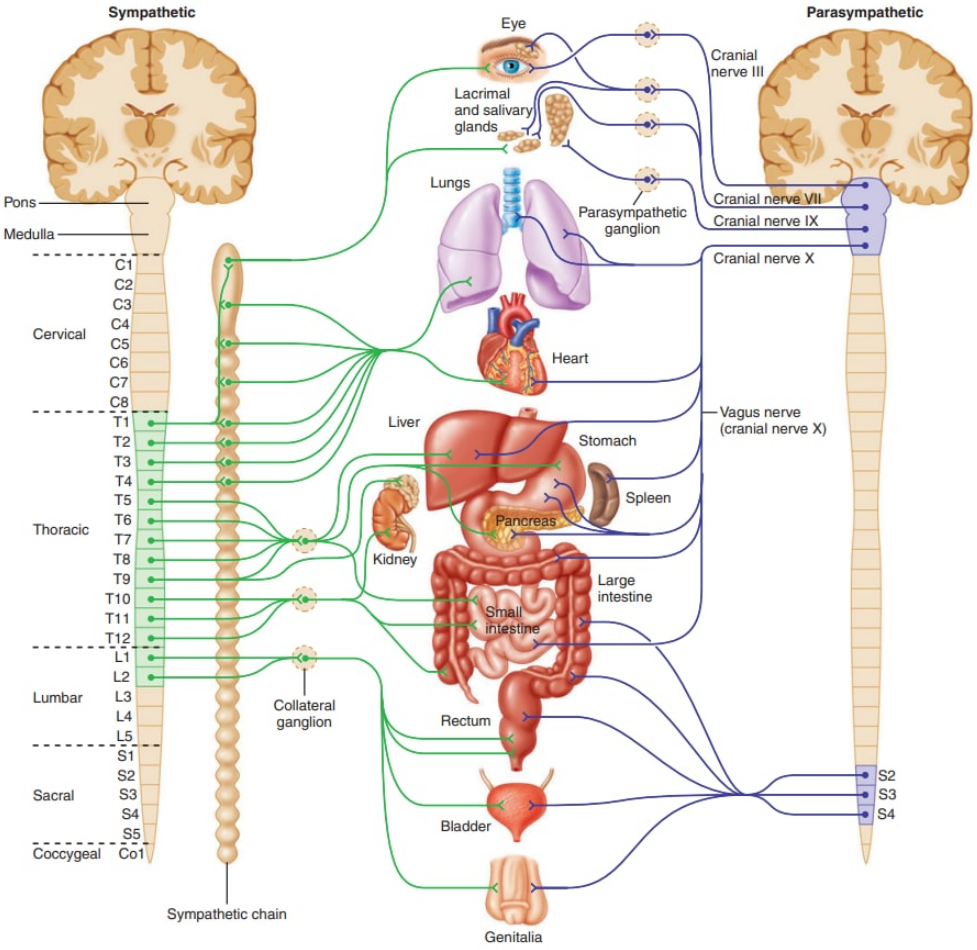

Figure 1.7: Dual innervation in the autonomic nervous system. Parasympathetic pathways are shown in purple and sympathetic pathways in green. Reproduced from [28, p. 305].

same organ if one branch causes excitation and the other one causes inhibition. The parasympathetic nervous system is most active during resting conditions, for example, it stimulates digestive organs or inhibits the cardiovascular system. The sympathetic nervous system is active during periods of excitation or physical activities, it works on the *fight-or-flight* response preparing the body to accomplish intense physical efforts [30].

1.2.1.1.1 Anatomy of the autonomic nervous system

The efferent pathway of the ANS (Figure 1.8) is made up of a series of two types of neurons that communicate with each other in the *autonomic ganglia*, which are peripheral structures: the *preganglionic neurons* travel from the CNS to the autonomic ganglia and the *postganglionic neurons* from the ganglia to the effector organs [28]. The sympathetic nervous system and parasympathetic nervous system are described in more detail below.

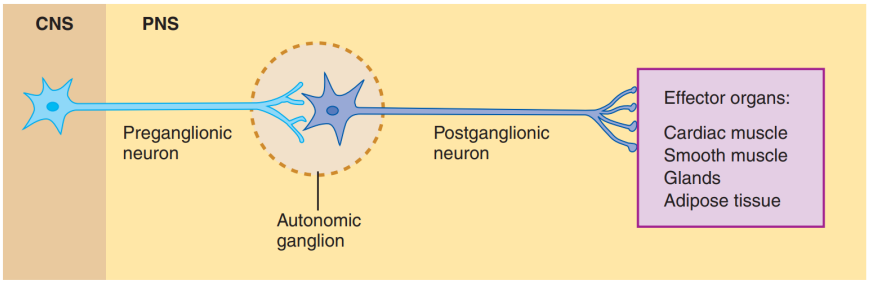

Figure 1.8: Anatomy of autonomic nervous system (ANS) pathways. CNS, central nervous system. PNS, peripheral nervous system. Reproduced from [28, p. 306].

- **Sympathetic nervous system:** the preganglionic neurons of the sympathetic nervous system originate in the thoracic and lumbar regions of the spinal cord (segments T_1-L_2, as it is shown in Figure 1.7). The ganglia where preganglionic and postganglionic neurons meet are located in the paravertebral sympathetic ganglion or in a different peripheral location (collateral ganglia) that is about halfway between the CNS and the effector tissue. Another pathway for the preganglionic neuron may be travelling from CNS to the *adrenal medullae*, the inner part of the adrenal endocrine gland which secretes *norepinephrine* and *epinephrine* into the bloodstream. A single preganglionic neuron can synapse with several postganglionic neurons in different ganglia, in fact, the ratio of preganglionic-postganglionic fibers is about 1:20. The postganglionic neurons travel within each one of the spinal nerves from the ganglia to various organs [31]. The distribution of sympathetic nerves to each organ is determined partly by the

locus in the embryo from which the organ originated [30].

- **Parasympathetic nervous system:** the preganglionic neurons of the parasympathetic nervous system originate in several nuclei of the brainstem (cranial nerves III, VII, IX, X) and from segments S_2-S_4 of the sacral region of the spinal cord (Figure 1.7). These neurons synapse with the postganglionic neurons in terminal ganglia close to or embedded in the target tissues. In many organs the ratio of preganglionic-postganglionic nerves is 1:1, for this reason, the effects of the parasympathetic system are more localized to specific tissues, in comparison with the sympathetic system [31]. About seventy-five percent of parasympathetic fibers are in the *vagus nerve* (VN), the cranial nerve X. The vagus nerves furnish parasympathetic innervation to the heart, lungs, esophagus, stomach, entire small intestine, proximal half of the colon, liver, gallbladder, pancreas, kidneys, and upper portions of the ureters [30].

1.2.1.1.2 Autonomic neurotransmitters and receptors

Neurotransmitters are chemical messengers. Communication via neurotransmitters occurs when a cell releases a chemical substance into the interstitial fluid and the target cell reacts to this messenger with certain proteins, called receptors, which specifically recognise and bind the messenger.
The two main neurotransmitters in the peripheral nervous system are *acetylcholine* and *norepinephrine*. Acetylcholine is released by preganglionic fibers of the sympathetic and parasympathetic nervous system and by postganglionic fibers of the parasympathetic nervous system; neurons that release this neurotransmitter are called *cholinergic*. Norepinephrine is released by the sympathetic postganglionic neurons, these neurons are referred to as *adrenergic*. The two neurotransmitters can bind to different classes of cholinergic and adrenergic receptors [28].
Nicotinic receptors and *muscarinic receptors* are the two major classes of cholinergic receptors. Nicotinic receptors are located on the cell bodies and dendrites of sympathetic and parasympathetic postganglionic fiber, on adrenal medullae and on skeletal muscle cells. We can find muscarinic receptors on target organs of the parasympathetic nervous system [29].

1.2.1.2 Electrical signaling and action potential of the neurons

A cell at rest has a potential difference across its membrane such that the inside of the cell is negatively charged compared to the outside. This potential difference is called "resting membrane potential" because the cell is not receiving or transmitting signals. It is an electrical force and for the neurons is about -70 mV [28].

The membrane of neurons is permeable to potassium and sodium ions, so both move through the membrane via the appropriate channels. The movement of each ion causes the membrane potential to approach the respective equilibrium potential. Since the membrane is much more permeable to potassium, the resting membrane potential, -70 mV, is much closer to the potassium than to the sodium equilibrium potential [28].

Neurons communicate by producing electrical signals in the form of changes in membrane potential. These occur when some ion channels open or close in response to certain stimuli, altering the permeability of the membrane to the ion in question. Small changes in membrane potential are called graded potentials. Generally, a single graded potential is not strong enough to trigger an action potential. However, when graded potentials overlap in time, they can add up in both time and space. Graded potentials generate action potentials if they depolarise a neuron to a certain threshold, which is a critical value of the membrane potential that must be reached or exceeded [28].

An action potential follows the all-or-none principle: "whether a membrane is depolarised to threshold or greater, the amplitude of the resulting action potential is the same; if the membrane is not depolarised to threshold, no action potential occurs" [28]. It consists of three different phases, described below and shown in Figure 1.9.

1. **Rapid depolarisation:** the membrane potential changes from -70 mV (rest) to +30 mV. This depolarisation is caused by a dramatic increase in permeability to sodium, followed by an increment in the movement of sodium ions into the cell. Since the permeability to sodium is now greater than the permeability to potassium, the membrane potential approaches the sodium equilibrium potential of +60 mV.

2. **Repolarisation:** within 1 ms after the sodium permeability rises, it falls rapidly again, reducing the sodium influx. At about the same time, the permeability of potassium increases. Potassium then migrates out of the cell across the electrochemical gradient, repolarising the membrane potential and bringing it back to the resting value (-70 mV).

3. **After-hyperpolarisation:** potassium permeability remains elevated for a short time (5-15 ms) after the membrane potential reaches the resting membrane potential, resulting in post-hyperpolarisation. During this time, the membrane potential is even more negative than at rest as it gets closer to the potassium equilibrium potential (-94 mV) [28].

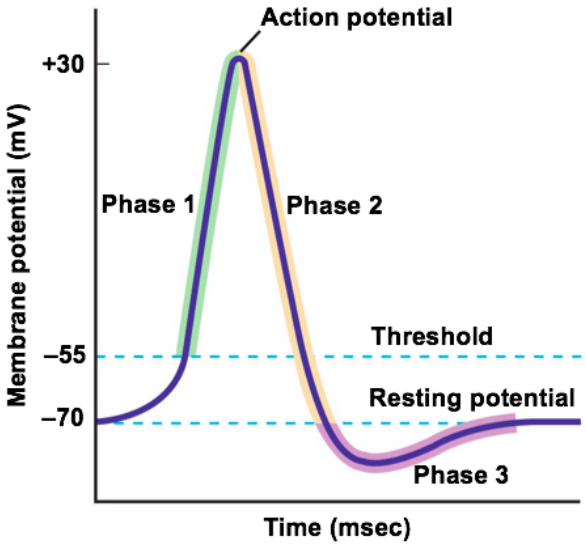

Figure 1.9: Three phases of an action potential. Modified from [28, p. 183].

An action potential triggered at any point, on an excitable membrane, usually excites neighbouring parts of the membrane, causing the action potential to propagate along the membrane. The average nerve trunk contains about twice as many unmyelinated fibers as myelinated fibers (large fibers over 1-2 μm in diameter) [30].

The action potential propagates differently in unmyelinated and myelinated fibers.

In unmyelinated fibers, the peak value of the action potential inside is about +40 mV compared to the outside, so the positive charge moves to the adjacent area of the membrane and depolarises it. The voltage-gated sodium channels in that region of the membrane are opened, therefore a "new" action potential is triggered and so on in the adjacent regions [28].

In myelinated fibers, the central core is the axon (which is filled at its centre with axoplasm, an intracellular fluid) surrounded by a myelin sheath, often much thicker than the axon itself, deposited by Schwann cells around the axon. About every 1 to 3 mm along the myelin sheath is a node of Ranvier, a small non-isolated area (2-3 μm) where ions can still flow easily through the axon membrane between the extracellular and intracellular fluids inside the axon. In myelinated fibers, action potentials are conducted from node to node according to the principle of saltatory conduction (see Figure 1.10), where ions can only flow through the nodes of Ranvier [30].

During the absolute refractory period, no new action potential can arise in an excitable fiber as long as the membrane is still depolarised by the previous action potential. This period is about 0.4 ms in large myelinated nerve fibres [30].

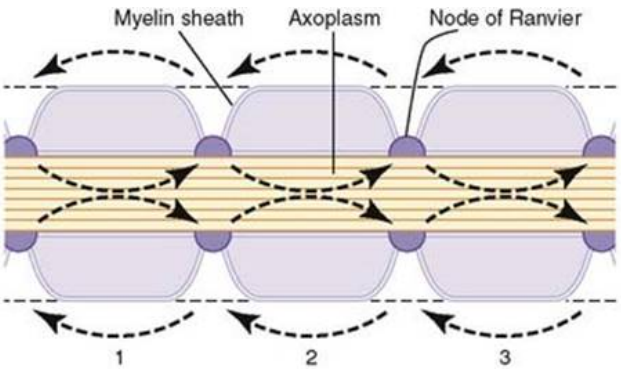

Figure 1.10: Saltatory conduction of action potential along a myelinated axon. Reproduced from [30, p. 72].

1.2.1.3 Nervous control of the heart

A dromotropic agent is an agent that affects the conduction of excitation in the AV (atrioventricular) node and thus the speed of electrical impulses in the heart. A positive dromotropic agent intensifies the conduction velocity [32].

Inotropic action is about changing the force or energy of muscle contractions. Positively inotropic agents increase the strength of muscle contraction [33].

Mechanisms that affect heart rate are said to have a chronotropic effect: those that increase heart rate have a positive chronotropic effect. The central nervous system controls the heart rate by varying the impulse frequency in sympathetic and parasympathetic nerve fibers terminating in the SA (sinoatrial) node, they are continuously active and change the rate of the spontaneous depolarization of that node [34]. The heart rate may increase as a result of decreased vagus nerve inhibition of the SA node or increased sympathetic nerve stimulation. Resting bradycardia (slow heart rate) of trained athletes is due to high vagus nerve activity. In the cardiac control center, more precisely in the medulla oblongata of the brain stem, the activity of the autonomic innervation of the heart is coordinated[29].

1.2.1.3.1 Vagus nerve

The vagus nerve is the longest nerve of the body, the X cranial nerve (as shown in Figure 1.7), which originates in the medulla oblongata and innervates the lungs, heart, stomach, spleen, pancreas, intestines, and liver [28]. It comprises two nerves, right and left (Figure 1.11), and it carries both afferent (80 %) and efferent (20 %) fibers, it, therefore, enables bi-directional communication between the brain and the different organs of the body. It consists of three kinds of fibers A, B, and C, with

different conduction properties (shown in Table 1); their stimulation threshold is determined by the total charge delivered to the nerve fiber, which depends on the stimulation current, pulse duration, stimulation frequency, and waveform. A-fibers are recruited first, followed by B-fibers and C-fibers [1].

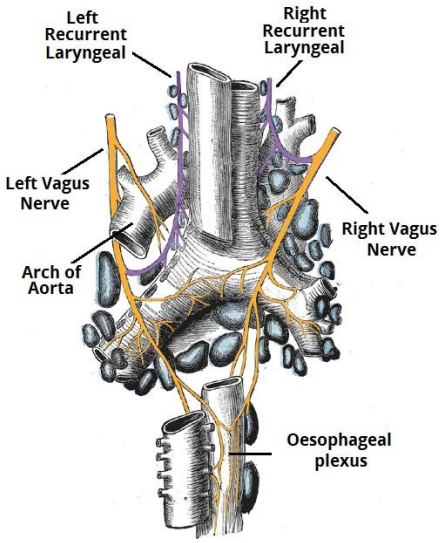

Figure 1.11: Cervical right and left vagus nerve anatomy. Reproduced from [35].

Type of fiber	Diameter (μm)	Myelination	Conduction velocity (ms)
A-fiber	5 - 20	yes	> 4.5
B-fiber	1 - 3	yes	2 - 4.5
C-fiber	0.4 - 2	yes	< 2

Table 1: Characterization of vagus nerve fibers [1].

1.2.2 Vagus nerve stimulation

Vagus nerve stimulation (VNS) is a medical treatment that delivers electrical impulses to the vagus nerve, using different combinations of stimulation parameters. It was developed and employed for animal experiments in the 19th century. A century later, in the 1990s, it was approved for human applications to treat refractory epilepsy [36],[37],[38], and then for refractory depression [39],[40]. In the literature one can find applications of VNS for the treatment of cardiovascular diseases, inflammatory disorders, metabolic disorders, and pelvic dysfunctions [1]. Abdominal VNS has been tested for the treatment of morbid obesity [41]. Cervical VNS has been used to treat inflammation, like Crohn's disease [42] and for cardiac application, such as heart failure (HF) [43],[44].

In most non-cardiac applications stimulation is delivered to the left vagus nerve, whereas for cardiac applications VNS is typically delivered to the right vagus nerve because it has a strong influence on the heart rate, attributed to its predominance in SA innervation [45].

VNS can be performed using both an invasive and a non-invasive techniques. The first consists of implanting electrodes connected to a programmable pulse generator under the skin (in the upper chest area). To avoid the complications associated with the surgical implant, researchers have developed non-invasive VNS devices such as transcutaneous cervical (TC-VNS) and transcutaneous auricular (TA-VNS) [1].

In the context of using invasive VNS for HF patients, some devices were developed: the CardioFit system and the Boston Scientific VNS device [46]. The CardioFit system is designed to measure heart rate via an intracardially implanted electrode, and deliver stimulation at preset delays relative to the electrocardiogram R-wave (representing the electrical stimulus as it passes through the main portion of the ventricular walls), as well as stop stimulation when the heart rate reaches the bradycardia limit (eg, 55/min). The implantation is performed under general anaesthesia in an outpatient setting, the electrodes (3 spiral coils: anode, cathode, and anchor tether) are attached to the left vagus nerve. The stimulation lead is an asymmetric, bipolar, multi-contact cuff electrode that allows unidirectional activation directed to the heart [47]. The Boston Scientific VNS device bi-directionally activates VN fibers without incorporating heart rate sensing for timing stimulation to the cardiac cycle [46].

One of the non-invasive devices is gammaCore, a TC-VNS device, which stimulates both the afferent and efferent cervical branch of the vagus nerve with a 1 ms pulse, repeated at 25 Hz. The stimulation intensity is self-controlled (up to 24 V and 60 mA) and lasts 2 minutes [47]. It obtained the CE marks for use in bronchoconstriction, primary headache, epilepsy, anxiety, depression, and gastric motility disorders [1]. Among the TA-VNS devices, an example is NEMOS, which uses an intra-auricular electrode to stimulate the afferent nerve fibers of the vagal auricular branch and delivers biphasic pulses (25 V, 10 Hz, 0.3 ms pulse), it is applied through a patient-controlled stimulation session that usually lasts for 1 hour, 3-4 times per day. This device received European clearance for the treatment of epilepsy, depression and pain relief. Non-invasive VNS may be a viable treatment option for systemic lupus erythematosus (SLE), which is a chronic systemic autoimmune disorder that commonly affects the skin, joints, kidneys, and central nervous system [1].

Among cardiovascular diseases, persistent sinus tachycardia (chronic elevation of the resting heart rate) has serious implications for patient health. More recently, researchers have investigated the use

of closed-loop vagus nerve stimulation for control of the heart rate [48],[49]. The VNS stimulation models are described more in detail in the next paragraph.

1.2.3 Vagus nerve stimulation models in cardiovascular applications

VNS therapy is commonly delivered using an open-loop approach, with fixed stimulation parameters (no feedback regulation) but it does not permit precise continuous control of the heart rate. Furthermore, variations in nerve fiber activation thresholds may lead to inconsistent heart rate responses for certain stimulation intensities. Therefore, a closed-loop control may be necessary to optimize and adapt the response to the therapy and minimize the side effects of neurostimulation devices. To design such closed-loop control algorithms, traditionally mathematical models of the control systems are used. That allows them to develop and optimize control strategies before their experimental evaluation [48].

In silico experiments, in which mathematical models are developed and tested on a computer, are a mixture of in vivo and in vitro techniques. Like in vivo techniques, they aim to mimic the behaviour of organisms in their entirety. However, as with in vitro experiments, no actual animal testing is required. Moreover, the conditions can be controlled down to the smallest detail, so they can easily be used for the control design of a system [50].

H. M. Romero Ugalde et al. [51] proposed a model-based design (MBD) framework, based on a closed-loop control system, to design proportional-integral (PI) controllers which work synchronously with the heart period, for regulating the heart rate by changing the stimulation current applied to the vagus nerve. A computational model was developed containing three different interconnected sub-models: 1) the cardiovascular system, 2) the baroreflex, including vagal activity, and 3) the vagus nerve stimulator. Sensitivity analyses were performed by varying parameters of the control system and the physiological model in order to find the most influential VNS parameters. A PI control system was designed, using the computational model, and subsequently optimised. The parameter ranges determined from the sensitivity analysis were experimentally validated in six sheep, with electrodes placed around the previously removed VN. These electrodes were connected to the PI control system. The in-vivo experiments performed showed promising results.

Another example is the model implemented by M. Haberbusch et al. [45] that can precisely predict the acute cardiac effects of electrical stimulation of the right cervical vagus nerve in anaesthetized animals. It incorporates the mechanisms of nerve fiber recruitment of external electrical stimulation

and also the acetylcholine (ACh) release dynamics at the cardiac vagal nerve terminals. A sensitivity analysis of the model parameters, associated with the vagal cardiac control was performed to identify the most influential parameters on the output: they are related to the description of the electrode nerve interface and the ACh release model.

Later, M. Haberbusch et al. [48], in 2022, developed two closed-loop heart rate control strategies based on that in-silico model, which were evalueted in ex-vivo Langendorf perfused rabbit hearts with intact vagal innervation, showing promising results.

Helmers S. L. et al. [52] developed a digital model of the vagus nerve to understand the role of combinations of output current and pulse width on the activation of A and B nerve fibers. Acute and chronic stimulation conditions were incorporated into the model to investigate the effects of tissue encapsulation at the site of electrode placement. To achieve optimal stimulation, the output current should be between 0.75 and 1.75 mA and the pulse width 250 or 500 μs.

1.2.4 Nerve cell numerical models

To study neuronal physiology [53] and simulate electrical nerve stimulation [54], as well as to find the right stimulation parameters, numerical models of the nerve cell membrane were developed. The Hodgkin-Huxley (HH) model is a very powerful system of four differential equations developed for the analysis of the properties of the squid membrane of neurons. The electrical circuit of the HH model is shown in Figure 1.12 (a), and the trend of the four variables over time is shown in Figure 1.12 (b). Today, it is the most accurate model for predicting membrane behaviour for arbitrary shapes of stimulation signals. The equations of the HH model are given below [2].

$$\frac{dV}{dt} = \left[i_{ist} - \bar{g}_{Na} m^3 h (V - V_{Na}) - \bar{g}_K n^4 (V - V_K) - g_L (V - V_L)\right] / C \tag{14}$$

$$\frac{dn}{dt} = [\alpha_n(V)(1 - n) - \beta_n(V) n] \cdot k \tag{15}$$

$$\frac{dm}{dt} = [\alpha_m(V)(1 - m) - \beta_m(V) m] \cdot k \tag{16}$$

$$\frac{dh}{dt} = [\alpha_h(V)(1 - h) - \beta_h(V) h] \cdot k \tag{17}$$

with the coefficient k for temperature T (in °C) [2]:

$$k = 3^{0.1T-0.63} \tag{18}$$

where [2]:

$$V = V_i - V_e - V_{rest} \tag{19}$$
$$i_{ist} = \frac{I_{inj}}{2\pi rl} \tag{20}$$
$$\alpha_n(V) = \frac{1-0.1V}{10(e^{1-0.1V}-1)} \tag{21}$$
$$\beta_n(V) = 0.125 \cdot e^{-\frac{V}{80}} \tag{22}$$
$$\alpha_m(V) = \frac{2.5-0.1V}{e^{2.5-0.1V}-1} \tag{23}$$
$$\beta_m(V) = 4 \cdot e^{-\frac{V}{18}} \tag{24}$$
$$\alpha_h(V) = 0.07 \cdot e^{-\frac{V}{20}} \tag{25}$$
$$\beta_n(V) = \frac{1}{1+e^{3-0.1V}} \tag{26}$$

The values of the constants and the parameters of the HH model are described in Table 2.

The resting state conditions are [2]:

$V(0) = 0$; $n(0) = 0.32$; $m(0) = 0.05$; $h(0) = 0.6$.

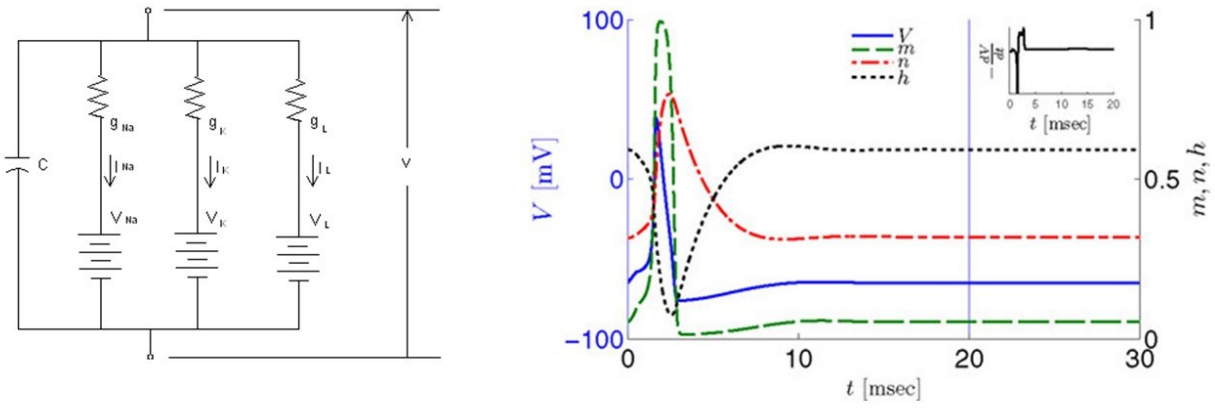

Figure 1.12: (a) Electrical circuit of the HH model. Reproduced from [55]. (b) Fitted HH model variables V, m, n, and h after an action potential (AP) relax to steady state on a shorter timescale than the minimal stimulation period used (T $\geq$ 20 ms). Inset: -dV/dt during an AP, which is proportional to extracellular V. Reproduced from [56, p. 3].

Parameter	Description	Value
V	reduced membrane voltage (Equation 19)	mV
V_{rest}	resting potential	-70 mV
V_i	intracellular potential	mV
V_e	extracellular potential	mV
I_{inj}	current of electrode (Equation 20)	μA
i_{ist}	stimulating current density	$\mu A/cm^2$
C	capacity of membrane per cm^2	1 $\mu F/cm^2$
r	radius of fiber	0.024 cm
g_{Na}	max. Na-conductance	120 $k\Omega^{-1}cm^{-2}$
g_K	max. K-conductance	36 $k\Omega^{-1}cm^{-2}$
g_L	max. leakage-conductance	0.3 $k\Omega^{-1}cm^{-2}$
V_{Na}	voltage generated by different sodium concentration on both sides of the membrane	115 mV
V_K	potassium voltage	-12 mV
V_L	leakage voltage	10.6 mV
m, n, h	probabilities for opening the ionic channels	-
T	temperature	6.3 °C

Table 2: Hodgkin-Huxley model parameters and constants [2].

The four states (V, m, n, h) of the HH model can be reduced to two variables. Figure 1.12 (b) shows that the states V and m have quite a similar shape, as do the slow variables n and h, therefore it is possible to use only equations (14) and (17) [2].

The so-called Fitz-Hugh-Nagumo (FHN) model is another mathematical model, originally used to describe the action potential activity of neurons. It utilises two variables (a scaled voltage and a recovery variable) and is simpler than the HH model. It is based on the principle that the injection of a small current through a cell membrane causes the cell to react passively with a small voltage deviation until a certain threshold is reached, after which the reaction becomes more pronounced [57]. The FHN model is described more in detail in the methods (paragraph 2.1.2).

1.3 Cardiovascular system model

This section first describes the anatomy of the cardiovascular system, then details are given about the conduction system of the heart and the regulation of the heart's pumping action. Finally, some models of the cardiovascular system are presented and described.

1.3.1 Anatomy and physiology of the cardiovascular system

The cardiovascular system includes the heart and blood vessels, its main functions are the transport of oxygen and nutrients and the washout of metabolic waste products. The cardiovascular system is also part of a control system, in which it secretes and distributes hormones and, it also plays a vital role in temperature regulation [58].

The heart, shown in Figure 1.13, is made up of two separate pumps: a right heart that pumps blood through the lungs, and the left heart that pumps blood through the systemic circulation to provide blood flow to all organs and tissues of the body. Each part is a pulsatile two-chamber pump composed of an atrium and a ventricle. The atria are designed to be weak primer pumps for the ventricles which supply the main pumping force that propels the blood either through pulmonary circulation or through systemic circulation [30].

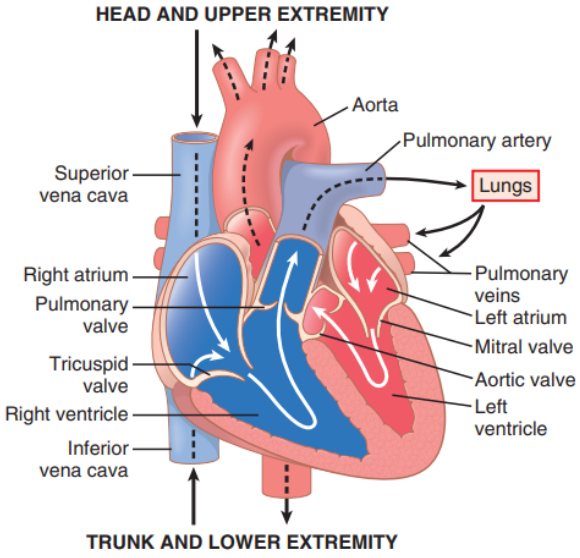

Figure 1.13: Structure of the heart and course of blood flow through the heart chambers and heart valves. Reproduced from [30, p. 109].

1.3.1.1 Blood circulation

The blood circulation, how it is shown in Figure 1.14, consists of *pulmonary circulation* and *systemic circulation*. The elements of the first system are *arteries*, *arterioles* and *capillaries*, their function is to transport oxygenated blood from the heart to the tissues. For pulmonary circulation, *capillaries*, *venules* and *veins*, collect deoxygenated blood from the tissues and conduct it back to the heart.

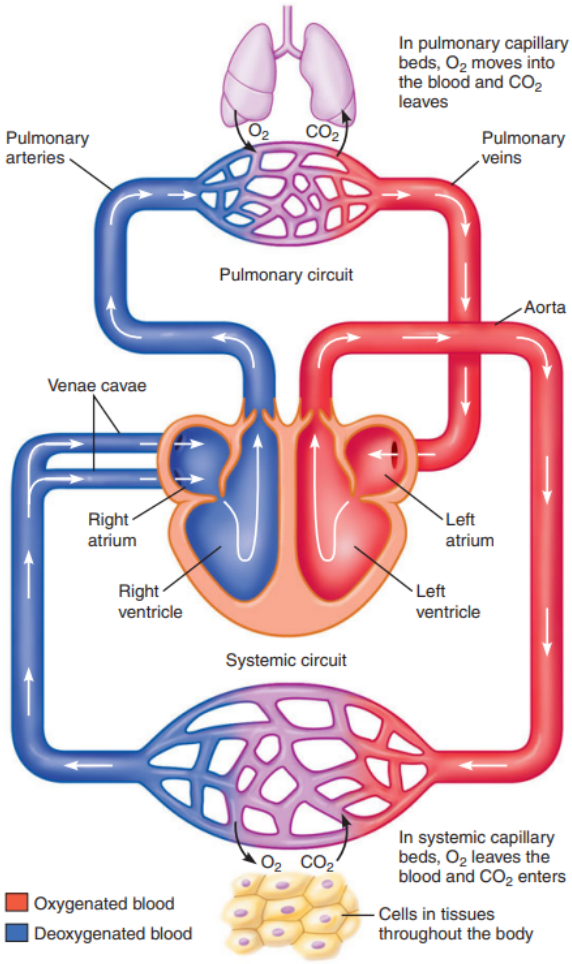

Figure 1.14: Pulmonary and systemic circulation: the path of the blood flow through the cardiovascular system. The direction of the blood flow is indicated by arrows. Reproduced from [28, p. 362].

1. **Pulmonary circulation**

   Venous blood enters the right atrium from the superior and inferior venae cavae, then flows through the tricuspid valve into the right ventricle. The ventricle receives the blood while it is in a state of relaxation (diastole). The next step is the ventricle (systole) contraction that pumps the blood out through the pulmonary artery and into the lungs at a low pressure [58]. In the alveoli (lung air sacs), gases exchange takes place: oxygen (O_2) passes from the air to the blood, and carbon dioxide (CO_2) leaves the blood [28], afterwards the oxygenated blood returns via the pulmonary veins to the left atrium.

2. **Systemic circulation**

   The blood passes through the mitral valve to reach the left ventricle, it ejects the same volume of blood as the right ventricle, but with more pressure, to the aorta and the branching arterial system. At the level of capillaries dissolved gases and nutrients diffuse to tissue cells, at that moment the deoxygenated blood is conducted through the venous system to the venae cavae [58].

1.3.1.2 Cardiac output

The cardiac output (CO) is the ejected volume over time, measured in L/min, and it depends on both the volume ejected per contraction (stroke volume, SV) and the number of contractions per minute (heart rate, HR). The proportional relationship among them is the following:
$CO = HR \times SV$.
The cardiac output varies greatly with the body's activity level. The following factors mostly affect CO: the basic level of body metabolism, whether the person is exercising, the person's age, and the size of the body.
For young, healthy men, resting CO is about 5.6 L/min; for women, this value is around 4.9 L/min; in round numbers, it is often stated to be 5 L/min [58].

1.3.1.3 Cardiac cycle

The *cardiac cycle* includes all events occurring from the beginning of one heartbeat to the beginning of the next one. The total *duration* of the cardiac cycle (heart period) is the reciprocal of the heart rate. Each cycle is initiated by the spontaneous generation of an action potential in the sinus node. The action potential travels from there rapidly through both atria and then through the atrioventricular bundle into the ventricles (the conductive system of the heart is explained in

section 1.1.4).

Figure 1.15 shows the different events that occur during the cardiac cycle for the left side of the heart. The top three curves show the changes in pressure in the aorta, left ventricle, and left atrium, respectively. The fourth curve depicts the changes in left ventricular volume, the fifth depicts the electrocardiogram, the electrophysiological activity of the heart. The last curve represents the phonocardiogram, which is a recording of the sounds produced by the heart valves during the blood pumping [30].

The cardiac cycle begins when both the atria and ventricles are in diastole and consists of four phases, described below.

1. **Ventricular filling:** during ventricular systole, large amounts of blood accumulate in the right and left atria due to the closed A-V valves. As soon as systole is over and ventricular pressures return to their low diastolic values, the moderately elevated pressures that developed in the atria during ventricular systole immediately push open the A-V valves and allow blood to flow rapidly into the ventricles, as shown by the rise in the left ventricular volume curve in Figure 1.15. This period is called the period of *rapid filling of the ventricles* and lasts for about the first third of diastole. During the middle third of diastole, there is usually only a small amount of blood flowing into the ventricles. During the last third of diastole, the atria contract and drive more blood into the ventricles [30].

2. **Isovolumetric contraction:** immediately after the onset of ventricular contraction, ventricular pressure rises abruptly, as shown in Figure 1.15, causing the A-V valves to close. Subsequently, the ventricle requires a further 20 to 30 ms to build up sufficient pressure to force the semilunar (aortic and pulmonary) valves open against the pressure in the aorta and pulmonary artery. During this period, the ventricles contract but there is no emptying because the valves are closed, so the volume of blood in the ventricles remains constant. During this period, the tension of the heart muscle increases, but there is little or no shortening of the muscle fibers [30].

3. **Ejection:** when the left ventricular pressure rises slightly above the aortic pressure, which is typically about 80 mm Hg, it pushes the semilunar valves open. Immediately, blood begins to flow out of the ventricles into the aorta and pulmonary arteries, during which, ventricular pressure reaches a peak and then declines. About 60 percent of the blood that is in the ventricle at the end of diastole is ejected during systole. About 70 percent of this proportion

ejects during the first third of the ejection period (*rapid ejection*), and the remaining 30 percent empties during the next two-thirds (*slow ejection*). When the ventricular pressure falls below the aortic pressure, the semilunar valves close [30].

4. **Isovolumetric relaxation:** at the end of systole, ventricular relaxation begins, rapidly decreasing the pressure in both the right and left ventricles. The increased pressure in the distended large arteries, which have just been filled with blood from the contracted ventricles, immediately pushes the blood back into the ventricles, closing the aortic and pulmonary valves. For another 30 to 60 ms, the ventricular muscle continues to relax, even though the ventricular volume does not change, during this period, the intraventricular pressures rapidly decrease back to their low diastolic levels. Then the A-V valves open to begin a new cycle of ventricular pumping [30].

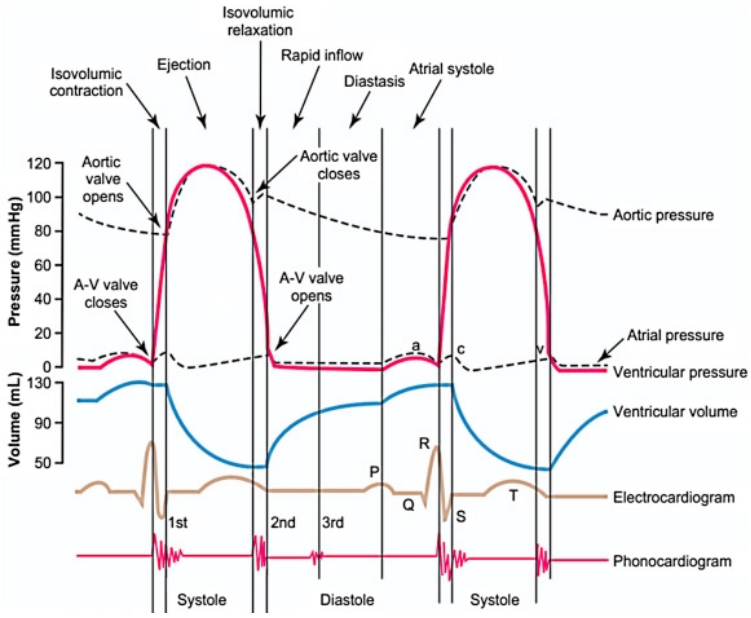

Figure 1.15: Events of the cardiac cycle for left ventricular function, showing changes in left atrial pressure, left ventricular pressure, aortic pressure, ventricular volume, electrocardiogram, and phonocardiogram. A-V, atrioventricular. Reproduced from [30, p. 114].

1.3.1.4 Conduction system of the heart

The cardiac muscle contractions are triggered on a periodic basis by signals originating in the heart itself. This ability of the heart is called *autorhythmicity* and is due to the rhythmic depolarization of cells. There are two types of autorhythmic cells: *pacemaker cells*, which generate action potentials

and determine the pace of the heartbeat, and *conduction fibers*, which quickly conduct the action potentials from place to place in the cardiac muscle. Together, these cells make up the conductive system of the heart, a sequence of electrical events that normally triggers the heartbeat [28]. The pathway of the electrical signal is shown in Figure 1.16. The heartbeat is initiated by the sino-atrial (SA) node, located on the posterior wall of the right atrium, close to the superior vena cava. The nodal cells rhythmically depolarize and generate action potentials. These electrical impulses are created by a repeating sequence of ion fluxes through specialized channels in the membrane of cardiomyocytes because of their unstable resting membrane potential [58]. In humans, in resting conditions, that leads to a cardiac contraction almost every second, which makes the heart pump at about 60 beats per minute (bpm). The action potential, passing through anterior *interatrial band*, reaches the left atrium and through the *internodal pathways* reaches the atrioventricular (AV) node. Because of this particular arrangement of the conduction system from the atria to the ventricles, there is a delay of more than 100 ms in the conduction of the cardiac impulse from the atria to the ventricles. This delay allows the atria to contract before ventricular contraction, thus pumping blood into the ventricles before the strong ventricular contraction begins. The AV node is located in the posterior wall of the right atrium, behind the tricuspid valve and it is also capable of generating spontaneous action potentials, but with a lower frequency in comparison with the SA node. From the AV node, the impulse travels through the *bundle of His* and afterwards, it splits into left and right branches located in the left and right ventricles. Finally, the impulse reaches the *Purkinje fibers*, a network of branches that drives the impulse through the rest of the cardiac cells [28].

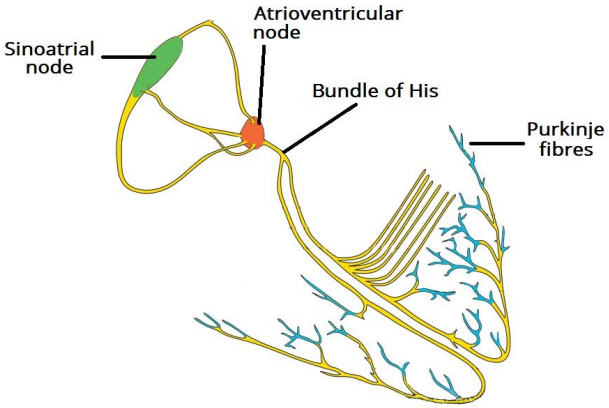

Figure 1.16: Cardiac conduction system. Reproduced from [59].

1.3.1.5 Regulation of heart pumping

Depending on the current level of activity of a person, the heart should adapt the amount of blood to pump into all the vessels of the body: the volume of the blood pumped in a minute can vary from 4-6 liters for a person who is resting, to seven times this amount for a person who is doing exercises. The volume pumped by the heart is controlled by two main processes (1) intrinsic cardiac regulation of pumping in response to changes in the volume of blood flowing into the heart and (2) control of heart rate and strength of the heart pumping by the sympathetic and parasympathetic branches of the autonomic nervous system [30], as described below in more detail.

1. **Intrinsic regulation of heart pumping:** the amount of blood the heart pumps per minute is normally determined almost exclusively by the rate of blood flow from the veins into the heart (venous return). The intrinsic ability of the heart to adapt to increasing volumes of incoming blood is called the *Frank-Starling mechanism of the heart*, in honour of Otto Frank and Ernest Starling, two great physiologists of a century ago. Basically, the Frank-Starling mechanism means that the more the heart muscle is stretched during filling, the greater the force of contraction and the greater the amount of blood pumped into the aorta [30].

2. **Control of the heart by the autonomic nervous system:** the pumping action of the heart is also controlled by the sympathetic nerves and vagus nerve (X. cranial nerve, the largest nerve of the parasympathetic nervous system), as shown in Figure 1.17. For a given atrial pressure, the amount of blood pumped per minute can often be increased by more than one hundred percent by stimulation of the sympathetic nervous system. In contrast, vagal (parasympathetic) stimulation can reduce the output to almost zero. Strong *sympathetic stimulation* can increase the heart rate in young adult humans from the normal rate of 70 beats/min to as high as 200 beats/min. In addition, sympathetic stimulation increases the force of heart contraction to twice the normal rate, increasing the volume of blood pumped and ejection pressure. Strong *stimulation of the parasympathetic nerve fibers* in the vagus nerve to the heart may stop the heartbeat for a few seconds, but then the heart normally "escapes" and beats at a rate of 20 to 40 bpm as long as the parasympathetic stimulation continues. In addition, strong vagal stimulation can reduce the strength of heart muscle contraction by 20 to 30 percent. The effect of vagal stimulation is mainly a reduction in heart rate rather than a large reduction in the strength of cardiac contraction because the vagal fibers are distributed mainly in the atria and very little in the ventricles, where the force contraction of the heart

takes place [30].

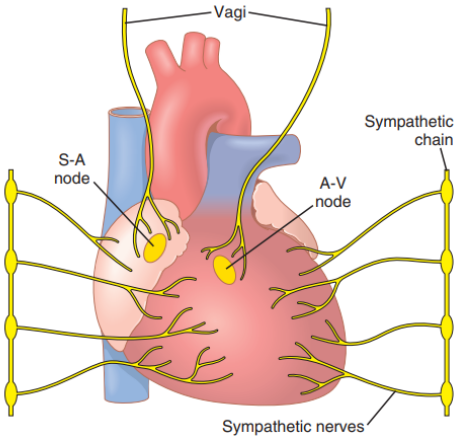

Figure 1.17: Cardiac sympathetic and parasympathetic (vagal) innervation. Reproduced from [30, p. 120].

1.3.2 Cardiovascular system models

The scientific community has been trying to model the cardiovascular system for several years now, and it is an interesting research topic because of its inherent mathematical complexity, the multi-scale nature of the physiological processes involved, and the need to develop reliable and efficient computational methods, also because of the increasing impact of cardiovascular diseases in our lives [60]. Moreover, it is an interdisciplinary challenge, requiring knowledge of physiology, mechanics of materials, fluid dynamics, and biochemistry [61].

Numerical and mathematical models can help us to better understand underlying mechanisms of human physiology, but they are also important for diagnostics, development of prostheses, novel treatment strategies, medical devices and they can be used for therapy selection and intervention planning [62],[63].

For instance, M. Haberbusch et al. [64] developed an integrated numerical model that is capable of predicting resting and exercise hemodynamics of heart transplantation recipients with good accuracy. This model consists of three components: a closed-loop lumped-parameter model of the human cardiovascular system, intrinsic control of heart rate represented by a Hodgkin-Huxley single-cell SA node model, and autonomic cardiac control mediated by the arterial baroreflex and pulmonary stretch reflex.

Moscato et al. [65] developed a control strategy for a ventricular assist device (VAD) that provides

an explicitly definable loading condition for the failing ventricle. Using a mathematical model of the cardiovascular system with an axial flow VAD, implemented by using Matlab-Simulink, the control strategy was tested in a failing left ventricle, slight physical activity and a recovery scenario. The cardiovascular system was modelled as a closed-loop hydraulic circuit comprising active atria and ventricles, systemic and pulmonary circuits, autoregulatory feedback loops for arterial systemic resistance (Ras) and the unstressed volume of venous systemic compliance (Cvs).

To evaluate the haemodynamic responses of patients with a left ventricular assist device (LVAD) and to compare the results with the responses reproduced by systematic cardiorespiratory numerical simulations, a computational simulator was developed by Gross et al. [66]. The simulator included a model with lumped parameters representing the cardiovascular and respiratory systems together, specifically adapted to heart failure conditions.

1.4 Aim of the thesis

The aim of this Master's thesis is to apply a deep neural network-based Koopman analysis to linearise complex physiological system models.

Specifically, a deep autoencoder was used to identify intrinsic coordinates, linearised by the Koopman operator, and decode these coordinates to recover the input of the autoencoder.

For that purpose, various deep neural network architectures were investigated and their performances in obtaining the Koopman operator were evaluated for three applications: a 2D non-linear dynamical system with a discrete eigenvalue spectrum (a toy model), the Fitz-Hugh-Nagumo model (a mathematical model of the nerve cell membrane) and, at a preliminary stage of development, a model of the cardiovascular system.

2 Methods

A brief description of the sections of the methods is provided below.

The first section (2.1) gives an outline of the three non-linear dynamical models analysed, a 2D non-linear dynamical system model with a discrete eigenvalue spectrum, the Fitz-Hugh-Nagumo model and a model of the cardiovascular system.

Then, section 2.2 describes the deep autoencoder technique for linear embeddings of non-linear dynamics, with an overview of the network architecture and the loss function.

Section 2.3 deals with the implementation of the deep neural network and includes the hardware specifications and the description of the code. In particular, the training and the parameters used for the training, the structure of the input files, the functions of the network architecture, and the generated validation code are described in detail.

Section 2.4 describes the datasets and the parameters set for training the three non-linear dynamical models, as well as the pre-processing of the datasets. At the end of subsection 2.4.2, Sobol's variance decomposition is described, it also lists the parameters used to perform the sensitivity analysis and gives the ranges for each of them. The last section (2.5) deals with the metrics for evaluating the results used in the three different models.

2.1 Description of non-linear dynamical models

2.1.1 Two-dimensional non-linear dynamical system model

As a toy model, a simple non-linear system, proposed in [3], was analysed. It consists of two first-order differential equations, with a single fixed point at the origin $x_1 = x_2 = 0$ and a discrete eigenvalue spectrum. The equations of the model are given below.

$$\begin{cases} \dot{x_1} = \mu x_1 \\ \dot{x_2} = \lambda(x_2 - x_1^2) \end{cases} \tag{27}$$

where $\mu = -0.05$ and $\lambda = -1$.

2.1.2 Fitz-Hugh-Nagumo model

The FHN model originates from the HH model, which consists of four first-order differential equations obtained from experiments with the squid axon (described in more detail in section 1.2.4). The two first-order differential equations from [2] that describe the FHN model are:

$$\begin{cases} \dot{v} = [c(w + v - \frac{1}{3}v^3 - I]\beta \\ \dot{w} = \frac{1}{c}(-v + a - bw)\beta \end{cases} \tag{28}$$

where I corresponds to the current applied to the membrane and is the bifurcation parameter, the variable $v(t)$ is the voltage-like variable with a cubic non-linearity that allows physical regeneration and self-excitation, $w(t)$ is the recovery variable that allows slower negative feedback. The parameters a (0.7), b (0.8) and c (3) are dimensionless and positive. The steady state values are $v_{rest} = 1.2$ and $w_{rest} = -0.625$. β is the time transformation factor, it can take a value between 4 and 7, so the solution will be 4-7 times faster than in the original problem with $\beta = 1$ [2]. The membrane voltage V (in mV) can be approximated as follows [2]:

$$V = 25(-v + 1.2) \tag{29}$$

2.1.3 Cardiovascular system model

The mathematical modelling of the cardiovascular system is complex since there are many parameters to consider. It can first be modelled using independent core components, each describing a single functionality. The next step is to integrate the core models into an overall system, which requires the introduction of suitable coupling conditions and novel numerical strategies for a stable, robust and computationally effective solution of the overall problem [60].

Haberbusch et al. [45] proposed a computational model of the acute cardiac effects of VNS. The model integrates a lumped-parameter representation of the cardiovascular system, comprising the heart with its chambers and the circulation with its venous and arterial compartments with a model of vagal nerve fiber stimulation and acetylcholine dynamics in cardiac-vagal synapses. For a detailed description of the cardiovascular system with all the equations to obtain the model, see the earlier work by Haberbusch et al. [67]. The developed cardiovascular system numerical model is shown in Figure 2.1.

In their study, Haberbusch et al. applied VNS in synchronization with the cardiac cycle, using the R-wave of the electrocardiogram as the trigger for the stimulator. As the presented model lacks the mechanism of myocardial depolarisation wave propagation, stimulation is instead triggered at the onset of left ventricular isovolumic contraction, identified by the left ventricular pressure (LVP) signal (Figure 2.2), as it corresponds well with the R wave. The stimulation parameters are explained in Figure 2.2.

For the control of the cardiovascular system using VNS, two cardiovascular markers were of particular interest: HR (heart rate) and MAP (mean arterial pressure).

The instantaneous HR was derived from the time differences between the peaks of the successive action potentials of the SA node and fed into the hemodynamic model. The MAP is the average arterial pressure during one cardiac cycle, systole, and diastole and it was calculated from systolic blood pressure (SBP) and diastolic blood pressure (DBP) as follows: $MAP = (SBP + 2 \times DBP)/3$ [67].

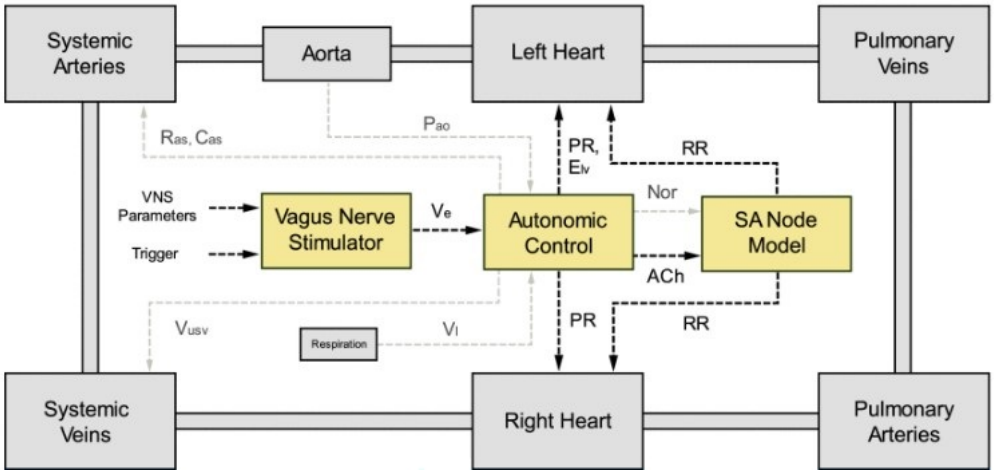

Figure 2.1: Cardiovascular system numerical model developed in [45, p. 614]. The autonomic control system governed by the arterial baroreflex and the pulmonary stretch reflex continuously integrates receptor information on blood pressure (Pao) and lung volume (Vl) and modulates total peripheral resistance (Ras), arterial compliance (Cas), venous unstressed volume (Vusv), left ventricular elastance (Elv), atrioventricular conduction time (PR), and sinoatrial (SA) node depolarization rate through the release of noradrenaline (Nor) and acetylcholine (ACh) and thus heart period (RR) respectively. The vagus nerve stimulator (VNS) is configurable, receiving stimulation parameters as input and generates a stimulation signal Iel. The VNS is operated in synchronization with the cardiac cycle using the rate of change in the left ventricular pressure as a trigger signal.

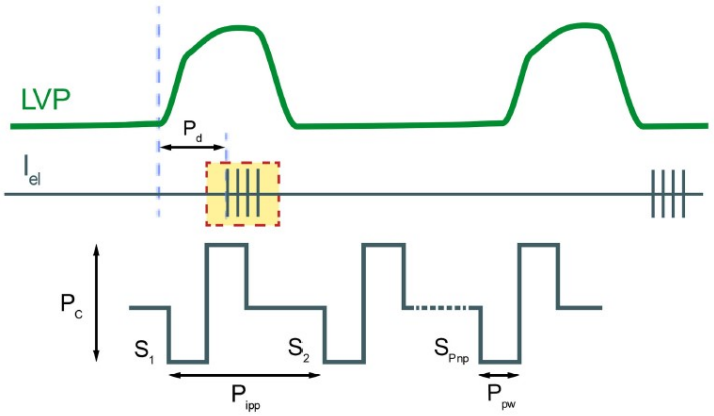

Figure 2.2: Illustration of the vagus nerve stimulation parameters. To synchronize the stimulation with the cardiac cycle, it is triggered through the peaks in the left ventricular pressure signal (LVP). The features of the stimulation signal (Iel) are defined by the current amplitude Pc, pulse width Ppw, number of pulses Pnp, inter-pulse period Pipp, and stimulation delay Pd. Reproduced from reference [45, p. 615].

2.2 Deep autoencoder for identification of linear embeddings

A. L. Branen in his master thesis [14], as future work in the "Recovering Linear Dynamics" section, proposed an approach using the Koopman operator to provide a coordinate transformation from a non-linear model into a linear model. This technique uses an autoencoder for dimensionality reduction: an encoder to reduce the model, afterwards, the Koopman analysis to linearise it, and a decoder to reconstruct the physiological activity.

Subsequently, the strategy developed by Lusch et al. in *"Deep learning for universal linear embeddings of non-linear dynamics"* [3] has been considered and deeply investigated for the purpose of linearisation of a highly non-linear lumped-parameter model of the cardiovascular system and the effects of vagus nerve stimulation.

Briefly, Lusch et al. [3] proposed a deep learning approach to obtain Koopman eigenfunctions' representations from the trajectories of dynamical systems, precisely they identified non-linear coordinates on which the dynamics are globally linear, using a modified auto-encoder.

They considered discrete-time dynamical systems, $x_{k+1} = F(x_k)$, where $x \in \mathbb{R}^n$ is the state of the system and **F** represents the dynamics that project the state of the system forward in time. The dynamics of **F** are usually non-linear or unknown, often only measurements of the dynamics are available. Representing non-linear dynamics with a linear system permits the achievement of

non-linear prediction, estimation, and control, using the theory available for linear systems [3].
The proposed network can identify some important intrinsic coordinates $y = \varphi(x)$, which can be obtained from a set of Koopman eigenfunctions $\varphi : \mathbb{R}^n \to \mathbb{R}^p$, together with a dynamical system $y_{k+1} = \mathbf{K}y_k$, using an autoencoder network to recover the input state $\mathbf{x}$ and an auxiliary network to construct a low-rank model of the intrinsic coordinates $\mathbf{y}$. In Figure 2.3 a schematic overview of the deep learning approach is presented. In Figure 2.3 (a), the deep-auto-encoder is shown, able to identify intrinsic coordinates $y = \varphi(x)$ and decode these coordinates to recover $x = \varphi^{-1}(y)$. In Figure 2.3 (b), the fundamental concept of the prediction of future time steps is demonstrated, while in Figure 2.3 (c) the idea of linearity enforcement is shown [3].
The employed approach implements three loss functions for the training of the model:

- **Intrinsic coordinates useful for reconstruction**: an auto-encoder is employed, made up of an encoder φ that identify intrinsic coordinates, and a decoder φ^{-1} to recover the initial state. See Figure 2.3 (a).
- **Linear dynamics**: to discover Koopman eigenfunctions, linear dynamics $\mathbf{K}$ are learned through the intrinsic coordinates. See Figure 2.3 (c).
- **Future state prediction**: the intrinsic coordinates enable future state prediction. See Figure 2.3 (b).

The detailed description along with the respective mathematical definitions are given later, in the next section.

2.2.1 Network architecture

The architecture of the autoencoder of the main network consists of two input layers, two hidden layers in the form $W_x + b$, followed by activation with the rectified linear unit (ReLU): $f(x) = max(0, x)$, and two linear output layers $(W_x + b)$ [11].
For systems with continuous eigenvalue spectrum, an auxiliary network (cf Figure 2.4) was developed to parameterise the continuous spectrum. The eigenvalues of the matrix $\mathbf{K}$ were allowed to vary, parameterised by the function $\lambda = \Lambda(y)$. The inputs of the auxiliary network are the intrinsic coordinates $\mathbf{y}$ and it outputs the parameters for the eigenvalues $\lambda_{\pm} = \mu \pm i\omega$ of $\mathbf{K}(\mu, \omega)$. For each complex conjugate pair of eigenvalues, the network defines a function Λ mapping $y_1^2 + y_2^2$ to μ and ω,

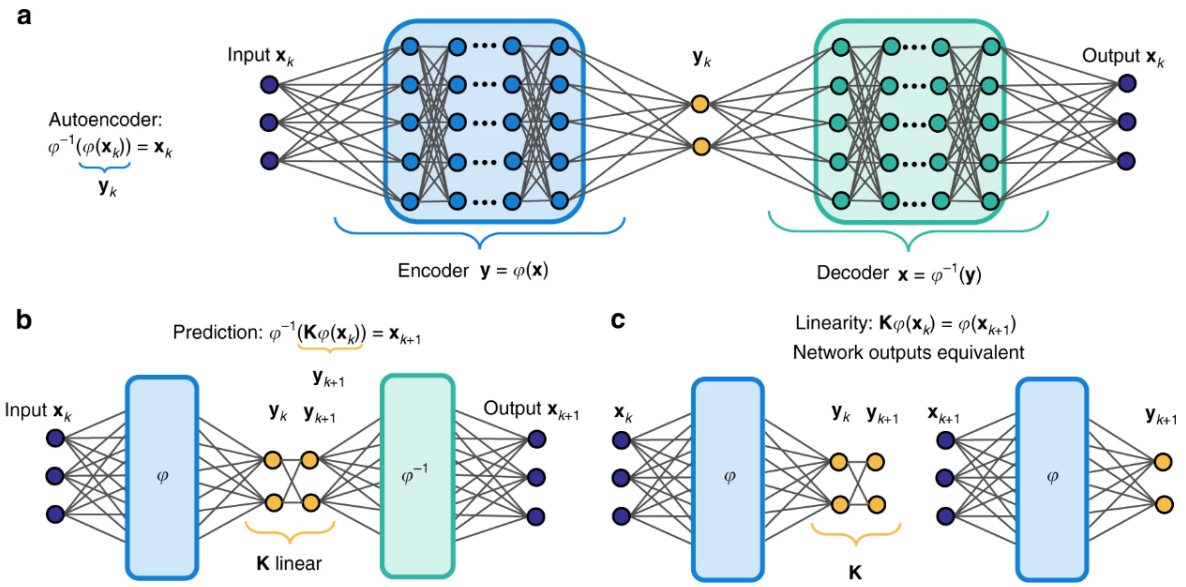

Figure 2.3: Diagram of the deep learning network to identify Koopman eigenfunctions $\varphi(x)$. (a) Intrinsic coordinates useful for reconstruction. (b) Future state prediction. (c) Linear dynamics. Reproduced from [3, p. 3].

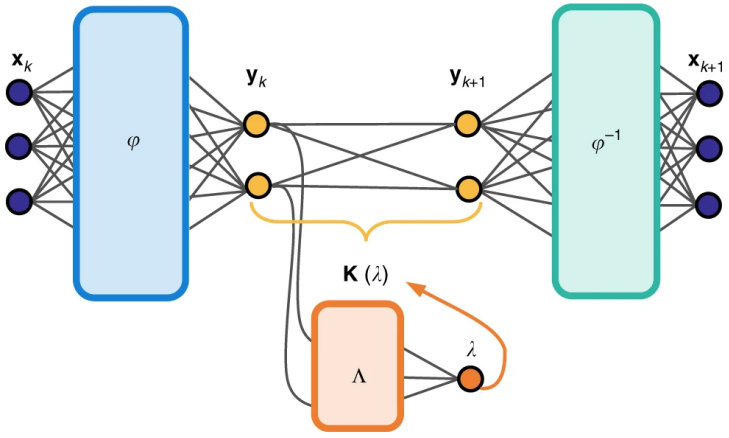

Figure 2.4: Diagram of the auxiliary network used to parameterize the continuous eigenvalue spectrum. Reproduced from [3, p. 4].

where y_j and y_{j+1} are the corresponding eigenfunctions. The architecture of the auxiliary network consists of three hidden layers [3].

2.2.2 Loss function

The loss function is composed of three weighted mean square error components: reconstruction accuracy, prediction of a future state, linearity and dynamics [3].
The loss function for the autoencoder is:

$$\mathcal{L}_{recon} = ||x_1 - \varphi^{-1}(\varphi(x_1))||_{MSE} \tag{30}$$

The intrinsic coordinates enable future state prediction with the loss function:

$$\mathcal{L}_{pred} = \frac{1}{S_p} \sum_{m=1}^{S_p} ||(x_{m+1}) - \varphi^{-1}(K^m \varphi(x_1))||_{MSE} \tag{31}$$

where S_p is a hyperparameter for how many steps to check in the prediction loss.
Linear prediction over m time steps is achieved with the loss function:

$$\mathcal{L}_{lin} = \frac{1}{T-1} \sum_{m=1}^{T-1} ||\varphi(x_{m+1}) - K^m \varphi(x_1)||_{MSE} \tag{32}$$

where T is the number of time steps in each trajectory.
There is also a term to penalize the data point with the largest loss, the infinite norm on autoencoder loss and one-step prediction loss:

$$\mathcal{L}_{\infty} = \|x_1 - \varphi^{-1}(\varphi(x_1))\|_{\infty} + \|x_2 - \varphi^{-1}(K\varphi(x_1))\|_{\infty} \tag{33}$$

Furthermore, a term with L_2 regularization on the weights W is added, to prevent overfitting. All the loss functions are summed up, obtaining:

$$\mathcal{L} = \alpha_1(\mathcal{L}_{recon} + \mathcal{L}_{pred}) + \mathcal{L}_{lin} + \alpha_2 \mathcal{L}_{\infty} + \alpha_3 \|W\|_2^2 \tag{34}$$

where $\alpha_1, \alpha_2, \alpha_3$ are the weights of the loss functions, tuned as hyperparameters.

2.3 Implementation of the deep neural network

2.3.1 Hardware specifications and code description

The laptop used for conducting the training, generating the datasets of the three models and post-processing the results has the features listed in Table 3. The dedicated GPU (graphics processing unit) was used for training the model.

The code is mostly written in Python and uses the Python API (application programming interface) for the TensorFlow framework [68] and the Adam optimiser for training [3], a method for efficient stochastic optimisation that requires only first-order gradients with low memory requirement[69].

The Python libraries and the respective versions needed to run the scripts are listed in Table 4.

The code includes several sections: the data containing MATLAB scripts to generate datasets, the Python scripts to set parameters and perform a random parameter search, the training code to train the neural network, the network architecture code, the helperfns code and the post-processing, Jupyter notebooks to analyse the results.

Lusch et al. [3] made the code available online at github.com/BethanyL/DeepKoopman.

Hardware specifications	
CPU	8 core Intel i7-4710HQ 2.50 GHz
GPU	NVIDIA GeForce GTX 960M
RAM	16 GB
OS	Windows 10

Table 3: Specifications of the hardware used: CPU, central processing unit, GPU, graphics processing unit, RAM, random access memory, OS, operating system.

Library	**Version**
Tensorflow	2.11.0
Numpy	1.23.5
Scikit-learn	1.2.0
Pandas	1.5.3
Matplotlib	3.6.2
Scipy	1.9.3

Table 4: Python libraries required.

2.3.1.1 Parameters to set and training of the neural network

The main script "DiscreteSpectrumExampleExperimentBestParams.py", where the parameters can be set, contains settings related to the dataset, saving results, network architecture, loss functions, training and timing. They are described in more detail in Table 5 for the main network and in Table 6 for the auxiliary network. In the end, there is a loop to run 200 experiments with random parameters and initialisations of the weights.

The script "DiscreteSpectrumExampleExperiment.py" performs a random search to find the optimal hyperparameters for the experiment, it is shown in Listing 1. The tuned hyperparameters include: the batch size, whether to train the autoencoder first, the weights of the loss functions, the number of layers for the autoencoder of the main network and of the auxiliary network, and the widths of the hidden layers.

Parameters to set for the main network		
Settings related to	**Parameter**	**Explanation**
DATASET	data_name	name of the dataset file
	data_train_len	number of training data files
	len_time	length of each simulation
	n	dymension of the system
	num_initial_conditions	number of initial conditions
	delta_t	time stepping in data
SAVING RESULTS	folder_name	name of the folder where results are saved
NETWORK ARCHITECTURE	num_evals (k)	dimension of y-coordinates
	w	number of hidden units of the autoencoder
	widths (d)	number of layers of the encoder and decoder
LOSS FUNCTION	num_shifts	number of steps penalised for prediction
	num_shifts_middle	number of steps penalised in the linearity loss
	max_shifts	max between num_shifts and num_shifts_middle
	num_examples	num_initial_conditions×(len_time-max_shifts)
	recon_lam	weight on losses involving reconstruction
	Linf_lam	weight for the infinite norm on autoencoder error and one prediction step loss function
	L1_lam	weight to regularize loss 1

Continued on next page

Table 5 – *Continued from previous page*		
	L2_lam	weight to regularize loss 2
	auto_first	if 1 it does the pre-training of the autoencoder first, if 0 not
TRAINING	num_passes_per_file	how often the training dataset is shuffled
	num_steps_per_batch	number of steps per batch before moving to next training file
	learning_rate	learning rate
	batch_size	number of samples processed before the model is updated
	steps_to_see_all	num_examples/batch_size
	num_steps_per_file_pass	steps per training file before moving to next one
TIMING	max_time	maximum time of training
	min_Xmin	X minutes of training
	min_Xhr	X hours of training
	min_halfway	max_time/2

Table 5: Parameters to set in the main network for the training [3].

Parameters to set for the auxiliary network		
Settings related to	**Parameter**	**Explanation**
NETWORK ARCHITECTURE	num_real	number of real eigenvalues
	num_complex_pairs	number of complex eigenvalues
	$w0$	number of hidden units of the auxiliary network
	hidden_widths_omega	width of the hidden layers of the auxiliary network

Table 6: Parameters to set in the auxiliary network for the training [3].

```
for count in range(200):  # loop to do random experiments
   # Batch size
    params['batch_size'] = int(2 ** (r.randint(7, 9)))
    steps_to_see_all = num_examples / params['batch_size']
    params['num_steps_per_file_pass'] = (int(steps_to_see_all) + 1) * params['num_steps_per_batch']

```

```
    # Whether to train the autoencoder first
    if r.random() < .5:
        params['auto_first'] = 1
    else:
        params['auto_first'] = 0

    # Weights of the loss functions
    params['L2_lam'] = 10 ** (-r.randint(13, 15))
    if r.random() < .5:
        params['Linf_lam'] = 0.0
    else:
        params['Linf_lam'] = 10 ** (-r.randint(6, 10))

    # Number of layers and widths for the main network
    d = r.randint(1, 4)
    if d == 1:
        wopts = np.arange(50, 160, 10)
        w = wopts[r.randint(0, len(wopts) - 1)]
        params['widths'] = [2, w, k, k, w, 2]
    elif d == 2:
        wopts = np.arange(15, 45, 5)
        w = wopts[r.randint(0, len(wopts) - 1)]
        params['widths'] = [2, w, w, k, k, w, w, 2]
    elif d == 3:
        wopts = np.arange(10, 25, 5)
        w = wopts[r.randint(0, len(wopts) - 1)]
        params['widths'] = [2, w, w, w, k, k, w, w, w, 2]
    elif d == 4:
        wopts = np.arange(10, 20, 5)
        w = wopts[r.randint(0, len(wopts) - 1)]
        params['widths'] = [2, w, w, w, w, k, k, w, w, w, w, 2]

    # Number of layers and widths for the auxiliary network
    do = r.randint(1, 4)
    if do == 1:
        wopts = np.arange(20, 110, 10)
        wo = wopts[r.randint(0, len(wopts) - 1)]
        params['hidden_widths_omega'] = [wo, ]
    elif do == 2:
```

```
        wopts = np.arange(10, 25, 5)
        wo = wopts[r.randint(0, len(wopts) - 1)]
        params['hidden_widths_omega'] = [wo, wo]
    elif do == 3:
        wopts = np.arange(5, 20, 5)
        wo = wopts[r.randint(0, len(wopts) - 1)]
        params['hidden_widths_omega'] = [wo, wo, wo]
    elif do == 4:
        wopts = np.arange(5, 15, 5)
        wo = wopts[r.randint(0, len(wopts) - 1)]
        params['hidden_widths_omega'] = [wo, wo, wo, wo]

    training.main_exp(copy.deepcopy(params))
```

Listing 1: 'DiscreteSpectrumExampleExperiment.py': loop for hyperparameters search. The explanation of the parameters can be found in Tables 5 and 6.

The code for a custom training loop was provided, it consists of four sections.
In the first section, shown in Listing 2, the loss functions for the training are defined: the autoencoder loss (lines 9-18), the prediction loss (lines 21-35), the linearity loss (lines 38-66) and the infinite norm on the autoencoder and the one-step prediction loss (lines 69-84), then they sum all the above loss functions (line 86).

```
def define_loss(x, y, g_list, weights, biases, params):
    # Minimize the mean squared errors.
    # subtraction and squaring element-wise, then average over both dimensions
    # n columns
    # average of each row (across columns), then average the rows
    denominator_nonzero = 10 ** (-5)

   # autoencoder loss
    if params['relative_loss']:
        loss1_denominator = tf.reduce_mean(tf.reduce_mean
        (tf.square(tf.squeeze(x[0, :, :])), 1)) + denominator_nonzero
    else:
        loss1_denominator = tf.to_double(1.0)

    mean_squared_error = tf.reduce_mean(tf.reduce_mean(tf.square(y[0] -
    tf.squeeze(x[0, :, :])), 1))
    loss1 = params['recon_lam'] *
```

```
    tf.truediv(mean_squared_error, loss1_denominator)}

    # gets dynamics/prediction
    loss2 = tf.zeros([1, ], dtype=tf.float64)
    if params['num_shifts'] > 0:
        for j in np.arange(params['num_shifts']):
            # xk+1, xk+2, xk+3
            shift = params['shifts'][j]
            if params['relative_loss']:
                loss2_denominator = tf.reduce_mean(
                    tf.reduce_mean(tf.square(tf.squeeze
                    (x[shift, :, :])), 1)) + denominator_nonzero
            else:
                loss2_denominator = tf.to_double(1.0)
            loss2 = loss2 + params['recon_lam'] * tf.truediv(
                tf.reduce_mean(tf.reduce_mean(tf.square(y[j + 1]
                - tf.squeeze(x[shift, :, :])), 1)), loss2_denominator)
        loss2 = loss2 / params['num_shifts']

    # K linear
    loss3 = tf.zeros([1, ], dtype=tf.float64)
    count_shifts_middle = 0
    if params['num_shifts_middle'] > 0:
        # generalization of: next_step = tf.matmul(g_list[0], L_pow)
        omegas = net.omega_net_apply(params, g_list[0], weights, biases)
        next_step = net.varying_multiply(g_list[0], omegas,
        params['delta_t'], params['num_real'],
        params['num_complex_pairs'])
        # multiply g_list[0] by L (j+1) times
        for j in np.arange(max(params['shifts_middle'])):
            if (j + 1) in params['shifts_middle']:
                if params['relative_loss']:
                    loss3_denominator = tf.reduce_mean(
                        tf.reduce_mean(tf.square(tf.squeeze
                        (g_list[count_shifts_middle + 1])), 1))
                        + denominator_nonzero
                else:
                    loss3_denominator = tf.to_double(1.0)
                loss3 = loss3 + params['mid_shift_lam'] *
```

```
                tf.truediv(tf.reduce_mean(tf.reduce_mean
                (tf.square(next_step - g_list[count_shifts_middle +
                1]), 1)),loss3_denominator)
                count_shifts_middle += 1
            omegas = net.omega_net_apply(params, next_step, weights, biases)
            next_step = net.varying_multiply(next_step, omegas,
            params['delta_t'], params['num_real'],
            params['num_complex_pairs'])

        loss3 = loss3 / params['num_shifts_middle']

    # inf norm on autoencoder error and one prediction step
    if params['relative_loss']:
        Linf1_den = tf.norm(tf.norm(tf.squeeze(x[0, :, :]), axis=1,
        ord=np.inf), ord=np.inf) + denominator_nonzero
        Linf2_den = tf.norm(tf.norm(tf.squeeze(x[1, :, :]), axis=1,
        ord=np.inf), ord=np.inf) + denominator_nonzero
    else:
        Linf1_den = tf.to_double(1.0)
        Linf2_den = tf.to_double(1.0)

    Linf1_penalty = tf.truediv(
        tf.norm(tf.norm(y[0] - tf.squeeze(x[0, :, :]), axis=1,
        ord=np.inf), ord=np.inf), Linf1_den)
    Linf2_penalty = tf.truediv(
        tf.norm(tf.norm(y[1] - tf.squeeze(x[1, :, :]), axis=1,
        ord=np.inf), ord=np.inf), Linf2_den)
    loss_Linf = params['Linf_lam'] * (Linf1_penalty + Linf2_penalty)

    loss = loss1 + loss2 + loss3 + loss_Linf

    return loss1, loss2, loss3, loss_Linf, loss
```

Listing 2: 'Training.py': loss functions. The explanation of the parameters can be found in Tables 5 and 6.

This training function requires as inputs the following arguments:

1. **x** = placeholder for input
2. **y** = list of output of the network for each prediction step
3. **g_list** = list of the output of the encoder for each step in x

4. **weights** = dictionary of weights for all the networks
5. **biases** = dictionary of biases for all the networks
6. **params** = dictionary of parameters for the experiments.

The function returns the following arguments:

1. **loss1** = autoencoder loss function
2. **loss2** = dynamics/prediction loss function
3. **loss3** = linearity loss function
4. **loss_Linf** = inf norm on autoencoder loss and one-step prediction loss
5. **loss** = sum of the above four losses.

In the second section of the training, the regularisation is defined and added to the loss functions. The third section deals with conducting a random experiment for specific parameters and dataset, it also includes the training of the model, and at the end of the training the results are stored.
The last section is about setting up and running a random experiment.
To prevent overfitting, it is possible to change the settings for the timing. The maximum training time is set to 4 hours by default, but it can be changed. The model is trained for 5 minutes and, if the validation error does not decrease enough, the training is stopped and a new experiment is started. After 20 or 40 minutes and then after 1, 2 or 3 hours, the same approach is followed.

2.3.1.2 Structure of input files

The function 'stack_data' has been implemented to convert the dataset from a 2D array to a 3D array in order to stack the simulations for different initial conditions. The inputs to this function are the dataset (it has as many columns as the number of inputs to the system), the number of shifts (time steps) the losses will use (the maximum is len_time - 1), and the length of each simulation. The output is the dataset converted into a 3D array, with the shape: (length of a simulation, number of simulations, number of inputs).

2.3.1.3 Network architecture functions

The script "networkarch.py" contains some functions regarding the network architecture, the main functions are described below.

- **Create_omega_net:** it creates the auxiliary (omega) network, which has the intrinsic coordinates and the parameters of the experiment, as inputs and returns the parameters for

the eigenvalues of **K** (μ and ω), and the dictionaries of weights and biases, as outputs.

- **Omega_net_apply:** it applies the omega network to the intrinsic coordinates. It has the parameters of the experiment, the intrinsic coordinates, the dictionaries of weights and biases, as inputs and returns a list, the omega network applied to the intrinsic coordinates, as outputs.
- **Create_koopman_net:** it creates a Koopman network that encodes, advances in time and decodes, as shown in Listing 3. It takes the parameters of the experiment as inputs and returns:
  1. **x** = placeholder for input
  2. **y** = a list, the output of the decoder applied to each shift: $g\_list[0]$, $K * g\_list[0]$, $K^2 * g\_list[0]$, ..., length num_shifts + 1
  3. **g_list** = a list, the output of the encoder applied to each shift in the input
  4. **weights** = dictionary of weights
  5. **biases** = dictionary of biases.

  The first step is to create the encoder network (lines 7-9), then apply this encoder to the input data (lines 13-14), following, the auxiliary network is created using the "create_omega_net" function (line 18), then a decoder network is created and applied to the intrinsic coordinates (lines 26-35). Finally, the coordinates are advanced in time with the function "varing_multiply" and the function "omega_net_apply" is used to apply the omega network to the intrinsic coordinates (lines 39-50).

```
1  def create_koopman_net(params):
2      depth = int((params['d'] - 4) / 2)
3      max_shifts_to_stack = helperfns.num_shifts_in_stack(params)
4      encoder_widths = params['widths'][0:depth + 2]  # n ... k
5
6      # Creation of the encoder network
7      x, weights, biases = encoder(encoder_widths, dist_weights=params['dist_weights'][0:depth + 1],
8                                   dist_biases=params['dist_biases'][0:depth + 1], scale=params['scale'],
9                                   num_shifts_max=max_shifts_to_stack)
10     params['num_encoder_weights'] = len(weights)
11
12     # Apply the encoder to the input data
13     g_list = encoder_apply(x, weights, biases, params['act_type'], shifts_middle=params['shifts_middle
14             '], num_encoder_weights=params['num_encoder_weights'])
15
16     # Creation of the auxiliary network
```

```
    # g_list_omega is list of omegas, one entry for each middle_shift of x (like g_list)
    omegas, weights_omega, biases_omega = create_omega_net(params, g_list[0])

    weights.update(weights_omega)
    biases.update(biases_omega)
    num_widths = len(params['widths'])
    decoder_widths = params['widths'][depth + 2:num_widths]  # k ... n

    # The decoder network is created and applied to the intrinsic coordinates
    weights_decoder, biases_decoder = decoder(decoder_widths, dist_weights=params['dist_weights'][
                depth + 2:],dist_biases=params['dist_biases'][depth + 2:], scale=params['scale'])
    weights.update(weights_decoder)
    biases.update(biases_decoder)

    y = []
    # y[0] is x[0,:,:] encoded and then decoded (no stepping forward)
    encoded_layer = g_list[0]
    params['num_decoder_weights'] = depth + 1
    y.append(decoder_apply(encoded_layer, weights, biases, params['act_type'], params['
    num_decoder_weights']))

    # The coordinates are advanced in time
    # g_list_omega[0] is for x[0,:,:], pairs with g_list[0]=encoded_layer
    advanced_layer = varying_multiply(encoded_layer, omegas, params['delta_t'], params['num_real'],
                        params['num_complex_pairs'])

    for j in np.arange(max(params['shifts'])):
        # considering penalty on subset of yk+1, yk+2, yk+3, ...
        if (j + 1) in params['shifts']:
            y.append(decoder_apply(advanced_layer, weights, biases, params['act_type'], params['
            num_decoder_weights']))

        omegas = omega_net_apply(params, advanced_layer, weights, biases)
        advanced_layer = varying_multiply(advanced_layer, omegas, params['delta_t'], params['num_real'
                        ], params['num_complex_pairs'])

    return x, y, g_list, weights, biases
```

Listing 3: 'Networkarch.py': create_koopman_net function. The explanation of the parameters can be found in Tables 5 and 6.

2.3.1.4 Validation code

The Jupyter notebook (version 1.0.0), a tool for interactive development and presentation of projects, was used to create a validation script to evaluate the results after the training.
First, the training and validation errors were plotted, then the reduced model signals and the original signals were plotted in the same plot to compare them, next, the plot of the eigenvalues was performed to ensure that they were constant. After that, some trajectories with the corresponding predictions were plotted to visualise the performance of the model for exemplary simulations.
Moreover, a long prediction was performed given only the initial conditions, some simulations were plotted to compare the difference between the reduced model and the original model, and also the average log10 prediction error, as the number of prediction steps increased, was plotted to see how it evolved over time. Finally, the average root mean squared error (ARMSE) was calculated to quantify the performance of the model in prediction.
The following functions were used to calculate the predictions.

- **PredictKoopmanNetOmegas:** it applies the network to just the initial conditions of each simulation. It takes the initial conditions, the weight and biases, and the parameters of the experiment, as inputs and returns the data transformed to intrinsic coordinates for three steps, the reconstructed input and the predictions for three steps, as output.
- **ApplyKoopmanNetOmegasFull:** it applies the network to the full dataset. It takes the stacked dataset, the weights and biases and the parameters of the experiment, as inputs and returns the list of predictions and the list of the intrinsic coordinates, as outputs.

2.4 Dataset and parameters for the training of non-linear dynamical models

The code was first used to replicate the prediction of the dynamics of the 2D non-linear dynamical system model, as it was done by Lusch et al. in [3]. Subsequently, the Fitz-Hugh-Nagumo's model was created using Simulink. The code parameters have been changed to fit the new model. Finally, a model of the cardiovascular system was used to predict the heart rate and the mean arterial pressure. The code was modified again to fit the parameters of the new dataset.

2.4.1 Two-dimensional non-linear dynamical system model

2.4.1.1 Preprocessing of the dataset

The dataset was generated using a MATLAB script that implements the model. 5000 simulations were created starting from random initial conditions, where $x_1, x_2 \in [-0.5, 0.5]$, since this part of the phase space is sufficient to capture the dynamics. The dataset was then split into training, test and validation sets in the ratio 70:10:20. For each initial condition, the differential equations were solved for a time period $t = 0, 0.02, ..., 1$.

2.4.1.2 Parameters for the training

The parameters set for training this model can be found in Table 7. The results were compared with the results obtained in [3].

Settings related to	Parameter	Value
DATASET	num_initial_conditions	3500
	time_span	1 s
	len_time	51
	delta_t	0.02
	n (system dimension)	2
LOSS FUNCTION	num_shifts	30
TRAINING	batch_size	256
	learning_rate	0.001

Table 7: Parameters to set for the training of the 2D non-linear dynamical system model.

2.4.2 Fitz-Hugh-Nagumo model

Mathworks MATLAB® R2022b (version 9.13.0) was used to generate the initial conditions of the model, the model was built via SIMULINK® (version 10.6), using the Ode45 solver. Based on these values of the initial conditions, $v_0 \in [-0.5, 1.5]$ and $w_0 \in [-0.625, 0]$, 1000 initial conditions were created using Latin Hypercube Sampling, and 1000 simulations were also generated using Simulink. The parameters and ranges for the creation of the Fitz-Hugh-Nagumo model are given in Table 8. For a, b, and c the nominal value $\pm$ 50 % of the nominal value was considered as range, for v_0, w_0 and β the ranges proposed in [2] were considered.

Figure 2.5 shows the implementation of the FHN model with Simulink, which follows equations (28). The model consists of two structures with two integrators: the first equation is shown above in black and the second below in green. The stimulus current I, in blue, is generated using two-step blocks, the raising step has a step time of 200 ms, initial value of 0 and final value I, the lower step has a step time of 900 ms, 0 as initial value and -I as final value. The output variables are v and w to investigate the dynamics of the model. The red blocks represent equation (29) and are useful for observing the membrane voltage V. The sampling time is set to 0.1 s.

The stimulus current I should be in a certain range to excite the system and obtain a positive signal as a nerve response. However, it is not possible to give a unit for the stimulus current I as it depends on the parameterisation of the model and the physiological significance should be taken into account.

Parameter	**Range**
a	[0.35, 1.05]
b	[0.4, 1.2]
c	[1.5, 4.5]
v_0	[-0.5,1,5]
w_0	[-0.625,0]
β	[3, 7]

Table 8: Parameters to set for the creation of the Fitz-Hugh-Nagumo model.

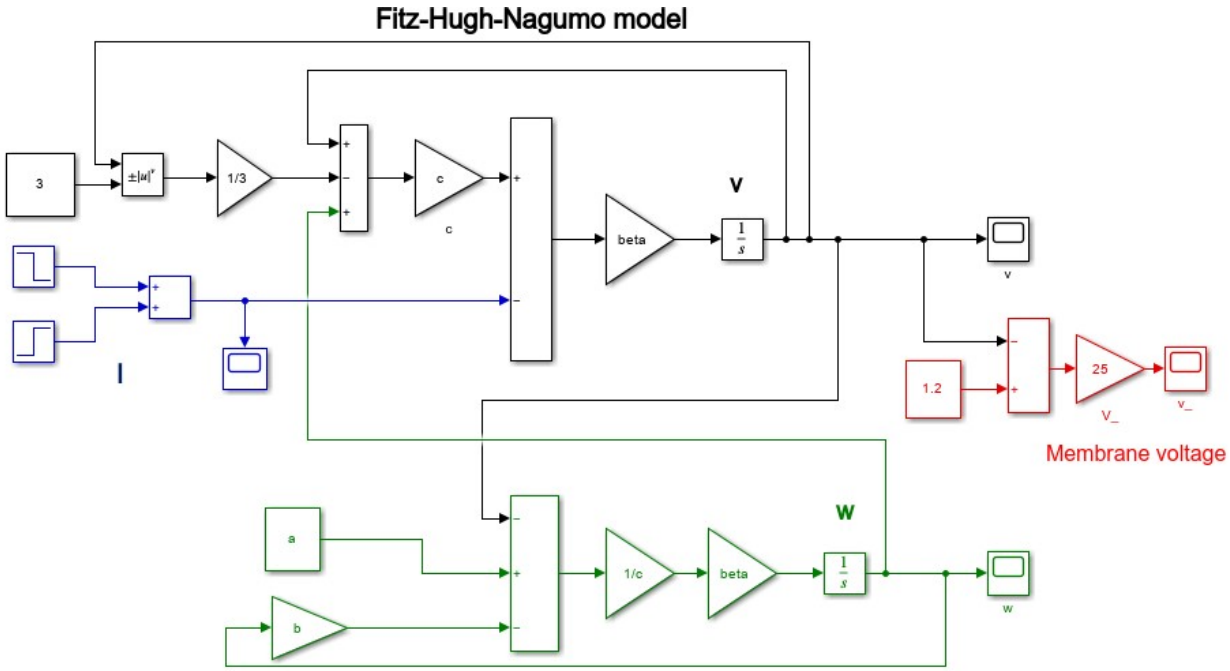

Figure 2.5: Simulink implementation of the Fitz-Hugh-Nagumo model.

2.4.2.1 Preprocessing of the dataset

To speed up the simulations, a Python script was created to generate the dataset. It uses the Python ODE (ordinary differential equation) solvers to solve the two equations (28). The dataset was then split into training (80%: 800 simulations), test (10%: 100 simulations) and validation (10%: 100 simulations) sets.

It was afterward standardised using the Z-score method with a mean of zero and unitary standard deviation.

2.4.2.2 Parameters for the training

The parameters set for training the Fitz-Hugh-Nagumo model can be found in Table 9. The batch size (bs) is not fixed because several values of this parameter were tried for training the model, see Table 16 for all the bs values used.

Originally a simulation of 6 s was considered, 6001 time steps were generated with a time step of 0.01, but the training of the model did not work, so a simulation of 3 s was chosen, 30 time steps were used with a time step of 0.1.

With the parameter num_shifts set to 29 ($len\_time - 1$), the best results were obtained.

Some hyperparameters were tuned, one at time, to get a small prediction error, Table 10 shows the

hyperparameters tuned and their ranges.

Settings related to	Parameter	Value
DATASET	num_initial_conditions	800
	time_span	3 s
	len_time	30
	delta_t	0.1
	n (system dimension)	2
LOSS FUNCTION	num_shifts	29
TRAINING	batch_size	not fixed
	learning_rate	0.001

Table 9: Parameters to set for the training of the Fitz-Hugh-Nagumo model.

Settings related to	Parameter	Range of values
MAIN NETWORK	w	30, 64, 128, 256, 512
	d (widths)	2, 3, 4
AUXILIARY NETWORK	$w0$	10, 32, 64, 128, 256
TRAINING	batch_size	32, 64, 128, 256

Table 10: Hyperparameters tuned for the training with their ranges. For a description of the tuned hyperparameters refer to Tables 5 and 6.

2.4.2.3 Sobol's variance decomposition

Sobol's variance decomposition, a global variance-based sensitivity analysis, is applied to the FHN model to quantify the relative importance of each input parameter to the output and consequently decide which parameter ranges to use for creating a new dataset and re-training the model.

The Sobol method measures the effect of a parameter relative to the variability of the observed output $\mathbb{V}[Y]$ [49]. It is convenient to calculate some indices to understand which inputs are most crucial for predicting the output.

The **first order effect**, denoted by $E_i = \mathbb{V}[Y|S[i]]$, represents the effect on output explained only by the variability of the parameter S[i], all other parameters remaining constant.

The **nth-order effect**, denoted as $E_{i...k} = \mathbb{V}[Y|S[i], ..., S[k]]$, is the effect explained by the joint variability of several parameters $S[i], ..., S[k]$.

The **total effect**, defined as $E_{Ti} = E_i + \sum_j E_{ij} + \sum_{j,m} S_{ijm} + ...$, characterises the effect caused by S[i] plus the effect caused by all possible interactions with the other parameters [49].

The sensitivity analysis was computed with Python, using the SALib library.
With SALib one can generate the model inputs, using one of the sample functions, and compute the sensitivity indices from the model outputs, using one of the analyze functions. A typical sensitivity analysis using SALib follows four steps [70]:

1. Determine the model inputs (parameters) and their sample range;
2. Run the sample function to generate the model inputs;
3. Evaluate the model using the generated inputs, saving the model outputs;
4. Run the analyze function on the outputs to compute the sensitivity indices.

The Saltelli sampler was chosen to generate the input samples, it generates N(2*D+2) samples, where N is usually a power of two and D is the number of model inputs. In this case, N was 1024 and D was 6, so 14336 samples were generated. All the values of the parameters used for the sensitivity analysis were equally distributed.
As output of the sensitivity analysis, the frequency of spikes of v was calculated as the reciprocal of the difference between a peak time and the previous peak time. This parameter could correspond to the heart rate in the cardiovascular system. The inputs and outputs included in the sensitivity analysis are listed in Table 11, and the ranges of the value are also provided.

	Parameter	**Range values**
INPUT	a	[0, 1]
	b	[0, 1]
	c	[0, 4]
	v_0	[-1, 2]
	w_0	[-1, 0.5]
	β	[1, 7]
OUTPUT	frequency of v	-

Table 11: Inputs and output parameters, with their ranges of value, used for the sensitivity analysis of the Fitz-Hugh-Nagumo model.

2.4.3 Cardiovascular system model

The model of the cardiovascular system considered in this thesis to extract the parameters is an unpublished modified version, developed in our group, of the model described in [45], which uses a rabbit SA node model [71] rather than a human one. The parameters considered in this model are

listed and described in Table 12. The unit and range of the values are also given. The first four parameters are the inputs to the autoencoder, the stimulation parameters, and the last two are the outputs of the autoencoder.

The charge Q was not included in the dataset and it was determined by multiplying the current amplitude by the pulse width. This is useful for reducing two parameters to one.

The reason for this choice of parameters is that a sensitivity analysis of the parameters for the stimulation of the vagus nerve, as described by Haberbusch in [72], showed that C (current amplitude), PW (pulse width), NP (number of pulses per burst) and F (frequency of stimulation) have the greatest influence on the cardiac response. The results of the sensitivity analysis are shown in Figure 2.6.

	Parameter	**Description**	**Unit**	**Value**
INPUT	C	Current amplitude	mA	[0, 5]
	PW	Pulse width	µs	[50, 300]
	F	Frequency of stimulation	Hz	[5, 50]
	Q	Charge	nC	[0, 1.5]
OUTPUT	HR	Heart rate	bpm	[129, 167]
	MAP	Mean Arterial Pressure	mmHg	[21, 199]

Table 12: Input and output parameters of the cardiovascular system model.

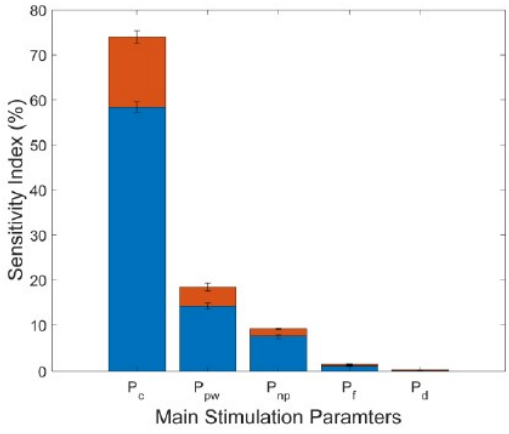

Figure 2.6: Results of the sensitivity analysis of the parameters for the stimulation of the vagus nerve, for the first order and total order effects. It is evident that the current (C) and pulse width (PW) have the greatest influence on the chronotropic response. P_c, current amplitude, P_{pw}, pulse width, P_{np}, number of pulses per burst, P_f, frequency, P_d, stimulation onset delay. Reproduced from [72, p. 108].

The original dataset contained two additional parameters: LVP (left ventricular pressure) and

CO (cardiac output), not considered for the training of this cardiovascular system model because they were of secondary interest.

In the original signals, each simulation was 12901 time steps long, but to achieve reasonable training performance and time, a shorter time frame (between samples 4800 and 5500) was chosen to capture the dynamics. The stimulation was on from the samples 4900 to 9900, as shown in Figure 2.7.

A simulation of the dataset, until sample 10000 for the time evolution of the output signals and the stimulation trigger signal (time sampling = 0.01 s) is shown in Figure 2.7. All stimulation parameters are step-like, as the stimulation trigger signal. The values used to generate this response were: C = 5 mA, PW = 90 μs, F = 23 Hz.

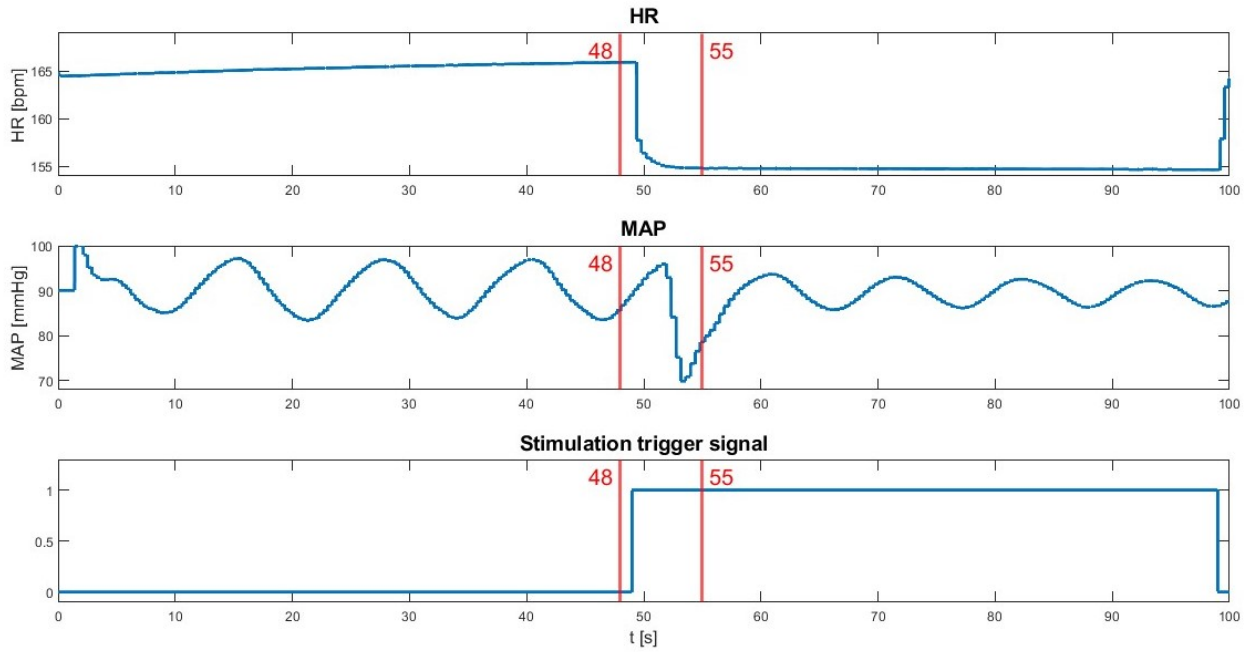

Figure 2.7: An example, among all the simulations of the dataset (until sample 10000), of the outputs and the stimulation trigger signal of the cardiovascular system model. The red vertical lines are the limits of the selected time frame (700 samples). The stimulation parameters used to produce this response were:
C = 5 mA, PW = 90 μs, F = 23 Hz.

2.4.3.1 Preprocessing of the dataset

The dataset consisted of 2010 simulations, with an original time sampling of 0.01 s.

It was then segmented and only 700 samples were selected from each simulation (originally 12901 time steps long) to cover the time frame in which the stimulation occurred, as shown in Figure 2.7 with the red vertical lines.

The dataset was first standardised with the Z-score method, to obtain a mean of zero and a unitary

standard deviation. Afterward, it was rescaled in the range [-0.5, 0.5]. Then the dataset was down-sampled by a factor of 10 to obtain only 71 time steps. Finally, the dataset was divided into training (70%), test (10%) and validation (20%) sets. All these steps are resumed in Figure 2.8.
The total number of simulations for the training set was 1407: the first 704 simulations (50 %) led to a heart rate response (this means the heart rate decreased until it reached a steady state), and the last 50 % of the simulations did not.

Figure 2.8: Steps to preprocess the cardiovascular system dataset.

2.4.3.2 Parameters for the training

The parameters used for training the cardiovascular system model are listed in Table 13. The dimension of the system is not fixed as different combinations of input parameters to the autoencoder were tried for training the model. Table 14 lists all the input combinations used.
It was then evaluated which combinations of inputs made the training perform better in terms of training and validation errors, computational cost per step and ARMSE, which are discussed in more detail in section 2.5.

Settings related to	Parameter	Value
DATASET	num_initial_conditions	1407
	time_span	7 s
	len_time	71
	delta_t	0.1
	n (system dimension)	not fixed
LOSS FUNCTION	num_shifts	70
TRAINING	batch_size	32
	learning_rate	0.001

Table 13: Parameters to set for the training of the cardiovascular system model.

Input combinations	System dimension (n)	Output
HR, C	2	HR
HR, C, PW, F	4	HR
HR, Q	2	HR
HR, Q, F	3	HR
HR, Q, MAP	3	HR, MAP
HR, Q, F, MAP	4	HR, MAP

Table 14: Input combinations for the training of the cardiovascular system model with relative output. The system dimension (n) is a parameter related to the dataset (Table 5).

2.5 Metrics to evaluate the reduced model performance

The 2D non-linear dynamical system model, the Fitz-Hugh-Nagumo model and the cardiovascular system model were evaluated in terms of training error, validation error, computational cost to predict one time step, ARMSE for all the simulations, and prediction error ranges, the average log_{10} prediction error as the number of prediction steps increases.

Furthermore, a representation of the eigenvalues of the system obtained in the paper [3], in which the eigenvalues may vary with the auxiliary network used for the continuous spectrum, is given. If the eigenvalues are relatively constant when allowed to vary with respect to their eigenfunctions, one can conclude that the system is linearised.

A phase space diagram was created, for the Fitz-Hugh-Nagumo model, in which the dynamics of the system with the two state variables (membrane potential v and recovery variable w) can be investigated. A comparison was also made between the phase space of the original signal and the phase space of the reduced model signal.

The neural network model is adapted for the training dataset, therefore the training error is calculated for an already sifted dataset. The validation dataset is only used to compare different models and select the best one. The validation error estimates how well the model will generalise with a new dataset.

The computational cost of generating one time step is calculated, for the original model, by subtracting the time instants between the beginning and the end of the generation of the dataset for a simulation and dividing this time difference by the number of steps in a simulation.

For the reduced model, it is instead calculated by subtracting the time instants between the start

and end of the prediction and then dividing this time difference by the number of steps in a simulation.

The RMSE is the square root of the mean squared error between the predicted and actual values:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}} \tag{35}$$

where $\hat{y}_i$ are predicted values, y_i are actual values and n is the number of observations. An advantage of the RMSE is that the metric it generates is on the same scale as the predicted unit.
The ARMSE is the average value of the RMSE calculated for all simulations.

3 Results

3.1 Evaluation of the models

3.1.1 Two-dimensional non-linear dynamical system model

The results of the 2D non-linear dynamical system model were compared with those obtained by B. Lusch et al. in [3] and are presented in Table 15. Please note that they did not calculate the computational cost per time step and the ARMSE.
The training and validation errors obtained in this thesis were larger than in the original work, but of the same order of magnitude (10^{-7}). The computational cost per step was 4.1 ms and the ARMSE was 0.2, indicating that there is a good fit between the original and the reduced model.

The average log_{10} prediction error with increasing number of prediction steps also had a wider range ([6.7, 5.3]) than that in the original work ([7.0, 5.7]), although the shape was similar. It is shown in Figures 3.1 (a), from [3] and in 3.1 (b), from this work.

Metric	**Paper [3]**	**This thesis**
Training error	$1.43 \cdot 10^{-7}$	$2.51 \cdot 10^{-7}$
Validation error	$1.43 \cdot 10^{-7}$	$2.84 \cdot 10^{-7}$
Prediction error ranges	[7.0, 5.7]	[6.7, 5.3]
Computational cost per step (ms)	-	4.1
ARMSE	-	0.2

Table 15: Comparison between the results of the 2D non-linear dynamical system model obtained in the work [3] and those obtained in this thesis.

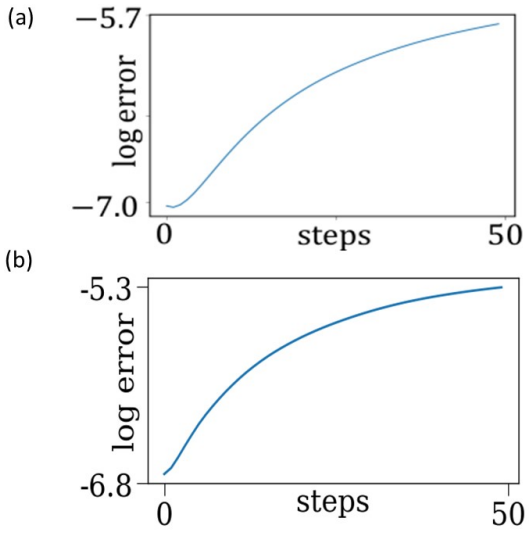

Figure 3.1: Average log_{10} prediction error as the number of prediction steps increases. (a) Obtained in the paper, modified from [3, p. 12]. (b) Obtained in this thesis.

In order to make some assumptions about the linearity of the system, it is possible to check whether the eigenvalues are constant when they can vary with respect to their eigenfunctions. Therefore, a representation of the eigenvalues of the system from [3] (Figure 3.2 (a)), where the eigenvalues are allowed to vary with the auxiliary network used for the continuous spectrum, is compared with that obtained in this thesis (Figure 3.2 (b)). In both studies, the eigenvalues remain relatively constant as the ranges are close to the true values of $\mu = -0.05$ and $\lambda = -1$ (see equation (27)).

It was also interesting to see how the reduced model performed in predicting longer sequences, given only the initial conditions. To understand this, see Figure 3.3 for an example of a model simulation prediction. The original signal, x_1 and x_2 over time, is compared with the long prediction made with the reduced model.

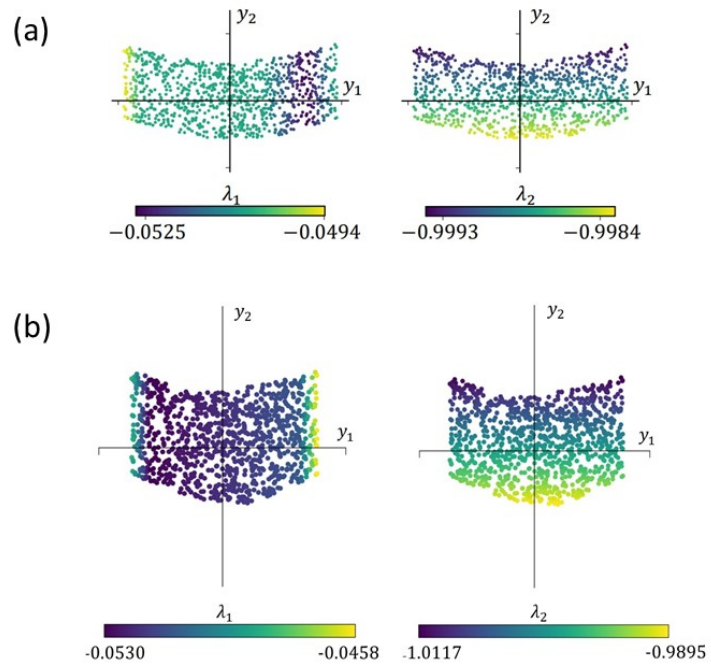

Figure 3.2: Eigenvalues of the 2D non-linear dynamical system dataset when they are allowed to vary in terms of the eigenfunctions (y_1, y_2). (a) Obtained in the paper, modified from [3, p. 12]. (b) Obtained in this thesis.

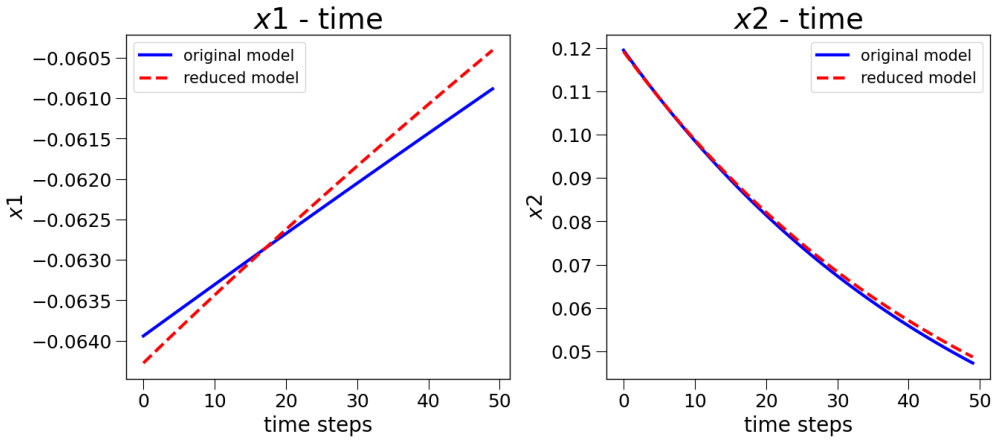

Figure 3.3: Comparison between the original model and the long prediction of the reduced model, $x1$ and $x2$ over time, for a simulation of the 2D non-linear dynamical system model. The solid blue line stands for the original signal and the red dashed line for the reduced model signal.

3.1.2 Fitz-Hugh-Nagumo model

A similar analysis was carried out for the FHN model as for the 2D non-linear model. It was evaluated in terms of training error, validation error, computational cost for predicting one step and ARMSE for all simulations between the original signals and the reduced model signals.

An exemplary simulation of the dataset of the FHN model is shown in Figure 3.4 (a). It shows the states v and w over time (with $v_0 = 1.2$, $w_0 = -0.625$, $I = 1.1$), the membrane voltage V over time and the input to the system, the stimulus current I over time. Figure 3.4 (b) shows the response of the system to a given threshold I: v over time without response ($I = 0.5$) is shown above, v over time with response ($I = 1.1$) is shown below.

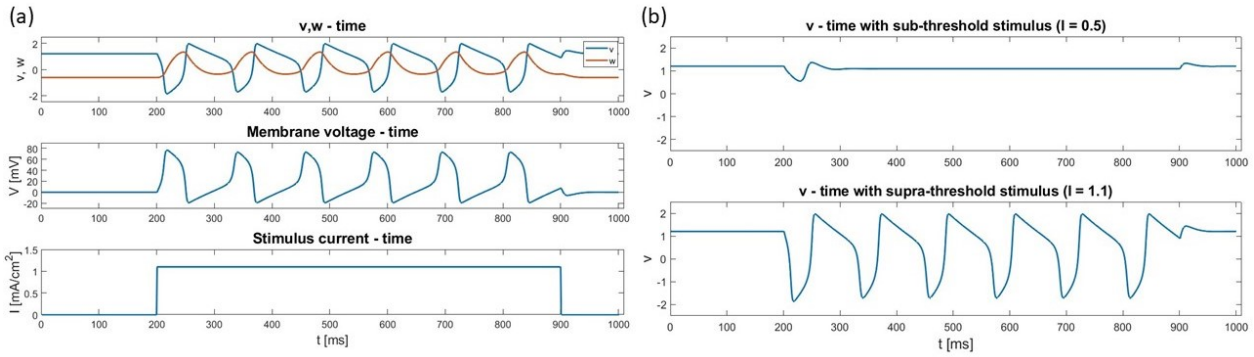

Figure 3.4: (a) Output of the system: dynamics of v and w over time with $v_0 = 1.2$, $w_0 = -0.625$, $I = 1.1$. Observable: membrane voltage V over time, following the equation (29). Input of the system: step stimulus current I over time. (b) v over time with no response ($I = 0.5$ under the threshold) and v over time with response ($I = 1.1$ over the threshold).

A random search of hyperparameters was performed to find the best architecture leading to the greatest prediction accuracy in terms of training and validation errors and ARMSE.
Selected hyperparameters were varied at a time and the performances of each model architecture (48 in total) were analysed. These results are summarised in Table 16. The tuned hyperparameters, listed in Table 10 with their ranges, were w (number of hidden units for the autoencoder), d (number of layers for the encoder and decoder) and w0 (number of hidden units for the auxiliary network) for the network architecture and bs (batch size) for the training settings. Figure 3.5 shows the training and validation errors as well as the computational costs for the prediction of a time step and the ARMSE for all combinations of the tuned hyperparameters listed in Table 16.

	Hyperparameters				Results			
Co.	w	d	$w0$	bs	**Training error**	**Validation error**	**Comp. cost per step (ms)**	**ARMSE**
1	64	2	32	32	$1.03 \cdot 10^{-3}$	$1.65 \cdot 10^{-3}$	1.4	15.3
2	64	3	32	32	$2.88 \cdot 10^{-4}$	$4.14 \cdot 10^{-4}$	1.8	14.5
3	64	4	32	32	$3.27 \cdot 10^{-4}$	$2.61 \cdot 10^{-4}$	2.2	14.7
4	64	2	32	64	$3.99 \cdot 10^{-3}$	$3.54 \cdot 10^{-3}$	1.3	20.2
5	64	3	32	64	$9.43 \cdot 10^{-4}$	$5.63 \cdot 10^{-4}$	1.6	15.5
6	64	4	32	64	$5.50 \cdot 10^{-4}$	$4.43 \cdot 10^{-4}$	1.9	14.8
7	64	2	32	128	$7.14 \cdot 10^{-3}$	$6.56 \cdot 10^{-3}$	1.4	27.1
8	64	3	32	128	$1.90 \cdot 10^{-3}$	$1.51 \cdot 10^{-3}$	2.3	16.9
9	64	4	32	128	$6.72 \cdot 10^{-4}$	$7.18 \cdot 10^{-4}$	3.1	15.0
10	64	2	32	256	$7.42 \cdot 10^{-3}$	$7.85 \cdot 10^{-3}$	3.0	27.8
11	64	3	32	256	$4.41 \cdot 10^{-3}$	$4.42 \cdot 10^{-3}$	1.5	22.9
12	64	4	32	256	$2.44 \cdot 10^{-3}$	$2.88 \cdot 10^{-3}$	4.3	18.9
13	128	2	64	32	$2.07 \cdot 10^{-3}$	$3.19 \cdot 10^{-3}$	2.9	19.1
14	**128**	**3**	**64**	**32**	$\mathbf{8.79} \cdot 10^{-5}$	$\mathbf{1.11} \cdot 10^{-4}$	**3.9**	**14.1**
15	128	4	64	32	$2.12 \cdot 10^{-4}$	$3.34 \cdot 10^{-4}$	4.9	14.8
16	128	2	64	64	$3.22 \cdot 10^{-3}$	$3.97 \cdot 10^{-3}$	2.7	21.3
17	128	3	64	64	$1.65 \cdot 10^{-4}$	$1.77 \cdot 10^{-4}$	3.1	14.4
18	128	4	64	64	$3.26 \cdot 10^{-4}$	$3.54 \cdot 10^{-4}$	4.6	14.8
19	128	2	64	128	$3.16 \cdot 10^{-3}$	$3.67 \cdot 10^{-3}$	2.7	20.0
20	128	3	64	128	$6.27 \cdot 10^{-4}$	$6.48 \cdot 10^{-4}$	5.5	15.0
21	128	4	64	128	$4.37 \cdot 10^{-4}$	$4.75 \cdot 10^{-4}$	7.5	14.5
22	128	2	64	256	$7.54 \cdot 10^{-3}$	$7.87 \cdot 10^{-3}$	2.8	27.9
23	128	3	64	256	$2.59 \cdot 10^{-3}$	$2.65 \cdot 10^{-3}$	2.9	19.2
24	128	4	64	256	$1.03 \cdot 10^{-3}$	$9.54 \cdot 10^{-4}$	3.5	15.2
25	256	2	128	32	$4.76 \cdot 10^{-4}$	$5.79 \cdot 10^{-4}$	8.2	14.6
26	256	3	128	32	$2.72 \cdot 10^{-4}$	$3.32 \cdot 10^{-4}$	27.3	14.6
27	256	4	128	32	$3.85 \cdot 10^{-4}$	$4.77 \cdot 10^{-4}$	38.1	15.3
28	256	2	128	64	$1.57 \cdot 10^{-3}$	$1.67 \cdot 10^{-3}$	12.8	16.6
29	256	3	128	64	$9.34 \cdot 10^{-3}$	$9.35 \cdot 10^{-3}$	13.8	30.4
30	**256**	**4**	**128**	**64**	$\mathbf{4.54} \cdot 10^{-4}$	$\mathbf{5.12} \cdot 10^{-4}$	**19.4**	**13.4**

Continued on next page

Table 16 – *Continued from previous page*

31	256	2	128	128	$2.72 \cdot 10^{-3}$	$2.58 \cdot 10^{-3}$	13.4	19.7
32	256	3	128	128	$7.54 \cdot 10^{-3}$	$7.87 \cdot 10^{-3}$	5.1	27.9
33	256	4	128	128	$4.34 \cdot 10^{-4}$	$3.46 \cdot 10^{-4}$	19.4	14.2
34	256	2	128	256	$8.44 \cdot 10^{-3}$	$8.54 \cdot 10^{-3}$	9.3	29.8
35	256	3	128	256	$6.56 \cdot 10^{-3}$	$6.58 \cdot 10^{-3}$	15.2	26.8
36	256	4	128	256	$8.06 \cdot 10^{-4}$	$8.11 \cdot 10^{-4}$	32.8	15.4
37	512	2	256	32	$1.03 \cdot 10^{-2}$	$1.05 \cdot 10^{-2}$	21.8	31.8
38	512	3	256	32	$1.53 \cdot 10^{-3}$	$1.92 \cdot 10^{-3}$	39.5	16.8
39	512	4	256	32	$3.95 \cdot 10^{-3}$	$4.11 \cdot 10^{-3}$	48.9	23.0
40	512	2	256	64	$8.19 \cdot 10^{-3}$	$8.43 \cdot 10^{-3}$	30.7	29.7
41	512	3	256	64	$1.53 \cdot 10^{-3}$	$1.55 \cdot 10^{-3}$	42.8	15.8
42	512	4	256	64	$7.54 \cdot 10^{-3}$	$7.87 \cdot 10^{-3}$	4.7	27.9
43	512	2	256	128	$4.63 \cdot 10^{-3}$	$4.87 \cdot 10^{-3}$	38.2	20.2
44	512	3	256	128	$1.06 \cdot 10^{-2}$	$1.05 \cdot 10^{-2}$	45.4	31.7
45	512	4	256	128	$1.24 \cdot 10^{-2}$	$1.23 \cdot 10^{-2}$	48.3	32.8
46	512	2	256	256	$1.07 \cdot 10^{-2}$	$1.06 \cdot 10^{-2}$	28.7	31.9
47	512	3	256	256	$1.10 \cdot 10^{-2}$	$1.09 \cdot 10^{-2}$	39.5	32.0
48	512	4	256	256	$1.03 \cdot 10^{-2}$	$1.01 \cdot 10^{-2}$	86.0	31.2

Table 16: Combinations of hyperparameters tuned for training the FHN model and results. The hyperparameter values leading to the best prediction accuracy are highlighted in bold. Co., combination; w, number of hidden units for the autoencoder; d, number of layers for the encoder and decoder; $w0$, number of hidden units for the auxiliary network; bs, batch size; ARMSE, average root mean squared error. For a description of the tuned hyperparameters, see Tables 5 and 6.

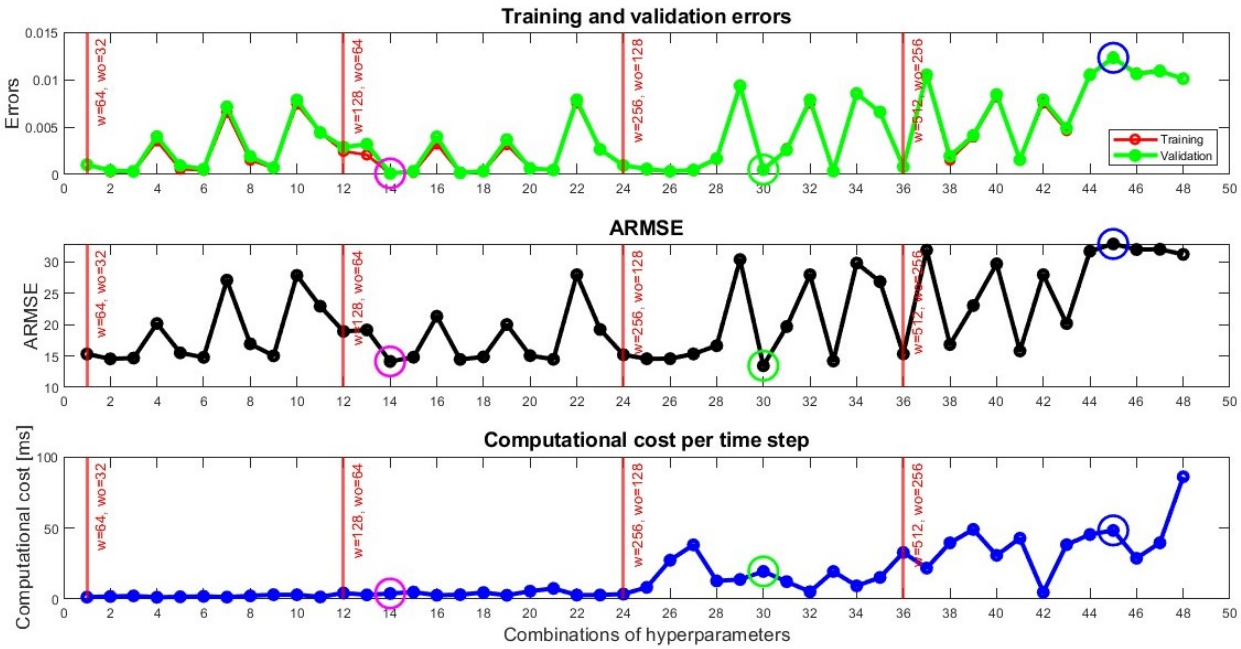

Figure 3.5: Plots of training and validation errors, ARMSE (average root mean squared error) and computational cost to predict one time step for all the combinations of hyperparameters listed in Table 16. The vertical lines in red indicate the sets of 12 combinations of hyperparameters, with the same number of hidden units, for the autoencoder (w) and the auxiliary network ($w0$). The magenta circle markers identify the best combination of hyperparameters in terms of training and validation errors, the green circle markers identify the best combination of hyperparameters in terms of ARMSE and the blue circle markers identify the worst combination of hyperparameters, from Table 16.

As with the previous model, the performance of this reduced model was examined in predicting longer sequences, given only the initial conditions. An example of a long prediction can be found in Figure 3.6, which shows the comparison between the original signal and the long prediction using the reduced model, for the evolution of v and w over time, for a simulation. It corresponds to the best set of hyperparameters (Table 16): $w = 128, d = 3, w0 = 64, bs = 32$. It led to the following results for the training and validation errors, computational cost and ARMSE: $8.79 \cdot 10^{-5}$, $1.11 \cdot 10^{-4}$, 3.9 ms, 14.1, respectively.

Another example of a long prediction is shown in Figure 3.7, it corresponds to the worst set of hyperparameters (Table 16): $w = 512, d = 4, w0 = 256, bs = 128$. It led to the following results for the training and validation errors, computational cost and ARMSE: $1.24 \cdot 10^{-2}$, $1.23 \cdot 10^{-2}$, 48.3 ms, 32.8, respectively.

The computational cost to generate one time step was also calculated, during the generation of the dataset, resulting in 0.7 ms, about 6 times faster than reduced model, considering the best set of hyperparameters (Table 16).

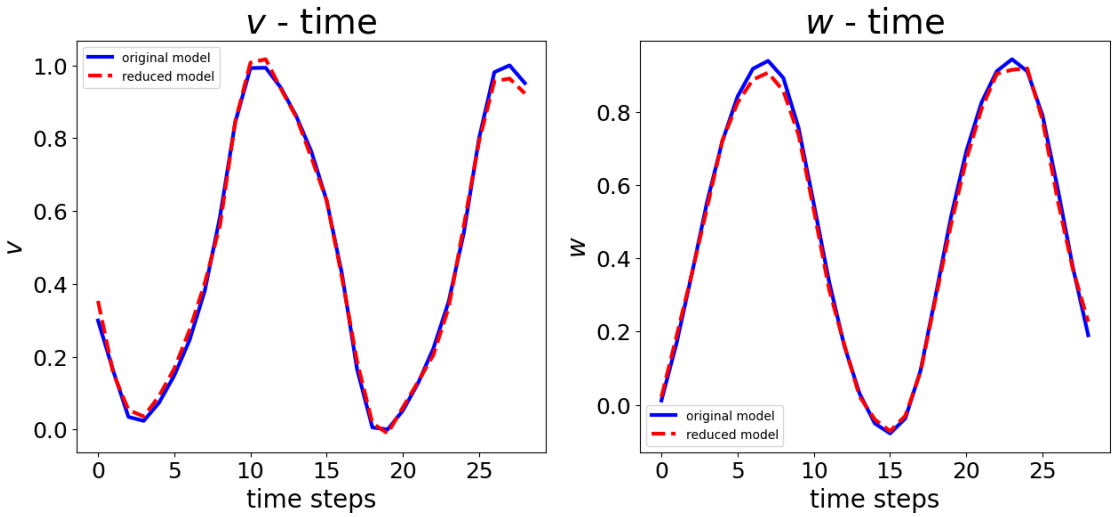

Figure 3.6: Comparison between the original model and the reduced model, v and w over time, for a simulation corresponding to the first best combination of hyperparameters from Table 16 ($w = 128, w0 = 64, d = 3, bs = 32$). The solid blue line stands for the original signal and the dashed red line for the reduced model signal.

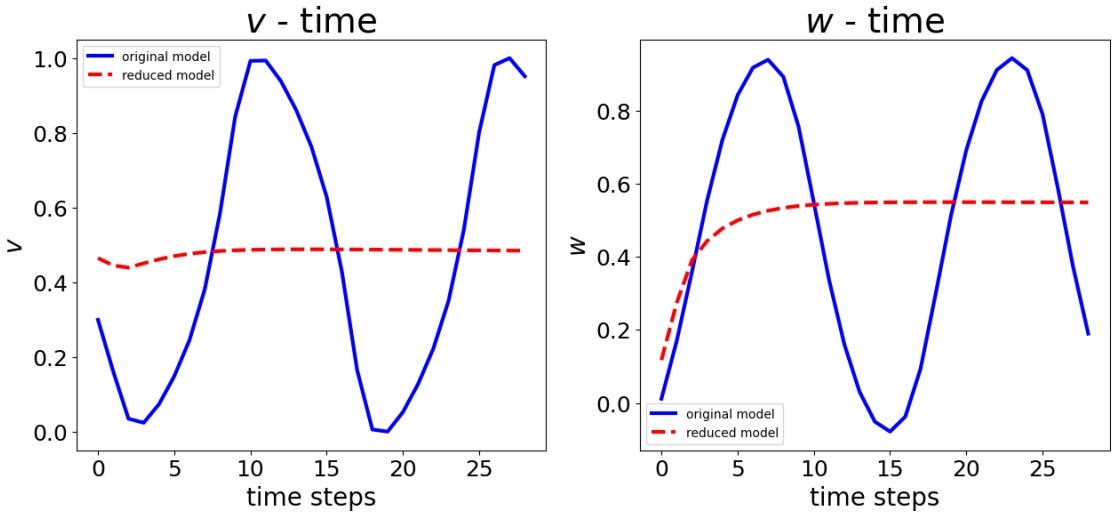

Figure 3.7: Comparison between the original model and the reduced model, v and w over time, for a simulation corresponding to the worst combination of hyperparameters from Table 16 ($w = 512, w0 = 256, d = 4, bs = 128$). The solid blue line stands for the original signal and the dashed red line for the reduced model signal.

The phase space allows to study the dynamics of the FHN model with two state variables: the membrane potential v and the recovery variable w. The phase space of the original signal and the reduced model signals, for the 14th parameter combination, were compared in Figure 3.8, where it can be seen that the two signals overlap quite well.

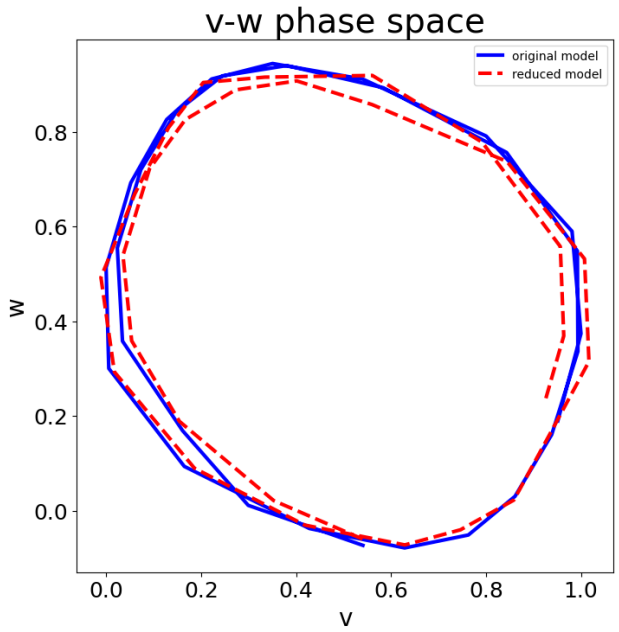

Figure 3.8: Phase space of the FHN model: comparison between the original model and the reduced model for the first best combination of hyperparameters from Table 16 ($w = 128, w0 = 64, d = 3, bs = 32$). The solid blue line stands for the original signal and the dashed red line for the reduced model signal.

The average log_{10} prediction error with increasing number of prediction steps was calculated for 29 time steps, for the combination of hyperparameters 14 (Table 16). It had the range [-1.87, -0.81] and the shape is shown in Figure 3.9.

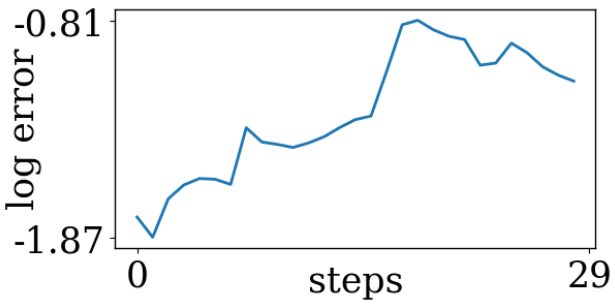

Figure 3.9: Average log_{10} prediction error as the number of prediction steps increases for the FHN model.

The eigenvalues for this model were also checked to make some assumptions about the linearity of the system. The two eigenvalues, μ_1 and μ_2, for all simulations of the FHN model are shown in Figure 3.10.

They refer to the first best combination of hyperparameters ($w = 128, w0 = 64, d = 3, bs = 32$, from

Table 16). They are both relatively constant, with the exception of simulation 336, as the dataset of the original model for this simulation has only 0 values.

The eigenvalues (μ_1, μ_2) were 1.050 ± 0.017 and 0.589 ± 0.002, respectively, indicating linearity of the reduced model.

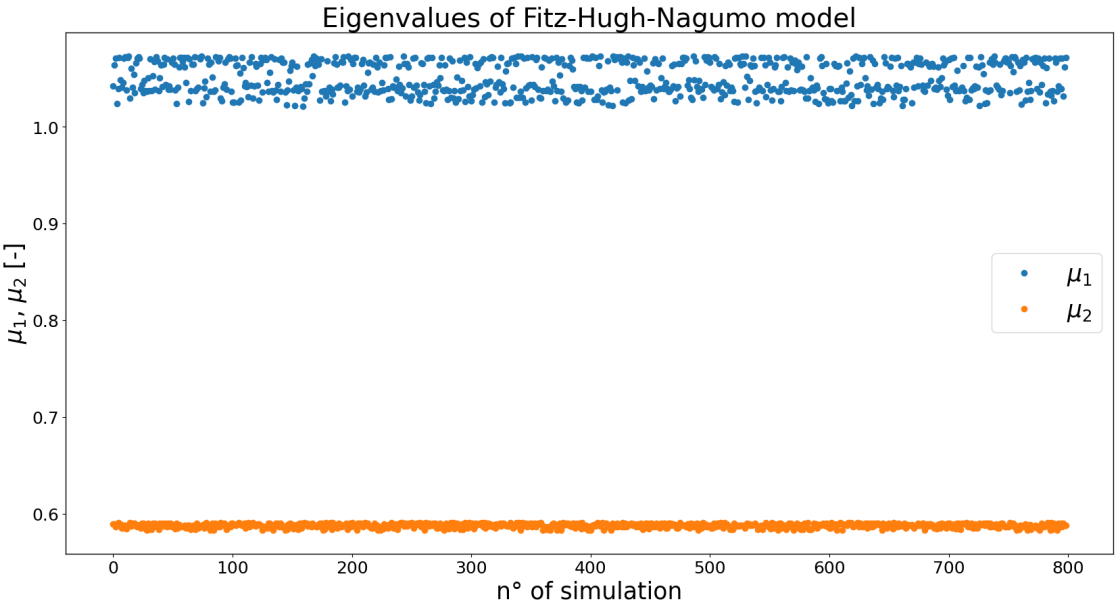

Figure 3.10: The two eigenvalues, μ_1 and μ_1, for all simulations of the FHN model. They refer to the first best combination of hyperparameters ($w = 128, w0 = 64, d = 3, bs = 32$, Table 16).

It was interesting to understand if some model parameters have a greater influence on the output (the frequency of v) than the others, because then it would make sense to re-sample these parameters with more values, respectively. Therefore, a global sensitivity anaylsis was performed to assess the influence of the model parameters $(a, b, c, \beta, v_0, w_0)$ on the spiking frequency of the FHN model.

The sensitvity analysis revealed, that except for c, none of the parameters had a substantial impact on the model output (together accounting for < 50 % of the observed variability in the output). On the other hand c explained more than 50 % of the variability in the model output, hence, for future work it might be of interest to include more values for c for the generation of the training dataset in anticipation to even further improve the model prediction accuracy.

3.1.3 Cardiovascular system model

For the cardiovascular system model, a similar analysis was performed as for the previous two models. It was evaluated in terms of training and validation errors, computational cost for predicting one time step and ARMSE for all simulations between the original signals and the reduced model signals.
The computational cost to generate one time step was also calculated during the generation of the dataset, with Simulink, resulting in 7.9 ms.

It was interesting to look at the prediction performance of the reduced model, so some example results are shown. For training the model, different combinations of inputs and outputs (Table 14) of the haemodynamic signals, such as heart rate (HR), mean arterial pressure (MAP), current amplitude (C), charge (Q), pulse width (PW) and frequency of stimulation (F), were examined to check the differences in performance. For the combination of inputs where MAP was also present, two values for the ARMSE were calculated, one for HR and the other for MAP, and then the mean between these two values was calculated. The results are shown in Table 17.

The first combination of inputs (HR, C) showed the best performance in terms of training and validation errors ($1.94 \cdot 10^{-5}$ and $4.98 \cdot 10^{-5}$, respectively) and computational cost for predicting one time step (1.5 ms, about 5 times slower than the original model to generate a simulation for one time step). The ARMSE was 7.3.
The worst performance was obtained with four inputs (HR, C, PW, F) with the larger training and validation errors, ARMSE and computational cost to predict one time step ($1.47 \cdot 10^{-3}$, $1.24 \cdot 10^{-3}$, 8.3 and 3.1 ms, respectively). The reduced model was 2.5 times slower at predicting a time step, with this combination of inputs, compared to the original model.

Figures 3.11 and 3.12 show the comparison of HR or MAP (for a simulation with response) between the original signals and the reduced model signals over time for all the combinations of inputs investigated (Table 14).
Figures 3.13 shows the comparison of HR and MAP (for a simulation without response) between the original signals and the reduced model signals over time for the last combination of inputs (Table 14).

Co.	Input	Output	Training error	Validation error	Comp. cost per step (ms)	ARMSE
1	**HR, C**	**HR**	$\mathbf{1.94} \cdot 10^{-5}$	$\mathbf{4.98} \cdot 10^{-5}$	**1.5**	**7.3**
2	HR, C, PW, F	HR	$1.47 \cdot 10^{-3}$	$1.24 \cdot 10^{-3}$	3.1	8.3
3	HR, Q	HR	$2.96 \cdot 10^{-4}$	$3.73 \cdot 10^{-4}$	1.7	7.2
4	HR, Q, F	HR	$1.13 \cdot 10^{-3}$	$1.06 \cdot 10^{-3}$	2.1	8.0
5	HR, Q, MAP	HR, MAP	$3.22 \cdot 10^{-4}$	$4.84 \cdot 10^{-4}$	2.1	6.7
6	HR, Q, F, MAP	HR, MAP	$8.08 \cdot 10^{-4}$	$1.21 \cdot 10^{-3}$	2.4	6.5

Table 17: Results of the cardiovascular system model for the different input combinations. The input combination leading to the best prediction accuracy is highlighted in bold. Co., combination of inputs; ARMSE, average root mean squared error.

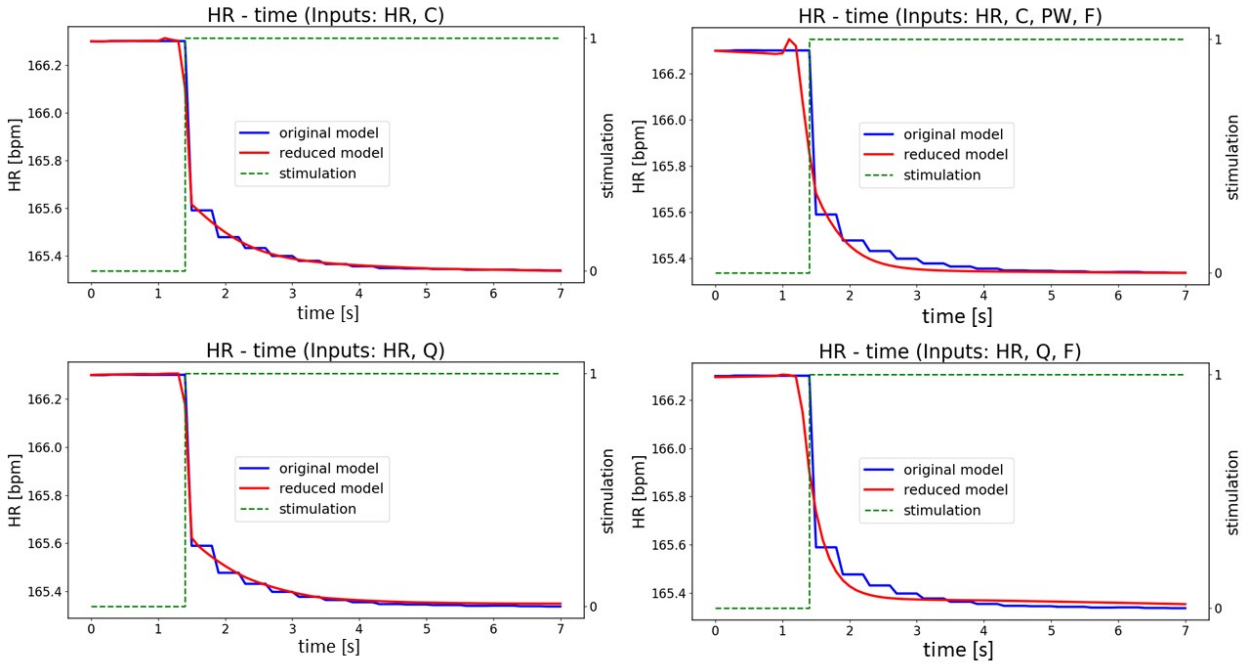

Figure 3.11: Simulations with response. Heart rate (HR) over time for the first 4 combinations of inputs (Table 17): comparison between original signals (blue line) and reduced model signals (red line), with the stimulation signal (green dashed line). C, current amplitude; Q, charge; PW, pulse width; F, frequency of stimulation.

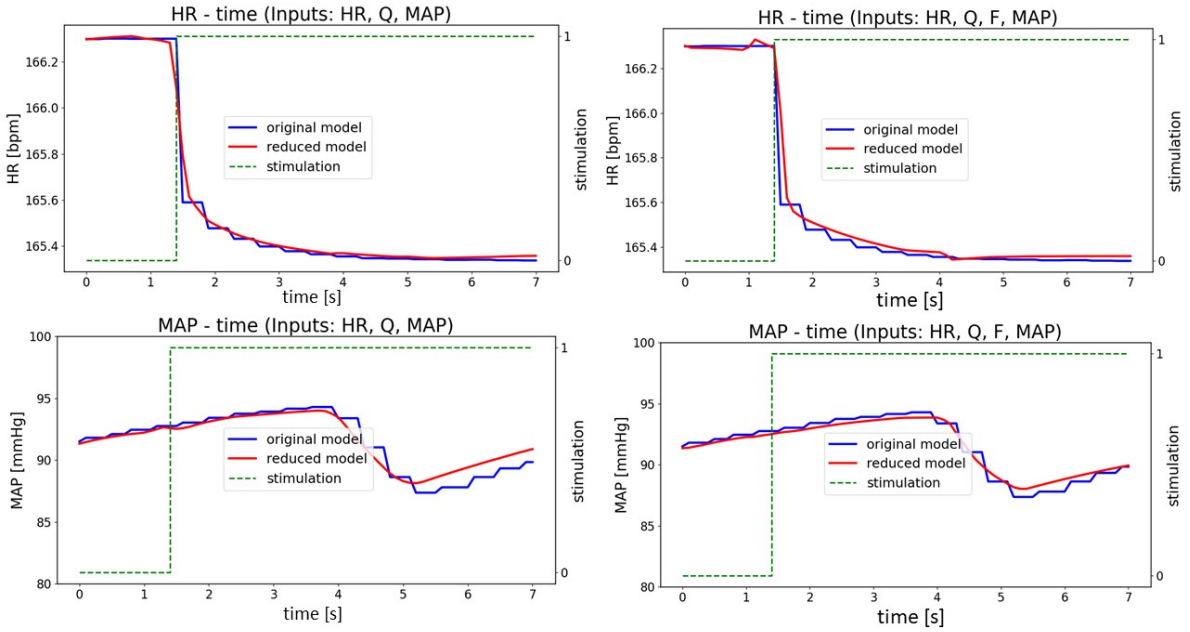

Figure 3.12: Simulations with response. Heart rate (HR) and mean arterial pressure (MAP) over time for the last 2 combinations of inputs (Table 17): comparison between original signals (blue line) and reduced model signals (red line), with the stimulation signal (green dashed line). C, current amplitude; Q, charge; PW, pulse width; F, frequency of stimulation.

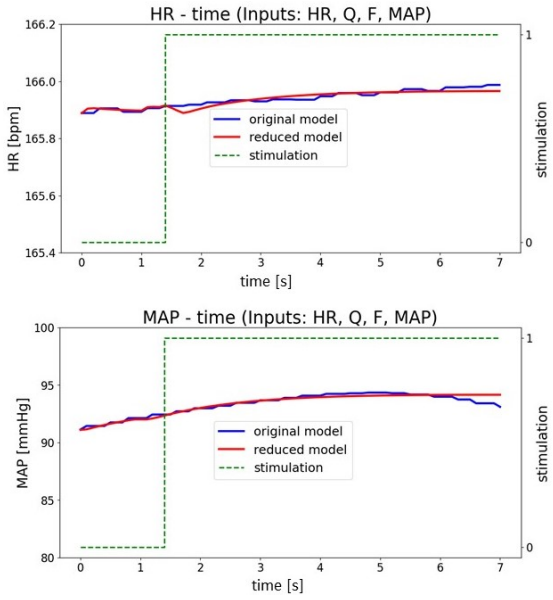

Figure 3.13: Simulations without response. Heart rate (HR) and mean arterial pressure (MAP) over time for the last combination of inputs (Table 17): comparison between original signals (blue line) and reduced model signals (red line), with the stimulation signal (green dashed line). C, current amplitude; Q, charge; PW, pulse width; F, frequency of stimulation.

Figure 3.14 shows the results for all the combinations of the inputs listed in Table 17 in terms of training and validation errors and computational cost to predict one time step and ARMSE.

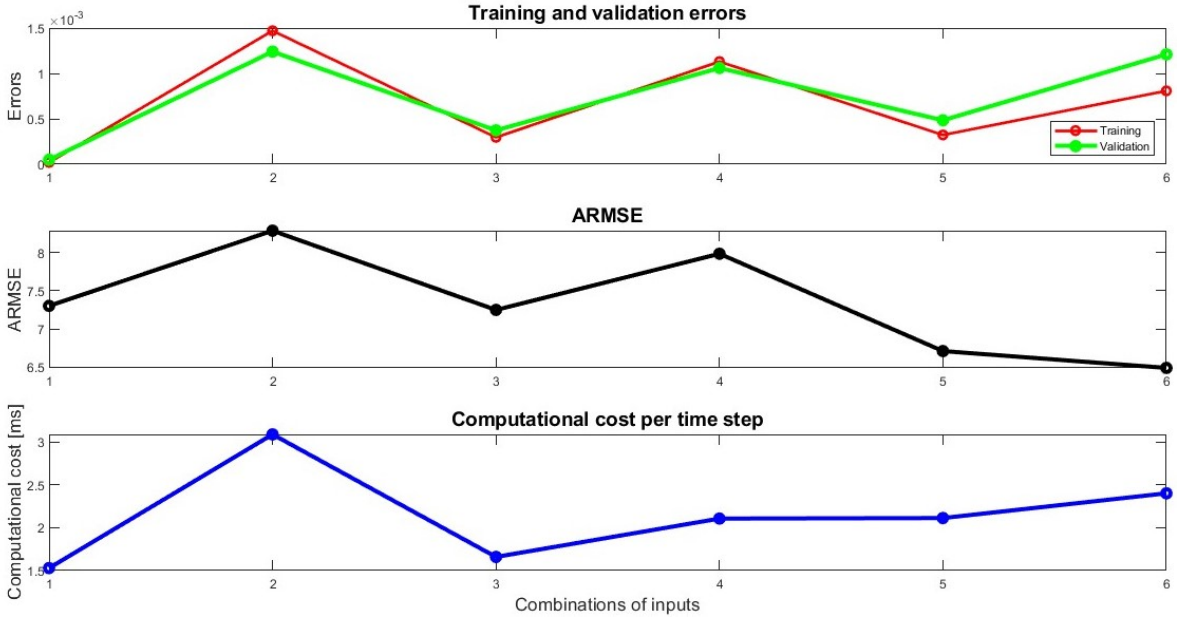

Figure 3.14: Plots of training and validation errors, ARMSE (average root mean squared error) and computational cost to predict one time step for all the combinations of inputs for the cardiovascular system model listed in Table 17.

The average log_{10} prediction error with increasing number of prediction steps was calculated for 70 time steps, for the first combination of inputs (Table 17), resulting in a range of [-1.46, -0.32]. Its shape is shown in Figure 3.15.

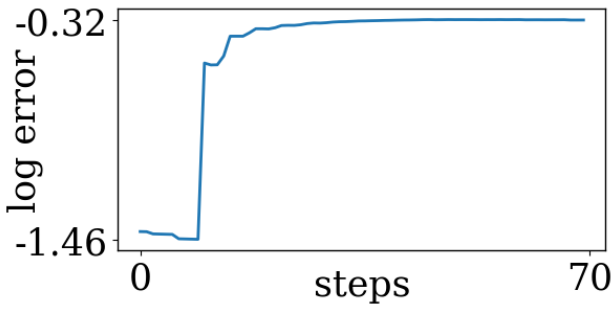

Figure 3.15: Average log_{10} prediction error as the number of prediction steps increases for the cardiovascular system model.

The dimension of the reduced linearised model equals to the number of state signals used in each training run, since the number of eigenvalues is equal to the number of inputs to the system. The eigenvalues of a two-dimensional model, μ_1 and μ_2, for all simulations of the cardiovascular system model, are shown in Figure 3.16. They refer to the first combination which has HR and C as inputs (Table 17). They are relatively constant and the step at simulation 705 divides the plot into two parts: the eigenvalues of the simulations with HR response (1-704) and those without HR response (705-1407). The eigenvalues were on average $\pm$ standard deviation 0.093 ± 0.001 (simulations with response) and 0.078 ± 0.000 (simulations without response) for μ_1 and 1.583 ± 0.002 (simulations with response) and 0.861 ± 0.005 (simulations without response) for μ_2.

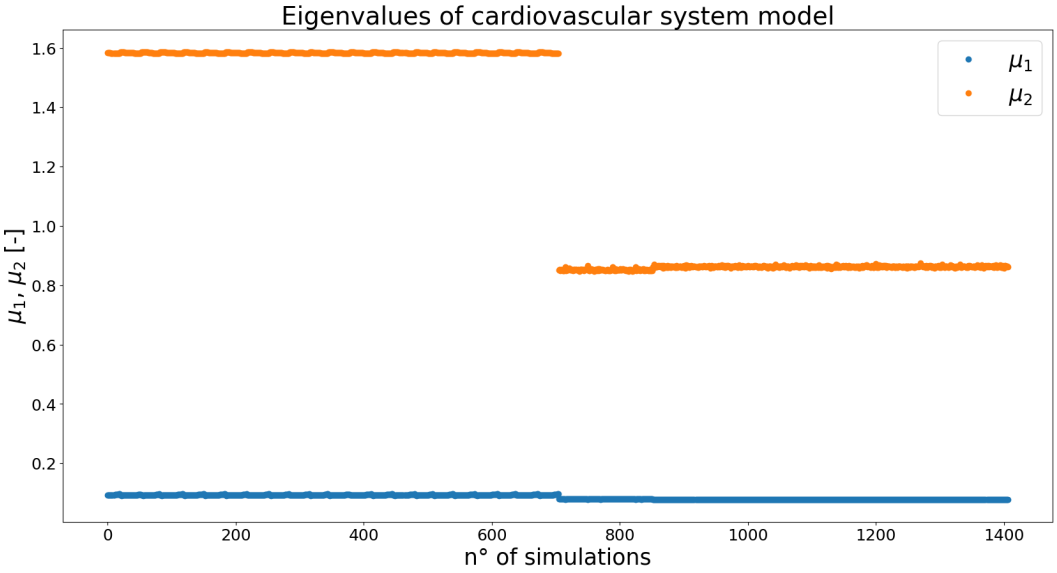

Figure 3.16: The two eigenvalues, μ_1 and μ_2, for all simulations of the cardiovascular system model. They refer to the first combination of inputs (HR and C, Table 17).

4 Discussion

Reducing the order of a model is important since non-linear dynamical systems usually have a large number of degrees of freedom, making them hard to analyse. Therefore, it is important to simplify these models to reduce their complexity so that they are not only easier to interpret, but also reduce the computational costs required to solve them.

Moreover, dealing with linear models is easier because we have a huge collection of optimal control techniques at our disposal for linear models. Therefore, it is also convenient to linearise a complex non-linear model, while maintaining the original nature of the model. Specifically, in the biomedical field, the ability to solve the non-linear equations of the human cardiovascular system would contribute significantly to a better understanding of the physiology of the model [73].

Model order reduction and linearisation are useful for model predictive control, as described by Choroszucha et al. in [74], where the balanced truncation based on empirical Gram matrix is investigated for reducing the nonlinear model order of a diesel airpath model to facilitate the design and implementation of nonlinear model predictive control.

In [75], Peitz and Klus presented a new approach for solving both open- and closed-loop control problems using reduced-order Koopman operator-based models. This method enabled them to control infinite-dimensional non-linear systems with finite-dimensional linear models.

In recent years, machine learning and big data strategies have emerged to reduce the order of complex models, as Shen et al. did in [76]. In particular, they proposed a new framework for reducing non-linear models that combines classical reduced deformable simulation with data-driven approaches using deep learning.

The overall goal of this thesis was to apply a deep neural network-based Koopman analysis technique to a model of the cardiovascular system. Since this is a very ambitious aim, it was done starting from easier problems first.

In the beginning, the results for a well-known toy model from the literature [3], a 2D model of a nonlinear dynamical system, were reproduced. The same technique was then applied to a more complex biomedical model created with MathWorks Simulink, the Fitz-Hugh-Nagumo model. Finally, it was also used for the more complex model of the cardiovascular system. However, at the time of writing this work is only at a preliminary stage.

For each model, further details about interpreting the results are given below.

4.1 Two-dimensional non-linear dynamical system model

The first analysis carried out was the training of the same non-linear dynamical system model (equations (27)), as used by B. Lusch et al. in [3]. The results of the training error and the validation error, obtained in this thesis, are of the same order of magnitude ($\sim 10^{-7}$) as their results [3].

Next, it was of particular interest to see the performance of the model in predicting longer time series, given only the initial conditions. 50 time steps into the future were predicted and the average log10 error was calculated as the number of prediction steps increased. Figures 3.1 (a) and (b) show the average log10 prediction error obtained in [3] and in this thesis, respectively. Both have comparable shapes and the ranges in which they vary do not differ significantly.

To ensure that a linearised model was obtained, the variations of the eigenvalues with the auxiliary network used for the continuous spectrum were examined, as was done in [3]. The results obtained in this thesis are slightly different from those obtained in their study. This could be explained by the random initialisation of the weights. Indeed, if the training is repeated several times, this can lead to different models and errors. However, as can be seen in Figure 3.2, the eigenvalues remain relatively constant in both studies.

To see how well the model can predict the system dynamics for multiple time steps in the future, given only the initial condition, a long prediction of 50 time steps was performed.
A comparison of the state signals x_1 and x_2 of the reduced model to the ones of the original model for an exemplary simulation is presented in Figure 3.3. One can see an almost perfect fit of the prediction of the reduced model for x_2 with the original data. Although there is a slightly worse prediction accuracy for state x_1, the overall error is still very low with an ARMSE of 0.2.
Overall, the reduced model was able to reproduce the original data with great accuracy.

4.2 Fitz-Hugh-Nagumo model

4.2.1 Prediction accuracy

The next step was to apply the deep neural network approach and Koopman analysis (DNN-KA) to a more complex problem that is also relevant in the field of biomedical engineering, a modified version of the Fitz-Hugh-Nagumo model from Rattay's book [2]. It is a model capable of reproducing the action potentials in the giant squid axon in shape and time.
In order to find the best model architecture that leads to the highest prediction accuracy while respecting the linearity condition, a random search for hyperparameters was performed as previously done by [3], [77], [78], [79].
The best model architecture, which leads to the best prediction accuracy in terms of training and validation errors and ARMSE, was researched among the several neural network architectures investigated (Table 16).
The hyperparameters included in the random search are listed in Table 10 with their ranges. The number of hidden units in the autoencoder (w) was chosen from values [30, 64, 128, 256, 512]; the number of layers (d) in the encoder and the decoder each from values [2,3,4]; the number of hidden units for the auxiliary network, used to construct the linearised reduced model, ($w0$) from values [10, 32, 64, 128, 256] and the batch size (bs) was chosen from values [32, 64, 128, 256].
The ranges were chosen based on experience from an initial manual exploration of model parameters and also to keep it computationally performant (the limit of the upper boundary). Therefore, for w and $w0$ (number of hidden units), 30 and 10 were the original values chosen by the authors of the paper [3], the other values are powers of two chosen so that w is always twice as large as $w0$.
High values for the hidden units and layers may affect the generalisability of the model, as the network architecture is more complex. Therefore, early stopping of the training was employed, ending the training after 1 hour if the performance did not further improve below a cost function value of 0.00008.

Figure 3.5 gives an overview of the errors for all combinations of hyperparameters examined (listed in Table 16). It can be seen that the training and validation errors are comparable for the most part, except for the following combinations of hyperparameters (from Table 16): 4, 5, 7, 8, 10, 12, 13, 16, 19, 38, 42 and 43. For these combinations of hyperparameters, the validation error is slightly larger

than the training error. We could explain this by the fact that the neural network has been trained with the training dataset and is, therefore, better able to predict future states with the same dataset.

The trends of training and validation errors and ARMSE are similar in all the combinations of parameters investigated, as expected. In these trends, a peak is detectable in every three combinations analysed: between the 4th and the 22nd, and between the 34th and the 40th combinations (from Table 16). These represent the combinations of hyperparameters where the number of layers in the encoder and decoder (d) is equal to 2, which means that this width is not sufficient for the model.

Moreover, the following sets of hyperparameters, including number of hidden units of the autoencoder $w = 512$, and auxiliary network $w0 = 256$, number of layers in encoder and autoencoder $d = 2, 3, 4$, batch size $bs = 128$, 256 (the last 5 combinations in Figure 3.5), result to have the highest training and validation errors ($\sim 10^{-2}$) and ARMSE (> 31.5). They share large values for $w, w0$ and bs, so in this case, the number of hidden units and the values for the batch size might be too high. However, since the explored combinations of hyperparameter values were restricted by computational time, we cannot exclude that along with other hyperparameter value combinations the results are different.
The choice of the batch size often requires some experimentation to find the optimal value and it can have a significant impact on the performance of the training in several ways. A larger batch size means that the model processes more samples in each iteration, which can result in faster training, but it requires more memory to store the samples during the training, so it may exceed the available memory on the GPU. Moreover, large values of batch size can result in a worse generalisation performance of the model because it is exposed to less diverse samples during training. A larger batch size can also result in the model overfitting the training data, particularly if the dataset is small.

The best combination of hyperparameters is $w = 128, w0 = 64, d = 3, bs = 32$ (Figure 3.5 highlighted in magenta), considering the training and validation errors, which are $8.79 \cdot 10^{-5}, 1.11 \cdot 10^{-4}$ respectively, the lowest among all possible combinations examined. The ARMSE was 14.1. For this combination, it can be seen in Figure 3.6 that the original and long prediction signals of the reduced model overlap for most time steps.

Instead, considering the ARMSE, the best combination of hyperparameters is $w = 256$, $w0 = 128$, $d = 4$, $bs = 64$ (Figure 3.5 highlighted in green) with an ARMSE of 13.4. For this combination, the training and validation errors are $4.54 \cdot 10^{-4}, 5.12 \cdot 10^{-4}$, respectively.
This fact shows that there is a discrepancy between the prediction accuracy when quantified by ARMSE and the cost function that was used for training/test/validation. That means that the networks performing best in terms of the cost function and ARMSE are not the same. This might be explained by the fact that the cost function not only takes into account the prediction error but also the model linearity.
The ARMSE for the first best combination resulted to be 5 % more than that of the second best combination, but the training and validation errors were 5 times smaller.

The worst combination of hyperparameters resulted to have $w = 512, w0 = 25, d = 4, bs = 128$. The training and validation errors for this combination were the highest among all the combinations examined ($1.24 \cdot 10^{-2}, 1.23 \cdot 10^{-2}$, respectively), and the ARMSE was 32.8. In addition, the reduced model signals for this combination do not overlap at all with the original signals (Figure 3.7). On the other hand, combinations 25-27 (from Table 16) had a small value of training, validation errors and ARMSE ($\sim 10^{-4}, \sim 10^{-4}, \sim 15.0$, respectively), they all share the same hyperparameters, but the number of layers in the encoder and decoder varies between 2 to 4.

4.2.2 Linearity

The eigenvalues of the model were checked to make some assumptions about the linearity of the system. Regarding the eigenvalues, μ_1 and μ_2, for all simulations of the Fitz-Hugh-Nagumo model (Figure 3.10), we can deduce that they are relatively constant.

4.2.3 Computational cost

It was desirable to improve the computational performance by model order reduction, therefore the computational cost was examined. The time needed to generate one time step simulation using the original model was compared to the time needed for one time step prediction using the reduced model.
The trend in computational cost per time step (Figure 3.5) can be clearly divided into two parts. 1 to 8 ms duration for the prediction of one time step refers to the first half of the trend (until the

24th combination from Table 16), in the second half a by no means constant behaviour with clear peaks and spikes downwards can be seen. It can be observed that with a larger number of hidden layers of the main and auxiliary networks ($w = 256, 512, w0 = 128, 256$), the prediction of a time step takes more time. In fact, the last combination of hyperparameters (the 48th from Table 16, with hyperparameters: $w = 512, w0 = 256, d = 4, bs = 256$) has the highest computational cost ($\sim$ 86 ms).

Larger layer sizes lead to increased computational time because they require more computations and calculations to be performed per unit. Moreover, larger layers can also require more memory to store the weights and activations of the units, which can increase memory usage and lead to slower processing times.

The computational time needed for generating one time step using the original model was 0.7 ms, instead for predicting one time step using the reduced model was 3.9 ms (6 times bigger). This result could be explained by the complexity of the neural model. Hence, it is possible to conclude that the linearisation of the model comes at a cost of computational efficacy.

Therefore, more work is needed to see if we can improve the computational performance while maintaining linearity, such as the inclusion of other hyperparameters in the random search (weights of the cost function for example).

4.2.4 Sensitivity analysis

It was interesting to understand whether some model parameters have a greater impact on the frequency of v (output of the model) than others. If so, it could be useful to re-sample these parameters more densely than the others to generate another dataset for re-training the model. According to the sensitivity analysis, c is the most important parameter as indicated by the Sobol's first-order effect index, which shows that c explains >50 % of the observed variability in the spiking frequency. The other parameters together accounted for less than 50 % of the variability in the output, hence can be considered to only have a minor impact.

However, the sensitivity analysis was performed at the end of the work to understand if there might be the possibility to further improve, the already excellent model prediction accuracy by a different sampling strategy for the generation of the training set. The results suggest, that in future it could be a viable approach to generate another dataset including more different values of c in anticipation of an improved prediction accuracy.

4.3 Cardiovascular system model

4.3.1 Prediction accuracy

Finally, after applying the DNN-KA to the FHN model, the next step was to investigate the approach for the reduction and linearisation of the complex cardiovascular system model, which was the ultimate goal of the thesis. As this model has plenty of states, first, only a small subset of them was considered (including just HR and MAP) and explored the ability of the DNN-KA to find linear embeddings in these signals. This part of the work is still at a very preliminary stage, however, the results are encouraging.
Since for the application of VNS to cardiovascular conditions, HR and MAP are the most important hemodynamic parameters these were chosen along with the most important main stimulation parameters of cardiac vagus nerve stimulation. These parameters were C, PW and F as a previously performed sensitivity analysis showed that they have the greatest influence on the cardiac response to VNS. To further reduce the parameters, C and PW were combined into charge (Q).

From the comparison of the different combinations of inputs to the system (from Table 14, results shown in Figure 3.14), the first one (HR, C) was the one associated with the smallest values of training and validation errors ($1.94 \cdot 10^{-5}, 4.98 \cdot 10^{-5}$ respectively), and the third one (HR, Q, F) was the one associated with the smallest value of ARMSE (7.2), so they can be defined as the best combinations.
Including MAP in the dataset, the best combination of inputs (HR, Q, MAP) led to the training and validation errors $3.22 \cdot 10^{-4}$ and $4.84 \cdot 10^{-4}$, respectively (Table 14).
Figure 3.14 shows that the trends of training and validation errors and ARMSE are similar until the fifth combination of inputs (from Table 14), then training and validation errors increase and ARMSE decreases.

Unexpectedly, some of the tested combinations (Table 14) led to higher training errors than validation errors. This could come from a bad selection of training and validation sets. Another possible explanation could be that the training set contained many 'difficult' cases that had to be learned compared to the cases that had to be predicted in the validation set. A cross-validation approach (repeated shuffling of the data set) and/or a larger dataset can be useful to adress this problem in

the future.

4.3.2 Linearity

The eigenvalues of the model were checked to make some assumptions about the linearity of the system. From simulation 705 (Figure 3.16), we can see a change in the average eigenvalue visually dividing the plot into two parts. Presumably, this is because the first half of the simulations (1-704) resulted in a heart rate response, while the second half (simulations 705-1407) did not.
Regarding the eigenvalues, μ_1 and μ_2, for all simulations of the cardiovascular system model, it is possible to conclude they are relatively constant for the simulations with response and the ones without response.

4.3.3 Computational cost

On average the computational cost to predict one time step using the reduced model was 2.3 ms. The first combination of inputs had the smallest value of computational cost (1.5 ms), while the second combination had the highest (3.1 ms). These results can be considered acceptable if the specifications of the hardware used (Table 3) are taken into account.
In the experiment, the best combination of inputs for the original model had a computational cost of 7.9 ms for generating a time step of the dataset, which is 5 times less than the computational cost of 1.5 ms for predicting a time step. This significant difference in computational cost indicates that the reduced model has achieved a substantial improvement in this aspect.

4.4 Limitations

The main limitation of this thesis was the limited exploration of hyperparameters in both the FHN and the cardiovascular system models, due to the vast constraints in computation time. For example, the FHN model lacked the tuning of the hyperparameters regarding the dimension of intermediate space and loss function settings.
The small dataset in both studies might have resulted in a limited generalisation. Although the similar training and validation errors do not indicate a lack of generalisation, additionally to the early stopping of the training, the use of a larger dataset should be considered.
However, the overall results are encouraging showing that the linearised model could reproduce the

dynamics of the original model with great accuracy.
For the model of the cardiovascular system, only a small subset of states was considered (including just HR and MAP), so the robustness of the approach to variations in the model parameters was not investigated. Further analysis with a more complete set of state signals is recommended.
Moreover, only a single neural network architecture was used for generation of the results in the thesis. Therefore, to further improve the prediction accuracy of the reduce model, a hyperparamter optimization, similar to the one applied for the FHN model should be performed in future work. Finally, also a wider range of input parameters and a larger number of samples is recommended to improve the results.

4.5 Outlook

This thesis has successfully applied deep-learning-based Koopman analysis to two models in the biomedical domain. The results are encouraging and further add to the growing body of evidence that demonstrates the viability of this approach for model order reduction.
The next step in this research is to further focus on the reduction and linearisation of the cardiovascular system model, which will eventually help in the development of physiological control strategies for biomedical devices such neuroprosthesis or left ventricular assistive devices, e.g. by enabling the use of control methods such as model predictive control or optimal control.
Therefore, in future work, a larger number of hemodynamic signals will be obtained from the lumped-parameter model comprising multiple heartbeats, and the DNN-based Koopman analysis approach will be used to identify linear embeddings in the hemodynamic signals.
Finally, the reduced order model will be validated not only in terms of prediction accuracy and evaluated regarding the computational performance, but also in terms of its ability to reproduce physiological phenomena such as the Frank-Starling mechanism. This work has the potential to enhance our understanding of the cardiovascular system and enable the development of novel control strategies that improve human health.

5 Conclusion

In conclusion, this thesis successfully demonstrated the potential of deep neural network approach to Koopman analysis (DNN-KA) for model order reduction and linearisation in the field of biomedical engineering. The results are promising, with encouraging prediction accuracy and evidence of the applicability of this approach for reducing the complexity of dynamical systems, including the cardiovascular system model.

The potential applications of this approach for the design of physiological control algorithms for vagus nerve stimulation and regulating cardiovascular system function are significant, with the possibility of improving clinical outcomes. The identification of linear embeddings for hemodynamic signals using DNN-KA is a novel and exciting development in this field.

Despite the promising results, further work is needed to optimise hyperparameters, expand the number of used hemodynamic signals, and evaluate the computational performance of the reduced-order model in reproducing physiological phenomena.

In summary, this thesis has contributed to the growing body of evidence supporting the usefulness of DNN-KA for model order reduction and linearisation in biomedical engineering. The findings of this study have the potential to make significant contributions to the development of physiological control algorithms and the design of novel treatment strategies in the future.

References

[1] Ramkissoon C. M., Güemes A., and Vehi J. Overview of therapeutic applications of non-invasive vagus nerve stimulation: a motivation for novel treatments for systemic lupus erythematosus. *Bioelectronic medicine*, 7(1), 2021.

[2] F. Rattay. Electrical nerve stimulation: Theory, experiments and applications. *Springer Wien*, 1990.

[3] Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1), 11 2018.

[4] Zhenghai Wang, D. Xiao, Fangxin Fang, R. Govindan, Christopher Pain, and Yike Guo. Model identification of reduced order fluid dynamics systems using deep learning. *International Journal for Numerical Methods in Fluids*, 86, 07 2017.

[5] Steven H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering*. Westview Press, 2000.

[6] Francisco J. Gonzalez and Maciej Balajewicz. Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems, 2018.

[7] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938, 2018.

[8] Anders Krogh. What are artificial neural networks? *Nature biotechnology*, 26:195–7, 03 2008.

[9] Magdi Zakaria, Mabrouka Amhamed Al-Shebany, and Shahenda Sarhan. Artificial neural network : A brief overview. *Journal of Engineering Research and Applications*, 4:07–12, 02 2014.

[10] Saul Dobilas. Lstm recurrent neural networks — how to teach a network to remember the past., 2022.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[12] Will Badr. Auto-encoder: What is it? and what is it used for? (part 1), 2019.

[13] Francisco Javier Gonzalez and Maciej Balajewicz. Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems. *ArXiv*, 2018.

[14] Branen Andrew. Data-driven modeling and control of cardiac system. *Theses and Dissertations Collection, University of Idaho Library*, 08 2021.

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[16] Maryam Farajzadeh-Zanjani, Ehsan Hallaji, Roozbeh Razavi-Far, and Mehrdad Saif. Generative adversarial dimensionality reduction for diagnosing faults and attacks in cyber-physical systems. *Neurocomputing*, 440:101–110, 2021.

[17] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, jan 2018.

[18] Enoch Yeung, Soumya Kundu, and Nathan O. Hodas. Learning deep neural network representations for koopman operators of nonlinear dynamical systems. *CoRR*, abs/1708.06850, 2017.

[19] Nishaal Parmar, Hazem H. Refai, and Thordur Runolfsson. A survey on the methods and results of data-driven koopman analysis in the visualization of dynamical systems. *IEEE Transactions on Big Data*, 8(3):723–738, 2022.

[20] Proctor Joshua L., Brunton Steven L., and Kutz J. Nathan. Dynamic mode decomposition with control. *SIAM Journal on Applied Dynamical Systems*, 15, 09 2014.

[21] J. Nathan Kutz, Steven L. Brunton, Bingni W. Brunton, and Joshua L. Proctor. *Dynamic Mode Decomposition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2016.

[22] Manjula Devak and C. T. Dhanya. Sensitivity analysis of hydrological models: review and way forward. *Journal of Water and Climate Change*, 8(4):557–575, 06 2017.

[23] Emanuele Borgonovo and Elmar Plischke. Sensitivity analysis: A review of recent advances. *European Journal of Operational Research*, 248(3):869–887, 2016.

[24] Chaudhry Aqeel Afzal, Buchwald Jörg, and Nagel Thomas. Local and global spatio-temporal sensitivity analysis of thermal consolidation around a point heat source. *International Journal of Rock Mechanics and Mining Sciences*, 139:104662, 01 2021.

[25] Melito Gian, Jafarinia Alireza, Hochrainer Thomas, and Ellermann Katrin. In *Sensitivity Analysis of a Hemodynamic-based Model for Thrombus Formation and Growth*, 07 2020.

[26] Sebastian Burhenne, Dirk Jacob, and Gregor Henze. Sampling based on sobol' sequences for monte carlo techniques applied to building simulations. *Proceedings of Building Simulation 2011: 12th Conference of International Building Performance Simulation Association*, pages 1816–1823, 01 2011.

[27] J.C. Helton, J.D. Johnson, C.J. Sallaberry, and C.B. Storlie. Survey of sampling-based methods for uncertainty and sensitivity analysis. *Reliability Engineering and System Safety*, 91(10):1175–1209, 2006. The Fourth International Conference on Sensitivity Analysis of Model Output (SAMO 2004).

[28] Stanfield C.L. *Principles of Human Physiology.* Pearson Education, 13th edition, 2013.

[29] Fox S. I. *Human Physiology.* McGraw Hill Higher Education, 8th edition edition, 2003.

[30] John E. Hall. *Guyton and Hall Textbook of Medical Physiology.* W B Saunders, 13th edition, 2016.

[31] McCorry and Laurie. Physiology of the autonomic nervous system. *American journal of pharmaceutical education*, 71:78, 09 2007.

[32] Y Furukawa, D W Wallick, P J Martin, and M N Levy. Chronotropic and dromotropic responses to stimulation of intracardiac sympathetic nerves to sinoatrial or atrioventricular nodal region in anesthetized dogs. *Circulation Research*, 66(5):1391–1399, 1990.

[33] W. Berry and C. McKenzie. Use of inotropes in critical care. *Clinical Pharmacist*, 2:395, 2010.

[34] Robinson B. F., Epstein S. E., Beiser G. D., and Braunwald E. Control of heart rate by the autonomic nervous system. studies in man on the interrelation between baroreceptor mechanisms and exercise. *Circulation research*, 71:400–411, 1966.

[35] © TeachMe Series 2022. The origin of the recurrent laryngeal nerves, 12 2022. https://teachmeanatomy.info/head/cranial-nerves/vagus-nerve-cn-x/.

[36] Flavio Giordano, Anna Zicca, Carmen Barba, Renzo Guerrini, and L. Genitori. Vagus nerve stimulation: Surgical technique of implantation and revision and related morbidity. *Epilepsia*, 58:85–90, 04 2017.

[37] Elinor Ben-Menachem. Vagus-nerve stimulation for the treatment of epilepsy. *The Lancet Neurology*, 1(8):477–482, 2002.

[38] Giuseppina Barbella, Isabella Cocco, Elena Freri, Guia Marotta, Elisa Visani, Silvana Franceschetti, and Marina Casazza. Transcutaneous vagal nerve stimulatio (t-vns): An adjunctive treatment option for refractory epilepsy. *Seizure*, 60:115–119, 2018.

[39] L. B. Marangell, A. J. Rush, M. S. George, H. A. Sackeim, C. R. Johnson, M. M. Husain, Z. Nahas, and S. H. Lisanby. Vagus nerve stimulation (vns) for major depressive episodes: One year outcomes. *Biological Psychiatry*, 51(4):280–287, 02 2002.

[40] Hang Lv, Yan-hua Zhao, Jian-guo Chen, Dong-yan Wang, and Hao Chen. Vagus nerve stimulation for depression: A systematic review. *Frontiers in Psychology*, 10, 01 2019.

[41] Michael Camilleri, James Toouli, Bård Kulseng, Lilian Kow, Juan Pantoja, Ronald Mårvik, Gjermund Johnsen, Charles Billington, F.G. Moody, Mark Ph.D, Katherine Tweden, M Vollmer, R.R. Wilson, and Mehran Anvari. Intra-abdominal vagal blocking (vbloc therapy): Clinical results with a new implantable medical device. *Surgery*, 143:723–31, 06 2008.

[42] B. Bonaz, V. Sinniger, D. Hoffmann, D. Clarençon, N. Mathieu, C. Dantzer, L. Vercueil, C. Picq, C. Trocmé, P. Faure, J. L. Cracowski, and S. Pellissier. Chronic vagus nerve stimulation in crohn's disease: a 6-month follow-up pilot study. *Neurogastroenterology and motility : the official journal of the European Gastrointestinal Motility Society*, 28(6):948–953, 2016.

[43] Michael R. Gold, Dirk J. Van Veldhuisen, Paul J. Hauptman, Martin Borggrefe, Spencer H. Kubo, Randy A. Lieberman, Goran Milasinovic, Brett J. Berman, Sanja Djordjevic, Suresh Neelagaru, Peter J. Schwartz, Randall C. Starling, and Douglas L. Mann. Vagus nerve stimulation for the treatment of heart failure: The inovate-hf trial. *Journal of the American College of Cardiology*, 68(2):149–158, 2016.

[44] R. K. Premchand, K. Sharma, S. Mittal, R. Monteiro, S. Dixit, I. Libbus, L. A. DiCarlo, J. L. Ardell, T. S. Rector, B. Amurthur, B. H. KenKnight, and I. S. Anand. Extended follow-up of patients with heart failure receiving autonomic regulation therapy in the anthem-hf study. *Journal of cardiac failure*, 22(8):639–642, 2016.

[45] Max Haberbusch, Silvia Frullini, and Francesco Moscato. A numerical model of the acute cardiac effects provoked by cervical vagus nerve stimulation. *IEEE Transactions on Biomedical Engineering*, 69(2):613–623, 2022.

[46] Cracchiolo Marina, Ottaviani Matteo, Panarese Alessandro, Strauss Ivo, Vallone Fabio, Mazzoni Alberto, and Micera Silvestro. Bioelectronic medicine for the autonomic nervous system: clinical applications and perspectives. *Journal of Neural Engineering*, 18, 02 2021.

[47] Yuan H. and Silberstein S. D. Vagus nerve and vagus nerve stimulation, a comprehensive review: Part ii. *Headache*, 56(2):259–266, 2016.

[48] Haberbusch Max, Kronsteiner Tina, Anne-Margarethe Kramer, Kiss Attila, Podesser Bruno, and Moscato Francesco. Closed-loop vagus nerve stimulation for heart rate control evaluated in the langendorff-perfused rabbit heart. *Scientific Reports*, 12:18794, 11 2022.

[49] Ojeda D., Le Rolle V., Romero-Ugalde H. M., Gallet C., Bonnet J. L., Henry C., Bel A., Mabo P., Carrault G., and Hernández A. I. Sensitivity analysis of vagus nerve stimulation parameters on acute cardiac autonomic responses: Chronotropic, inotropic and dromotropic effects. *PloS one*, 11(9), 2016.

[50] Richard B. Colquitt, Douglas A. Colquhoun, and Robert H. Thiele. In silico modelling of physiologic systems. *Best Practice & Research Clinical Anaesthesiology*, 25(4):499–510, 2011. New Approaches in Clinical Research.

[51] Romero Ugalde Hector, Ojeda David, Rolle Virginie, Andreu David, Guiraud David, Bonnet Jean-Luc, Henry Christine, Karam Nicole, Hagège Albert, Mabo Philippe, Carrault Guy, and Hernandez Alfredo. Model-based design and experimental validation of control modules for neuromodulation devices. *IEEE Transactions on Biomedical Engineering*, 11 2015.

[52] Helmers S. L., Begnaud J., Cowley A., Corwin H. M., Edwards J. C., Holder D. L., Kostov H., Larsson P. G., Levisohn P. M., De Menezes M. S., Stefan H., and Labiner D. M. Application of

a computational model of vagus nerve stimulation. *Acta Neurologica Scandinavica*, 126(5):336–343, 2012.

[53] I Parnas and I Segev. A mathematical model for conduction of action potentials along bifurcating axons. *The Journal of Physiology*, 295(1):323–343, 1979.

[54] H. Motz and F. Rattay. A study of the application of the hodgkin-huxley and the frankenhaeuser-huxley model for electrostimulation of the acoustic nerve. *Neuroscience*, 18(3):699–712, 1986.

[55] Hodgkin-huxley experiments. `https://cheever.domains.swarthmore.edu/Ref/HH/HHmain.htm`. Accessed: 2023-02-06.

[56] Daniel Soudry and Ron Meir. Conductance-based neuron models and the slow dynamics of excitability. *Frontiers in computational neuroscience*, 6:4, 02 2012.

[57] Onyejekwe Okey Oseloka. Dynamical computations of the FitzHugh- nagumo equation. *Archive of Biomedical Science and Engineering*, pages 027–036, 09 2021.

[58] Levick andJ Rodney. *An introduction to cardiovascular physiology*. Butterworth-Heinemann, 2013.

[59] © TeachMe Series 2022. Overview of the individual components of the heart conduction pathway, 12 2022. https://teachmeanatomy.info/thorax/organs/heart/conducting-system/.

[60] Quarteroni A., Manzoni A., and Vergara C. The cardiovascular system: Mathematical modelling, numerical algorithms and clinical applications. *Acta Numerica*, 26:365–590, 2017.

[61] Benjamin Owen, Nicholas Bojdo, Andrey Jivkov, Bernard Keavney, and Alistair Revell. Structural modelling of the cardiovascular system. *Biomechanics and Modeling in Mechanobiology*, 17:1–26, 10 2018.

[62] Van de Vosse F. Mathematical modelling of the cardiovascular system. *Journal of Engineering Mathematics*, 47:175–183, 2003.

[63] M. Haberbusch, B. Kronsteiner, A. Kiss, and F. Moscato. Model-based development of a closed-loop heart rate control strategy using vagus nerve stimulation. Presented at Proc. Annual Meeting of the Austrian Society for Biomedical Engineering 2021, 10 2021.

[64] Haberbusch Max, De Luca Daniela, and Moscato Francesco. Changes in Resting and Exercise Hemodynamics Early After Heart Transplantation: A Simulation Perspective. *Frontiers in Physiology*, 11:579449, 2020.

[65] Francesco Moscato, Maurizio Arabia, Francesco M. Colacino, Phornphop Naiyanetr, Guido A. Danieli, and Heinrich Schima. Left ventricle afterload impedance control by an axial flow ventricular assist device: A potential tool for ventricular recovery. *Artificial Organs*, 34(9):736–744, 2010.

[66] Christoph Gross, Libera Fresiello, Thomas Schlöglhofer, Kamen Dimitrov, Christiane Marko, Martin Maw, Bart Meyns, Dominik Wiedemann, Daniel Zimpfer, Heinrich Schima, and Francesco Moscato. Hemodynamic exercise responses with a continuous-flow left ventricular assist device: Comparison of patients' response and cardiorespiratory simulations. *PLOS ONE*, 15(3):1–17, 03 2020.

[67] Haberbusch Max, De Luca Daniela, and Moscato Francesco. Changes in resting and exercise hemodynamics early after heart transplantation: A simulation perspective. *Frontiers in Physiology*, 11, 2020.

[68] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016.

[69] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[70] Jon Herman and Will Usher. SALib: An open-source python library for sensitivity analysis. *The Journal of Open Source Software*, 2(9), jan 2017.

[71] A. Garny, P. Kohl, P.J. Hunter, M.R. Boyett, and D. Noble. One-dimensional rabbit sinoatrial node models: benefits and limitations. *The Journal of Cardiovascular Electrophysiology*, 14:S121–S132, 2003.

[72] Haberbusch Max. *Development of phenomenological models and control strategies to restore vagal heart rate control through closed-loop vagus nerve stimulation.* PhD thesis, Medical University of Vienna, 2023. Unpublished thesis.

[73] G Adomian, G E Adomian, and R E Bellman. Biological system interactions. *Proceedings of the National Academy of Sciences*, 81(9):2938–2940, 1984.

[74] R.B. Choroszucha, J. Sun, and K. Butts. Nonlinear model order reduction for predictive control of the diesel engine airpath. In *2016 American Control Conference (ACC)*, pages 5081–5086, 2016.

[75] Sebastian Peitz and Stefan Klus. Koopman operator-based model reduction for switched-system control of pdes. *Automatica*, 106:184–191, 2019.

[76] Siyuan Shen, Yang Yin, Tianjia Shao, He Wang, Chenfanfu Jiang, Lei Lan, and Kun Zhou. High-order differentiable autoencoder for nonlinear model reduction, 2021.

[77] Mohammad Reza Keshtkaran and Chethan Pandarinath. Enabling hyperparameter optimization in sequential autoencoders for spiking neural data. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[78] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In Ryan P. Adams and Vibhav Gogate, editors, *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 367–377. PMLR, 22–25 Jul 2020.

[79] Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. *CoRR*, abs/1803.09820, 2018.