POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering (LM-25)

Master's Degree Thesis

Deep Learning technique for Model Dynamic Identification and Forecasting

Supervisors

Prof Giorgio GUGLIERI

PhD. Francesco MARINO

Candidate

Antonino GANDOLFO

Marzo 2023

Table of Contents

List of Tables

List of Figures

Acronyms

AI
Artificial Intelligence

ANN
Artificial Neural Networks

NN
Neural Networks

ML
Machine Learning

SIL
Software-in-the-Loop

LTI
Linear time invariant

PgNN
Physics-guided neural network

PiNN
Physics-informed neural network

PeNN
Physics-encoded neural network

UDE
Universal Differential Equation

Abstract

The mathematical modeling of a system is an activity as complex as it is useful for many fields of work and research.
Technological development in areas such as **machine learning** opens the door to new approaches in data analysis and data knowledge extrapolation.
What we propose in this work is a new approach to system identification that takes the advantage of the newest Deep Learning research for the virtualization of models from measurements.

NeuralODEs are a new formulation of the classical Neural Network.
NeuralODE set itself the objective not of imitating the pattern of a system but really learning its dynamics from data. The aim of the work is to validate this framework for the generation of a surrogate model able to learn the set of ODEs that represent the model under study.

We propose different test cases with different dynamics to test the abilities and limitations of this tool.
Furthermore we test the NeuralODEs for different "*unseen*" conditions and input values.

The framework studied shows interesting ability in system identification for systems with a content complexity, showing forward good scalability.

Chapter 1

Introduction

In this work we investigated structural identification problems that usually involve complex setups, difficulties in the training data-driven algorithm and lack of scalability of the system.
In this approach, the aiming aspect is to exploit the features of Neural Ordinary Differential Equation (**Neural ODE**) that can include the dynamic of a system that leads to formulating the Physics-Encoded Neural Networks (**PeNN**).

The proposed framework was tested in different scenarios condition and with different implementations to explore the capabilities and limits of the proposed tool.

The most inspiring formulation that has brought us to this research comes with the benefits of generalising the dynamic model with a grey box modellization in which the true system $f(x)$ is replaced with a generalization of the model, and the remaining governing dynamics will be captured by means of the neural network.

In the first chapter, a detailed explanation of the motivation that moves this work and the objectives we want to achieve was proposed.
Additionally an abstract of state of the art concerned about this approach and the uses of this Deep Learning technique.

Follows, in the second chapter provides a comprehensive explanation of the technological and theoretical tools used for this approach. An introduction about the Deep Learning technique, basic concepts of **Artificial Neural Networks**(**ANN**) and a deep clarification about the main tool used, the **NeuralODE**.

In the third chapter a series of case studies and technical reports will be presented, will also be explained in detail the methodology developed and made a comparison between the various policies that we wanted to test. Particular emphasis on the last case returns us to a true application example of this method.

In the last two chapters we analyze the results and through that motivate our conclusion and future works, develops and applications.

Chapter 2

Motivation and Objectives

In the field of engineering modelling and prototyping a common powerful methodology is the **Model-based Design**.
Model-based design is a method for developing complex policy using mathematical models to represent the system's behaviour. This approach is commonly used in engineering, finance, and other fields to analyze and design systems that involve multiple variables and relationships.

In model-based design, a mathematical model is developed to represent the system's behaviour under various conditions. This model can be used for time series *predictions*, optimal control strategy and identify potential instabilities.
The model can also validate the system's design and ensure it meets the required specifications and performance criteria.
Model-based design has several advantages for the integration of different models and software tools into a single design environment, facilitating and speeding up different testing phases (*Software in the Loop* - **SIL**).

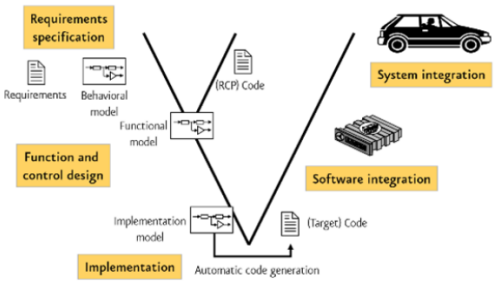

Figure 2.1: Model Based Design

2.1 System Identification

In the context *structural-system identification* (**SI**) [1, 2, 3], describes the methodologies to define an adherent guess of the mathematical model of the system of interest from data response of the system itself.

Researcher have been carried out a lot of techniques to transform measured response into models, is possible to summarize them into two main categories, **data-driven methods** and **physics-driven methods**.

2.1.1 Data-driven methods

Data-driven methods refer to those techniques that take advantage of a large number of measurements available to reconstruct, through different approaches, the system that has generated certain output by given input.
Examples of data-drive policy for system identification could be divided in time domain related and the one related to frequency domain.

Example of **time domain** are eigensystem realization algorithm [4], that proposes a procedure, by using the Hankle matrix, to recostruct the state-space representation of a *linear time invariant* system (**LTI**) from measurement.

In **frequency domain** a well tested method is *frequency domain decomposition* where the decomposition is performed simply by analyzing each of the estimated spectral density matrices [5].

These approaches are examples of classical identification tools that are commonly related to the LTI system.

A more contemporary approach, due to the computational end technological innovation, to the problem is exploiting the use of machine learning techniques.
The big amount of data used for the inverse modelling process is a good fit with the use of these kinds of techniques such as Neural Networks (**NN**).

The neural networks prediction may be physically inconsistent due to their architecture [6], moreover, their learning ability results restricted for mimicking the dynamics of a system for which it has been trained.

2.1.2 Physics-driven methods

The physics-drive methods exploit a prior knowledge of the system, that allows to *lean* on a general mathematical formulation of the system.

The ability to not only map the input and the output of a system but also bring it together considering the system dynamics, has become a field of particular interest

in engineering.

This second approach finds promising tools in the field of Machine Learning where big amount of measurements could be analyzed with the integration of domain knowledge. This approach enhances interpretability, robustness and physical consistency. The most suitable approach that allows us to integrate the dynamic structure into its architecture is the neural networks.

Generally, in literature, we can distinguish three neural network frameworks to encode physics constraints in training: *physics-guided neural networks* (**PgNN**), *physics-informed neural networks* (**PiNN**) and *physics-encoded neural networks* (**PeNN**) [6].

- **Physics-guided Neural Network (PgNN)**

  Constructs the model as a black box to learn a map between inputs $\mathbf{x}$ and outputs $\mathbf{y}$; the function $\mathbf{y} = \mathbf{F}(\mathbf{x}, \mathbf{w})$ with the neural network's parameters $\mathbf{w}$ were minimized w.r.t the parameters as a classical LossFunction($\mathbf{w}$).
  The particularity of this framework is the data generation, the data are generated *ad hoc* in a controlled environment to capture all the possible physical interactions of the model in the study[7]. Typically the training data set were generated by experimentation (e.g. phenomenon observation), the solution of the governing ordinary differential equation (ODE) or partially differential euquations (PDE), etc.

  PgNN has been used since the study of Lee and Chen [8] for estimation of fluid dynamics properties. The use of ANN in fluid mechanics gives the possibility to alleviate the numerical computational problem of solving the Navier-Stokes equations. In Yang et al. [9] the PgNNs have been used as a part of the resolution process of fluid mechanics simulations showing to be significant in particular for large-scale of fluid flow computation. PgNN has also been applied in aerodynamics due to their capability the speeding uo computation. Wang et al. [10] applied ANN for a modeling process of swirling flow in combustor. In the field of hypersonic turbulent flows [11] a ANN-surrogate model was embedded in flow simulator to reduce the high computational cost.

  PgNN has benn applied also in topology optimization health condition assessment, structural analysis etc. An example of PgNN application in the field of solid mechanics was proposed by Tadesse et al. [12] for the deflection prediction of composite bridge. The PgNN has been shown a good capability in term of alleviate computational problems but they suffers from several limitations due to their ability to generate models based on statistics variation learned by the proposed data set. This aspect gives to the ANN a limited knowledge of the system's physic.

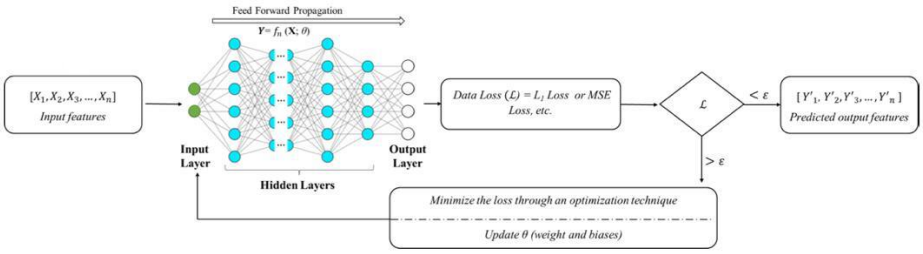

Figure 2.2: PgNN architecture

- **Physics-informed Neural Networks (PiNN)**

  Due to the lack of robustness and the impossibility of generalization of the problem by PgNN the physics behaviour is incorporated outside of the neural network structure.

  This framework typically involves spatial-temporal input and PDE (partial derivative equation) or ODE's (ordinary differential equation) solution as outputs. The models are *informed* about the physical law by adding next to the output layer of a classical MLP (3.1.2) a differentiation layer.
  The solution obtained by differentiating w.r.t the input is used to optimize the parameters of the NN by minimizing a suitable loss function that will take into account boundary condition, data set measurement, governing equations.[6]

  In literature is possible find different application for PiNN, they find a wide application from fluid dynamics to electromagnetic modellization and application. In [13] Fang and Zhan has been developed a PiNN for designing of electromagnetic meta-materials used for specific electrotecnical application like rotor component, DC motors etc. The PiNN were applied also in non-distructive material evaluation, Shuka et al. [14] provide a surrogate model of the poly-cristalline nickel properties. In field of fluids mechanics Depina et al.[15] demostrates the advantages of PiNN in governing Richards PDE and estimate the van Genuchten model parameters through the modellization of unsaturated groundwater flow problem. Wessels et al. [16] develop a Lagrangian method based on PiNN for the solution of inviscid Euler equations of incompressible free surface flow. In these work, it was demostrated the validity of the use of PiNN, it was able to mimic the governing equations of incompressibility condition.

  PiNN training process deal with several problematic. The loss function in PiNN optimization problem contains a lot of terms that affect the final solution, currently there is no guidelines to optimize this process. The PiNNs due to this problem faces difficulties in good convergence or may encouter

the gradient vanishing problem, this occur when during the backpropagation algorithm the partial derivative of the Loss Function w.r.t. the NN parameters can't reache the deeper layer due to the rapid decreasing of the derivative.

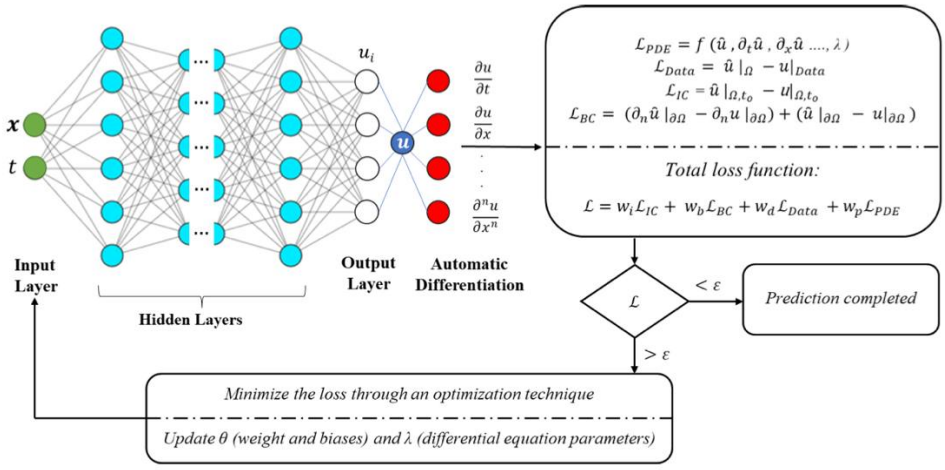

Figure 2.3: PiNN architecture

- **Physics-encoded Neural Networks (PeNN)**

  This family of neural networks directly integrate the physical-driven constraints into their own mathematical structure. Their peculiar architecture allows them to have better computational efficiency, generalization and robustness compared to PgNN and PiNN [17]. Physics-encoded Neural Network propose different approach for encoding physics law into neural networks.

  Li et al. in their work [18] proposed an innovative way to interpret the resolution of a classica neural networks. Fourier Neural Operator (**FNO**) moves the solution field for the optimization of a neural network from time space to Fourier space.

  A different attitude was proposed by Rao et al. [17] in witch with the Physics-encoded Recurrent Convolutional Neural Network (**PeRCNN**) they reformulate basic element of the classical neural network substituting to the nonlinear activation functions a novel elementwis product operation in order to simulate the non linearity of the system. PeNN extend learning ability from instance to continuous learning, representing in the case of Neural Ordinary Differential Equations (**NeuralODE**), the dynamics of a system using a set of continuous-time differential equations.3.3
  NeuralODE also due to their ability to learn the dynamic knowledge of a

system give us the possibility to generalize a system and so have the chance to test this surrogate learned dynamic system under never seen initial condition or different system-forcing.

2.2 Universal Differential Equation

The link that enabled us to combine the benefits of a data-driven machine learning approach with the rigour of more traditional scientific models and thus exploit the characteristics of our chosen typology of PeNN is **Universal Differential Equation**(**UDE**) [19].

The UDE is a mathematical object that represents a differential equation in which the equation is partially or fully established by a so-called "*universal approximator*", a parametrizable black box with the ability to mimic any possible function (i.e. Neural Networks).

This policy opens the door to an immense variety of possible tools to be developed,from the simple case in which the part of physic law is missing (f_{phy}) and fall in the case of NeuralODE 3.3 to more articulated tools in which the encoded knowledge can be found internally or externally to the structure of the neural network. Here in this work we explore a part of these application varieties by analyzing their characteristics, strengths and weaknesses. $NN(\mathbf{h}(t), t, \theta)$.

$$\frac{d\mathbf{h}(t)}{dt} = f_{phy}(\mathbf{h}(t), t, \mathbf{u}(t)(t)) + NN(\mathbf{h}(t), t, \theta) \tag{2.1}$$

where:

- $\mathbf{h}(t)$: states vector $[\mathbf{x}(\mathbf{t}), \dot{\mathbf{x}}(\mathbf{t})]$
- $\mathbf{u}(t)$: system input
- $\boldsymbol{\theta}$: trainable parameters of the neural networks

2.3 Physics-encoded Neural ODEs

In this work, we look through the different possible implementations and develop a procedure that allows a strong generalization ability without a lack of accuracy in the model description.
The PeNN are a typology of the neural network, as expressed before, requires a model to be encoded inside the formulation of the mathematical model of the neural network itself.
In this work our proposal is to carry out an optimization problem formulation and

the implementation of a generalized model description in the presence of incomplete knowledge of the non-linear system (**grey-box modelling**) was performed.

2.3.1 Grey-box modeling

In a black-box approach, we do not have any prior knowledge of the system and so any mathematical relations between the variables in the system will be estimated only by the use of a large amount of data.
On the other hand, white-box modelling presupposes a precise and accurate knowledge of the phenomenon and therefore all the constitutive equations that describe it.

In different cases is difficult to know the whole dynamic of a system and it is

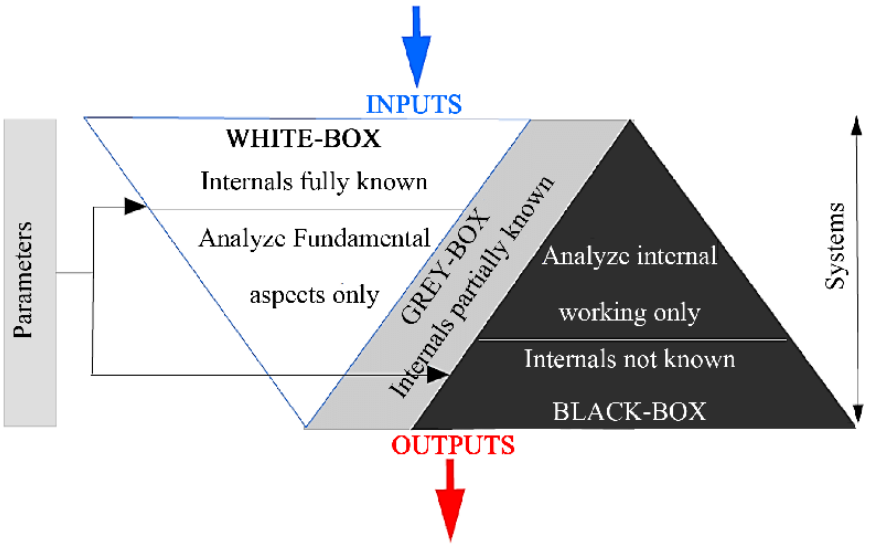

Figure 2.4: Modeling representation

difficult to have a consistent and varied amount of data available for a purely black-box approach. In this case, **grey-box** approach allows us to exploit partial prior knowledge of the system, and the unknown part of the system dynamics are estimated by data measurement.

Chapter 3

Technological Background

Machine Learning (**ML**) is a very broad branch of Artificial Intelligence (**AI**). Given the variety of topics that comprise it and the number of fields of application, Machine Learning is characterized by different methods, tools and techniques.
More in general ML is a set of methods that allows an AI to improve its capabilities over time.
Thus, the AI will be able to increase its own experience and improve itself continuously in carrying out a specific task.
One of the most popular branches of ML is the Artificial Neural networks (**ANN's**). The name and logic behind ANN's take inspiration from the same biological structures, the **neuron**.

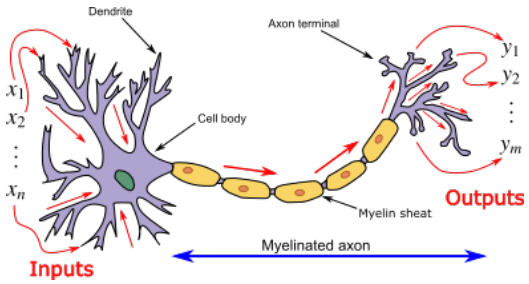

Figure 3.1: Biological neuron

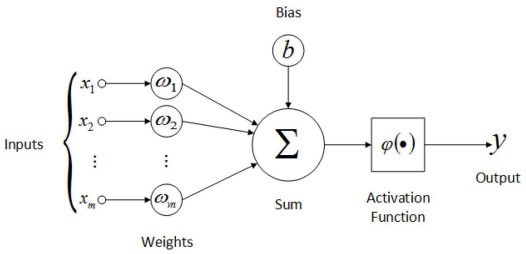

Figure 3.2: Artificial neuron

The first model of a rough neuron was developed in 1943 by Warren McCulloch and Walter Pitts. The ANNs is a dynamical mathematical model that has the capability, in certain conditions, to *learn*, to map input and output, to *reconstruct* any nonlinear function.
ANNs use learning algorithms that can make adjustments independently and thus improve their performance over time. They work in the same way as a biological neural network. Receives information from the previous layer of neurons performs

an evaluation and communicates its result to the next neuron layer.
Due to their flexible mathematical structure and their strong efficiency, ANNs have a wide field of applications.
In **finance** different machine learning startegies have been used for stocks analysis or in the context of ani-financial crime, the big amount of bank transaction well match the possibility of this instrument to extract reasonable information from data.
Furthermore in **medical** sector ANN are well used in prognostic phase in order to better analyze heath records and for the definition of prevention treatment.

Generally Machine Learning policy are concretizing the importance of data analysis, that tools are giving us the possibility to extract knowledge from big amount of data of every context.

3.1 Neurons

3.1.1 Biological Neuron

A neuron is an electrically excitable cell, w.r.t figure 3.1, it's composed by:

Dendrites
Short nerve fibres, similar to branches that, receive messages from the axons of other neurons(the input of the neuron) and transmit them to the nucleus of the cell inside the body cell.

Soma
The cell body where all the signals are processed.

Axon
Long nerve fibre that transmits messages from the body of the neuron to the dendrites of other neurons (the output of the neuron).

Synapses
The virtual point of contact between neurons through which a neuron passes the signal to another neuron.

When a neuron receives stimuli through its dendrites, depending on the kind of stimulus and the kind of neurons, there is a possibility that the neuron passes in an activation state. In this state, the neuron produces an electrical signal that will be propagated via the axon to the following synapses to communicate with another neuron. Once they reach the axon termination, these impulses (called action potentials) lead to the release of neurotransmitters at the level of synapses which could inhibit or propagate the signal to the next neuron.
A neural network consists of the interconnection of a multitude of neurons grouped

in different ways that allow the development of various stimuli.

3.1.2 Artificial Neuron

Even for the Artificial Neural Network, the structural unity is the **neuron**.
An artificial neuron is a simple mathematical structure that establishes a correlation between inputs and output values.

This relationship will be a function of:

Weights(**w**) arbitrarily assigned to each input.

Activation function Φ that maps output with the various inputs considered.

Developed for the first time in 1943 by Warren McCulloch and Walter Pitts in the first time the structure was similar to a unit step function with a threshold. From now on we refer to the **MCP** model [20]. This model consists in a classifier, which maps a vector of n input to a value of output.
With reference to the figure 3.2 here a **MCP** policy:

- At first the **dot product** is computed between the *input vector* **x** and the *weight/parameter vector* **w**; in this way the importance for each input is defined. In the same way as a biological neuron, different stimuli can trigger in different ways the same neuron.

$$\mathbf{x} \cdot \mathbf{w} = \sum_{i=1}^{n} x_i w_i = z \tag{3.1}$$

- Once the information of a neuron is created, "z" will be the input for the activation function "**Φ**" of the neuron. This step recreates the passage of information between one neuron and to another. The value of the activation function is used to classify the input vector. There are different kinds of activation functions for the specific nature of mapping.

$$\Phi\Big(\sum_{i=1}^{n} x_i w_i\Big) \tag{3.2}$$

  An example of activation function could be:

$$\Phi(z) = \begin{cases} \text{z} & \text{if } \sum_{i=1}^{n} x_i w_i + \varphi \geq 0 \\ 0 & \text{else } \sum_{i=1}^{n} x_i w_i + \varphi < 0 \end{cases} \tag{3.3}$$

where φ is the bias and set the threshold for the activation of function Φ .

In this way the output of every single neuron is a parametrization of the input wrt the weights **w**.

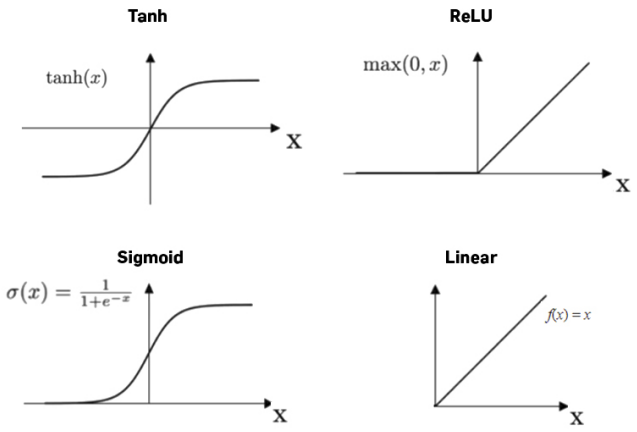

Figure 3.3: Example of activation function

3.2 How an ANN's learns

ANN propagates the signal of the input forward through its network, computes its output, and so **back-propagates** in reverse through the network the information about the output and the target to be reached.

3.2.1 Feed-forward

The first layer that takes the input is the input layer. Each neuron is initialised with a random weight and computes its output through the activation function. Afterwards, the output of each neuron is passed to the next layer with the same policy described above. The process of information distribution from one layer to the next one defines the ANN's as a **feed-forward network**.

The structure described in 3.1.2 refers to a single operating unit.
A series of n connections between neurons with common input makes up the layers, and a series of them makes up *Multi Layer Perceptron* (**MLP**). This is a **nonlinear feed-forward network** that can compute every kind of function.
In figure 3.4 an example of MLP neural network structure.
The learning ability of an ANN's, in the first instance, is a function of the dimensional characteristics of the network itself *number of layers*, *number of neuron per*

layer and consequently number of connections.

The structure described creates a very sensitive function that connects input and output via the weight of every single neuron. These simple networks are suitable for mathematical readjustments.

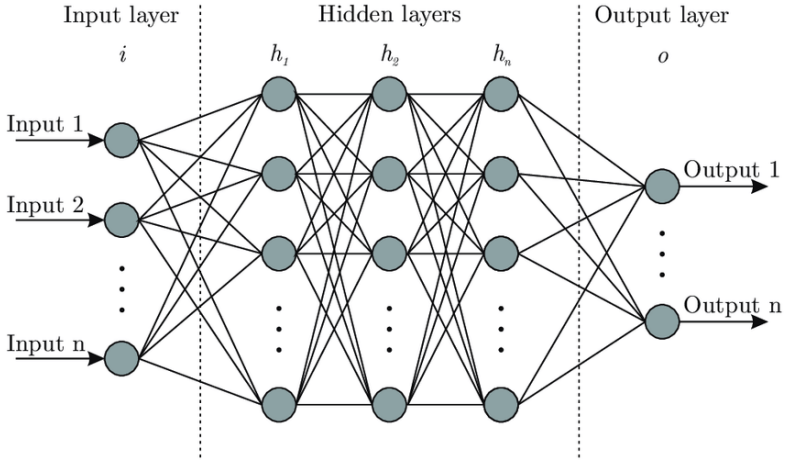

Figure 3.4: ANN structure

3.2.2 Backpropagation

The aim is to understand how to efficiently modify the ANN's weight in order to have a function of the inputs and weight $\tilde{y} = F_{NN}(\mathbf{w}, \mathbf{x})$ that recreates the original function under study $y = F(\mathbf{x})$. This is an *optimization problem* whose aim is to minimize the loss function wrt the set of weight.

$$L(y, \tilde{y}) = L(y, F_{NN}(w, x)) = \min\ w\ L(y, F_{NN}(\mathbf{w}, \mathbf{x})) \tag{3.4}$$

a possible choice of L is mean square error so the 3.4 became:

$$L(y, \tilde{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2 \tag{3.5}$$

For this purpose, the ANN are trained using a training data set, a collection of data in which inputs and outputs are correlated (x, y). The corresponding true output values y in the training data set will allow the ANN to efficiently modify the weights. This problem of optimization could be difficult to compute. In massive ANN's there is a lot of weight to be adjusted and each of these has a complicated influence on the result.

The solution is a **backpropagation** algorithm that allows us to propagate the

error (the result of the loss function) to each weight that it composes. It consists of the computation of the loss gradient ∇L with respect to net weights $\mathbf{w}$.

The aim is to know how much a change in w_i affects the total error $L(y, \tilde{y})$. Taking into consideration the figure 3.2.

By applying the chain rule:

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \tilde{y}} \frac{\partial \tilde{y}}{\partial z} \frac{\partial z}{\partial w_i} \tag{3.6}$$

First term.

$$\frac{\partial L}{\partial \tilde{y}} = 2\,\frac{1}{2}(\tilde{y} - y) \tag{3.7}$$

With $L(y, \tilde{y}) = \frac{1}{2}(y - \tilde{y})^2$.

Second term.

$$\frac{\partial \tilde{y}}{\partial z} = z(1 - z) \tag{3.8}$$

Choosing as activation function *sigmoid* $\tilde{y} = \Phi(z) = \frac{1}{1+e^{-z}}$

Third term.

$$\frac{\partial z}{\partial w_i} = \frac{\sum_{i=1}^{n} x_i w_i}{\partial w_i} \tag{3.9}$$

wrt equation 3.1.

Update of the parameters

Once the loss gradient $\nabla L(w)$ has been evaluated, it will be used to update the network weights.

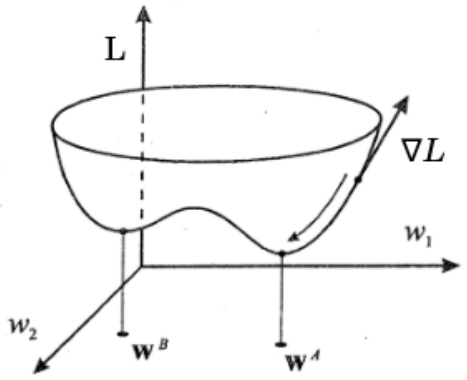

Figure 3.5: Two dimensional Loss function «L»

ANN's exploit iterative *gradient descent algorithms* to optimize their network and find the minimum of the loss function.

$$w_i^{new} = w_i^{old} + \Delta w_i \tag{3.10}$$

where:

$$\Delta w_i = -\eta \nabla L(w_i) \tag{3.11}$$

The variable η (> 0) is the learning rate and the step size for modifying the ANN's weight. It's one of the most important hyperparameters, whose correct choice ensures that the optimization problem can be brought to convergence, avoiding local minima. Choosing a large value could generate instability in the optimization algorithm. Instead, a small value might prevent the algorithms from convergence.
One possible solution to the problem is introducing another parameter μ called *momento*.

$$\Delta w_i = -\eta \nabla L(w_i) + \mu \Delta w_i \tag{3.12}$$

where μ ($0<\mu<1$) in a certain way this hyperparameter represents the *inertia* of the neural network's parameters to be changed by the training algorithm.
It takes into account the contribution of the current variation of the weight. In this way the correction term takes care of the Loss function dynamic. Before moving forward, a clarification is due to introduce the terms of **epochs** and **batch**.

- Epochs -> is a hyperparameter representing the number of times the learning algorithm will work using the whole training data set.
- Batch -> is a hyperparameter that defines the number of measurements used before the algorithm updates the NN's parameters

3.3 Neural ODE

NeuralODE is a particular kind of Deep Neural Network. The idea is similar to the mathematical structure of the *Residual Network*, (**ResNet**) [21] in which the chain of output layers can be seen as a numerical procedure for solving ordinary differential equations [Euler's method]:

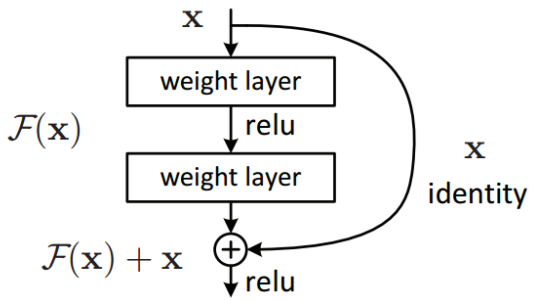

Figure 3.6: Block diagram of Residual Network

$$h_{t+1} = h_t + f(h_t, \mathbf{w}) \tag{3.13}$$

The analogy between the structures of Euler's method with the ODE solver and the definition of the ResNet network is at the base of the NeuralODE. In this kind of ANN, the mathematical structure of the chain output of the hidden layer is replaced by another ODE solver with better accuracy. In this way, it is possible to parameterize the internal mathematical structure of the hidden layer as an ordinary differential equation.

$$\frac{dh(t)}{dt} = f(h(t), \mathbf{w}, t) \tag{3.14}$$

Thus the connection between the layers is determined by the accuracy of the ODE solver we want to use. In other words, we take the basic structure of a single layer of a NN and solve it as a step of our ODE solver. This routine iterates over from the "layer" $h(0)$ to $h(T)$, where 0 (zero) and T integration extremes give us the solution of the ODE initial value problem.

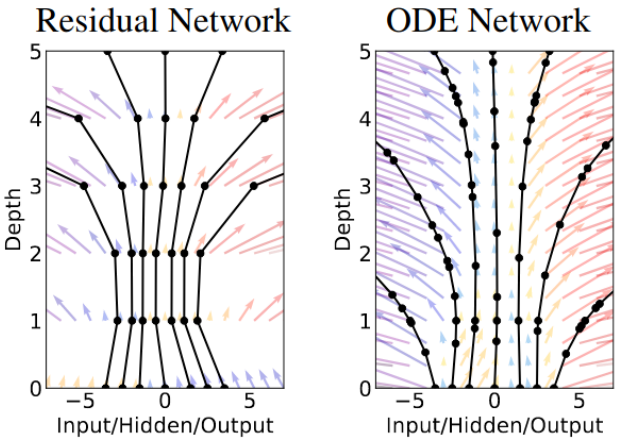

Figure 3.7: Comparison of sequence of transformation

The backpropagation

The NeuralODE are composed by the same elements and procedure of a common ANN, the most problematic aspect was the backpropagation of the error.
As explained in the section **3.2.2** the central aspect of ANN's training is the learning algorithm so the backpropagation of the error. With reference to the original paper [22] the Loss function is defined as:

$$L\Big(\mathbf{z}(t_1)\Big) = L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)\, dt\right) = L(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)) \quad (3.15)$$

where:

$\boldsymbol{\theta}$: dynamics parameters

$\mathbf{t_0}$: start time

$\mathbf{t_1}$: stop time

$\mathbf{z}(\mathbf{t_1})$: final state

In NeuralODE's training the ODE solver is treated as a black box element and the gradient of the Loss function was computed using the adjoint sensitivity method. [23] It is a numerical method to compute the gradient of a function in a numerical optimization problem.

The gradient of 3.15 is computed by solving a second, augmented ODE backwards in time, and this procedure is available to all ODE solvers. This approach allows us to compute at first the gradient with respect to the hidden states.

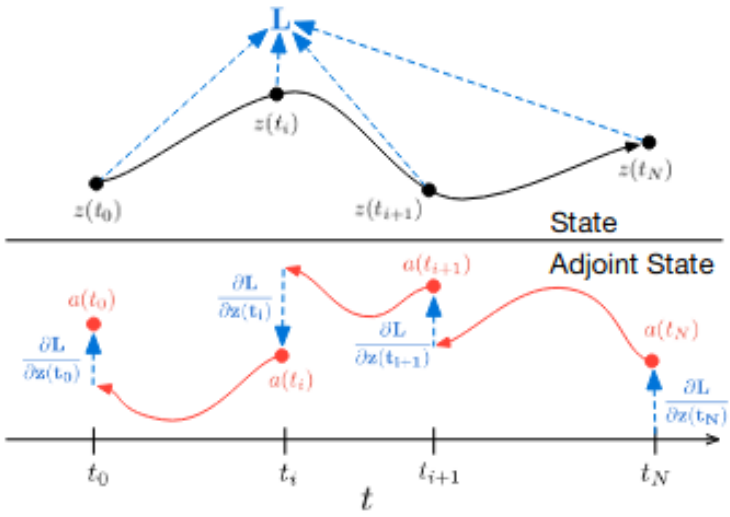

Figure 3.8: Differentiation of an ODE solution

This kind of ANN's has the following advantages:

Memory efficiency treating the ODEsolve as a black box element and using adjoin sensitivity method allow us to train our NN with constant memory cost as a function of size.

Adaptive computation the possibility to use a more suitable ODEsolver allows us to adjust more precisely the computational cost and the accuracy we want to give. Also after training.

Continuous time-series models NeuralODE works continuously. This does not require a constant discretization of input data, they can reconstruct the dynamics equally even with non-sampled data.

3.4 Experimental Method

In this section, a practical illustration of the procedure used to the various test cases presented is given. According to the nativity development of the paper already mentioned [19] we decide to reach our aims by using a specific program language called **Julia**. It's a dedicated programming language for scientific and numerical computing, it combines the versatility of **Python** syntax and the performance of **C**.

All the test cases are developed using the *SciML: Open Source Software for Scientific Machine Learning* a differentiable programming software for scientific machine learning. It contains a variety of modules for automating the process of model discovery and fitting. The main module that allows use to approach this study is *DiffEqFlux.jl*. This library enables the training of embedded neural networks inside of differential equations (neural differential equations or universal differential equations) for discovering unknown dynamical equations.

3.4.1 Experimental set-up

All the experimental that we propose for this work follow the same procedure and they differ for the application case and the problem generalization used to push the developed toll to the limit of its capabilities.
Going more specifically into this topic what we did was to train PeNN to mimic the behavior of a set of ordinary differential equations with fixed initial conditions and input. What we expect to receive after training is a surrogate model of partially known differential equations (grey box) that will allow us to simulate the learned system also for other inputs and/or initial conditions.

Data Generation

For all the experiments the data sets were generated by solving the differential equations (ODE) that described the respective dynamics.
Due to testing the capability of this approach to use a contained number of examples to train our NN's parameters all training end validation data set contain a limited number of measurements.
A traditional approach provides the generation of multiple data sets, one is intended to be used to train the network (**training set**), another to validate the training (**validation set**) and ultimately for testing the correct learning of the dynamics in "*never seen*" conditions. In this case due to the system dynamics of the example used to train we generate 2 measurements for the validation set. The choice is about the nature of the examples chosen, they take stable dynamics with a very limited transient.

Neural Network set-up

Our purpose is to use the NN mathematical structure as a supplemental degree of freedom to find the feasibility parameter set (**FPS**) that mixed with the already known dynamics identify the model.
To do that small NN with a content number of layers and neurons was used, the reason to limit the FPS in which the network must seek the solution of the

optimization problem.

Generally, the NN was constructed by 3∼5 layers and for each layer 10∼50 neurons. According to the experiment, the layers of NN will be interspersed with customized layers $h_{phy}(\theta, t)$ that describe the approximate dynamics of the system. This is possible thanks to the ecosystem that is being used that allows us to define sets of differential equations as custom layers of the network.

The activation functions chosen are the swish function $f(x) = x \cdot \text{sigmoid}(x)$ and the sigmoid function $f(x) = \frac{1}{1+e^(-x)}$ based on the degree of non-linearity that best suited the optimization problem of the Loss Function.
For all the test cases we decide to use the same gradient descendent algorithms, as in the official page of the library above DiffEqFlux.jl we opted to train our nets twice.
The first one involved a rough training that would allow us to avoid bad local minima to which the algorithm would otherwise have settled. This was performed with ADAM algorithm [24] a stochastic gradient descent algorithm; following the first training a second training was carried out with Broyden–Fletcher–Goldfarb–Shanno algorithm (**BFGS**) [25] a local search optimization algorithm. In all the test cases the *Loss Function* to be minimized has been the *Mean Squared Error* (MSE):

$$\text{LossFunction}(\theta, t) = \text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2 \tag{3.16}$$

w.r.t n number of elements, y measurement, $\tilde{y}$ NN's output.

3.4.2 Training

Bad local minima are one of the most common problems when approaching gradient descent optimization. Neural Networks solution space can be big sp that the presence of many local minima is indeed probable. To minimize the possibility of getting stuck in a local minimum we employ three different techniques to train our "*nets*".

Mini-Batching

Single batch approach could drop the algorithm in a non-optima local minimum due to their policy to upload the parameters weight once for entire training data sets. One possible approach to this problem is to divide the measurements in a parametrizable number of batch. To do this the NN's parameters will have a higher update rate that gives us a better ability to move through bad local minima.

Algorithm 1 Mini-batching algorithm

Require: Training data set = (X,Y), Batch size = N
  $batches \leftarrow (X, Y) grouped\ in\ N.\ elements$
  for Number of epochs **do**
    for each batch (X, Y) **do**
      Evaluate NeuralODE solution
      error $\leftarrow Loss\ Function(\mathbf{w})$
      Backpropagation error
      $w_{new} = w_{old} + \Delta w \leftarrow$ Update weighs
    end for
  end for

Multiple shooting

This technique is a method for training NeuralODEs in which the time span $[t_0, t_{fin}]$ of the defined dynamic problem is partitioned in N intervals. In this way, the time series is divided into N initial value problems that will be resolved simultaneously. This segmented trajectory is used to evaluate the optimization problem (Loss Function) as in a single shooting.
For each interval $N-1$ the solution of initial value problem gives a discontinuity trajectory, in order to take trace of the discontinuity gap between the $N-1$ trajectories and guide the algorithm to an optimum minima a penalty terms has to be considered.
Continuity term is a hyperparameter that will penalize the gap between the intervals thus guiding the training algorithm to create a continuous trajectory.

Algorithm 2 Multiple shooting algorithm

Require: N intervals, Continuity term μ, Time span $[t_0, t_f]$
  $t_0 = \tau_0 \leq \tau_1 \dots \tau_N = t_f \quad N+1$ points
  for $i = 1 \dots N$ **do**
    $NN_i(x_i, [\tau_0^i, \tau_f^i], \theta)$
  end for
  $solve(NN_i, \dots, NN_N)$
  Loss Function
  $\min_w \quad Loss\ Function(\mathbf{w}) = MSE(\mathbf{w}) + \rho Q(\mathbf{w})$
    s.t. $\quad Q(\mathbf{w}) = x_f^i - x_0^{i+1} = 0$
  Evaluate the *error*
  Update weights: $w_{new} = w_{old} + \Delta w$

Chapter 4

Case Studies

This section will present the case studies and the related experiments we carried out. The intent is to show the capabilities and limits of the **PeNN** using the frameworks chosen and the implementation developed. Each case presents a specific neural network implementation to adapt to the problem under consideration. Also, we elaborate on different training policies and different versions of PeNN suitable for testing their ability for system identification.

4.0.1 Simple Pendulum

In this first example, we test the ability of this framework to mimic the dynamic of a simple system to understand its performance in incorporating the dynamic and test it for different initial conditions $\mathbf{x_0}$.

In this case, we do not incorporate any dynamic layer inside the neural network structure.

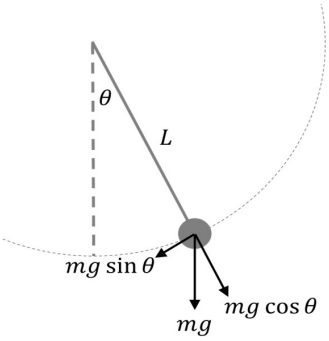

Figure 4.1: Simple pendulum

System dynamic description

The dynamic of simple pendulum 4.1 is a well-treated problem in the literature. It consists of a point of mass m, subjected to a gravitational field of constant g, suspended thanks to an in-extensible wire of length L from a fixed support. The angle between the vertical axis and the wire is commonly defined as θ. Despite its simplicity, it is an excellent example of a non-linear system. The Lagrangian derivation of the equations of motion of the simple pendulum:

$$ml^2\ddot{\theta}(t) + mgl\sin(\theta) = F \tag{4.1}$$

Converting this second-order differential equation into a set of first-order differential equations and considering $F = 0$:

$$\begin{aligned}
\theta &= \theta_1 \\
\frac{d\theta_1}{dt} &= \theta_2 \\
\frac{d\theta_1}{dt} &= \theta_2 \\
\frac{d\theta_2}{dt} &= -\lambda \cdot \theta_2 - \frac{g}{L} \cdot \sin(\theta_1)
\end{aligned} \tag{4.2}$$

Data Collection

The training data set was collected by solving the equation 4.2. Above is an extract of the code that represents its formulation[fig code]. The initial condition $\mathbf{x_0}$ = [0,2], the timespan = [0s,10s]. The data were collected with a sample of 0,1s, so 100 measurements.

Neural-ODE structure

The neural network, defined above, represents the simplest case of PeNN wherein the f_{phy}, the portion of physics knowledge, is omitted, and then it's the case of a NeuralODE $x'(t) = NN'(\sigma, t)$. Since the structure we created will be treated as a differential equation, it will also have a solver, initial conditions, and a period (the same used for data generation). The NN hyperparameters are:

n. Neurons	$layer_1$	10
	$layer_2$	30
Hyperparameters	$Learning\,rate$	5×10^{-2}
	$Initial\,step$	1×10^{-2}
Activation function	swish	
n. Epochs	500	

Table 4.1: Simple Pendulum NN parameters

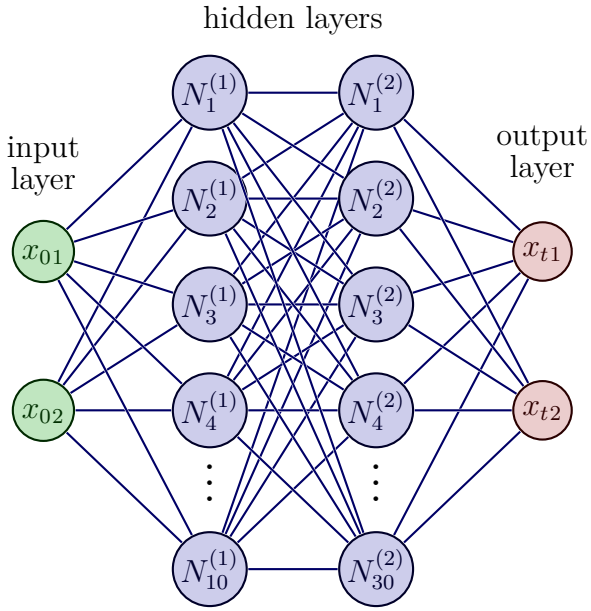

Figure 4.2: Simple pendulum NN structure

4.0.2 Triple Oscillating Mass

We have not incorporated the system's dynamics within the neural network's structure as in the previous case.
We focused on investigating whether dynamics integration also worked for more complex systems.

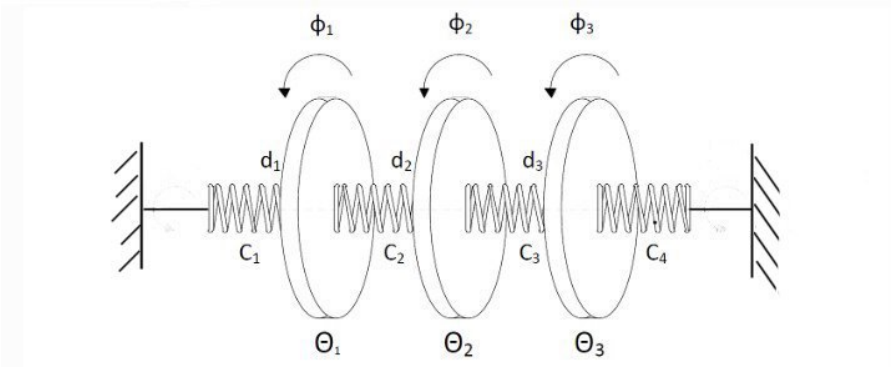

Figure 4.3: Triple oscillating mass

System dynamic description

Three rotating discs with inertia $[\Theta_1, \Theta_2, \Theta_3]$ are connected via springs with constants $[c_1, c_2, c_3, c_4]$. The two outermost discs are each connected to a fixed support with additional springs. Relevant parameters of the system are the inertia $\Theta_{1,2,3}$ of the three discs, the spring constants $c_{1,2,3,4}$ as well as the damping factors $d_{1,2,3}$, the state vector is $\mathbf{x} = [\phi_1, \phi_2, \phi_3]$. The set of equations that describe the system can be written as follows:

$$\begin{aligned}
\Theta_1\ddot{\phi}_1 &= -c_1\phi_1 - c_2(\phi_1 - \phi_2) - d_1\dot{\phi}_1 \\
\Theta_2\ddot{\phi}_2 &= -c_2(\phi_2 - \phi_1) - c_3(\phi_2 - \phi_3) - d_2\dot{\phi}_2 \\
\Theta_3\ddot{\phi}_3 &= -c_3(\phi_3 - \phi_2) - c_4\phi_3 - d_3\dot{\phi}_3
\end{aligned} \tag{4.3}$$

Reformulating the equation 4.3 in a first order ODEs:

$$\begin{aligned}
\dot{x}_1 &= x_4 \\
\dot{x}_2 &= x_5 \\
\dot{x}_3 &= x_6 \\
\dot{x}_4 &= -\frac{c_1}{\Theta_1}x_1 - \frac{c_2}{\Theta_1}(x_1 - x_2) - \frac{d_1}{\Theta_1}x_4 \\
\dot{x}_5 &= -\frac{c_2}{\Theta_2}(x_2 - x_1) - \frac{c_3}{\Theta_2}(x_2 - x_3) - \frac{d_2}{\Theta_2}x_5 \\
\dot{x}_6 &= -\frac{c_3}{\Theta_3}(x_3 - x_2) - \frac{c_4}{\Theta_3}x_4 - \frac{d_3}{\Theta_3}x_6
\end{aligned} \tag{4.4}$$

Data Collection

The "*Triple Oscillating Mass*" in case of a non-linear system taken from the example gallery of **Do-MPC** [26], a Python toolbox for the implementation of a *model predictive control*. This toolbox was used only for convenience in the integration of the ODEs.
After having declared the system by using the equation 4.3, the following boundary condition are used to solve the initial value problem: $\mathbf{x_0} = [\pi, \pi, -\frac{3}{2}\pi, \pi, -\pi; \pi]$, timespan = [0,20] s

Neural-ODE structure

As in the previous case, the same considerations were made about the structure neural network due to test it in a more challenging scenario.

n. Neurons	$layer_1$	70
	$layer_2$	70
	$layer_3$	70
Hyperparameters	*Learning rate*	1×10^{-3}
	Initial step	1×10^{-3}
Activation function	swish	
n. Epochs	500	

Table 4.2: Triple Oscillating Mass NN parameters

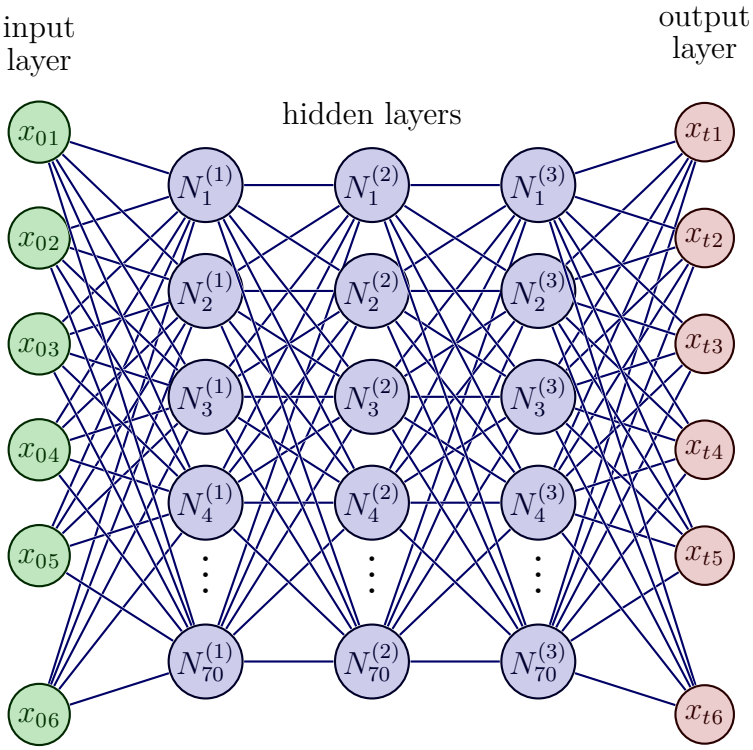

Figure 4.4: Triple rotating mass NN structure

4.0.3 Mass Spring Damper

Previously we dealt with a free dynamic system. In this test case, we want to focus on forced one. Different from the previous example, now we create a specific neural network framework able to bring up the dynamic and test it for different initial

conditions $\mathbf{x_0}$ and different values of input signal $\mathbf{F}$.

For this test case, we proposed an alternative way to exploit NeuralODE. Until now, we present NeuralODE as an NN able to learn not the trend of a function over time $f(x,t)$ but to learn the variation itself during time $f'(x,t)$.

In this case, an alternative approach was offered, in the opposite way of the previous case, within the dynamic was fully represented by the NN and tan solved like an ODE function. In this case, we test to solve an ODE formulation like an NN layer.

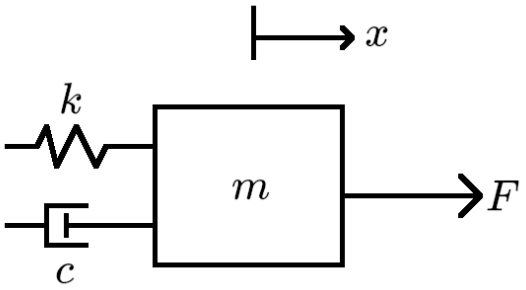

Figure 4.5: Mass damper spring

System dynamic description

The system comprises a mass m, a spring with elasticity constant k and a damper of constant c. The mass is attached to the spring and the damper and the latter are attached to a fixed reference. The system is subject to an impulsive force F. The measured states are the displacement $x(t)$ of the mass and its velocity $\dot{x}(t)$. The dynamic formulation of the problem is:

$$m\ddot{x} + c\dot{x} + kx = F \tag{4.5}$$

Referred as first ODEs:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{1}{m}(F - cx_2 - kx_1) \end{aligned} \tag{4.6}$$

Data Collection

The training data set are collected solving the equation 4.6 for the initial condition $\mathbf{x_0} = [0,0]$, timespan = [] and with an impulsive force $F = 5\,N$.

Furthermore, we decide to add noise to the measurements to test the training algorithm's convergence ability.

Neural-ODE structure

In this case, we create a custom layer in which the connection of the various learnable parameters is represented by the set of ODEs of the system. A different configuration due to test different ways the ability of the frameworks to respond to different stimuli.

n. Parameters	*dynamic layer*	3
Hyperparameters	*Learning rate*	1×10^{-2}
	Initial step	1×10^{-4}
Activation function	swish	
n. Epochs	500	

Table 4.3: Mass Damper Spring NN parameters

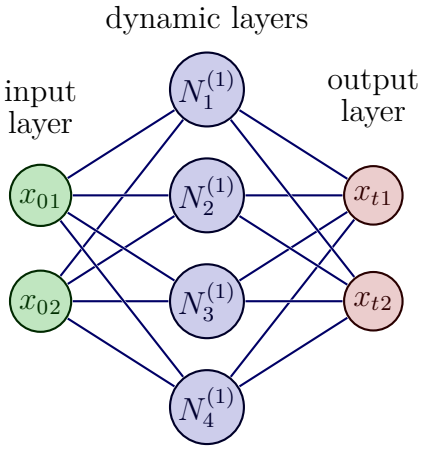

Figure 4.6: Mass Spring Dumper NN structure

4.0.4 Electrohydraulic actuator

In this last case, we elaborate a policy to train our frameworks to mimic the dynamic of an electrohydraulic actuator. These are very complex devices whose functionality is crucial in the avionics industry. Servo valves allow the setting of actuators that are essential for optimal control of primary and secondary flight systems. Due to their importance in the flight control system, component monitoring action is important. Creating a surrogate model for model-based fault detection ad identification becomes a challenge as difficult as useful for preventing critical operating conditions. This experiment bases its interest in the field of prognostic [27], a sub-set of system identification problems about failure prediction of some system components.

Concerning that application area, we've tried to create a framework that would scale up the problem in a way that no longer refers to a specific actuator but, based on the available measurements, rebuild the dynamics of each specific actuator for specific failure tests.

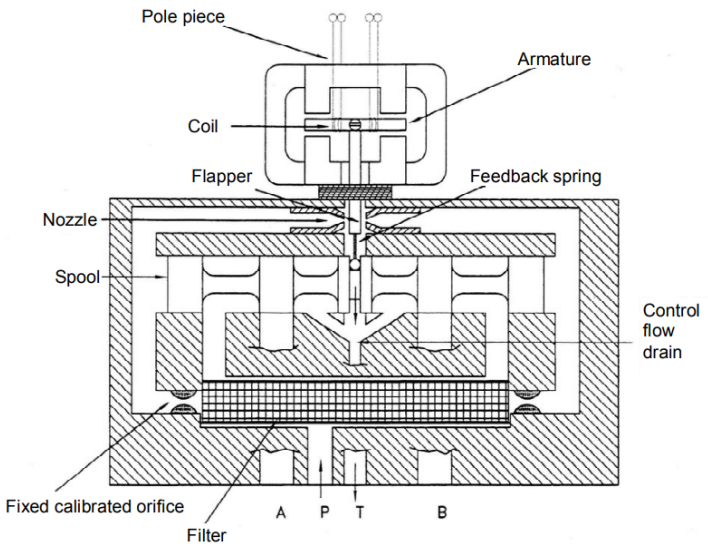

Figure 4.7: Schematic of the flapper-nozzle servovalve

System dynamic description

Regarding figure 4.8, the dynamics of the electrohydraulic actuator can be separated by the interaction of three systems.

The **controller subsystem** is usually a PID (proportional-integral-derivative) an electronic controller which, according to the designated logic, will command through a servo-amplifier (a low-power electrical actuating) signals to the next system.

The **Electrohydraulic two-stage servovalve** commands the pressure hydraulic fluid based on the received signals.

The high-pressure fluid, in turn, will control the stroke of a **hydraulic piston**.

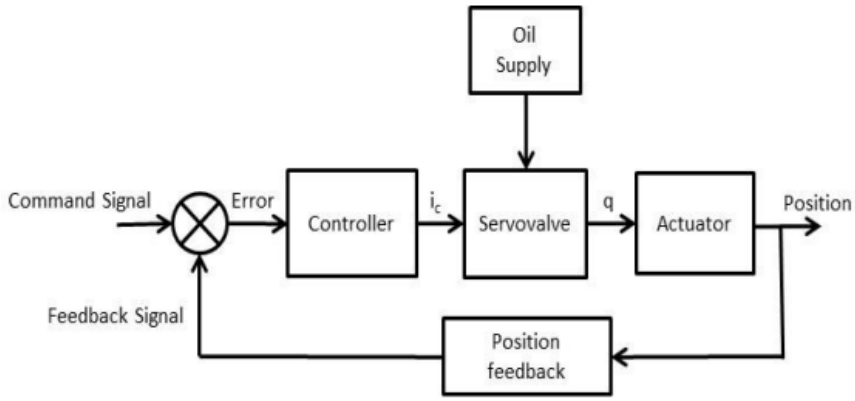

Figure 4.8: Feedback control loop of electrohydraulic actuator

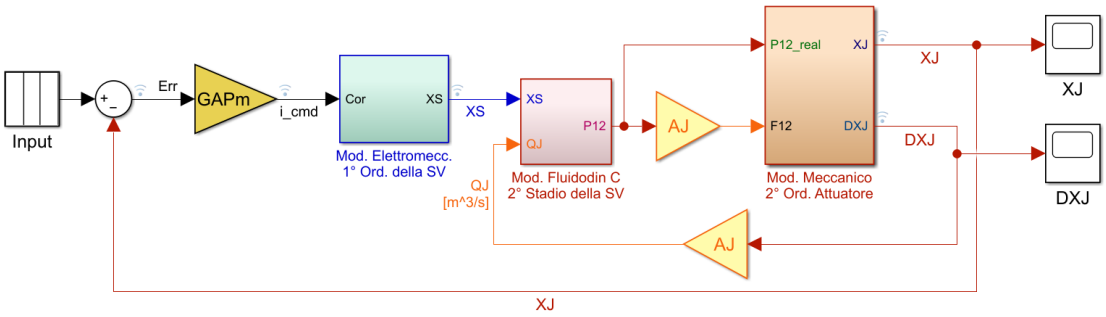

Figure 4.9: Simulink representation electrohydraulic actuator

Data Collection

All data sets, in this case study refer to work done by [27]. The measurements come from a simulation in MATLAB exposed in their work. The state of our interest is the position and velocity of the spool $\mathbf{x} = [x, \dot{x}]$.

Neural-ODE structure

Being aware that the dynamics of the servo actuator consists of several connected systems, our intuition has moved us to replace the difficult parts modeled as if they were NeuralODE and implement a simplification of the mechanical part with a quite good guess (grey-box approach). We decide to approximate as if it was a "mass-spring dumper" system 4.6.

n. Neurons	$layer_1$	10
	$layer_2$	30
Hyperparameters	$Learning\,rate$	5×10^{-2}
	$Initial\,step$	1×10^{-2}
Activation function	swish	
n. Epochs	500	

Table 4.4: Electrohydraulic actuator NN parameters

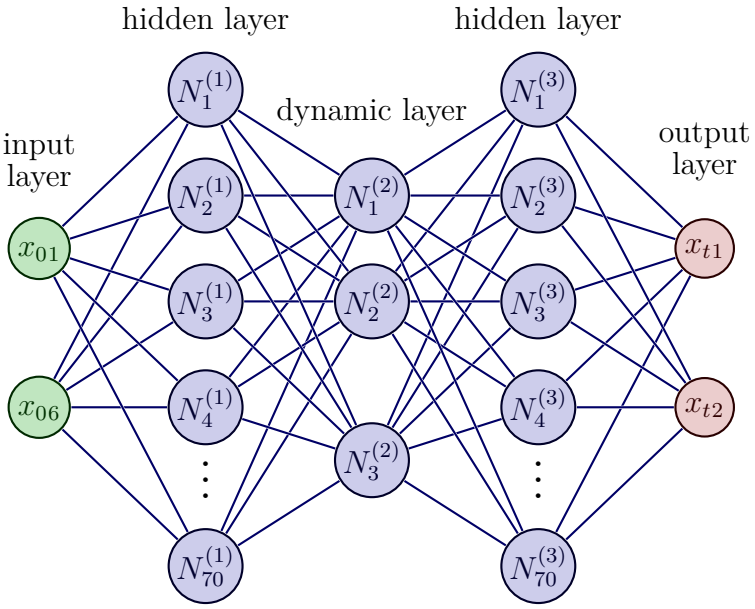

Figure 4.10: Electrohydraulic actuator NN structure

Chapter 5

Results Analysis

5.1 Simple Pendulum

5.1.1 Training and Result

The NN was trained over a single batch method at two-step training to avoid non-optimal minima.
At first, a more shallow training was performed with the ADAM algorithm then through the BFGS algorithm, we tried to optimize the loss function more precisely, avoiding the bad local minima. As shown in 5.1, which describes the course of the

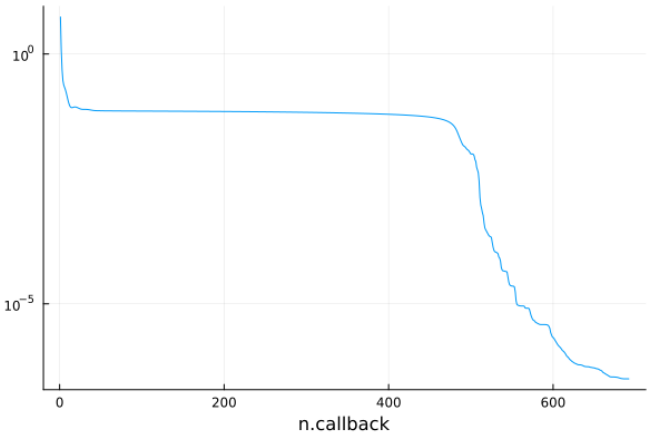

Figure 5.1: Loss Function

value of the loss function during the NN training, we early find bad local minima from which the first algorithm used to train the NN will not be able to escape from. After the chosen epochs, the developed policy changes the algorithm skipping from **BFGS** that will allow a substantial improvement in performance, reaching a

Loss Function = 9,671 $\times$ 10^{-9}.

In the validation process, we compare the response of the NeuralODE model with

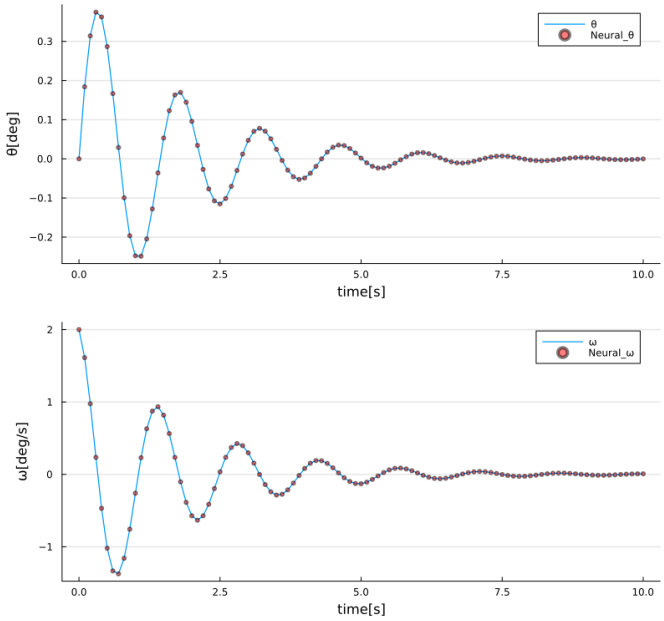

Figure 5.2: Trained NeuralODE

the original one. The trained model was validated by submitting it to resolve different initial value problems with the new initial conditions. In this way, we tested if the developed model absorbed the system dynamics.
In figures 5.3,5.4,5.5 the solutions of the NeuralODE for 3 initial conditions $x_0^{(1)} = [0,1], x_0^{(2)} = [1,0], x_0^{(3)} = [2,0]$ compared with true one obtained by the solution of the original ODE.

It should be emphasized that the training section was carried out by using only **one** test case data set that describes a specific ODE solution, the one for $x_0 = [0,0]$. We can conclude that the trained NeuralODE has learned the dynamics of the simple pendulum problem.

	Loss (MSE)
Training	$9{,}671 \times 10^{-9}$
Validation $x_0 = [0{,}1]$	$1{,}641 \times 10^{-8}$
Validation $x_0 = [1{,}0]$	$1{,}541 \times 10^{-4}$
Validation $x_0 = [2{,}0]$	$1{,}211 \times 10^{-1}$

Table 5.1: Simple Pendulum results

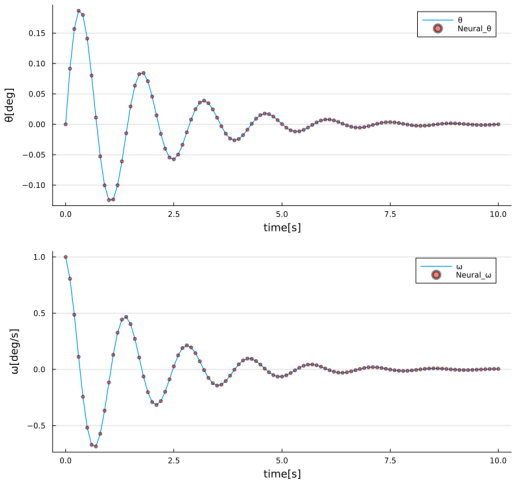

Figure 5.3: Validation $x_0 = [0{,}1]$

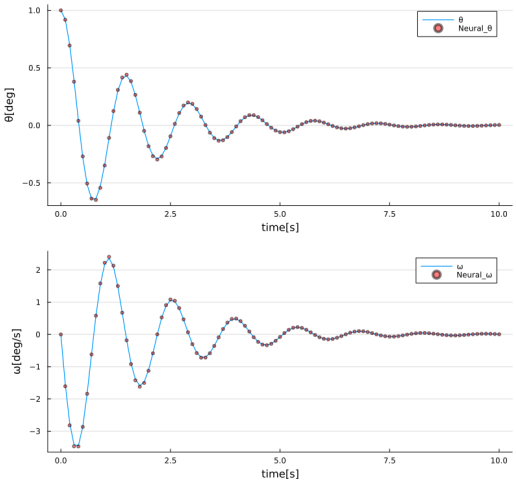

Figure 5.4: Validation $x_0 = [1{,}0]$

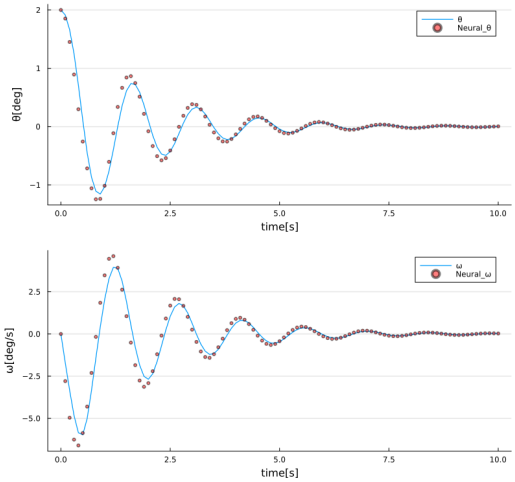

Figure 5.5: Validation $x_0 = [2{,}0]$

5.2 Triple Oscillating Mass

Due to the system's complexity, two training policies were proposed.

The first approach was to train the surrogate model using mini-batching training to avoid no-optimal minima in the loss function due to the oscillating trend. The mini-batching training has been sized to envelop a half cycle for each batch. Then a fine-tuning optimization with BFGS algorithm was performed in the whole training data set in a way to fine-tune the already trained network.

Another training approach is to use multiple shooting training policies. As for the mini-batching, the training data set was split into several groups. Differentially from the previous one, the groups are evaluated simultaneously. Also, in this case, the same consideration about the system behavior was made, and the groups was dimensioned in the same way. To better test the multiple shooting policy, we experimented with training the network both times, first with ADAM algorithm for rough training and after with BFGS for fine-tuning, with this splitting data set technique.

For both training cases, the NN could not converge to an optimal minimum due to the high complexity of the system in consideration.
The NN performs better for increasing complexity.

	Loss(MSE)	
	Mini-batching	**Multiple shooting**
Training	6,77	2,55
Validation $x_0 = [0,1]$	783,8	69,420

Table 5.2: Triple rotating mass results

5.2.1 Mini-batching

As shown in figure 5.6 the loss function does not have a monotone trend due to the fact for each iteration, the Loss function was evaluated for a portion of the whole training data set.

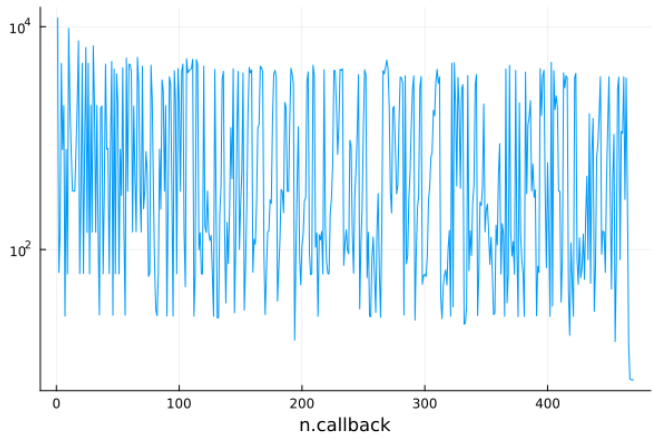

Figure 5.6: Loss Function

In the training section, as shown in figures 5.7,5.8, it is possible to notice that the NeuralODE didn't show the same behavior for all the evaluated states.
In particular, for the state corresponding to Θ_2 the rotating disc placed in the middle in the system, the training section shows the worst training result.
Regarding eqn. 4.4 we assumed this result could be produced by error propagation in the evaluation of the state related to the other two discs Θ_1 and Θ_3.

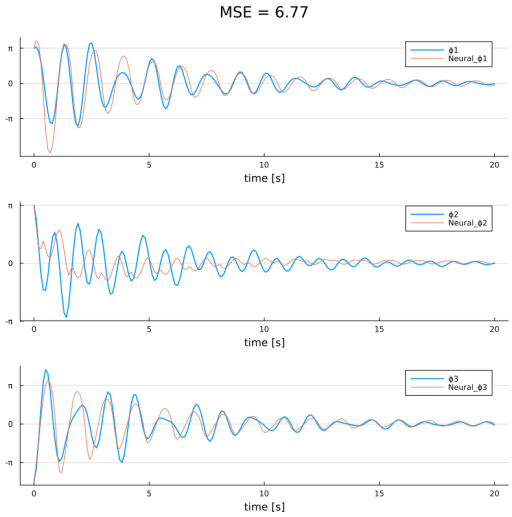

Figure 5.7: Training NeuralODE (position)

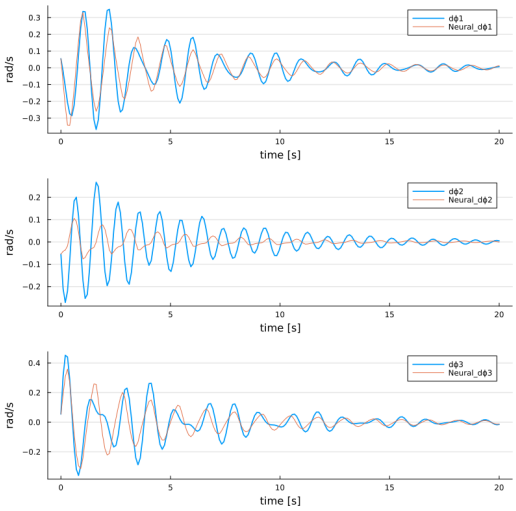

Figure 5.8: Training NeuralODE (velocity)

As a consequence, the validation test didn't return an interesting result.

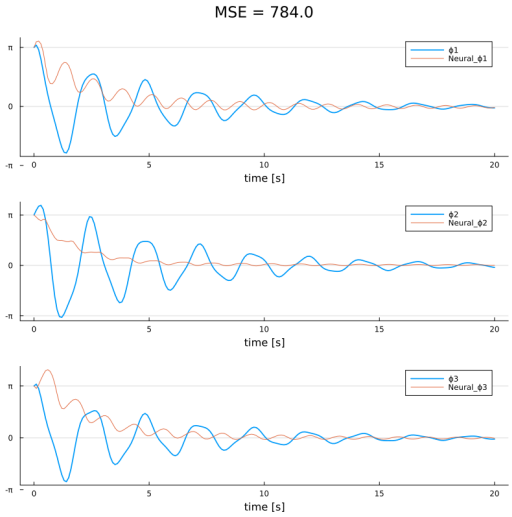

Figure 5.9: Validation NeuralODE (position)

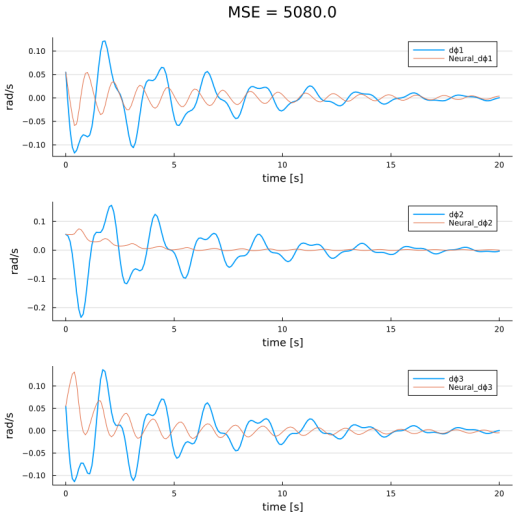

Figure 5.10: Validation NeuralODE (velocity)

5.2.2 Multiple shooting

Differently for the mini-batching case in figure 5.11 loss function presents a decreasing trend. In this case, the loss function was evaluated on the whole training data set divided in groups for the decided number of epochs. Figures 5.12,5.13

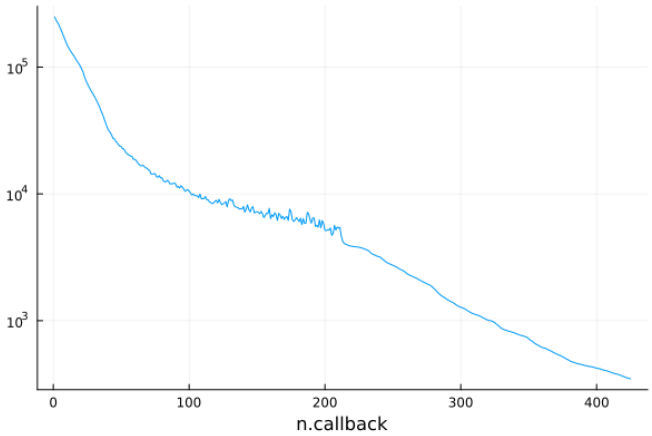

Figure 5.11: Loss Function

show a good result in the training phase. It seems the NeuralODE finds a good convergence to the training data set but the validation set proves that it isn't.

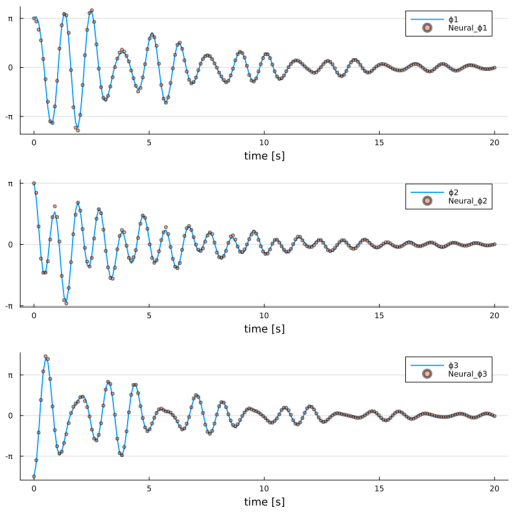

Figure 5.12: Training NeuralODE (position)

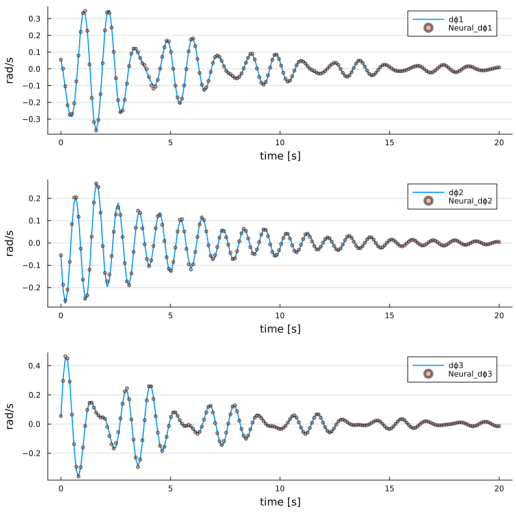

Figure 5.13: Training NeuralODE (velocity)

In the figures 5.14 and 5.15 we can notice that the trained model didn't acquire the original system dynamics.
This is an example of **overfitting**. This phenomenon occurs when the net is overtrained on a data set and loses its scalability. This event is proven when the value of the Loss function on the training set continues to decrease instead of the validation Loss that increases over training.

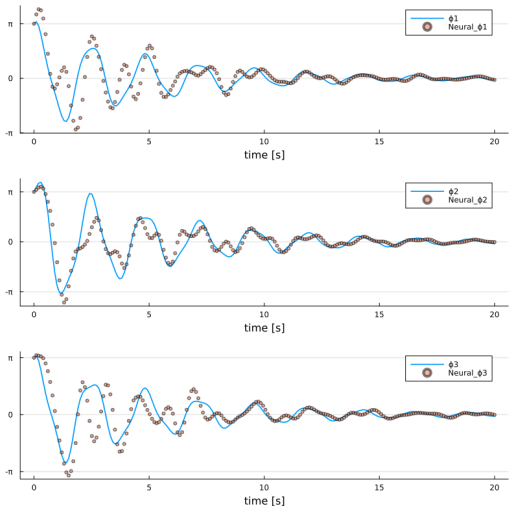

Figure 5.14: Validation NeuralODE (position)

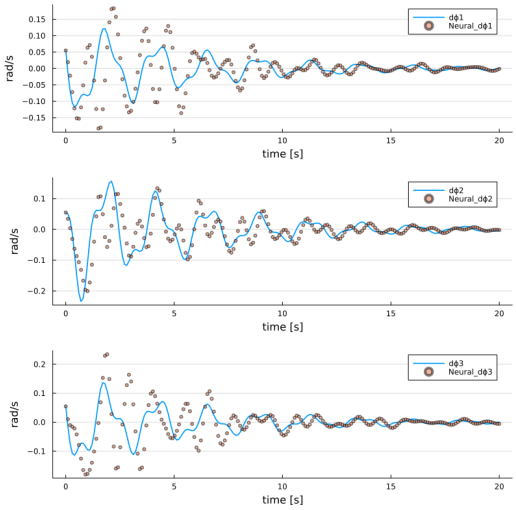

Figure 5.15: Validation NeuralODE (velocity)

5.3 Mass spring dumper

The Mass spring dumper problem differs from the previous test case. Here, we introduce our proposal for a custom NeuralODE layer.
Before this test case, we tested the NeuralODE system without external input to take our consideration about the learning ability and the different developed policies to train the network.
Here we introduce our *dynamic layer* that describes the MDS system which the network will treat as a normal layer. This aspect gives us the possibility to provide the network with an estimate of the dynamics, moreover, in this way, we could introduce to the network an external input not normally treated by the simple DiffEqFlux.jl libraries.

As in the previous cases, we investigated the two training policies, mini-batching and multiple shooting. In this case, we also decide to test the framework to consider noise in the measurement.
The resulting measurement results:

$$\begin{aligned} &y_{noise} = y + 0.02\mathcal{N}(\mu,\, \sigma^2) \\ &\mu = 0 \quad \sigma^2 = 1 \end{aligned} \tag{5.1}$$

	Loss(MSE)	
	Mini-batching	**Multiple shooting**
Training $F = 5N$	3.1021×10^{-1}	268,19
Validation $F = 10N$	$3{,}91 \times 10^{-9}$; $2{,}11 \times 10^{-4}$	5,88 ; $2{,}221 \times 10^{-1}$
Validation $F = 20N$	$1{,}591 \times 10^{-2}$; $8{,}521 \times 10^{-4}$	23,54 ; $9{,}081 \times 10^{-1}$
Validation $F = 50N$	$9{,}971 \times 10^{-2}$; $5{,}331 \times 10^{-3}$	147,15 ; $5{,}671 \times 10^{-1}$

Table 5.3: Mass Spring Dumper results

5.3.1 Mini-batching

Mini-batching training as shown in figure 5.16 has a oscillating trend as the previous test case.
This time is possible to observe three sections of decreasing trend that corresponds in a good fitting in training data set 5.22.
Despite the noisy data set, the NeuralODE has demonstrated an interesting capacity to find a good convergence for the optimization problem particularly in the second state $x_2 = \dot{x}$ where noise has more influence on measurements.

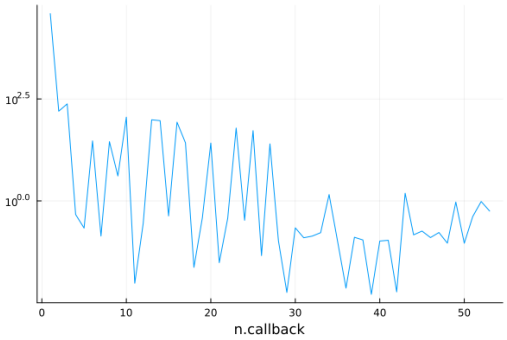

Figure 5.16: Loss Function

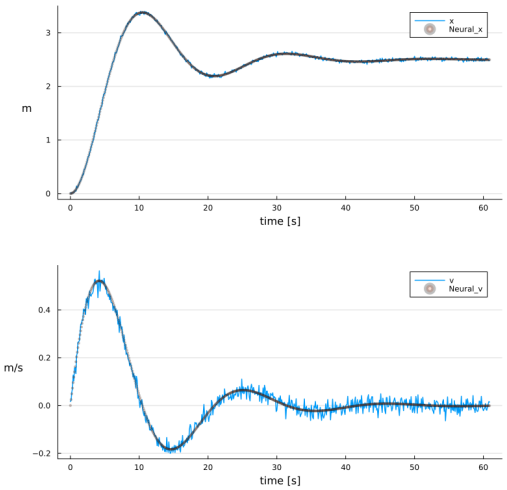

Figure 5.17: NeuralODE trainig

As a result of the training, the learned model proves to have embedded the dynamic of the original system even though the addiction of the noise in the data set.
The model developed was tested to respond in three scenarios, same initial condition $x_0 = [0,0]$ but with different input values $F_1 = 10N$,$F_1 = 20N$,$F_1 = 50N$ 5.18,5.19,5.20.

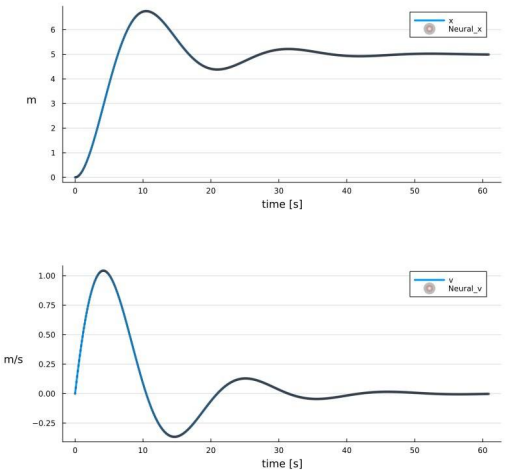

Figure 5.18: Validation $F = 20$N

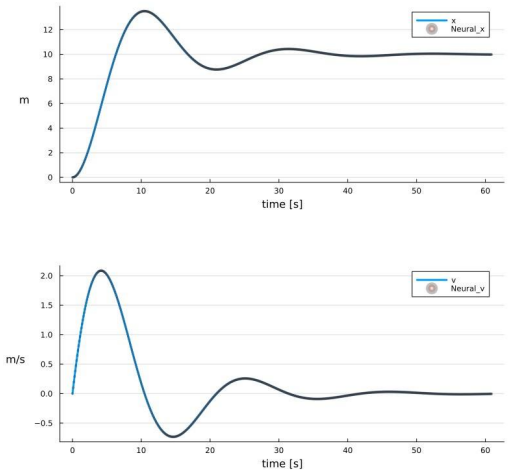

Figure 5.19: Validation $F = 20$N

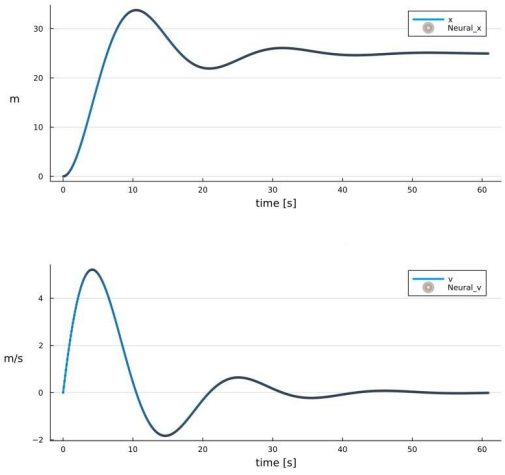

Figure 5.20: Validation $F = 50$N

5.3.2 Multiple shooting

As in the previous case, NeuralODE shows a good ability to take into account noisy data. The Loss Function trend proves a clear convergence, the *chattering* behavior at the end of the big first loss function reduction is to be attributed to the learning rate not being small enough that easily allows the function to escape from that bad local minimum.

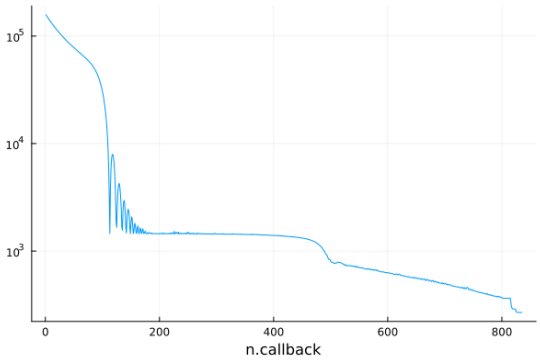

Figure 5.21: Loss Function

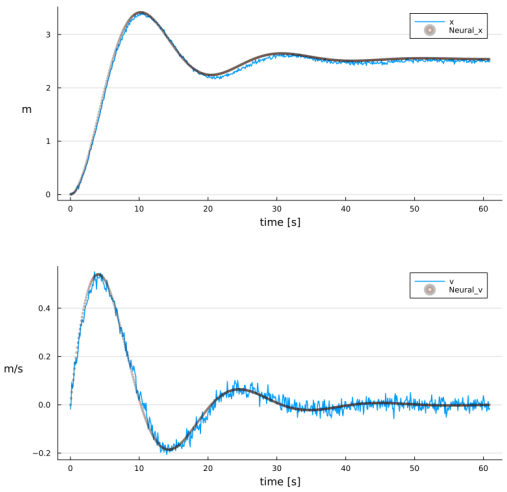

Figure 5.22: NeuralODE trainig

Also in this case to test the performances of the trained model we submitted the model to the same test as before.
Three scenarios with same initial condition $x_0 = [0,0]$ and different inputs $F_1 = 5N, F_2 = 20N, F_3 = 50N$. With respect to table 5.3 multiple shooting demonstrates a greater sensibility to the noise.

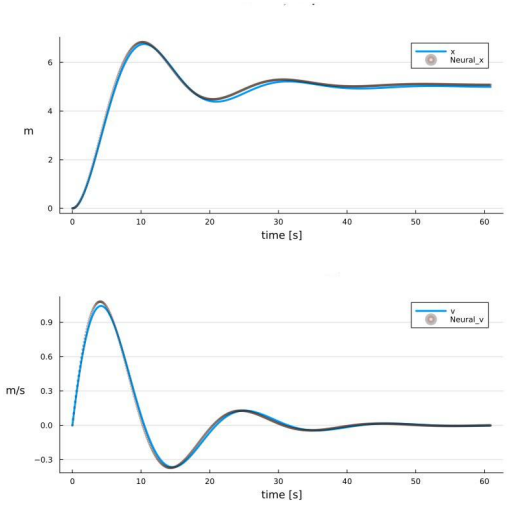

Figure 5.23: Validation $F = 10$N

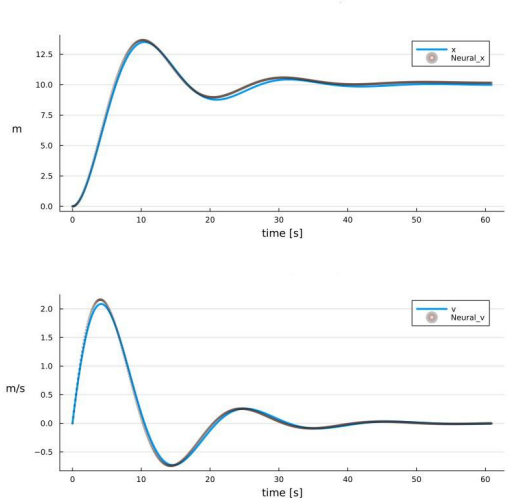

Figure 5.24: Validation $F = 20$N

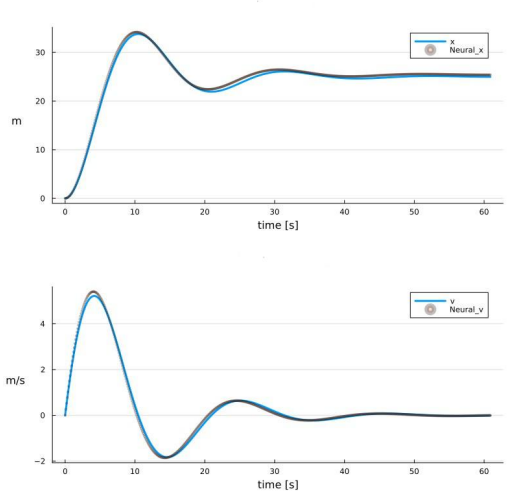

Figure 5.25: Validation $F = 50$N

5.4 Electrohydraulic actuator

The electrohydraulic actuator proved to be a challenge due to its strong multiple mechanical non-linearity. In the previous case we introduced the possibility to treat normal set of equations as an artificial neural networks layer. In this case we provided a grey box modelization proposal using the experimented *dynamic layer*, in that case we interconnected the dynamic layer to other classical fully connected ones. The intuition was to guide the ANN in the research of the function that better approximate the dynamic of the system to be learned. This strategy also involves a reduction of the dimension of the optimization problem formulated, decreasing the amount of parameters of the ANN to be established.
The ANN structure was built following the dynamic description of the system described in section 4.0.4, substituting to the unknown dynamics a fully connected layers and for the actuator dynamic an approximation given by the mass spring damper equations set. Training and validation session were coded in order to take a force F, related to the fluid pressure controlled by control unit, as input.

	Loss(MSE)	
	mini-batching	**Multiple shooting**
Training $F = 100N$	27,09 ; 1686,9	$1{,}11 \times 10^2$; $2{,}11 \times 10^2$
Validation $F = 50N$	53,09 ; 3486,9	$1{,}41 \times 10^4$; $3{,}91 \times 10^4$
Validation $F = 200N$	55,66 ; 3727,56	610,4 ; $4{,}91 \times 10^4$
Validation $F = 500N$	924,64 ; $2{,}91 \times 10^{-4}$	234,5 ; $3{,}41 \times 10^3$

Table 5.4: Electrohydraulic actuator results

5.4.1 Mini-batching

In figures 5.27 it's possible to notice not a complete adherence of the trained network with the original data set in transient phase. The NeuralODE otherwise seems to respect the steady state behavior of the system. This ANN configuration doesn't overcome the strong non linearity of the system due to the saturation condition on the spool speed.

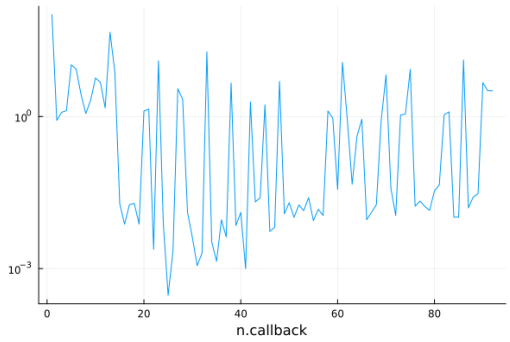

Figure 5.26: Loss function

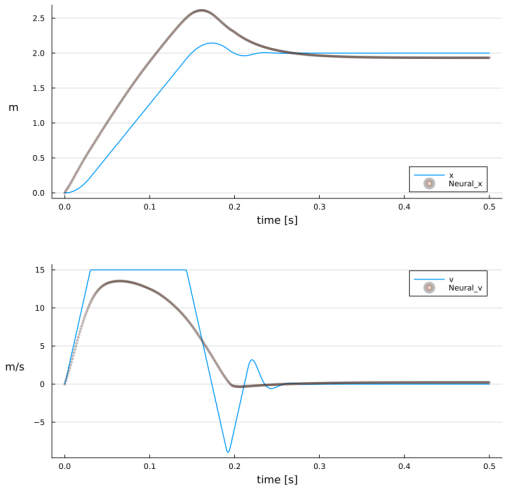

Figure 5.27: NeuralODE training

The validation test was performed subjecting, as in the previous case, the trained model for different input signals. The validation data sets describe the electrohydraulic actuator subjects to $F = [50,200,500]$. The results cannot be considered totally satisfactory, but it's interesting to notice in figures 5.28,5.29 that the model responds compatibly to the approximation learned from the data set. In figure 5.30 we notice that the input value is too far from the condition in which the ANN was trained causing a totally inconsistency in the output generated.

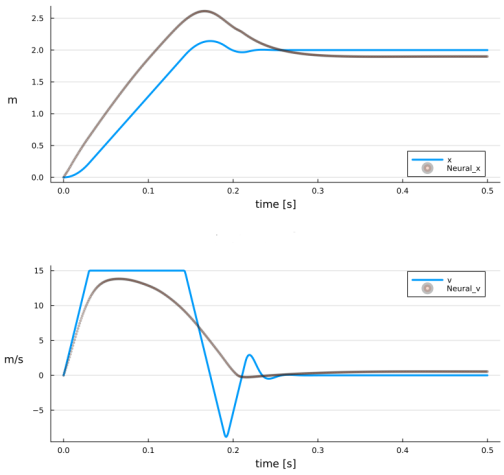

Figure 5.28: Validation $F = 50$N

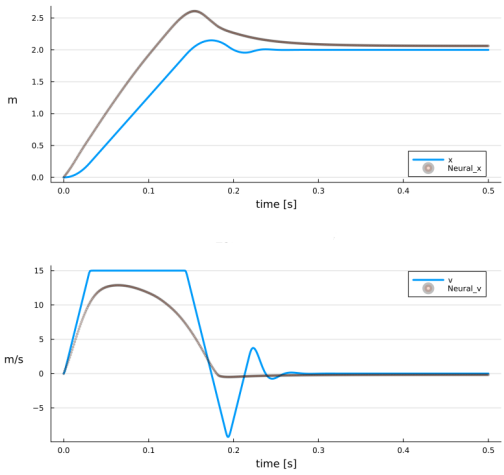

Figure 5.29: Validation $F = 200$ N

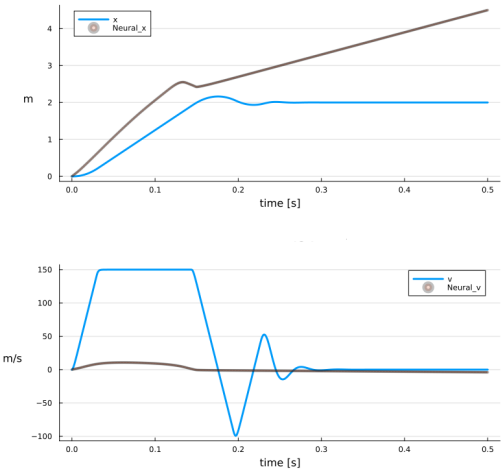

Figure 5.30: Validation $F = 500$N

5.4.2 Multiple shooting

Multiple shooting technique performed in a worse way than Mini-batching training. Figure 5.31 at velocity state, highlights the responds of the ANN. It perfectly follows the trend of the true system till the saturation value.
That behaviour in our opinion is due to the falls of the framework into a bad local minima.

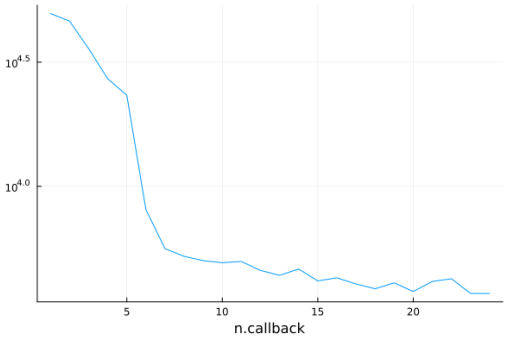

Figure 5.31: Loss function

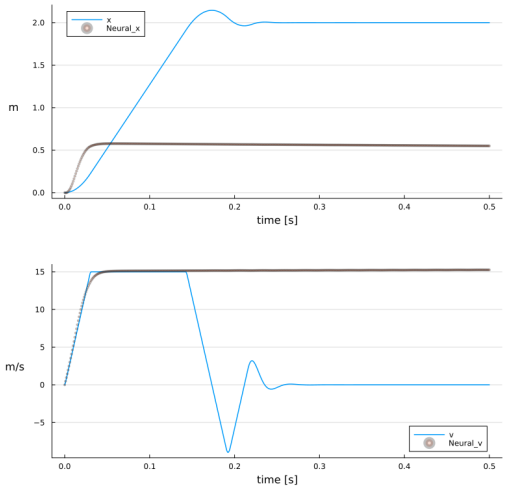

Figure 5.32: NeuralODE training

Figures 5.33-5.34-5.35 underline the bad result of the training phase of this case study. Differently from the other case study the ANN didn't acquire knowledge of the system dynamic

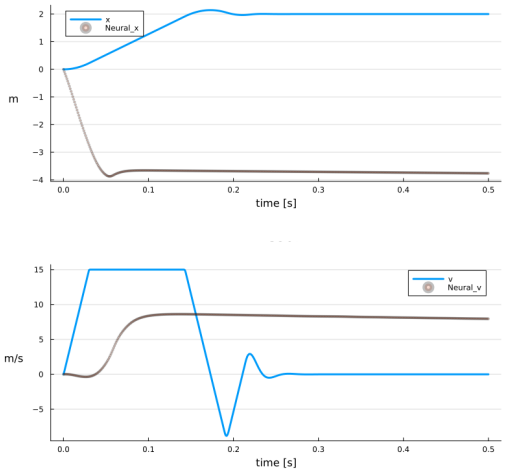

Figure 5.33: Validation $F = 50$N

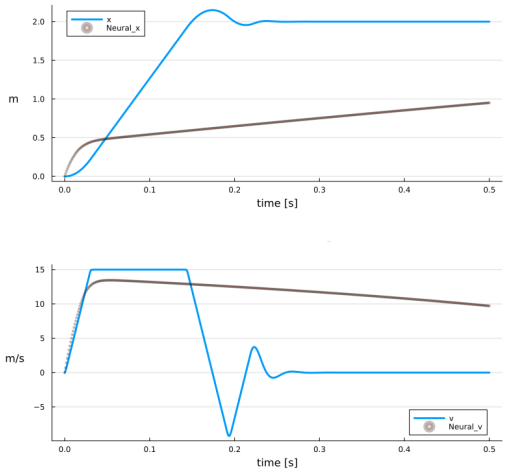

Figure 5.34: Validation $F = 200$ N

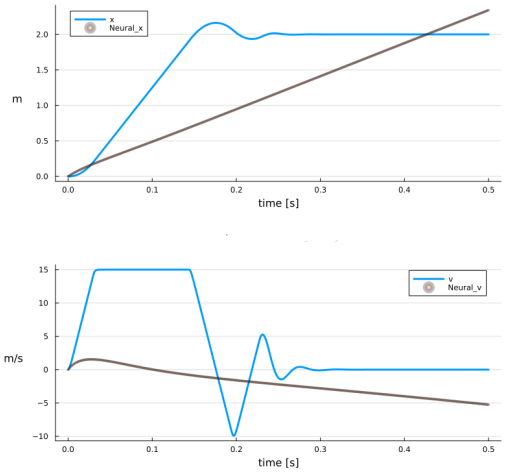

Figure 5.35: Validation $F = 500$N

Chapter 6

Conclusion

The aim of this thesis work was to explore the possibilities that Physics-encoded Neural Networks (PeNN) can provide in the context of system identification and especially in the possibility of creating *surrogate* models for Software in the Loop purposes. In particular, we focus our attention on testing the capability and limitations of one of the newest Deep Learning tools developed in the field of Universal Differential Equation **NeuralODE**.

This study aims to develop a neural network able to learn the dynamics of a system emulating the set of ordinary differential equations (ODEs) that represent the system itself. That approach allows us to simulate the different condition models even in the case we physically do not have the model or can't represent it mathematically.

The framework studied shows:

- Important features in terms of network architecture and training approach techniques.
- Allows us to easily implement PeNN architecture enveloping partial knowledge about the dynamic system to be estimated.
- Interesting ability to embed the dynamic of the system"*studied*" showing to mimic the original set of ODEs also for never trained boundary conditions. To be noticed that also for the problem with bad convergences of the loss function the networks gave an encouraging respond.

On the contrary, the tool used presents limitations when facing systems with strong non-linearity like the electrohydraulic actuator of this work or in the case of triple rotating mass with high complexity systems. This work ends with interesting proposals for future works based on the benefits and limitations observed during this study.

- **Technical upgrades**: investigation about better understanding of what the SciML ecosystem can offer and what starting from that can be developed together the help of consistent community knowledge.
- **Grey-boxing**: develop a methodology for generalising the problem of system approximation. A proposal is Polynomial Optimization Problem (POP), it could be a good tool to be implemented in support of the ANN approximation. POP does require general knowledge of the system to be analyzed and that fits the assumption of our formulated problem.
- **Constraints optimization**: focus on the optimization problem to emphasise the respect by the agent to the physical constraints of the system under study.

Bibliography

[1] Roger G. Ghanem and Masanobu Shinozuka. «Structural-System Identification. I: Theory». In: *Journal of Engineering Mechanics-asce* 121 (1995), pp. 255–264 (cit. on p. 3).

[2] L. Ljung. *System Identification: Theory for the User*. Prentice Hall information and system sciences series. Prentice Hall PTR, 1999. ISBN: 9780136566953. URL: https://books.google.it/books?id=nHFoQgAACAAJ (cit. on p. 3).

[3] Gaëtan Kerschen, Keith Worden, Alexander F. Vakakis, and Jean-Claude Golinval. «Past, present and future of nonlinear system identification in structural dynamics». In: *Mechanical Systems and Signal Processing* 20.3 (2006), pp. 505–592. ISSN: 0888-3270. DOI: https://doi.org/10.1016/j.ymssp.2005.04.008. URL: https://www.sciencedirect.com/science/article/pii/S0888327005000828 (cit. on p. 3).

[4] Jer-Nan Juang and Richard S. Pappa. «An eigensystem realization algorithm for modal parameter identification and model reduction». In: *Journal of Guidance, Control, and Dynamics* 8.5 (1985), pp. 620–627. DOI: 10.2514/3.20031. eprint: https://doi.org/10.2514/3.20031. URL: https://doi.org/10.2514/3.20031 (cit. on p. 3).

[5] Rune Brincker, Lingmi Zhang, and Palle Andersen. «Modal identification of output-only systems using frequency domain decomposition». In: *Smart Materials and Structures* 10.3 (June 2001), p. 441. DOI: 10.1088/0964-1726/10/3/303. URL: https://dx.doi.org/10.1088/0964-1726/10/3/303 (cit. on p. 3).

[6] Salah A Faroughi, Nikhil Pawar, Celio Fernandes, Subasish Das, Nima K. Kalantari, and Seyed Kourosh Mahjour. *Physics-Guided, Physics-Informed, and Physics-Encoded Neural Networks in Scientific Computing*. 2022. DOI: 10.48550/ARXIV.2211.07377. URL: https://arxiv.org/abs/2211.07377 (cit. on pp. 3–5).

[7] Salah A. Faroughi, Ana I. Roriz, and Célio Fernandes. «A Meta-Model to Predict the Drag Coefficient of a Particle Translating in Viscoelastic Fluids: A Machine Learning Approach». In: *Polymers* 14.3 (2022). ISSN: 2073-4360. DOI: 10.3390/polym14030430. URL: https://www.mdpi.com/2073-4360/14/3/430 (cit. on p. 4).

[8] Ming Jer Lee and Jui Tang Chen. «Fluid property predictions with the aid of neural networks». In: *Industrial & Engineering Chemistry Research* 32.5 (1993), pp. 995–997. DOI: 10.1021/ie00017a034. eprint: https://doi.org/10.1021/ie00017a034. URL: https://doi.org/10.1021/ie00017a034 (cit. on p. 4).

[9] Cheng Yang, Xubo Yang, and Xiangyun Xiao. «Data-driven projection method in fluid simulation». In: *Computer Animation and Virtual Worlds* 27.3-4 (2016), pp. 415–424. DOI: https://doi.org/10.1002/cav.1695. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cav.1695. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.1695 (cit. on p. 4).

[10] Zhikai Wang, Ka Gong, Wei Fan, Chao Li, and Weijia Qian. «Prediction of swirling flow field in combustor based on deep learning». In: *Acta Astronautica* 201 (2022), pp. 302–316. ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2022.09.022. URL: https://www.sciencedirect.com/science/article/pii/S0094576522004878 (cit. on p. 4).

[11] Bradley N. Bond and Luca Daniel. «Guaranteed stable projection-based model reduction for indefinite and unstable linear systems». In: *2008 IEEE/ACM International Conference on Computer-Aided Design*. 2008, pp. 728–735. DOI: 10.1109/ICCAD.2008.4681657 (cit. on p. 4).

[12] Zekarias Tadesse, K.A. Patel, Sandeep Chaudhary, and A.K. Nagpal. «Neural networks for prediction of deflection in composite bridges». In: *Journal of Constructional Steel Research* 68.1 (2012), pp. 138–149. ISSN: 0143-974X. DOI: https://doi.org/10.1016/j.jcsr.2011.08.003. URL: https://www.sciencedirect.com/science/article/pii/S0143974X11002173 (cit. on p. 4).

[13] Zhiwei Fang and Justin Zhan. «Deep Physical Informed Neural Networks for Metamaterial Design». In: *IEEE Access* 8 (2020), pp. 24506–24513. DOI: 10.1109/ACCESS.2019.2963375 (cit. on p. 5).

[14] Khemraj Shukla, Ameya D. Jagtap, James L. Blackshire, Daniel Sparkman, and George Em Karniadakis. «A Physics-Informed Neural Network for Quantifying the Microstructural Properties of Polycrystalline Nickel Using Ultrasound Data: A promising approach for solving inverse problems». In: *IEEE Signal Processing Magazine* 39.1 (Jan. 2022), pp. 68–77. DOI: 10.1109/msp.

2021.3118904. URL: https://doi.org/10.1109%5C%2Fmsp.2021.3118904 (cit. on p. 5).

[15] Ivan Depina, Saket Jain, Sigurdur Mar Valsson, and Hrvoje Gotovac. «Application of physics-informed neural networks to inverse problems in unsaturated groundwater flow». In: *Georisk: Assessment and Management of Risk for Engineered Systems and Geohazards* 16.1 (2022), pp. 21–36. DOI: 10.1080/17499518.2021.1971251. eprint: https://doi.org/10.1080/17499518.2021.1971251. URL: https://doi.org/10.1080/17499518.2021.1971251 (cit. on p. 5).

[16] Henning Wessels, Christian Weißenfels, and Peter Wriggers. «The neural particle method – An updated Lagrangian physics informed neural network for computational fluid dynamics». In: *Computer Methods in Applied Mechanics and Engineering* 368 (Aug. 2020), p. 113127. DOI: 10.1016/j.cma.2020.113127. URL: https://doi.org/10.1016%5C%2Fj.cma.2020.113127 (cit. on p. 5).

[17] Chengping Rao, Hao Sun, and Yang Liu. «Hard Encoding of Physics for Learning Spatiotemporal Dynamics». In: (2021). DOI: 10.48550/ARXIV.2105.00557. URL: https://arxiv.org/abs/2105.00557 (cit. on p. 6).

[18] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. *Fourier Neural Operator for Parametric Partial Differential Equations*. 2020. DOI: 10.48550/ARXIV.2010.08895. URL: https://arxiv.org/abs/2010.08895 (cit. on p. 6).

[19] Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, and Alan Edelman. *Universal Differential Equations for Scientific Machine Learning*. 2020. DOI: 10.48550/ARXIV.2001.04385. URL: https://arxiv.org/abs/2001.04385 (cit. on pp. 7, 18).

[20] F. Rosenblatt. *The perceptron - A perceiving and recognizing automaton*. Tech. rep. 85-460-1. Ithaca, New York: Cornell Aeronautical Laboratory, Jan. 1957 (cit. on p. 11).

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Deep Residual Learning for Image Recognition». In: (2015). URL: https://arxiv.org/abs/1512.03385 (cit. on p. 16).

[22] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. «Neural Ordinary Differential Equations». In: (2019). URL: https://arxiv.org/abs/1806.07366 (cit. on p. 17).

[23] L.S. Pontrjagin, V.G. Boltyanskii, R.V. Gamkrelidze, E.F. Mishchenko, and D.E. Brown. *The Mathematical Theory of Optimal Processes*. International series of monographs in pure and applied mathematics. Wiley, 1962. URL: https://books.google.fr/books?id=PcH9oAEACAAJ (cit. on p. 17).

[24] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980 (cit. on p. 20).

[25] Wei Zhao. «A Broyden–Fletcher–Goldfarb–Shanno algorithm for reliability-based design optimization». In: *Applied Mathematical Modelling* 92 (2021), pp. 447–465. ISSN: 0307-904X. DOI: https://doi.org/10.1016/j.apm.2020.11.012. URL: https://www.sciencedirect.com/science/article/pii/S0307904X20306648 (cit. on p. 20).

[26] URL: https://www.do-mpc.com/en/latest/getting_started.html (cit. on p. 26).

[27] Matteo Dalla Vedova, Paolo Maggiore, and Francesco Marino. «Proposal of a Fast Model-Based Prognostic Paradigm for Electrohydraulic Actuators affected by Multiple Failures». In: *WSEAS Transactions on Systems and Control* 11 (Jan. 2016), pp. 445–452 (cit. on pp. 30, 32).