Universidad de La Laguna

Facultad de Ciencias
Universidad de La Laguna

# Study of physical systems with machine learning techniques:

## Lorenz63 system

by

Víctor Manuel Naval Hernández

# Study of physical systems with machine learning techniques:

## Lorenz63 system

### Bachelor of Science in Physics


| | |
|---|---|
| Author: | Víctor Manuel Naval Hernández |
| Student ID: | 42221562W |
| Supervisor: | Dr. Albano José González Fernández |
| Second supervisor: | Luis Alonso Siverio |
| Project duration: | Oct 2022 – May 2023 |



Universidad de La Laguna

Facultad de Ciencias
Universidad de La Laguna

# Abstract

# Study of physical systems with machine learning techniques: Lorenz63 system

The dissertation presented in this document comprises the study of a physical system, in particular, the Lorenz63 system of equations in a chaotic solution range[1]. To do so, up-to-date machine learning techniques are applied. In such a framework, a description of the Lorenz63 system is presented, together with a description of the Neural Network models used in the study. Moreover, these Machine Learning models are trained in a novel approach that attempts to boost their generalisation abilities. Then, they are implemented to simulate the dynamics of the system given an initial point in phase space. The success of such simulations is assessed using chaos measuring techniques for time-series of data such as the Correlation Dimension ($D_2$) and Lyapunov Exponents ($\lambda$). In addition, not only short-range predictions but also the learning of the overall dynamic behaviour of the system is analysed and compared with a Runge-Kutta numerical integration of the equations, leading to success for some of the chosen Neural Networks. The main goal of the project is to present a guideline and example of how to confront the study of a physical system implementing Neural Networks as well as introducing some innovative alternatives for studying and solving numerical problems. In this context, Reservoir Computing is introduced as a concept and derived Neural Networks models are shown to successfully simulate the Lorenz63 system's behaviour, in agreement with [2]. Lastly, a set of conclusions—with the purpose of widening the scope of the previous discussion—wraps up the dissertation, providing the first steps to guide future research on the same lines.

---

The fulfilment of this project took place with the support of a collaboration scholarship granted by the MECD (Ministry of Education, Culture and Sport) conducted in the Department of Physics in the University of La Laguna, located in San Cristóbal de La Laguna, Tenerife, Spain.

Departamento de Física
Universidad de La Laguna

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN, CULTURA Y DEPORTE

# Resumen

# Estudio de sistemas físicos con técnicas de aprendizaje automático: Sistema Lorenz63

Este trabajo comprende el estudio de sistemas físicos, en particular el sistema de ecuaciones diferenciales Lorenz63, en un rango de soluciones caótico[1]. A tal efecto, se aplican técnicas actuales de aprendizaje automático. En esta línea de trabajo, se presenta una descripción del sistema Lorenz63 así como de los modelos de Redes Neuronales empleados en el estudio. Dichos modelos son entrenados a través de una aproximación novedosa que busca potenciar las capacidades de generalización de los mismos para, a continuación, ser implementados en la simulación de la dinámica del sistema dado un punto inicial. El éxito y calidad de dichas simulaciones se evalúa mediante técnicas diseñadas para cuantificar el caos en series temporales de datos, como son la Dimensión de Correlación ($D_2$) y los Exponentes de Lyapunov ($\lambda$). En dicha evaluación no solo se atiende a la predicción a corto plazo sino también al aprendizaje del comportamiento dinámico general del sistema, que ha mostrado ser satisfactorio para algunos de los modelos aplicados. El principal objetivo del proyecto es plantear una guía, correspondientemente ejemplificada, de cómo afrontar el estudio de un sistema físico empleando Redes Neuronales, además de introducir técnicas innovadoras que puedan ser de utilidad en la resolución de problemas numéricos. En este contexto, se menciona y aplica el concepto de reservorio computacional a través de una arquitectura específica de Red Neuronal, que muestra ser eficaz y exitosa en la tarea abordada. Este resultado concuerda con lo presentado en [2]. Para finalizar, se expone una serie de conclusiones que tratan de ampliar el ámbito de la discusión previa, aportando una base de partida para futuras investigaciones en este campo.

---

La realización de este proyecto tuvo lugar bajo el amparo de una beca de colaboración concedida por el MECD (Ministerio de Educación, Cultura y Deporte) y llevada a cabo en el Departamento de Física de la Universidad de La Laguna, emplazada en San Cristobal de La Laguna, Tenerife, España.

Departamento de Física
Universidad de La Laguna

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN, CULTURA Y DEPORTE

# Acknowledgements

*Without the constant support and help from my Supervisor, this project would have failed on the completing of the preconceived goals. My most sincere acknowledgements for Albano and his fellows that provided the best possible working environment. I also wish to extent this gratitude to my family and friends. Lastly, I wanted to acknowledge Luis for introducing me into the Machine Learning world and opened my eyes to a new field.*

---

*It is a pleasure to study and investigate knowing that there is always someone by your side willing to listen and with selfless support to supply.*

# Contents

# List of Figures

# List of Tables

# 1. Introduction: Lorenz63 System and Neural Networks

Esta primera sección introductoria asienta las bases para el posterior seguimiento del proyecto. Se comienza describiendo brevemente el estado del arte sobre Inteligencia Artificial (IA), Machine Learning (ML) y más en concreto, Redes Neuronales (RN). Seguidamente, se exponen los conceptos básicos sobre Redes Neuronales y algunos tipos y estructuras comunes que serán utilizados *a posteriori*. Para finalizar la introducción, se introduce el sistema objeto de estudio: Lorenz63. Para ello se presentan las ecuaciones de movimiento, seguidas del pertinente desarrollo para su obtención y rango de soluciones.

***Resumen***

## 1.1 Brief overview of Machine Learning and Artificial Intelligence

### 1.1.1 AI vs. ML

Artificial Intelligence and Machine Learning are concepts often use indifferently, however the latter is a branch or subcategory of the former. The University of Columbia, NY, defines **Machine Learning** as a **pathway** to Artificial Intelligence in which algorithms are used during learning to process data, gain insight and apply that learning to make better decisions. By exploring ML tools, programmers are testing the limits of how much the cognition and performance of a computer can be improved. This is a basic stone in **Artificial Intelligence** development, which ends supplying humanity with computers and robots that are able to analyse and, what is more, contextualise data in order to **provide** information and make decisions without human interference[3].

It seems that AI and ML techniques are nowadays applied in almost every aspect of human's live, nonetheless, the origins of such concepts and ideas began around the middle of the last century, closely attached to the study of the human brain, neurons and how the information travelled among them. The mathematical tools needed to found those studies were developed hundreds of years ago with the help of the greatest in calculus like Leibniz, Euler or Taylor. What pushed the implementation of such techniques was the expansion of computers since the 1950s. A short revision of how the present situation was reached is presented in the next section.

### 1.1.2 Machine Learning: Origins and State of the Art

It all started with a cell, as it has happened with several developments in science and natural history. In this case, it was a brain cell—a neuron— and how information and communication travelled and occurred between a group of them. The first description of a neuron using logic and setting the starting point for Neural Networks

was given by McCulloch and Pitts in 1943[4]. A few years later, in 1949, Donald Hebb presented a model of brain cell interactions and set the information theory basis to further developments[5]. Really close in time, Arthur Samuel presented a novel computer program for playing checkers in 1950 while working for IBM, which involved the computer in choosing the next move by measuring the chances of winning for each possible move based on a scoring function using the position of the pieces. He also designed a number of mechanisms that made possible for his algorithm to become better, laying the basic definitions and ideas[1] behind Machine Learning[6]. It is usually considered as the father of the term "Machine Learning", but it has not been able to prove such an assumption.

With respect to the brain cell study, the baton was taken by Frank Rosenblatt in 1958 when he presented the Perceptron[7], merging the concepts from Hebb and the ML learning efforts from A. Samuel. The Perceptron was intended to be an IBM machine but ended up being a computer program designed for image recognition (the program would now be known as a binary classifier). The next step was the addition of more complexity to the model, multilayer perceptrons arrived in the 1960s to stay[2], giving birth to FeedForward Neural Networks and the design and application of the backpropagation algorithm. Backpropagation had been presented in 1970 but not applied in this context[8], however, in 1982 it was first applied to a Multilayer Perceptron as it is now known[9]. In 1986 D.E.Rumelhart, *et al.* published an experimental analysis of the technique applied to Neural Networks[10].

Despite the seemingly impressive progress and possibilities of the new ML techniques, the decade of the 70s witnessed the separation between ML and AI. Mainly because AI researchers were more focused in logical, knowledge-based approaches rather than in algorithms, but also because Neural Networks were abandoned by ML and AI technicians. After the separation, ML experts worked on providing services by solving practical problems instead of training AI, running down the sector almost to extinction. It was not until 1990 when Neural Networks bloomed again as the availability of digital data increased with the growth of Internet. From then, ML was relocated as a key sector and progress started to quickly accumulate. In 1997, S.Hochreiter introduced a novel type of Neural Network called LSTM[11], which would outperform traditional speech recognition models in the coming years.

By the arrival of the new millennium, ML and AI had been shot to the moon and they still are now. With the help of big companies such as Google or Microsoft, there are very large departments destined to specifically investigate, progress and develop AI and ML, having shown unbelievable results to the world: Autonomous driving, face recognition, voice assistants, natural language processing, real-time recommendations and customisation and more that it is to come. All this work has converged in tools that are completely changing how the world is perceived or even how information is assimilated and taught: Chat GPT, Dall-E-2 O, Jenni AI,...

A sharp turn has been taken in the technology world and Physics should not be less than other scientific fields and adapt to the use of these revolutionary tools to make the best out of the already acquired knowledge and the one that will be discovered. But, is it not really complicated to use ML tools in a Physics' problem? The quick answer would be no. Nevertheless, deep knowledge of how the tools work and how the chosen system or problem is described and behaves, are crucial if plausible and useful results are desired.

[1] A. Samuel stated that computers could be programmed to learn from experience, eventually eliminating the need for more programming efforts.

[2] The own Rosenblatt introduced a perceptron with three layers but did not implement learning in all three layers.

### 1.1.3 A Use Case: Atmospheric Physics

Atmospheric Physics and Meteorology comprise one of the best use cases of ML and AI applied to Physics. Weather forecasting and the ability of providing a final forecast that pops up in the smartphones of millions of people is a complex task, but can be broken down into a few steps:

1. **Data Assimilation**: Data of the current state of the atmosphere needs to be collected and processed. The amount of meteorological data gathered every few hours all over the world is enormous and comes from very unalike sources. From weather stations on land or buoys in water, to satellites or airplanes. Combining all this information is a big challenge.

2. **Data processing**: Once the data has been collected, it has to be contextualised. Where, when and how it was obtained affects its reliability and quality. Also, it has to be merged with the rest of the data acquired and this needs of large computational and memory facilities to be achieved.

3. **Forecasting**: With the current state of the atmosphere sorted out and depicted by the data, it is used to evolve the atmospheric dynamical system using models that resemble the behaviour of the real Earth-Atmosphere couple. The main problem with the models is their size and complexity, since to give something similar to a forecast, physics systems of different scales need to be taken into account, for example, cloud micro-physics, which happens within a range of centimetres or large convective cells between latitudes, in grids with lengths of hundreds of kilometres. Both need to be considered working together. Supercomputers are indispensable in this step to be able to solve the equations of motion in a human-time scale.

4. **Estimating and correcting errors**: Once the forecast is produced by running the models in vast supercomputers, an estimation of the errors and a correction of them is compulsory in order to produce accurate results. Combining information from previous results and applying complex statistical techniques, the forecast is improved in accuracy.

5. **Managing the results and sharing them globally**: How to manage the amount of data produced and how to present that information to the final user is another not easy task. User's location and properly assigning the corresponding forecast is one of the issues, as well as storing all the new forecasts produced to use them in further improvements.

However, it is in this intricate procedure in which Machine Learning has been proved to be of the greatest help. Different algorithms and techniques are being applied in *all* the steps of weather forecasting. Managing the data and checking for anomalies depending on the geography location is now being assessed by AI tools. Physics Models that solve Navier-Stokes equations, are being faced with Graph Neural Networks that simulate the complete Earth-Atmosphere system, outperforming the former in efficiency and accuracy. Moreover, forecasts' errors are being corrected with the help of algorithms that identify patterns and apply statistical corrections, being able to learn from previous forecasts and inaccuracies. There is no single part in weather forecasting in which ML has not made his place to stay, and the results are not open to debate: faster and more accurate forecasts, innovative research and new discoveries about the behaviour of the atmosphere and even new physics that has been accounted for thanks to the implementation of such techniques. The relevance is so, that the European Centre for Medium Range Weather Forecasting

(ECMWF), in collaboration with the International Foundation on Big Data and Artificial Intelligence for Human Development (IFAB), has launched a Massive Open Online Course (MOOC) on Machine Learning in Weather and Climate during 2023. The objective is to train a wider community on the impact and use of machine learning in numerical weather and climate predictions[12].

### 1.1.4 Ingredients for a Machine Learning Project

As mentioned before, Physics should not be different from other fields like Business or the Automotive Industry in which ML models can be applied. In fact, the ingredients needed to study a problem with ML techniques are the same as they would be in any other field: knowing the problem, data, models and metrics.

- **Problem and system to study**: First of all, what is going to be studied needs to be clear. In this case, the physics involved in the system, how to describe them, approximations done and which are the relevant features and variables which are useful when describing the system. Having some insight and intuition about the system would help choosing the right ML tools and techniques, especially during the first attempts.

- **Data**: It is the source of information and the fuel of any ML algorithm. All information is extracted from the data and not any data source is valid for its study. It needs to be of a certain size, usually large, it also needs to be varied, representing the most possible behaviours of the system in question, and last but maybe the most important, it needs to be of good quality. This leads to the first step in any project of this kind, which is *data preprocessing*. Once acquired from the available sources, it needs to be treated in order to be useful. Managing missing values, different data types (boolean, categorical or numerical) or different numerical ranges that need to be scaled are some examples. If there is no knowledge about the working dataset, no success will be reached.

- **Model**: There is a huge variety of models available from where to choose. Neural Networks, Boosting Algorithms, Decision Trees, Nearest Neighbours and multiple modifications and evolutions of the basic classification and regression algorithms. Choosing the proper approach for the problem —and consequently the proper model— would make a big difference and can bring the desired results. However, it is usually not a simple decision. One can decide the type of model depending on the problem, maybe a Neural Network or a Decision Tree, but once in that group, hundreds of specific models have been designed to serve particular purposes. FeedForward Neural networks, Recurrent Neural Networks, Graph Neural Networks, Physics Informed Neural Networks or AdaBoost, XGBoost, LogitBoost as examples of these two types. The options could be endless because the most suitable one might be a combination of some of them, applied at different points during the implementation.

- **Metrics**: The measuring rule. Is the model producing good results in comparison with, for example, a physical model that numerically solves the equations? To answer this, metrics are needed. They are also called Key Performance Indicators (KPIs), since they indicate how well or bad is the model performing. As well as the models, there are different metrics depending on the type of the problem and some of them could be more appropriate than others. They usually work all the same way, comparing the results output by the model with previous results that are considered to be true. In which way the comparison

is done is what differs from one metric from the other. Classification and Regression Metrics are the two main categories. While the first are intended to measure the number of correct classifications against the incorrect ones (Confussion Matrix, Accuracy or AUC-ROC curves), the Regression ones are more related with the concept of error treatment in measures or deviation. There is a true value and a numerical output that might be close or not and metrics which measure such distance (Mean Squared Error, Mean Absolute Error or Mean Absolute Percentage Error). Nonetheless, defining or finding alternative metrics suitable for the problem is possible, as long as they fulfil their purpose and make it possible to compare model results with true results and extract conclusions.

The four above-mentioned ingredients will be considered, assessed and discussed for the case of study of this dissertation in the following sections.

## 1.2 Neural Networks in a Nutshell

Plenty of descriptions and introductions to Neural Networks (NN) or Artificial Neural Networks (ANN) can be found online or even in books[13][14][15]. However, for the purpose of presenting the inexperienced reader a brief introduction into what Neural Networks are, the very basics are now presented.

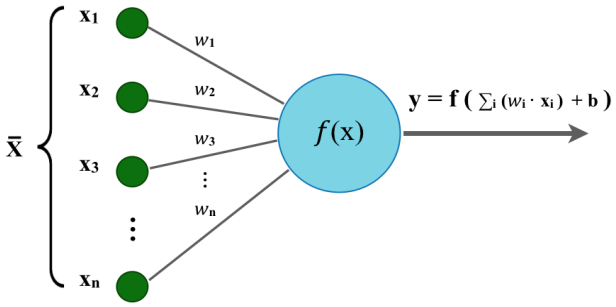
### 1.2.1 Neurons: the Basic Computational Units

As it happens with the human Brain, it is the union of all neurons what creates a functioning machine able to work with information. To be able to understand such a system, the basic units need to be firstly studied. In the case of ANN, neurons[3] are the basic units of computation which process information at the most simple level: they receive some input data, apply simple mathematical operations and provide some output data which has been transformed with respect to the input. Let's break down this with some more detail:

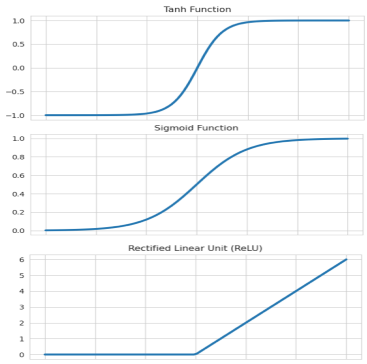
- **Input data** [ $\overline{\mathbf{X}}$ ]: The neuron can receive the input data from more than one source which could be other neurons or directly various external sources.
- **Assimilating the data**: Values coming from the inputs are combined using a weighted sum in which each weight, $\mathbf{w_i}$, is associated with an input. At this point, it can also be added some bias, $\mathbf{b}$, that shifts the values upwards or downwards.
- **Producing the output** [ $\mathbf{y}$ ]: The neuron produces an output by evaluating the assimilated data using an activation function $\mathbf{f}$. The activation function is a non-linear function which is in charge of introducing non-linearity in order to explore these kind of relations among the data. There are several types of activation functions, being the most used the Hyperbolic Tangent $[tanh(x)]$, the Sigmoid function $[\sigma(x)]$ or the Rectified Linear Unit $[max(0, x)]$.

The two parameters mentioned, the weights and biases ($w_i$, $b$) are the learnable parameters which will be updated during the training process of the model. A schematic representation of a neuron with its inputs, weights and biases, outputs and three examples of possible activation functions is pictured in Fig.1.1.

[3]They are also called nodes.




(a) Diagram of the Basic Computational Unit.




(b) Example of three activation functions: Tanh, Sigmoid and ReLU.

Figure 1.1: Scheme of a neuron and examples of activation functions $f(x)$.

### 1.2.2 Building an ANN from Scracth

#### Structure

Now that the building blocks are known, as if they were puzzle pieces, neurons can be used to build a Neural Network following some simple structure-wise recommendations:

- **Layers**: the nodes need to be organised in layers, which are nodes aligned in a sequence and often represented in vertical. Every Neural Network has two permanent layers. Those would be the *input layer* and the *output layer*. Between them, the rest of the desired layers, which are called *hidden layers* and in which the large part of the computation occurs. The data is fed into the network via the input layer, processed in between the hidden layers and then passed to the output layer from which the data arises already transformed. The Layers are connected between them and how these connections are designed would define the type of layer or even the type of NN. The basic layer type is the *dense layer* in which every neuron is fully connected with all the neurons in the next layer.

- **Input Layer's Dimension**: the number of units or neurons in the input layer is commonly defined by the input data, in particular, by the number of features in the data. For example, if the data contains cartesian positions of an object in 3D space, it would be common to use 3 neurons, as the input data would often be a three-featured vector.

- **Output Layer's Dimension**: As it happens with the input layer, the output layer should have as many units as elements in the desired output. In the above example, for the case of using the NN for predicting the global Energy of the system, the output layer could work with only one unit, since the NN should return one value given the 3D position vector.

- **Activation Function**: The selection of the activation function for all the neurons in a layer is also important. Some activation functions work better for certain type of problems[4]. However, it is usually a parameter that needs to be explored and it is a common practice to consider it within a range of functions to check which one gives better results. The activation functions

[4]ReLU is generally used in Regression problems while Tanh or Sigmoid are more used in Classification tasks.

will also impose the range in which the input data should be scaled. *Scaling* is one of the key preprocessing steps before running data through a model since activation functions work only within a certain range, often being (0,1) or (-1,1).

### Training and Learning

It has been explained how to build a ANN, however, about how they work and how they are able to learn little have been said. The main pillar on which the learning process is sustained is training. During training, the Neural Network processes the main part of the available dataset and will try to find the relations and connections hidden in the data. The goal is that after training, whenever some data is fed to the network, it will be able to produce an output that is in line with the true data used for training. Such a process of learning from the data is done by adjusting the weights and biases depending on how close the output is to the true data, using a *loss function* to measure this proximity.

Once the error inferred has been measured, the network needs to adapt its parameters in an attempt of minimising the loss and improving its results. In charge of that update is the *backpropagation algorithm* which, in a few words, would use Leibniz's chain rule to compute the gradient of the Loss Function in terms of the learnable parameters (again, weight and biases) and find the minimum for this function. It is called backpropagation because the error is obtained in the output layer and then the derivatives are calculated backwards until reaching the parameters in the input layer. The search for the minima can be done with different algorithms within which are found Stochastic Gradient Descent(SGD) or Adam. A detailed description of how backpropagation and SGD work can be found in a series of educating and brilliantly made videos done by Josh Starmer in the StatQuest[5] Youtube Channel[16]. Also a great introduction to how ANN works, with examples of these concepts, is artfully presented by C. Arrizabalaga[17].

### Implementation

Bearing this in mind and with some programming skills, one should be able to build a simple ANN. In the case of this project, everything has been programmed in `Python`. In particular, the tools selected were `Tensorflow` and `Keras`, since they are a good basis onto which to build knowledge about programming NN and ML and lots of information is available. The former is an end-to-end open source platform for Machine Learning, as defined in its documentation[18]. It provides the structure for managing data, building and training models, checking their performance and implementing them. It is suitable for learners and expert users with different interface's levels in which to have more or less control over each step of the process.


Built over Tensorflow, Keras is the high-level API, providing building blocks for the user to control the project as well as different levels of abstraction[19]. It is intended to reduce the number of user's actions to run common cases.

While Tensorflow contains and maintains the support structure for working with ML models (models, metrics, data structures, ...), Keras presents a user-friendly way of working with all these parts and tools, making possible

[5]StatQuest presents hundreds of videos about not only ML but also statistics and Data Science concepts in short and easy to follow format.

to build up a model from scratch with little knowledge on how Tensorflow works. To familiarise and learn about Tensorflow and how to create different types of models, a tutorial is referenced[20].

### 1.2.3 Feed-Forward Neural Networks

If one follows the scheme presented in the previous section, the NN built would be a Feed-Forward Neural Network or FFNN. As the name states, the information and data only travels *forward* through the network: data goes in through the input layer, crosses the layers till the output layer and goes out through it. There are no loops or connections through which the data could be fed back into the network. This types of NN present the simplest architecture and its workflow is easy to understand. An example is presented in Fig.1.2 where a FFNN with two hidden layers is depicted. The architecture would be described as having 3 neurons in the *input layer*, 5 and 4 neurons in the 1st and 2nd *hidden layers*, respectively, and 1 neuron in the final *output layer*. The *activation functions* would be $\mathbf{f_1}(\mathbf{x})$ and $\mathbf{f_2}(\mathbf{x})$ and as each neuron is connected with all the neurons in the next layer, the layers would be called *dense layers*.

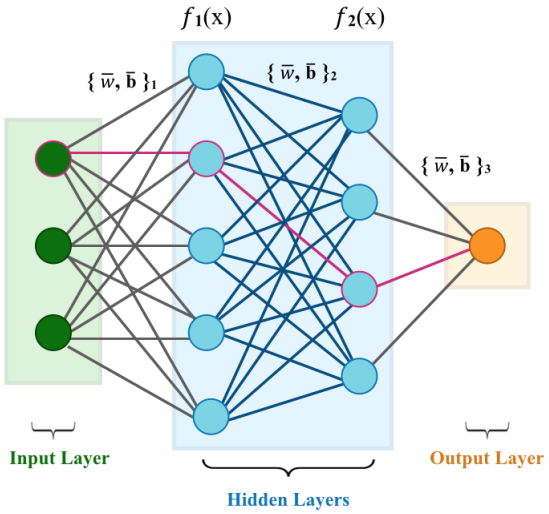



Figure 1.2: Example of a FFNN in which one of the paths that the data will follow is highlighted in pink. Weights and biases sets are also indicated for each connection between two layers.

### 1.2.4 Recurrent Neural Networks

Usually referred to as RNN, Recurrent Neural Networks differ from traditional FFNN because they are intended to work with sequential or time-series data. They introduce loops inside the workflow in such a manner that previous inputs can influence current inputs and outputs. In addition, they also share parameters across the layers, whereas FFNN had different sets of weights and biases per layer. This provides the NN with some kind of "memory". In comparison with FFNN that assume inputs to be independent, RNN work under the assumption that output depends on previous values within the sequence that has been fed in[21]. They are really powerful in language translation, speech recognition or natural language processing. Some of the most common architecture variants are LSTMs, already mentioned before, and Gated Recurrent Units or GRUs which are similar to LSTMs but with different number of components and slightly different functioning. As LSTMs will be implemented in this project, a more detailed explanation is further presented.

### Long Short-Term Memory Neural Networks

Recurrent Neural Networks use loops to deal with sequential data. Instead of passing the first element of the input sequence straight to the output, the first element is passed through a loop and combined with the second, then the result goes through the next loop and combines with the third... and this keeps going until the end of the input sequence, where the data finally passes to the output. However this recurrent use of the loops provokes the *vanishing/exploding gradient* problem, which, in a few words, can be explained as the uncontrolled increase or decrease of the values going through the network due to its recursive multiplication with the weights associated to each loop[6]. To solve this issue, LSTMs contain two separate paths for the information, one for Long-Term Memory and one for Short-Term Memory. Data going through the Long-Term path suffers only slight changes and flows easily to the output. The Short-Term path is more complex and the data is transformed and combined sequentially with previous data from the sequence using different weights and biases.

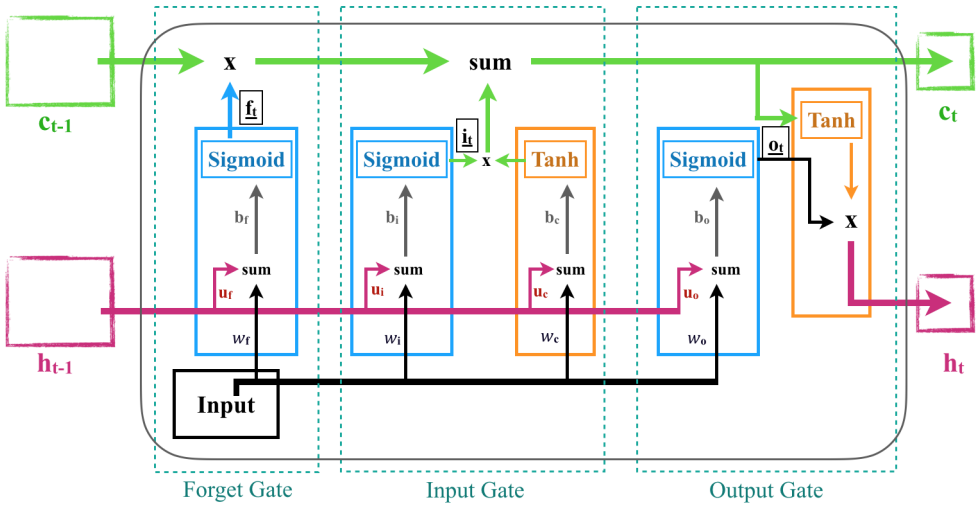



Figure 1.3: LSTM cell with all individual elements. Notice that there are two sets of four weights, the ones associated with the input ($w$, in black) and the ones associated with the hidden state coming from the previous cell ($u$, in red).

All these elements are combined in a basic unit called *cell*. The LSTM cell is built on three *gates* that manage the information that is contained and goes in and out of both memory paths. Inside the gates there are two types of *activation functions*, *sigmoid* and *tanh*, which determine what is forgotten or transformed, respectively. Each of these cells has two outputs, one corresponding to the Long-Term Memory and one corresponding to the Short-Term Memory which are called *cell state* and *hidden state*, respectively. This basic unit is the construction brick for LSTMs and is shown in Fig.1.3. The equations (1.1) describe the cell functioning:

$$\begin{aligned} f_t &= \sigma_g(W_f \times x_t + U_f \times h_{t-1} + b_f) \\ i_t &= \sigma_g(W_i \times x_t + U_i \times h_{t-1} + b_i) \\ o_t &= \sigma_g(W_o \times x_t + U_o \times h_{t-1} + b_o) \\ c_t &= f_t \cdot c_{t-1} + i_t \cdot \sigma_c(W_c \times x_t + U_c \times h_{t-1} + b_c) \\ h_t &= o_t \cdot \sigma_h(c_t) \end{aligned} \tag{1.1}$$

[6]For weights greater than one, the value will grow radically and vice-versa, for values lower than 1, it will quickly go to zero.

This set of equations can be followed with the help of Fig.1.3 and contain:

1. The *activation functions* tanh and sigmoid ($\sigma_c$ and $\sigma_g$).
2. The *weight matrices* for the *hidden state* and *input data* ($U_f$ and $W_f$).
3. The previous *cell* and *hidden state* ($c_{t-1}$ and $h_{t-1}$).
4. The *input data* ($x_t$) and the *biases* for each gate ($b_j$)

Each of these cells will process one time-step coming from the sequence. Hence, if the data is made of samples, for example, of three time-steps, three basic cells would be needed together one after the other. As a visual example, a LSTM with one layer for data of this type (three time-steps) and three neurons is depicted in Fig.1.4. How each individual cell is connected to the adjacent one is seen in the picture.

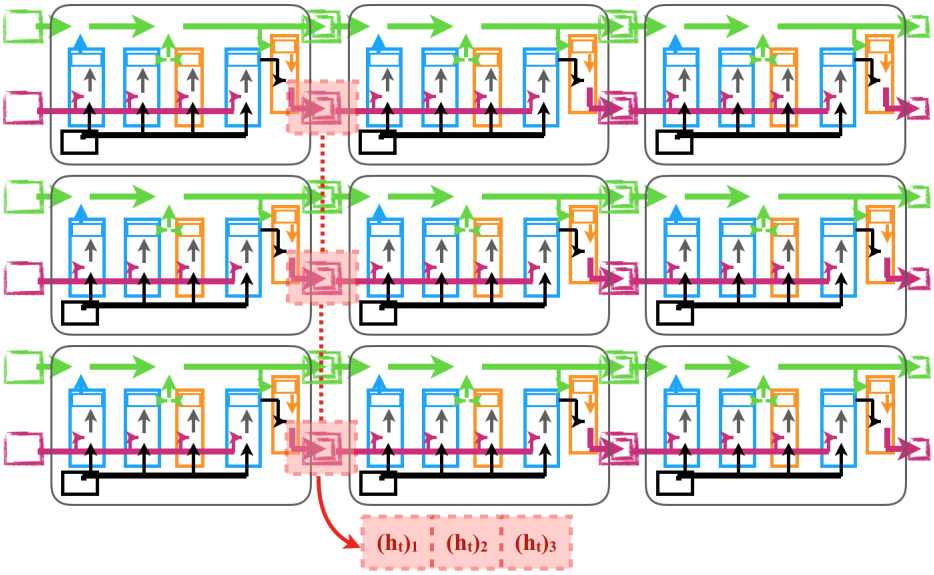



Figure 1.4: Example of a three neuron LSTM layer for working with sequence of three time instants. Highlighted in red is the hidden state vector, which contains as many hidden states as neurons in the layer.

In terms of functioning, LSTMs are trained as any other NN, however, having more trainable parameters, the training is heavier in terms of computation and takes longer.

### Echo State Networks

Echo State Networks (ESN) were designed as an alternative RNN architecture which was looking for a simpler and more efficient training process for RNNs. The concept had been discussed in the 90s but it was translated and adapted to the Machine Learning context in 2002 by Herbert Jaeger[22]. The basic structure consists in a hidden layer or *reservoir* of sparsely connected neurons in which the weights and connectivity scheme is fixed and is randomly assigned when creating the model. Then, an output layer is added and the connections between the output neurons and the reservoir are the only parameters that are modified during training. This fact makes the training processes lighter in terms of computation and possible to be done using quadratic regressions like Least Squares Fit, Ridge Regression or Lasso Regression, avoiding the tedious Backpropagation algorithm. Despite looking simple, the dynamic of the reservoir provides the non-linearity needed to be able to efficiently learn complex time series and datasets. A schematic representation of an ESN is shown in Fig.1.5.

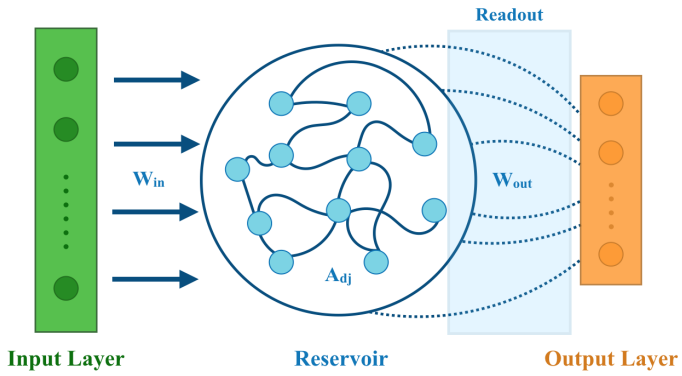



Figure 1.5: ESN structure scheme.

Nowadays, ESNs are found under the more general concept of Reservoir Computing, which is a computational framework derived from RNN. Such systems map input signals into a higher dimensional and non-linear space, the reservoir, to then translate the responses induced in each neuron by the input signal using a simple readout mechanism which is trained to map the state of the reservoir to the desired output.

When building an ESN, one can configure how the reservoir is created in terms of number of neurons, connectivity between them or the inertia that previous states have. The readout can also be customised regarding the fitting algorithm that uses or the regularisation parameter in the case of Lasso or Ridge Regression. Some other aspects of the model can be controlled, like how the input and output weights are initialised. Together, input, reservoir and readout can be connected in sequence in the basic configuration, with the data travelling from the input to the reservoir and then to the readout. However, more complex flows can be designed by the addition of loops between readout and reservoir as well as connecting more than one readouts or reservoirs.

While the rest of the models are implemented with the mentioned tools (`Tensorflow` and `Keras`), the reservoirs are going to be built using the `Python` module `ReservoirPy`[23], which provides an easy way of constructing and using this kind of RNN.

## 1.3 Lorenz System

The Lorenz system is a set of three Ordinary Differential Equations (ODEs) which are deterministic and non-linear. It was originally proposed by Edward Lorenz[7], Ellen Fetter and Margaret Hamilton back in 1963 while studying a simple mathematical model for atmospheric convection in a two-dimensional layer which was uniformly heated from below and similarly cooled from above[1]. The system in question is presented below:

$$\begin{cases} \dot{X} = \sigma \cdot (Y - X) \\ \dot{Y} = X \cdot (\rho - Z) - Y \\ \dot{Z} = X \cdot Y - \beta \cdot Z \end{cases} \tag{1.2}$$

where unknowns $X$, $Y$ and $Z$ are related with physical magnitudes of the system of study and so are the dimensionless parameters $\beta$, $\rho$ and $\sigma$. Their exact meaning and definitions can be found in the process —which is not straightforward— through which (1.2) is derived. While the equations look rather simple, a diverse spectrum of solutions is obtained depending on the values of the parameters, which define the behaviour of the variables in the different situations. From simpler periodic solutions to aperiodic or transient periodic and chaotic solutions, Lorenz system is the classical example of non-linear set of ODEs and it has been widely studied since its discovery

[7]Edward Norton Lorenz was an American mathematician and meteorologist who found the basis of atmospheric weather predictability and computer-aid studies and simulations of atmospheric physics. He is also known due to his contributions to the theory of modern chaos.

in 1963 in the frame of non-linearity and chaos theory. Several versions considering higher dimensions and reformulations of the equations have appeared over the years, including a 1996 iteration by the own Lorenz and others authors[24][25].

### 1.3.1 Derivation of the Equations

In order to reach the final set of equations, several steps and assumptions need to be done. A two-dimensional problem in the atmosphere is considered, in which thermal convection occurs between two parallel horizontal plates. The initial description of such a system would be represented by the momentum equation for viscous fluids in the most general situation:

$$\rho\frac{D\mathbf{v}}{Dt} = \mathbf{F_{vol}} - \nabla p + \mu\Delta\mathbf{v} + + \left(\frac{\mu}{3} + \lambda\right)\mathbf{grad\ div}(\mathbf{v}) \tag{1.3}$$

where $\mathbf{v}$ is the velocity field, $\rho$ the density of the fluid, $p$ the pressure and $\mathbf{F_{vol}}$ the volume forces acting on the system. Moreover, $\mu$ and $\lambda$ are the first and second coefficients of viscosity, being $\mu$ often referred to as the *dynamic* viscosity coefficient. Assuming the fluid to be incompressible and considering buoyancy as the unique volume force acting on the system, Navier-Stokes equations appear naturally. Choosing $x$ and $z$ as the directions of study and expanding the Lagrangian derivative[8], the decomposed Navier-Stokes system of equations:

$$\rho\frac{D\mathbf{v}}{Dt} = \rho\mathbf{g} - \nabla p + \mu\Delta\mathbf{v} \tag{1.4}$$

$$\rho\frac{\partial v_x}{\partial t} + \rho\mathbf{v}\cdot\nabla v_x = -\frac{\partial p}{\partial x} + \mu\nabla^2 v_x \tag{1.5}$$

$$\rho\frac{\partial v_z}{\partial t} + \rho\mathbf{v}\cdot\nabla v_z = -\rho g - \frac{\partial p}{\partial z} + \mu\nabla^2 v_z \tag{1.6}$$

From this point, some assumptions and reexpressions will take place. One of them will be applying the Boussinesq approximation, considering that density variations are only of relevance in the buoyancy term. The Boussinesq approximation is a widely known and used assumption applied in several physics studies for fluids since it provides a simple scheme suitable for a first approach to a variety of problems in fluid dynamics. However, it is often the first assumption dropped from the theoretical framework when searching for finer details of the system of study. In addition, it will be needed to consider the thermal effect of the heating in the lower plate of the layer and for that, a thermal forcing term is introduced. Considering the initial non convective state (motionless), the temperature variation in the layer will closely resemble a linear profile, which can be assessed as

$$T(x,z,t) = T_{bot} - \frac{z}{h}\delta T \tag{1.7}$$

being $z$ a certain height, $h$ the layer width and $\delta T = (T_{top} - T_{bot})$, with $T_{top}$ and $T_{bot}$ the values at the top and bottom of the layer respectively. Once convective motions have appeared, the temperature profile will departure from this linearity and, in fact, it is this departure what is measured and introduced in the equation through the function $\tau(x,z,t)$:

$$\tau(x,z,t) = T(x,z,t) - T_{bot} + \frac{z}{h}\delta T \tag{1.8}$$

[8]Details for $\frac{D\mathbf{v}}{Dt}$ can be found in Appendix A.

It can be proved[26] that (1.8) fulfils the equation for thermal diffusion[9] giving an extra equation for the system of study:

$$\boxed{\frac{\partial \tau}{\partial t} + (\mathbf{v}\cdot\nabla)\tau = D_T \cdot \nabla^2 \tau} \tag{1.9}$$

The next step is to account for the density variations in the layer, which can be related with temperature variations through the thermal expansion coefficient $\alpha = -\frac{1}{\rho_0}\frac{\partial \rho}{\partial T}$, being $\rho_0$ the density value for $\rho(T) = \rho(T = T_{bot})$. The full function for the density is:

$$\rho(T) = \rho_0 - \alpha\rho_0\left[-\frac{z}{h}\delta T + \tau(x,z,t)\right] \tag{1.10}$$

Introducing this in the $z$-component of Navier-Stokes equation (1.4) and assuming now that the perturbations in density are only of relevance in the buoyancy forces (Boussinesq approximation), the equation is expressed as:

$$\rho_0\frac{\partial v_z}{\partial t} + \rho_0\mathbf{v}\cdot\nabla v_z = -\rho_0 g - \alpha\rho_0\frac{z}{h}\delta T - \frac{\partial p}{\partial z} + \alpha\rho_0 g\tau + \mu\nabla^2 v_z \tag{1.11}$$

Since the first three terms on the right are zero when no convective motion is present (mind that when no motion is present, no departure from the linear temperature profile exists), it is possible to introduce an effective pressure gradient that also equals to zero when the fluid is motionless. This effective gradient is detailed below:

$$p' = p + \rho_0 g z + \alpha\rho_0\frac{z^2}{h}\frac{\delta T}{2} \quad ; \quad \frac{\partial p'}{\partial z} = \frac{\partial p}{\partial z} + \rho_0 g + \alpha\rho_0\frac{z}{h}\delta T \tag{1.12}$$

Making use of it, the 2D system of equations states:

$$\boxed{\begin{aligned} \frac{\partial v_x}{\partial t} + \mathbf{v}\cdot\nabla v_x &= -\frac{1}{\rho_0}\frac{\partial p'}{\partial x} + \nu\nabla^2 v_x \qquad (1.13)\\ \frac{\partial v_z}{\partial t} + \mathbf{v}\cdot\nabla v_z &= -\alpha g\tau - \frac{1}{\rho_0}\frac{\partial p'}{\partial z} + \nu\nabla^2 v_z \qquad (1.14) \end{aligned}}$$

where the *kinematic* viscosity or *momentum diffusivity* $\nu = \frac{\mu}{\rho}[\frac{J}{Kg}s]$ has been introduced.

### 1.3.2 Dimensionless Equations

In order to have a better understanding of the equations and how they describe the system, a dimensionless version is searched for. Through this procedure, important parameters controlling the behaviour of the system will arise and the dependence with $\delta T$ will be removed. It will also help to develop intuition about the balance between the different terms in the motion equations. Each variable is scaled by its range in which it can vary and re-labelled with a *dash*-variable:

- Time: $t' = \frac{D_T}{h^2}t$, where $\frac{h^2}{D_T}$ is the typical time for thermal diffusion to occur over a distance $h$.
- Distances: $x' = \frac{x}{h}$ ; $z' = \frac{z}{h}$
- Temperature: $\tau' = \frac{\tau}{\delta T}$
- x-Velocity: $v_x' = \frac{x'}{t'} = \frac{D_T}{h}v_x$
- z-Velocity: $v_z' = \frac{z'}{t'} = \frac{D_T}{h}v_z$
- Laplacian operator: $\nabla'^2 = h^2\nabla^2$

[9] $\frac{\partial T}{\partial t} + (\mathbf{v}\cdot\nabla)T = D_T\cdot\nabla^2 T$ ; $D_T$ = thermal diffusion coefficient.

Introducing the new variables and multiplying by the factor $\frac{h^3}{\nu D_t}$:

$$\frac{D_T}{\nu}\left[\frac{\partial v_x'}{\partial t'} + \mathbf{v}' \cdot \nabla' v_x'\right] = -\frac{h^2}{\nu D_T \rho_0}\frac{\partial p'}{\partial x'} + \nabla^2 v_x' \tag{1.15}$$

$$\frac{D_T}{\nu}\left[\frac{\partial v_z'}{\partial t'} + \mathbf{v}' \cdot \nabla' v_z'\right] = -\frac{\alpha g \delta T h^3}{\nu D_T}\tau' - \frac{h^2}{\nu D_T \rho_0}\frac{\partial p'}{\partial z'} + \nabla'^2 v_z' \tag{1.16}$$

At this point it is possible to introduce some dimensionless parameters, in particular, the Prandtl number and the Rayleigh number.

1. **Prandtl number** $\sigma$: It is the ratio between *kinematic* viscosity and the thermal diffusion coefficient, which compares the dissipation of energy due to friction (shear flow) and due to heat (thermal energy flow). Its definition is $\sigma = \frac{\nu}{D_t}$
2. **Rayleigh number** $R$: It is a dimensionless measure of the difference in temperatures $(T_{top} - T_{bot})$ which resembles a balance between the tendency to rise due to buoyant force associated with thermal expansion or the tendency to dissipate energy due to viscosity and thermal diffusion. Its definition is: $R = \frac{\alpha g h^3 \delta T}{\nu D_T}$

Moreover, as it had not appeared by itself earlier, a dimensionless pressure is now introduced and defined through $\Pi = \frac{p' h^2}{\nu D_T \rho_0}$. Gathering everything together, the final system of three equations is complete:

$$\begin{aligned}
&\frac{1}{\sigma}\left[\frac{\partial v_x}{\partial t} + \mathbf{v} \cdot \nabla v_x\right] = -\frac{\partial \Pi}{\partial x} + \nabla^2 v_x \qquad && \frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla \tau - v_z = \nabla^2 \tau \\
&\frac{1}{\sigma}\left[\frac{\partial v_z}{\partial t} + \mathbf{v} \cdot \nabla v_z\right] = -R\tau - \frac{\partial \Pi}{\partial z} + \nabla^2 v_z
\end{aligned}$$

Dashed notation has been removed for the sake of simplicity.

### 1.3.3 Reduced Equations

The next step in obtaining the Lorenz system as stated in (1.2) is out of the scope of this text in terms of complexity and mathematical procedures. As some of the procedures are commonly used in fluid dynamics, the sequence will be briefly commented.

First, once the information of the system is contained in the three before-stated equations, it is compressed into a *stream* function, $\Psi(x, z, t)$, which is defined as $v_x = -\frac{\partial \Psi(x,z,t)}{\partial z}$ ; $v_z = \frac{\partial \Psi(x,z,t)}{\partial x}$. This provides a system of equations for $\tau$ and $\Psi$.

In second place, a Fourier expansion is applied, rewriting $\tau$ and $\Psi$ as a series of sines and cosines. However, it is necessary to reduce the infinite set of ODEs that appear from the Fourier expansion. For this purpose, a Galerkin Truncation is applied using boundary conditions for $\tau$ and $\Psi$[10,11] . This limits the possible sine and cosine terms in the solutions, giving for $\tau$ and $\Psi$:

$$\begin{cases}
\tau(x, z, t) &= T_1(t) \sin(\pi z) \cos(ax) - T_2(t) \sin(2\pi z) \\
\Psi(x, z, t) &= \Psi(t) \sin(\pi z) \cos(ax)
\end{cases}$$

[10] $T_{bot}$ and $T_{top}$ are fixed due to linear profile, hence $\tau(z = 0, 1) = 0$

[11] As the fluid is retained within the plates, $\frac{\partial v_x}{\partial z}|_{z=0,1} = 0$. Also assuming no vertical motion in the boundary plates, $v_z(z = 0, 1) = 0$.

where $T_1(t)$ contains the difference in temperature between upper and downward moving parts of the convective cell and $T_2(t)$ the deviation from linear behaviour in the centre of the convective cell in terms of $z$. The parameter $a$ is a coefficient of the Fourier expansion to be determined.

After some manipulations and trigonometric operations, a final change in variable is needed, introducing the $X$, $Y$ and $Z$ appearing in the original system (1.2):

$$X(t) = \frac{a\pi}{(\pi^2 + a^2)\sqrt{2}}\Psi(t) \quad ; \quad Y(t) = \frac{r\pi}{\sqrt{2}}\Psi(t) \quad ; \quad Z(t) = r\pi T_2(t) \tag{1.17}$$

where $r$ is the reduced Rayleigh number $r = \frac{a^2}{(\pi^2+a^2)^3}R$. Introducing a new parameter labelled $\beta = \frac{4\pi^2}{(a^2+\pi^2)}$ and using $\rho$ instead of $r$, the expression in (1.2) is reached:

$$\begin{cases} \dot{X} = \sigma \cdot (Y - X) \\ \dot{Y} = X \cdot (\rho - Z) - Y \\ \dot{Z} = X \cdot Y - \beta \cdot Z \end{cases}$$

From now on, this system of ODEs will be referred to as **Lorenz63**, in agreement with the bibliography that identifies it with respect to other versions, such as Lorenz96.

### 1.3.4 Spectrum of Solutions for Lorenz63 System

The main advantage of the system of equations (1.2) being dimensionless is that its behaviour can be described almost entirely by the values of the three parameters: $\sigma, \rho$ and $\beta$. Depending on those, the solutions for the system will be completely different, comprising from chaotic and transient chaotic solutions, to simpler cases where trajectories converge to a single point when time evolves. A thorough study of this spectrum of possible solutions has been and it is still being done [27][28][29]. However, to understand the different possibilities that arise from the equations, some mathematical concepts are needed such as the definition of attractors or bifurcations. Some of them are presented in the Appendix B. As an example, the solutions for four different sets $(\sigma, \rho, \beta)$ are presented in Fig.1.6.

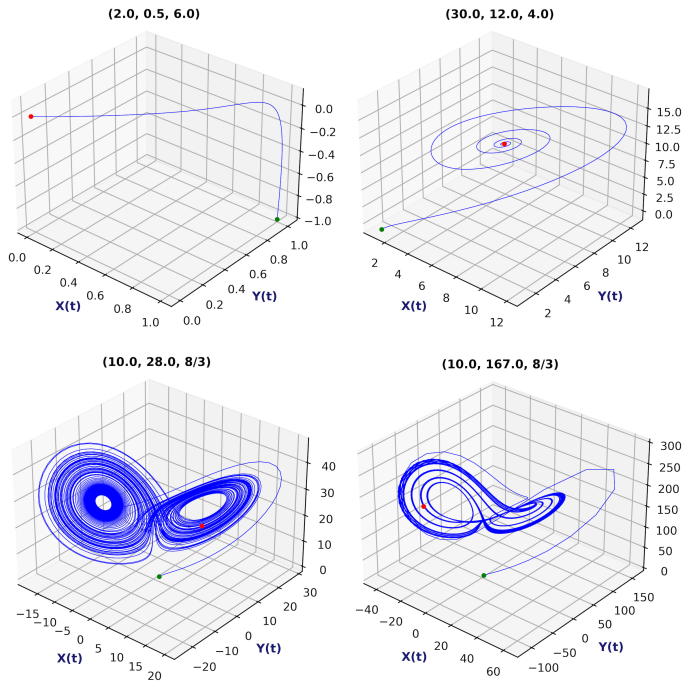



Figure 1.6: Phase Space for 4 different sets of the parameters $(\sigma, \rho, \beta)$

Similar graphics to those in Fig.1.6 are often found in the bibliography showing the phase space of solutions, constructed by representing the values of the three coordinates $[X(t), Y(t), Z(t)]$ in a 3D plot. The trajectories in this space consist of the sequence of states $\overline{\mathbf{S}} = \{\mathbf{X_1}, \mathbf{X_2}, \mathbf{X_3}, ...\}$ ; $\mathbf{X_i} = [X(t_i), Y(t_i), Z(t_i)]$ explored by the system. A globally stable solution is shown in the upper left corner, a case with a single attractor in the upper right and two chaotic solutions with two attractors each can be seen in the lower line of Fig.1.6.

Over the years, two particular cases that provide chaotic solutions have been the object of study. Lorenz originally focused on $(\sigma = 10.0, \rho = 28.0, \beta = 8/3)$, shown in Fig.1.6, but a large number of papers and publications related with the dynamics of the Lorenz system have worked with the set $(\sigma = 16.0, \rho = 45.92, \beta = 4.0)$, which provides a similar solution regarding the number of attractors and topology but different amplitudes[12]. However, the entirety of this project is based on the Lorenz63 system with $(\sigma = \mathbf{10.0}, \rho = \mathbf{28.0}, \beta = \mathbf{8/3})$. A detailed look over the three variables' time-series, $X(t)$, $Y(t)$ and $Z(t)$ for this set is framed in Fig.1.7:

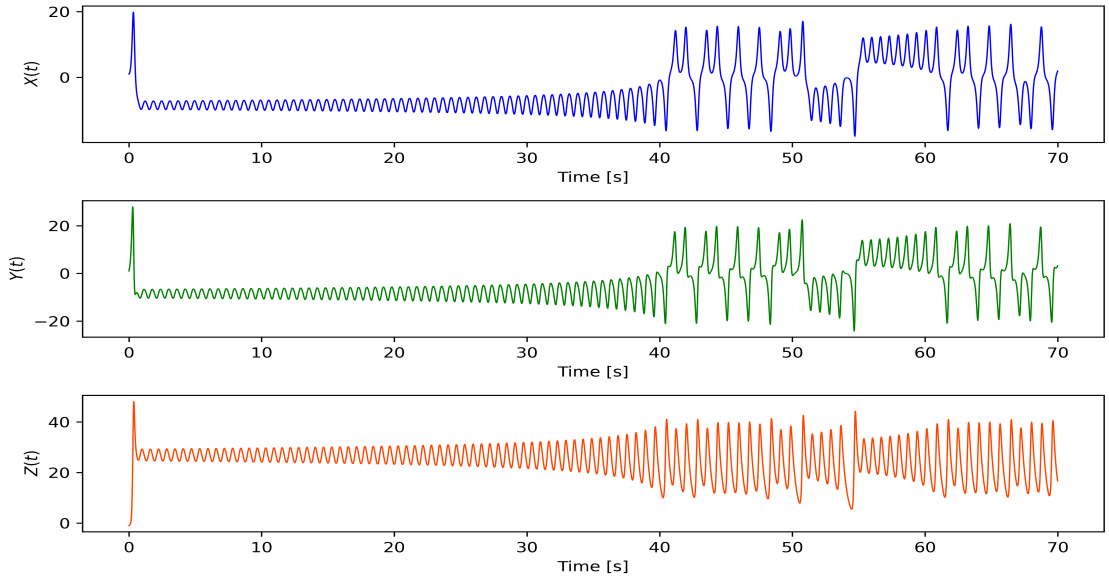



Figure 1.7: Time-series for the three variables corresponding to $(\sigma = 10.0, \rho = 28.0, \beta = 8/3)$ within a 70 secs. integration.

[12]Example shown in Appendix C

# 2. Methodology

La presente sección aborda la metodología del proyecto, describiendo con detalle los diferentes elementos necesarios para el estudio. En primer lugar, se presenta el conjunto de datos empleados en el entrenamiento. Seguidamente, se aporta una descripción completa de las arquitecturas usadas para cada tipo de Red Neuronal y una explicación del proceso de entrenamiento y ajuste de las mismas. Para finalizar, se presentan, teórica y computacionalmente, las métricas empleadas para la evaluación y comparación de los modelos.

***Resumen***

## 2.1 Training Dataset

The dataset used for training the different models has been obtained by integration of the equations (1.2) using a Runge-Kutta[1] fourth order method improved with Richardson's extrapolation method[30][31]. A time step of 0.01 was chosen and fifty one individual integrations of $10^4$ iterations, departing each from a different initial point, were performed. In total, the dataset is composed of $51 \cdot 10^4$ samples which are then split into train, test and validation datasets covering 88%, 2% and 10% respectively. The test set is of such small size because it is made just by taking the last individual integration of $10^4$ samples to be able to test the model with a complete and independent run for certain initial conditions.

In terms of statistics, each individual integration could not be considered a set of independent-identically distributed (IID) samples since each value has been obtained from the previous one through an integration method. Indeed, they are more closely identified with Markov sequences in which it can be assumed that each value depends only on the state in the previous time instant. However, while both cases are taken as assumptions in order to simplify the mathematical treatment underlying the computation of statistical analysis techniques like averages or probabilities, they are not often the real distribution of the data.

Once the data has been obtained, it is scaled between [0,1] or [-1,1] depending on the model and its activation function. Scaling is compulsory before feeding the models since Neural Networks work weighting the data run through it with weights $w \in (0, 1)$.

### 2.1.1 Training Process

With the dataset, models are trained carefully managing the interfaces between different integrations, since they are packages of data completely independent of each other. The idea behind the use of such a dataset is to provide the models with information from a variety of cases. This way, the generalisation skills of the NN are boosted, trying to avoid having to re-train the models with new information. The only exception was made with the novel ESN-RC model, since the implementation with `ReservoirPy` had some limitations in respect of the control of the training

[1]The integration rutine was programmed by the author in earlier years of the bachelor.

procedure. Despite trying different possibilities, such as 50 individual training processes and averaging the tuned weights, the model was only trained with one integration randomly selected from the set of 50. This opened the opportunity to push the abilities of this type of NN to the extreme, since it will be compared with the rest of the models having seen much less data than the rest.

## 2.2 Models

### 2.2.1 Feed-Forward Neural Networks

#### Illustrative baseline: one hidden layer Neural Netwrok

The basic configuration for a Neural Network is a FeedForward Neural Network in which the data is run through the model sequentially and there are no loops inside. The input data is passed from one layer to the next one until it reaches the output layer. It corresponds with the basic concept of the multi-layer perceptron mentioned in the introduction and exemplified in Fig.1.2. The dimension of the different layers depends directly on the data and the problem. The number of units in the input layer often coincides with the shape of the input data. Similarly, the number of neurons in the output layer is defined by the desired output of the model and it could be, for example, just one if the model performs a binary prediction (0 or 1). For the Lorenz system, each input data consists of a point in 3D phase space and the output should be the evolved state, another point in 3D space. Hence, input and output layers for the FFNN have 3 neurons each, a dimension that will be kept fixed.

As there are hyperparameters that can be tweaked (number of hidden layers, number of neurons, activation functions,...), multiple architectures could be chosen, giving different performances and results. Despite existing specific algorithms to tune this set of hyperparameters, this process has been implemented manually with loops in a way called "*gridsearch*". It is important to mention that the training of a neural network has a random component which can lead to worse or better results. In order to avoid or reduce that, the models are trained three times for each configuration and the best result is chosen. While this increases the running time of the programme, it provides better results. To illustrate the process, a simple network with one hidden layer has been trained using different number of neurons in the hidden layer (since input and output have fixed size) and different activation functions: Tanh, Sigmoid, ELU and ReLu. The results of such training are presented in Fig.2.1.

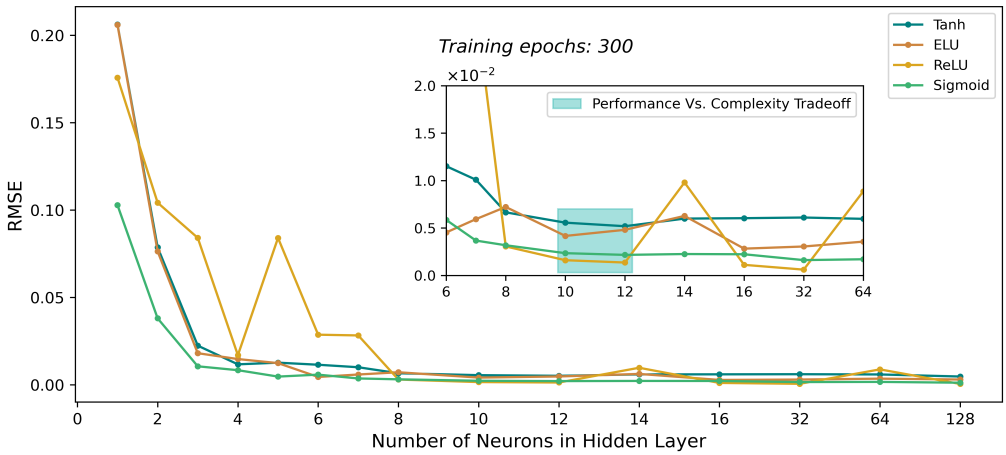



Figure 2.1: Gridsearch results for the 1 hidden layer FFNN using four different activation functions for a total of 300 epochs with each architecture.

There, the training error, measured over the validation data and using Root Mean Squared Error, is plotted versus the number of neurons in the hidden layer.

From Fig.2.1 it can already be seen how the training error measured through a RMSE, no longer decreases significantly after increasing the number of neurons to a certain number. In the blue box, what could be a good trade-off between complexity and performance has been highlighted. Nevertheless, this simple model has proved to be nowhere near to learn the chaotic system, though not being considered as a case of study.

### Two hidden layer Neural Network

#### Architecture

A two hidden layer FeedForward neural network is considered as one of the models to study. Increasing complexity from the previous example, an architecture with input layer, two hidden layers and output layers is put to test. To decide the final architecture, a gridsearch process is also conducted. The results of such a search are plotted using the measured training error with RMSE. As there are now two hidden layers, adding the RMSE creates the 3D plots in Fig.2.2. There, a general tendency to improve when the number of neurons in both hidden layers increases can be seen. This means that the *deeper* the model, the better the result, given that *deeper* is a common term to refer to a high number of neurons in hidden layers. It is worth mentioning the irregular behaviour of the ReLu and Sigmoid function. While providing the lowest errors, as presented in Table 2.1, they also randomly give really bad results. This fact is something that needs to be put into perspective when choosing the activation functions, since smoother behaviours, like the one shown by ELU, could ensure better and more consistent results. The best and worst configurations for each activation function are presented in Table 2.1.

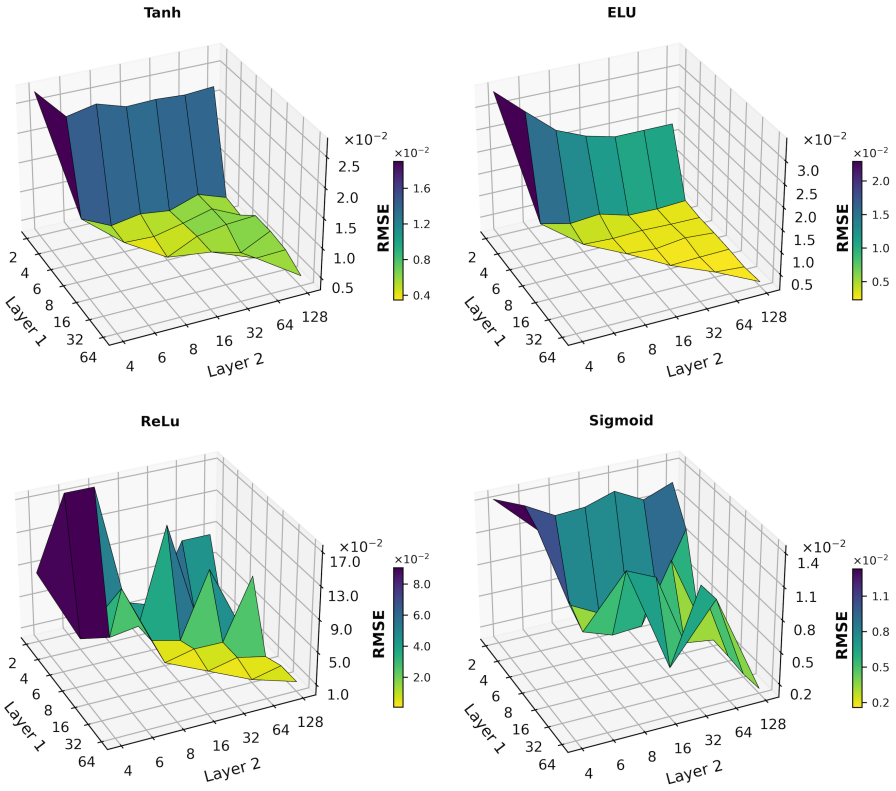



Figure 2.2: Gridsearch results for the 2 hidden layer FFNN using four different activation functions for a total of 300 epochs with each architecture.

| | Layer 1 | Layer 2 | **RMSE** | Layer 1 | Layer 2 | **RMSE** |
|---|---|---|---|---|---|---|
| **Tanh** | 8 | 16 | 0.003466 | 2 | 4 | 0.027465 |
| **ELU** | 32 | 128 | 0.002198 | 2 | 4 | 0.034180 |
| **ReLu** | 16 | 32 | 0.000757 | 2 | 6 | 0.174188 |
| **Sigmoid** | 64 | 128 | 0.001244 | 2 | 4 | 0.014391 |

Table 2.1: Best and worst configurations obtained from the 2 hidden layer FFNN. ReLu shows lowest RMSE for 16 and 32 neurons respectively.

In the light of these results and the difference in performance between ReLu and the other functions, the architecture that gave the best result is the one chosen: 16 and 32 neurons in the 1st and 2nd hidden layer, respectively, and ReLu as activation function.

### 2.2.2 Long-Short Term Memory Neural Networks

#### Architecture

Next model considered in the study belongs to the Recurrent Neural Networks (RNNs) and is the case of the LSTMs. As it happens with any neural network and was exemplified with the FFNN, almost infinite configurations exist for the set of hyperparameters. However, the LSTM is chosen to have one hidden layer with the LSTM structure (presented previously in Fig.1.4) and an input and output dense layers with the size fixed by the number of variables. The data re-structure step is crucial for LSTMs since it establishes how many time steps are going to be used to predict and how many time steps into the future are going to be predicted. The first number is called *lookback* and the second, *lookforward*. This reshaping is often done through a windowing process in which sub samples of the time series are extracted according to the *lookback* and *lookforward* parameters. Such procedure is illustrated in Fig.2.3. The new data structure introduces two other numbers whose ideal values need to be found. Therefore, the gridsearch for this kind of model will explore different values for the dimension of the hidden layers as well as different *lookbacks*, while the *lookforward* has been fixed to one for the simulation purposes of the model. Slightly different types of LSTM models are also explored.

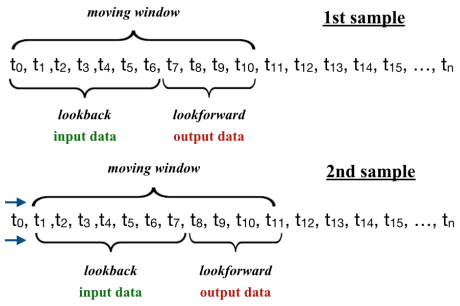



Figure 2.3: Windowing process example.

#### Vanilla LSTM

*Vanilla* in the Machine Learning jargon stands for "simple" or "soft" and in the case of LSTMs it refers to a model with the architecture and functioning described above, a single LSTM layer that retains the state of the long term memory during each sample, this is, during the *lookback* time steps. When varying the number of units in the LSTM layer, the *lookback* and the activation function in the output layer, the gridsearch outputs the results depicted in Fig.2.4. There, increasing the number of neurons shows to improve the training result, although some good configurations are also found between those with lower dimension. In concordance with Fig.2.2,

ReLu and Sigmoid present irregularities and the difference between the best ReLu architecture and the second best, is smaller than in the previous case. While the number of neurons is not determinant, in Table 2.2 can be seen how, using 25 time steps as input, provides the best results in all cases.

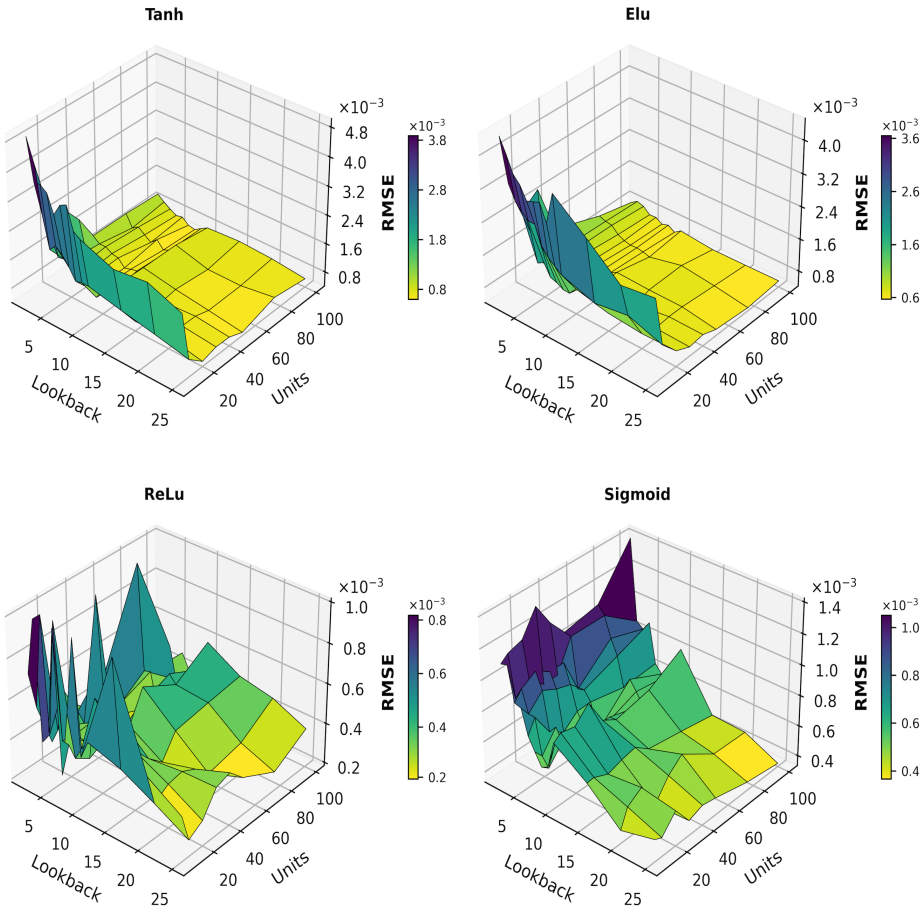



Figure 2.4: Gridsearch results for the LSTM model using four activation functions in the output layer.

Best and worst results are gathered in Table.2.2:

| | Lookforward | Units | **RMSE** | Lookforward | Units | **RMSE** |
|---|---|---|---|---|---|---|
| **Tanh** | 25 | 50 | 0.000487 | 2 | 5 | 0.004932 |
| **ELU** | 25 | 100 | 0.000355 | 2 | 5 | 0.004433 |
| **ReLu** | 25 | 10 | 0.000266 | 5 | 40 | 0.529048 |
| **Sigmoid** | 25 | 75 | 0.000504 | 2 | 100 | 0.001401 |

Table 2.2: Best and worst configurations obtained from the LSTM. ReLu shows lowest RMSE for *lookback* = 20 and 10 units.

From the gridsearch, the architecture chosen for future training and simulation should be (**ReLu, lookback = 25, units = 10**). However, the difference in performance with ELU is not drastically bigger and the latter is chosen to ensure regular results and smoother behaviour, making benefit of using a higher number of neurons in the LSTM layer. The final architecture will use input data with *lookback* = 25, 100 units in the LSTM layer and ELU activation function in the output layer (**ELU, lookback = 25, units = 100**).

#### Stateful LSTM

In contrast to what is commonly believed, "Long-Term Memory" in LSTM is not that *"long-term"*. Indeed, the number of time steps through which information is retained in a LSTM unit is defined by *lookback* and after each sample of shape (*lookback*, *variables*) is processed, the *hidden* and *cell states* of the unit are reset. This means that it does not matter if the whole dataset is time related —as it is the case here for each $10^4$ samples that conform an integration—, the LSTM will only store information from the data in each individual sample created during the windowing process. Despite this sounding useless or at least not a big improvement from a normal FFNN, this configuration is the most common one in LSTM applications and its referred to as a *stateless* model. In a situation in which retaining information for longer subsets of data could be useful, there is an option of going for a *stateful* model[32]. However, while making possible to explode auto-correlation features in a time series data, it also imposes some restrictions during training and simulation in terms of data splitting and processing with Keras. If implemented right, the LSTM will store information for a whole batch, meaning that the unit states will not be reset until a complete batch of $n$ samples has run through the network. This new operational scheme seemed suitable for the problem that is being studied and the same architecture that was chosen previously is checked to be compatible. As it is the case, the same architecture used for the *Vanilla* LSTM, is now implemented but in *stateful* mode.

#### Predicting Individual Variables

A different approach from *Vanilla* LSTM structure is also explored and implemented as an extension of the use of LSTMs. In all preceding cases, only one model was being trained and used to predict the value of the three variables $X(t)$, $Y(T)$ and $Z(t)$. Another possible framework could be to use three identical models to predict the variables individually while taking the same input data.

In order to put this to practice, the architecture needs to be adjusted —since the output is now one variable instead of three— by changing the dimension of the output layer from three to one. Next, these three identical models are trained with the same data as earlier models. In addition, the prediction and simulation scheme also needs corrections because it is necessary to combine the output of the three models into a 3D vector containing the three variables predicted in order to feed the models again. The performance of this structure is also tested.

### 2.2.3 Echo State Network

Similarly to other NNs, ESNs contain hyperparameters which optimum values need to be found. Some of them and their common baseline values are shown in Table 2.3. Detailed information and examples of the impact of each of the hyperparameters can be found in the `ReservoirPy` documentation and GitHub page[33][34].

| **Reservoir Hyperparamters** | |
|---|---|
| **Units** = 2000 | **Connectivity** = 0.3 |
| **Leak Rate** = 0.25 | **Input Connectivity** = 0.3 |
| **Spectral Radius** = 1.0 | **Regularisation** = $10^{-8}$ |
| **Activation Funct.** = Tanh | **Input Scaling** = 1.0 |

Table 2.3: Basic ESN Reservoir hyperparameters and baseline values.

The tuning process of the model can be done using a similar gridsearch scheme to the ones described previously. However, for the sake of simplicity, the basic configuration in Table 2.3 is chosen, together with a readout based on the Ridge Regression with a regularisation parameter of $10^{-8}$.

## 2.3 Key Performance Indicators (KPIs)

### 2.3.1 Conventional Regression KPIs

Whenever a regression problem is faced in Machine Learning, the perfomance's metrics or Key Performance Indicators (KPIs) are usually well known and defined: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean and Weighted Mean Absolute Percentage Error (MAPE and WMAPE), the Pearson Coefficient (R) or even a custom made metric that follows the lines of the above-mentioned ones[35][36][37]. While some of them are not applicable to all regression problems, RMSE, MAE, MAPE and WMAPE are flexible and should be suitable in general, providing an indicator of the differences between the model output data and the true data. This can be checked in the RMSE definition (2.1), where $\overline{\mathbf{y}}_{\mathbf{true}}$ is the true data and $\overline{\mathbf{y}}_{\mathbf{pred}}$ the data predicted by the model, with $N$ number of samples conforming the data.

Nonetheless, the Lorenz system is **chaotic** and the roots of this behaviour lie in the sensitivity to initial conditions, which can be translated into responsiveness to small variations in the trajectories. Previously mentioned metrics could be useful for short-term predictions. However, for long-term simulations, as the model will not predict the exact same values contained in the dataset[2], the differences between true and predicted data will rapidly end in two diverging trajectories. The issue is pictured in Fig.2.5. In fact, this divergence is exponential in time[3] and it is bounded, in the case of an attractor, by the size of its basin. Having said that the classic metrics measure this differences, their values will quickly increase not providing a real image of the performance of the model.

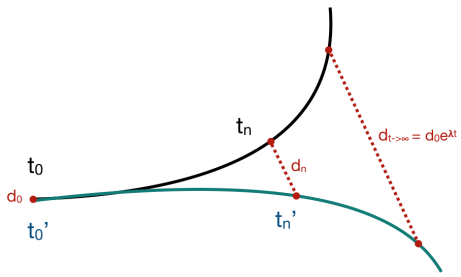



Figure 2.5: Divergence of initially coincident trajectories where $d$ represents the distance between the $t_i$ and $t'_i$ data.

$$RMSE = \sqrt{\frac{\sum_{i=0}^{N-1}(\mathbf{y_{pred,i}} - \mathbf{y_{true,i}})^2}{N}} \quad (2.1)$$

It is important to mention that due to this fast divergence, the **goal** of the model would not be to emulate the Lorenz dataset for long-time ranges, but to capture the structure of the phase space and reproduce the two-attractor arrangement. Despite this, short range simulations will do be assessed to check how well the model emulates the system starting from a common initial point. Such a task will be done by attending to the time horizon until which the predicted data is significantly close to the true time-series. Exponential divergence of the trajectories leads to one major problem regarding metrics: **How will the long-term or overall performance of the different models be measured and compared?**

[2]It does not matter how good the model is, there will always be differences due to computational precision.

[3]The exponent $\lambda$ will be studied and explained in the next section.

### 2.3.2 Long-Term Behaviour: Invariant Characteristics for Classifying Systems

Going back to the root of the problem, the sensitivity to initial conditions, it is necessary to find some kind of measure that eludes this responsiveness. Here, two features are introduced that come to be invariant under the evolution of the system and therefore are independent of changes in the initial conditions, as well as being independent on the coordinate system in which the trajectories are observed. It is precisely the lack of sensitivity what is remarkable about these two concepts, Fractal Dimensions and Lyapunov Exponents.

#### Fractal Dimensions: Correlation Dimension($D_2$)

The geometry that an attractor shows in its phase space and its dimensionality are strongly related with the nature of the dynamic behaviour of the system. What is more, this dimensionality is important in defining the possible dynamic behaviours. What departures from the general intuition about the idea of dimension is that there are some mathematical objects which have non-integer dimensionalities and those are the ones who play an important role in the dynamics of chaotic systems. These geometric objects are called *fractals*[38] and the attractors that show non-integer dimensionalities are referred to as *strange attractors*[4].

The term *fractal dimensions* refers to a host of dimensionality measures each of them with different, but sometimes close, definitions. The **Correlation Dimension** is one of the most used definitions, mainly because it is easy to compute and it gives better results even if there is a small amount of data available. Firstly, the fraction of observed points $\mathbf{y}(k)$ within a sphere of radius $r$ centred in a certain phase space point $\mathbf{x}$ is considered. This number is given by

$$n(\mathbf{x}, r) = \frac{1}{N} \sum_{k=1}^{N} \theta(r - |\mathbf{y}(k) - \mathbf{x}|) \tag{2.2}$$

where $\theta(a)$ is the Heavyside function. Secondly, focus on the moment $(q-1)$ of the function $n(\mathbf{x}, r)$ and use the density of points in phase space $\rho(\mathbf{x})$ —which is usually inhomogeneous for strange attractors— to evaluate the moment selected. Being then $f(\mathbf{x}) = n(\mathbf{x}, r)^{(q-1)}$ the object to evaluate, the function $\mathcal{C}(q, r)$ of two variables is defined by the mean of $f(\mathbf{x})$ over the attractor using the density as weighting factor:

$$\mathcal{C}(q, r) = \int d^d x \rho(\mathbf{x}) n(\mathbf{x}, r)^{(q-1)} \tag{2.3}$$

The idea of using this kind of sums for studying strange attractors is due to Grassberg and Procaccia[39], who discussed the case $q = 2$ introducing a concept of dimension based on the behaviour of the correlation integral (2.3), from where it inherits its name (Correlation Dimension, $D_2$). The Grassberg-Procaccia algorithm[39] takes advantage of the linear behaviour when $r$ is small, range in which fulfils a power law:

$$\mathcal{C}(q, r) = r^{(q-1)D_q} \longrightarrow r^{D_2} \quad ; \; q = 2 \tag{2.4}$$

$D_2$ can then be extracted by adjusting carefully the logarithm of the correlation sum versus the logarithmic variation of $r$, both computed from the phase space trajectories. This can be done using observational data, in this case, time-series data from the single variable $X(t)$ in the Lorenz system but first, it requires a reconstruction of the phase space. To do so, a technique called *Delay-Coordinate Embedding* "re-inflates"

[4]This is an alternative definition for strange attractors different from the one in the Appendix B.

from 1D to $n \cdot D$ the phase space using a time-delayed set of vectors and an embedding dimension $n = d_E$. While being extremely interesting and not difficult to follow, the detail of the process is out of the scope of this work but is worth mentioning that parameters of importance in this reconstruction such as $d_E$ or the *lag* applied to create the delayed vectors are of relevance in the Grassberg-Procaccia algorithm and, consequently, in the resulting value of $D_2$. Further details and a complete explanation of this reconstruction can be found in[40][41].

### $\mathbf{D_2}$ Computation

For the purpose of calculating the Correlation Dimension from time-series data simulated by the models, the function `corrDim[]` from `MatLab` was used due to its better performance. An example of the logarithmic representation and straight line fit required to obtain the dimension is presented in Fig.2.6:

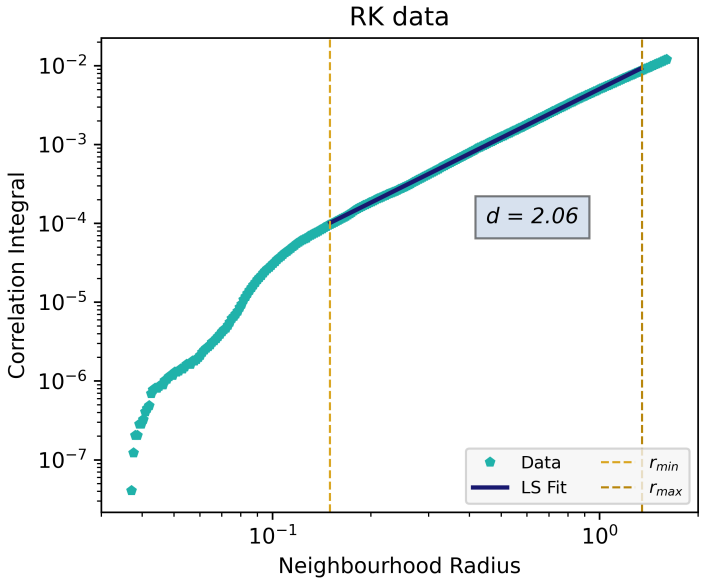



Figure 2.6: Correlation Dimension calculation via least squares fit to the straight part of the data. 500 values of the Corr. Integrals were taken using $r \in [0.002, 1.600]$.

### Lyapunov Exponents($\lambda$)

Lyapunov Exponents measure how small perturbations to an orbit in the attractor evolve. Considering a short trajectory, $\mathbf{y}$, and a small perturbation, $\delta$, the linearised dynamics provide the equation (2.5) for the time evolution of the perturbations:

$$\begin{aligned} \dot{\mathbf{y}} + \dot{\delta} &= \mathbf{F}(\mathbf{y} + \delta) = \mathbf{F}(\mathbf{y}) + \mathbf{J}(\mathbf{y}) \cdot \delta + ... \\ \dot{\delta} &\approx \mathbf{J}(\mathbf{y}) \cdot \delta + ... \end{aligned} \tag{2.5}$$

The system shows exponential time divergence ruled by the eigenvalues of the Jacobian matrix $\mathbf{J}(\mathbf{y})$. If a small trajectory segment is considered, the eigenvalues are known as **local Lyapunov Exponents** and when they are averaged over the whole attractor, they converge to the **Global Lyapunov Exponents ($\lambda$)** of the system. The whole set of eigenvalues is usually referred to as the *spectrum* of Lyapunov Exponents $\{\lambda_1, \lambda_2, ..., \lambda_n\}$ and there should be one per number of characteristic variables describing the system. In order to obtain the spectrum it is necessary to find a way of computing the Jacobian matrix and their eigenvalues with good accuracy, which is not an easy task. The mutiplicative ergodic theorem of Osledet[42] provides the theoretical background for the computation of the Jacobian to be correct,

whereas the evaluation of the eigenvalues lies on the ideas presented by Eckmann *et al.* [43]. There were some previous works on the calculation of the exponents from series of data by Wolf *et al.* [44], but the one of interest in this dissertation is the algorithm published by Rosenstein *et al.* in 1993 [45]. In such article, Rosenstein presents an alternative calculation which simplifies the process if only the largest Lyapunov Exponent is needed, which is usually enough to show whether the system behaves or not chaotically. The solution bypasses the complex management of the trajectories' tangent vectors which led to the Jacobian matrix in the other algorithms. Those required continuous re-normalisation to ensure that the correct directions (Lyapunov directions) were being followed, increasing the complexity and computational costs. Instead, Rosenstein relied on the Oseledet theorem to justify the use of an arbitrary direction and stated that the largest Lyapunov Exponent can be obtained from the relation (2.6):

$$d(t) = C \cdot e^{\lambda_1 t} \tag{2.6}$$

where $d(t)$ is the average distance at time $t$ and C is a normalisation constant related to the initial separation of the trajectories followed. A more visual idea of the concept can be given if one imagines a volume in phase space at a certain moment and tracks how it shrinks or expands during time. In that situation, the Lyapunov Exponents would define the rate of stretching or shrinkage (depending on the sign of $\lambda_i$) of line segments, areas and various dimensional subvolumes in phase space. It is not only more visual but it is also the approach behind some of the calculation methods that provide the full spectrum of Lyapunov Exponents. It also presents visual proof of all three exponents for the system. The case for Lorenz63 system ($\lambda_\mathbf{1} = \mathbf{0.906}$, $\lambda_\mathbf{2} = \mathbf{0}$, $\lambda_\mathbf{3} = \mathbf{-14.572}$)[46][47] has been exemplified in Fig.2.7. There, an initial spherical volume in phase space comprised by the green dots, has evolved freely for $7 \cdot 10^{-2}$ seconds, giving the resultant pink set of points.

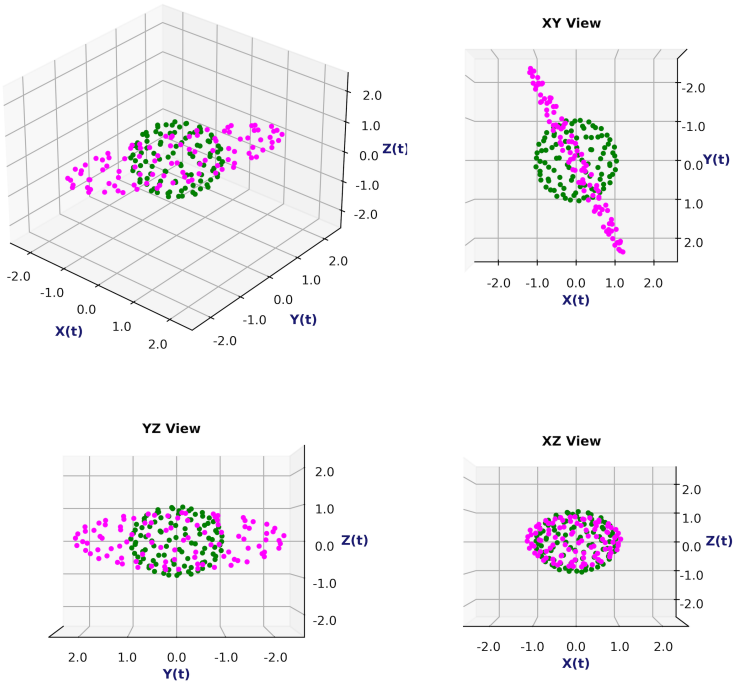



Figure 2.7: Evolution and consequent deformation (pink points) of an initial spherical volume in phase space (green points). There is an expanding direction ($\lambda_1$), a second direction in which it shrinks ($\lambda_3$) and a third one with respect to which responses statically ($\lambda_2$).

Different plane views are added to appreciate the different deformation depending on the direction. For example, along the $Z$ direction there is little contraction or expansion and, consequently, the Lyapunov direction corresponding to $\lambda_2 = 0$ can be identified with it. More radical shapes are adopted along the $XY$ directions. Notice that the Lyapunov directions do not strictly coincide with $X, Y$ and $Z$ directions. However, there is a great contraction in a direction close to $X$, corresponding to $\lambda_3 < 0$ and an expansion in a direction forming an angle with respect to $Y$, related with $\lambda_1 > 0$.

**Largest Lyapunov Exponent Computation**

Having mentioned the main algorithms for Lyapunov Exponents calculation, plenty of iterations and modifications to them have appeared over the years. Working with the Rosenstein algorithm, since it is the most computationally efficient for the needs of this project, several `Python` implementations were tried before deciding to use a custom implementation of the original algorithm designed by R.Hegger, H.Kantz and T.Schreiber. Such code was developed as part of their software project for Time Series Data Analysis(TISEAN)[48] and it was written in C and Fortran but adapted to `Python` for this work. Exponents are obtained starting from (2.6) and assuming that the $j$th pair of nearest neighbours should diverge at a rate close to

$$d_j(i) = C_j \cdot e^{\lambda_1 (i\Delta t)} \tag{2.7}$$

were $C_j$ is initial separation. By taking logarithms on both sides and considering several values of $j$ over a trajectory, the Lyapunov Exponent is obtained by a least-squares fit to the average line:

$$\ln d_j(i) \approx \ln C_j + \lambda 1 (i\Delta t) \longrightarrow y(i) = \frac{1}{\Delta t} \langle \ln d_j(i) \rangle \tag{2.8}$$

The execution of the algorithm requires again the reconstruction of the phase space using the *Delay-Embedding* technique mentioned previously, since the only data available is a 1D time-series. It is worth mentioning that all algorithms and implementations found for computing Lyapunov Exponents were highly sensitive to small changes in the parameters. Modifying the number of steps during which the neighbour trajectories are tracked or selecting a different *lag* for the delayed reconstruction had a huge impact on the output. It was decided then to tune the algorithm with the aim of providing the best possible results with true data (Runge-Kutta) and then, without changing the parameters, apply it to the data simulated by the models. This manual tuning was done individually for the 50 simulations performed with the trained models.

### 2.3.3 Recurrence Time

Recurrence of states of a dynamical system have been important in the study of statistical mechanics as well as in the study of chaos. The concept of recurrence makes reference to how often a region in phase-space is visited, and the calculation of this time frequency is rather simple compared to the previous concepts introduced in this section. In spite of this concept not being an invariant of the system as it happens with $D_2$ and $\lambda$, it is helpful when studying long-term behaviour of chaotic systems and will be also useful when comparing such behaviour between the models[49].

The definition of the recurrence time considers a neighbourhood of radius $\rho$ centred in a point of the attractor, $\mathbf{X_0}$, and the recurrence to this neighbourhood. This is, all the points $X$ within a distance $d \leq \rho$ to $\mathbf{X_0}$. Then, the attention is

focused on a trajectory with a certain length and denote by $M$ the subset of points from the trajectory that belong to the neighbourhood, $M = \{X_{t_1}, X_{t_2}, X_{t_3}, ...\}$. With such set, the **recurrence time** is $T(i) = t_{i+1} - t_i$, $i = 1, 2, ...$ However, it could happen that as $\rho$ increases, for trajectories with small time-step ($\tau$), several consecutive points of the trajectory lay inside the neighbourhood, giving $T(i) = \tau$. For those kind of points, the measured recurrence time is not correct since it is not the time between two different visits to the neighbourhood. The former are called Poincare points and the latter Soujorn points, which wrongly affect statistics for $T$ and ought to be removed during computation.

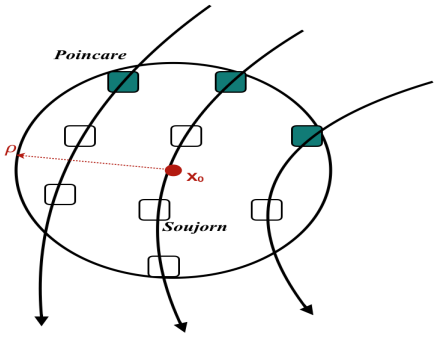



Figure 2.8: Recurrence point diagram.

Having the set of $T(i)$, the scaling law for the mean values resembles a power law:

$$\overline{T}(\rho) = \rho^{-\gamma} \tag{2.9}$$

where the exponent is negative, since an increase in $\rho$ diminish the re-visit time because the neighbourhood covers a larger part of the phase space. Here gamma can be related with some of the fractal dimension definitions since, as it was mentioned before, those dimensionalities determine the dynamical behaviour of the chaotic system, consequently having influence on the recurrence time to a certain zone of the attractor. It is important to consider that the recurrence time may not (and it will usually not) be the same throughout the whole phase space, having strong dependence with $\mathbf{X_0}$. This coincides with the statement of having an inhomogeneous point density mentioned in (2.3). This fact should also be considered when calculating the recurrence time.

### Recurrence Time Computation

The Python algorithm designed to implement the recurrence time calculation evaluates the distances between the points in the phase space and the randomly chosen $\mathbf{X_0}$[5] for a certain value of $\rho$. Those distances are then checked to belong or not to the neighbourhood and for those which fulfil the condition, revisit time is calculated using the location of those points in the time-series that comprises the trajectory. At this point, Soujorn samples are removed and the whole process is repeated for the next value of $\rho$ in the range selected. The main parameters in the algorithm are the $\rho$ range, the number of values in that range, which defines the number of points used for the curve, and the time step ($\tau$) of the time-series, which scales the recurrence times.
To obtain the value of $\gamma$, logarithms are taken on both sides of (2.9) giving:

$$\log \overline{T}(\rho) = -\gamma \ln \rho \tag{2.10}$$

Using the least squares method, a straight line is fitted to the points and the slope is extracted. In order to tackle the above-mentioned dependence on the $\mathbf{X_0}$ point, a hundred different points are taken and the results averaged. The $\rho$ range selected is $\rho \in (10^{-3}, 1)$

[5]As the density of points is not homogeneous, the $\mathbf{X_0}$ selection is not completely free but constrained to a certain zone of the attractor. Inside that zone its location is truly random.

### 2.3.4 Evaluating Models' Performance

Armed with the tools described in previous sections, the process followed to compare the output and results from different NN models is now described:

1. A single trajectory with a starting point ($X_0$, $Y_0$, $Z_0$) is integrated using Runge-Kutta method with time step 0.01. With the same starting point, the different models are left free to simulate their own trajectories. The time-series that arise from the Runge-Kutta method and the models are the ones that will be analysed.

2. The time-series data will be plotted in a short-range scale to compare the prediction horizon and when, the different models, start to diverge in comparison with the Runge-Kutta data. Figures like Fig.1.7 will be presented.

3. Then, the overall behaviour of the models will be assessed by computing the three concepts described: Correlation Dimension ($D_2$), Largest Lyapunov Exponent ($\lambda$) and Recurrence Time ($\gamma$). As they are intrinsic characteristics of the system, which itself is defined by the equations (1.2), had the models really captured and learned the dynamics, values should be close to those obtained from Runge-Kutta simulations and also reported in the correspondent bibliography.

4. In an attempt of providing some robustness to the results, fifty different simulations beginning in random initial points are run and their performance is measured. Final results represent the average of those runs.

# 3. Results and Discussion

En esta sección se presentan los resultados correspondientes a la aplicación de los modelos descritos en la Sección 2.2 construidos y entrenados siguiendo el esquema reflejado en la Secciones 2.2.1, 2.2.2 y 2.2.3. Dichos resultados se evalúan atendiendo a dos aspectos diferentes: las prediciones de los modelos a corto plazo y la eficacia al aprender y reproducir la dinámica general del sistema Lorenz63.

***Resumen***

After having trained the models with the architectures and hyperparameters described in the previous section, their performance is tested by performing several different simulations. Given a random initial point $[X,Y,Z]$ as input, each model **freely simulates** for $10^4$ time-steps of 0.01 secs. A *simulation* is understood as the process in which the model outputs are repeatedly fed as input for the next prediction. The main implication of such a procedure is that the errors made by the model in each prediction will accumulate iteration after iteration. This situation tries to test the NN in the framework in which they would work when substituting the original system of equations (1.2). It is important to remember that the main goal of the NN in this project is to completely replace the equations and correspondent numerical integration of them in the Lorenz attractor problem. In total, 50 distinct initial points randomly selected within a range where used, providing 50 time-series of the before-mentioned length for all three variables. An example of how this time-series look was introduced in Fig.1.7 and the $X(t)$ variable is recovered again in Fig.3.1.

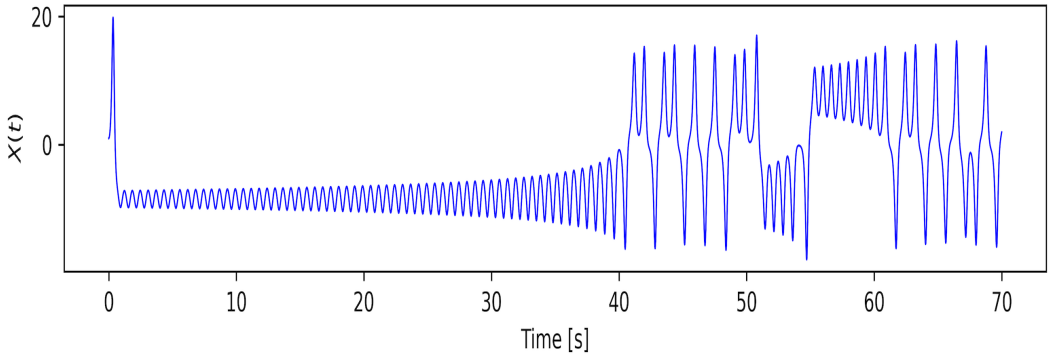
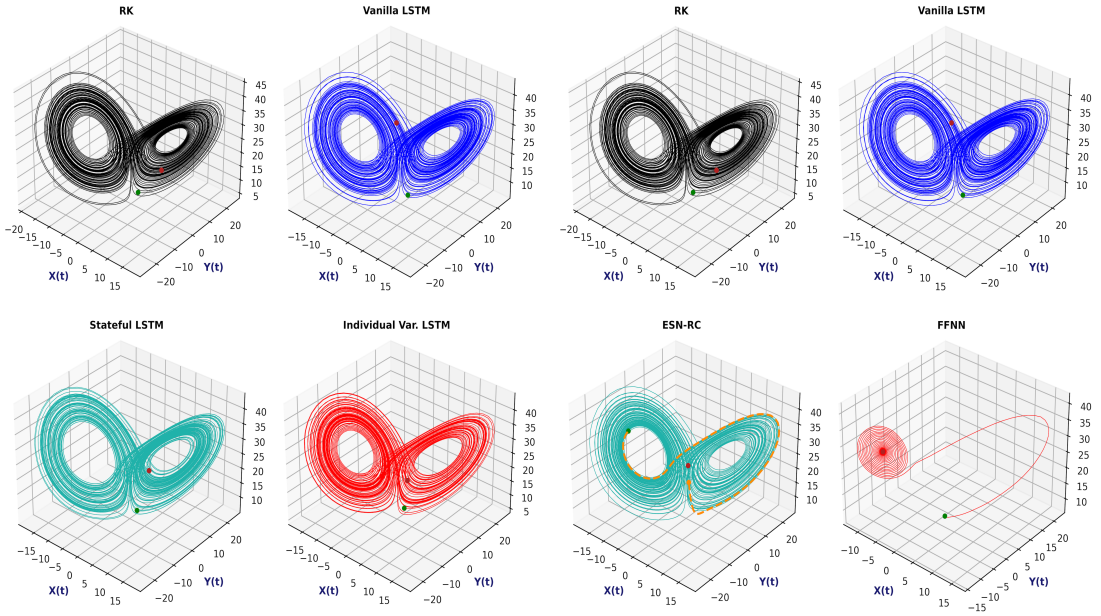



Figure 3.1: Time-series data example for $X(t)$.

To have a glance of how the different models behave in a simulation and as a prelude of the detailed upcoming results, the 3D phase space is considered in Fig.3.2, providing a visual comparison between the shape of the different attractors reproduced by the models and the Runge-Kutta integration.

For further analysis, the focus is put on two different aspects of the simulations. The first one attends to the ability of the model to perform accurate short-term predictions given a random initial point in phase space. In this situation, models are compared straightaway and their prediction horizons in time are assessed. The second one will study the overall skills of the different NN in reproducing the dynamics

of the Lorenz system, so as to try to answer if the Networks truly captured and effectively learnt the system dynamics.




(a) (b)

Figure 3.2: Visual comparison of the simulations performed by the different models in 3D phase space with a common initial point $(X = 1.5, Y = 1.5, Z = 7.5)$. $(a)$ Phase space of Runge-Kutta and LSTMs simulations. $(b)$ Phase space ofRunge-Kutta, Vanilla LSTM, ESN-RC and FFNN simulations.

## 3.1 Short-Term Simulations

Some of the time-series arising from the simulations made by the models are now presented. In the horizontal axis, the time is represented in seconds $(s)$, whereas the value of the variable $X(t)$, $Y(t)$, or $Z(t)$, is plotted in the vertical axis. The models are arranged in different groups for the sake of clearness and to avoid overloading the graphics. For the upcoming results, the simulation with the initial point $(X = 1.5, Y = 1.5, Z = 7.5)$ is chosen.

In Fig.3.3 the time-series for the three variables are plotted, showing the simulated series by the three LSTM variations compared with the Runge-Kutta integration. The vertical lines represent the *prediction horizon*, time until which the predictions are considered accurate. It is worth mentioning that the results are consistent for the three variables, something *a priori* not evident since each of them behaves differently in terms of amplitude and frequency.

A quick look at the prediction horizons highlights the Vanilla LSTM as the best performing model in this simulation among the ones depicted. In Fig.3.4, the ESN-RC model is shown against the Vanilla LSTM and Runge-Kutta. The results achieved by the ESN-RC, bearing in mind the particular training dataset mentioned in 2.1.1, are of relevance. It outperforms the best LSTM in all three variables, almost doubling its prediction horizon. The orange zone in Fig.3.4 corresponds to a series of 100 predictions, corresponding to 1 sec., used by the ESN-RC to warmup. This is needed to correctly initialise the reservoir state with respect to the input data and it is conceived under the assumption that after the warmup, the state of the reservoir somehow resembles a representative sample of the data.

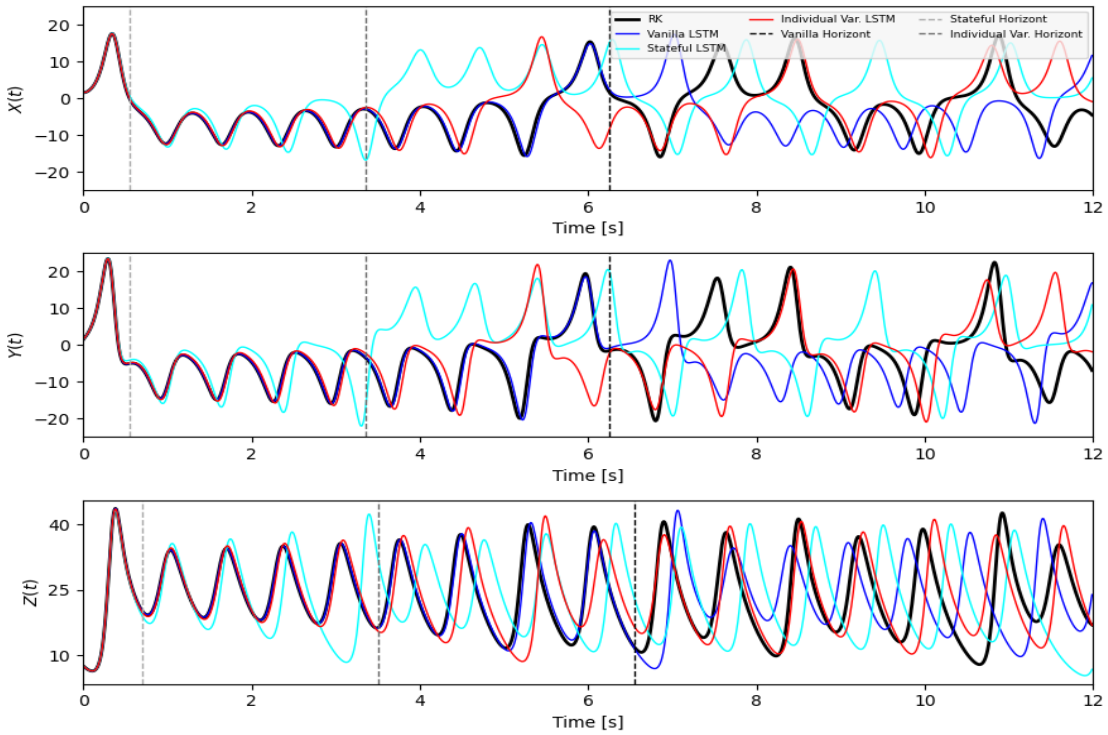



Figure 3.3: 12 secs. $X(t)$, $Y(t)$ and $Z(t)$ time-series from Runge-Kutta and LSTMs simulation.

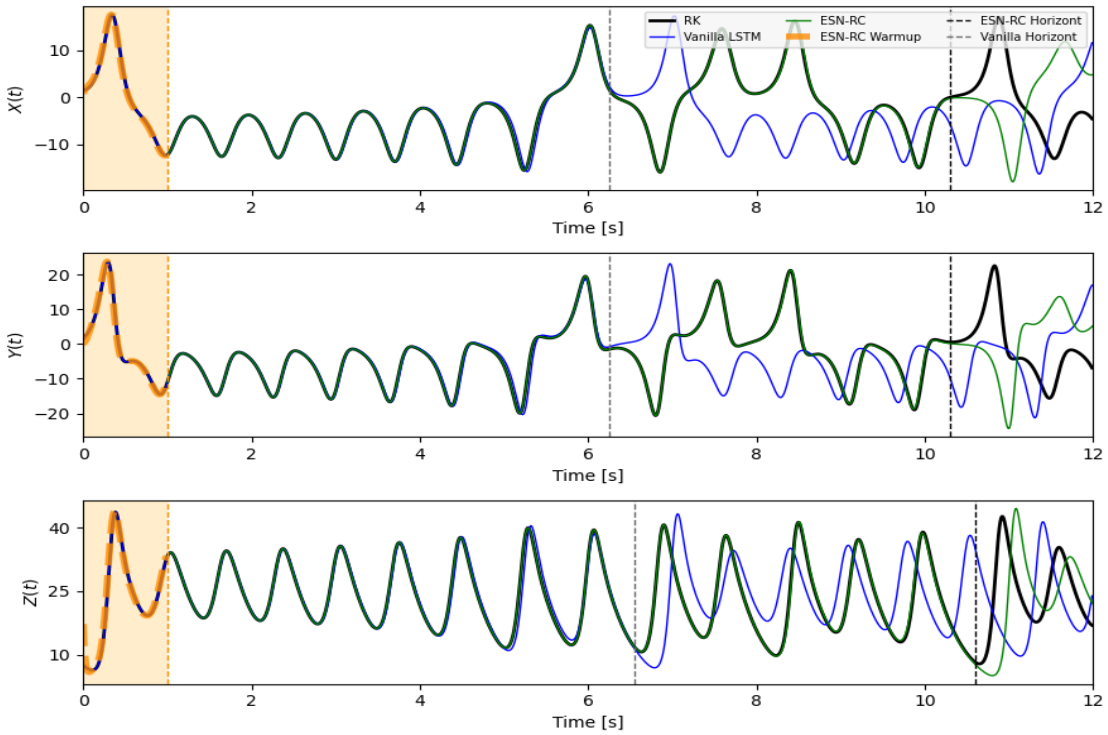



Figure 3.4: 12 secs. $X(t)$, $Y(t)$ and $Z(t)$ time-series from Runge-Kutta, Vanilla LSTM and ESN-RC simulation. The warmup stage is marked in orange.

More detailed comparisons are done focusing on the $X(t)$ time-series. In Fig.3.5, the simulation performed by the FFNN is also depicted. It can be seen how, despite the similar beginning, the FFNN behaves completely different to the Runge-Kutta simulation. This anomaly could be expected and is coherent with the shape of the

phase space shown in Fig.3.2b, where only one attractor was reproduced, causing all trajectories to converge to a single point.

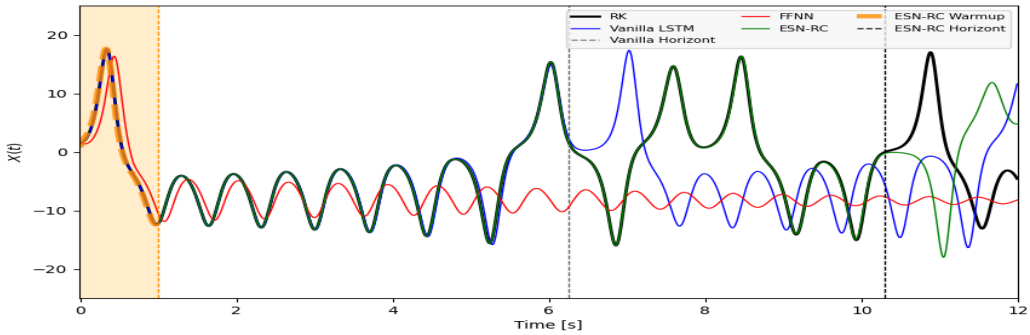



Figure 3.5: 12 secs. $X(t)$ time-series from Runge-Kutta, Vanilla LSTM, ESN-RC and FFNN simulation. Failure of the FFNN model is easy to notice in red.

Given the performance of the different LSTMs and FFNN[1], the attention from now on will be set on the Vanilla LSTM and ESN-RC models in order to test them in diverse simulation cases. While the Individual Variables LSTM also provides some good results, in particular for simulations of small amplitude and periodic behaviour, it is always surpassed by the Vanilla version, hence the decision.

In Fig.3.6, another simulation, in this case with the initial point $(X = 13.5, Y = 16.0, Z = 6.0)$, is represented . The trajectory starts with small amplitude dynamics, almost resembling a periodical signal, however, it gradually increases in amplitude and suddenly changes in behaviour. In the phase space, the first section illustrates a repeated motion around one of the attractors, just before jumping to the other attractor and starting to alternate between both. These jumps are identified by the high peaks and deep valleys in the time-series after the more regular first 25 secs. In spite of the difficulty that these abrupt deviations could imply for a numerical method, the ESN-RC is able, not only to extend the prediction horizon until 26 secs., but also to cope with the strong fluctuation (not without sacrificing some accuracy). The Vanilla LSTM was also able to correctly simulate for a larger time but not to do the latter as shown in the augmented frame in Fig.3.6. As mentioned before, the small amplitude and regular pattern at the beginning of the time-series help to predict further in time, since previous examples had prediction horizons below the 10 secs. for the LSTM and below 12 secs. for the ESN-RC.

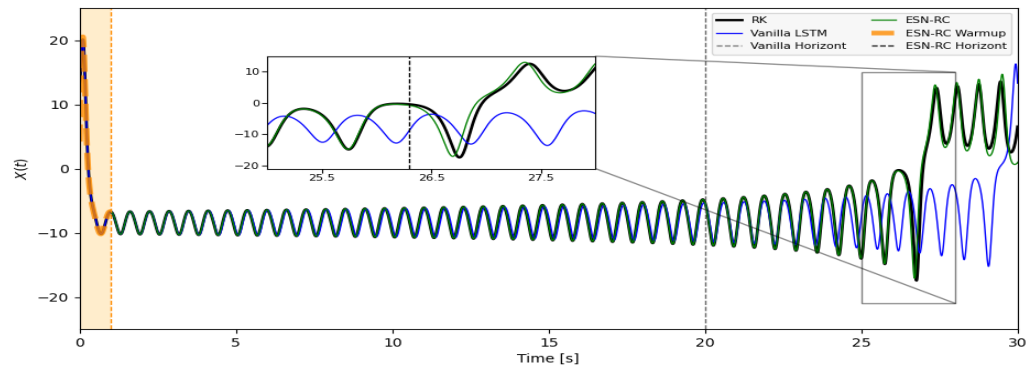



Figure 3.6: 30 secs. $X(t)$ time-series from Runge-Kutta, Vanilla LSTM and ESN-RC

[1]Their lack of skills was tested in different simulation, some of them shown in Appendix C.2

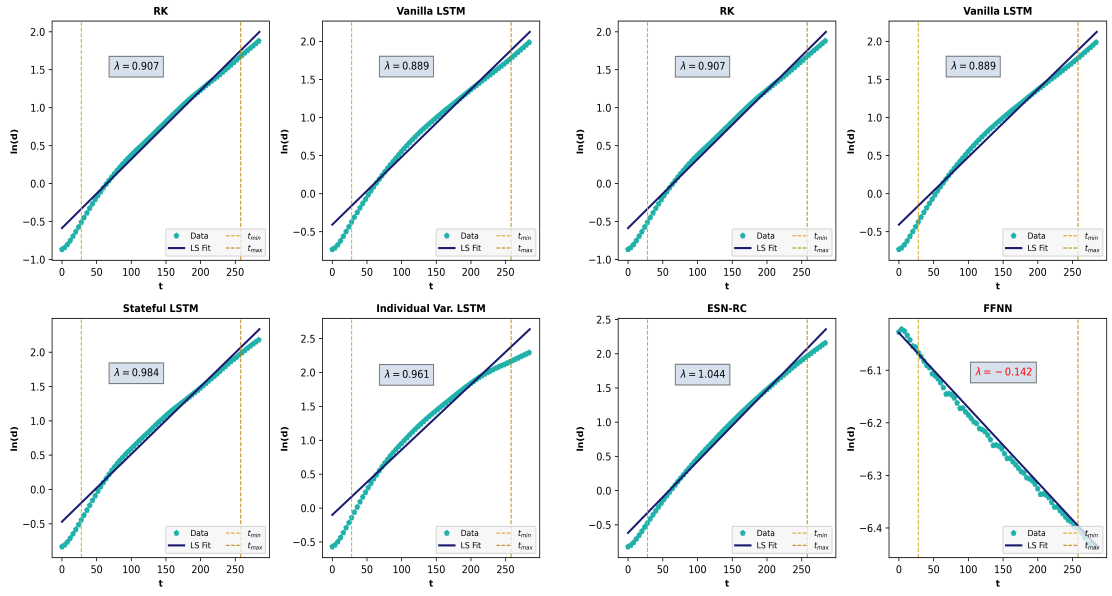
Once again, it is important to recall on the prediction skills of the ESN-RC. It can not be forgotten that the system is highly chaotic, with quickly diverging trajectories and, in such a situation, the model is able to predict more than 10 secs. and even 20 secs. in some of the cases presented. The 20 secs. stamp implies 2000 calculations in which numerical differences are accumulated and can quickly separate the Runge-Kutta simulation from the NN simulation. Despite that, the ESN-RC manages to produce such first-rate results. Moreover, the structure and training dataset of the ESN-RC makes the feat more awe-inspiring if possible.

## 3.2 Dynamical Behaviour

Due to the chaotic dynamics of the Lorenz system, long-term predictions are not suitable to be compared with, for example, a Runge-Kutta integration, since trajectories will diverge at a certain point and will continue doing so. However, turning to invariants of the system and topology, it is possible to evaluate if the model has learnt the dynamics of the Lorenz attractor and to what extent. The KPIs described in Chapter 2 will serve for this purpose. This is needed because just by analysing the visual aspect of the phase space, as in Fig.3.2, no conclusions can be extracted except for extreme bad performing cases such as the FFNN, already showing a different structure. The various metrics are now compared using *boxplots* that represent their values for the 50 simulations studied.

### 3.2.1 Largest Lyapunov Exponent

The *largest Lyapunov Exponents* for the simulation with the initial point ($X$=1.5, $Y$=1.5, $Z$=7.5) are outlined in Fig.3.7:




(a) (b)

Figure 3.7: Example of Lyapunov Exponent calculation for all models during a common individual simulation with initial point ($X$=1.5, $Y$=1.5, $Z$=7.5). ($a$) Lyapunov Exponent Least Squares Fit of the Runge-Kutta and LSTMs. ($b$) Lyapunov exponent Least Squares Fit of the Runge-Kutta, Vanilla LSTM, ESN-RC and FFNN.

The calculation already indicates that the FFNN reproduced a completely different dynamical behaviour, since a negative Lyapunov Exponent indicates converging

trajectories instead of diverging ones. However, the result is in line with the 3D plot in Fig.3.2 because it shows the trajectory collapsing to a single point, which will explain the negative character of the Lyapunov Exponent.

Accordingly to the value found in the bibliography ($\lambda = \mathbf{0.906}$)[46][47], the Runge-Kutta integration is correct, despite some minor differences that can be assigned to the numerical precision of the computer and the algorithm used. In addition, Vanilla LSTM shows again to be the best performing model between the LSTM variations and, in this case, the absolute best model, since it also outperforms the ESN-RC for this particular simulation.

Computing the exponents for all 50 simulations provides the results illustrated in Fig.3.8. Runge-Kutta integrations show consistency, as it could be expected given that the dynamics are fixed in the equations, which are numerically solved only changing the initial point. The values obtained from the FFNN are also within the expected range, confirming by being negative that the two attractors' structure and dynamics of the system were not successfully learnt.

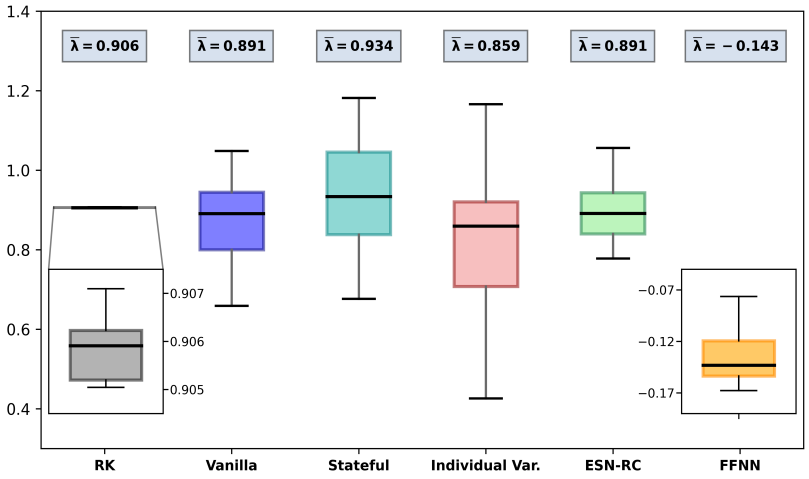



Figure 3.8: Largest Lyapunov Exponent boxplot obtained from the 50 Runge-Kutta, LSTMs, ESN-RC and FFNN simulations.

The Lyapunov Exponents extracted from the Stateful and Individual Variables LSTMs present a wider distribution, in particular in the case of the latter, which could indicate that the model is more likely to provide irregular results. Nonetheless, both values are below the 5% error margin with respect to the Runge-Kutta and theoretical value. For better performing models such as the Vanilla LSTM and ESN-RC, the error decreases to a value between 1%-2%, which, again, is a satisfactory result. Additionally, the ESN-RC shows a narrower distribution, which could lead to better consistency when simulating in different situations. A detailed look over these boxplots can be done using variations of Fig.3.8 located in Appendix C.2.

### 3.2.2 Correlation Dimension

The targeted value of the *Correlation Dimension* is $\mathbf{D = 2.05 \pm 0.01}$, as mentioned in [50][51]. Using the scheme indicated in 2.3.2, the dimension $D_2$ is obtained from each simulation. A boxplot is used to present the results of such calculations in Fig.3.9:

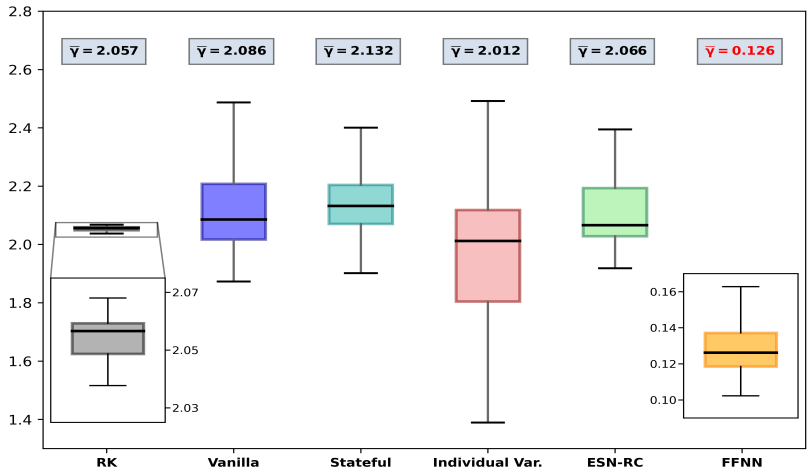



Figure 3.9: Correlation Dimension obtained from the Runge-Kutta, LSTMs, ESN-RC and FFNN 50 simulations.

The overall distribution of the results is similar to the one of the Lyapunov Exponent, which supports having chosen both metrics. The deviation from the Runge-Kutta value also matches with those of the before-mentioned metric. In this case, the dimension obtained from the ESN-RC simulations even lays inside the error margin provided by the bibliography, something of great significance. The Vanilla simulation also provides a great result, with a slight deviation lesser than 2%. In the case of the FFNN, it is once again confirmed by a different measurement that its reproduction of the Lorenz system is not correct, reflecting a totally different topology and a value that is close to the 94% of error with respect to the Runge-Kutta. Furthermore, the Correlation Dimension is a fractal dimension, a strong invariant of the system. This invites one to consider small deviations from the expected value as appreciable errors to take into account. Consequently, the wide distribution aroused from the Indivudal Variables LSTM could indicate significant discrepancies between the dynamics of the real and simulated systems. However, the Stateful LSTM seems to have captured the correct behaviour of the Lorenz equations even if its short-range prediction skills are second-rate, as proved in the previous section.

### 3.2.3 Recurrence Time

The third measurement of the overall performance is the *Recurrence Time*. The results are portrayed in Fig.3.10. Surprisingly, even the Runge-Kutta simulations show a wide distribution of values. This might be due to the fact that its value strongly depends on the point in the phase space where the algorithm is initialised. It can be deduced from the drawing of the trajectories in phase space that the time elapsed between two visits to a neighbourhood is not the same in an area close to one of the attractors than in a peripheral zone. In an attempt of providing robustness to the calculation, the starting point was always taken in the zone between both attractors, where trajectories jump from one to the other. This area is chosen because it roughly shows the same density of trajectories in most of the simulations. This procedure reduces the variability by situating the average values closer between them. This sensitivity also means that no unique values of the recurrence time can be found in bibliography.

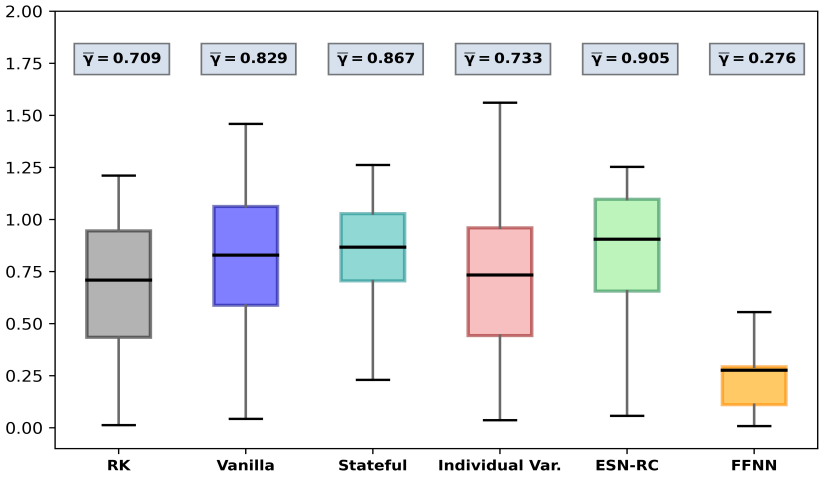



Figure 3.10: Recurrence Time boxplot obtained from the 50 Runge-Kutta, LSTMs, ESN-RC and FFNN simulations.

The fact that the behaviour of this metric, in comparison with Lyapunov Exponent and Correlation Dimension, does not follow the same line, could indicate that it is second-rate if considered as a metric. Notice how the best performing models (Vanilla LSTM and ESN-RC), also present significant dispersion when compared to, for example, the Stateful LSTM (the worst between LSTMs). However, the recurrence time is enough to reassure that the FFNN failed the learning task, since its deviation with respect to the Runge-Kutta mean value is greater than 60%. Other boxplots comparing the different models in more detail are contained in Appendix C.2.

# 4. Conclusions

Para concluir el proyecto, se muestra en esta última sección una evaluación completa y global de los resultados, así como las conclusiones extraídas del estudio. Además, se realiza una crítica sobre publicaciones previas relacionadas, para finalizar con sugerencias orientadas a posibles futuros estudios.

***Resumen***

## 4.1 Comprehensive Assessment of the Results

The first question that one may ask when using a Machine Learning model to emulate a physical system would be if the system accurately mimics the original one, reproducing with exactitude its behaviour. With respect to this, short-range predictions were studied, from where the ESN-RC model stepped out as the uppermost in terms of performance. Its prediction horizons vastly surpassed the rest. Moreover, the simple structure and its generalisation abilities need to be taken into account if one considers that it was trained with only a 2% of the original training dataset used on the rest of the models. A training time of 22.57 secs. was enough to provide the results presented in this dissertation. Although its performance was outstanding, the lack of training data reduces the confidence on the simulation aspect, leading towards considering also the Vanilla LSTM configuration. While being a more computationally heavy model in terms of training and architecture, it also performed consistently and provided accurate results. However, even if its prediction horizon is higher than any of the other LSTM configurations, it is greatly improved by the ESN-RC. Finally, the worst execution of more complex LSTM configuration, coincides with the results achieved by Chattopadhyay *et al.* [2] proving that the stateless Vanilla LSTM outclasses the Staetful one. In the same work, novel Reservoir Computing models similar to the ones applied here, also show greater prediction skills for the Lorenz96 system.

Ongoing with the study of the dynamics of the learnt systems, it was found that one of the biggest challenges when studying the viability of Machine Learning techniques to confront physical systems is to find adequate metrics to evaluate the performance of the different models. Being chaos native to the system of equations (1.2), classical regression metrics based on deviations from a reference value were not appropriate for the task. A comprehensive literature review on chaotic time-series was required in order to find plausible solutions, finally opting for Lyapunov Exponents ($\lambda$), Correlation Dimension ($D_2$) and Recurrence Time ($\gamma$). These set of measures made able to discuss the success or failure of the learning processes committed by the different Neural Network models. Results discussed in the previous section make it possible to reaffirm the ESN-RC as the model to highlight. The values obtained from the 50 simulations for the different metrics showed low dispersion and deviations (below 2%) from the reference mean values. This was the case for the first two metrics. Having mentioned this fact, the remaining models aside from the FeedForward Neural Network, successfully captured the general dynamics of the system, as it can be seen in Fig.3.2 and in the numerical values of $\lambda$ and $D_2$. Despite the failure of

the FeedForward architecture, both metrics have been proved to be consistent and trustworthy, since they reassured the expected results even in the outlier behaviour of this last model. The negative Lyapunov Exponent was the best example of such achievement. Having found these two measurements of significant usefulness, it is also important to discard the Recurrence Time as a purposeful metric in the required situation. As detailed in section 2.3.3, there is no unique reference value and the distribution arising from the 50 simulations is wide and disperse. Nonetheless, it is a simple concept, easy to implement and that could be useful to check performance in other types of dynamics and problems.

Despite the good results provided by Lyapunov Exponents and Correlation Dimension, its computational implementation, while widely studied over the years, is based on algorithms claimed to be robust against changes in their hyperparameters. Such affirmation was found to be inexact and, in occasions, misleading. Individual tuning of the algorithms was required to reach coherent and worthy results, a detail that was not able to be tracked down in the available bibliography. Some authors present results based on these metrics and similar comparisons to the ones described in this project that were not possible to be reproduced without further information that the one available in the published documents. This fact, together with the need of such individual fine tuning, creates concerns about some of the dissertations published about the use of these concepts. Due to this, the novel application of such techniques in a more general dataset framework (as it was the case) and the obtaining of sound results, support the effort and reinforce the value of the current study.

Taking everything into consideration, conducting a Machine Learning study in the field of Physics, requires clear knowledge of the problem itself and familiarity with Machine Learning models and their implementation in a computational language such as `Python`, `C++` or `Spark` if further scaling steps are desired. In addition, a reliable and generous data source is compulsory, as well as deciding the most suitable metrics for evaluating the results. The combination of these ingredients will be boosted if the scientist already has some experience in the field and is used to face with problems and the consequent search for solutions and answers. This recipe needs to be clear whenever facing a problem with novel AI techniques.

## 4.2 Future Research

The success of the ESN-RC structure, considering its simple architecture and the low amount of training data needed, furnish promising prospects for the implementation of such Neural Networks in these kind of problems. More complex architectures, combining various reservoirs and readouts or even creating hybrid models with LSTMs, could support the spread of these novel structures among the scientific community.

With respect to metrics, Machine Learning is now being applied to almost any possible field and not all the problems can be classified in one of the classical regression or classification groups. Further studies on how to measure and assess the learning, performance and general behaviour of the models, mainly long-term, could directly benefit upcoming projects.

For further extension of this study of the Lorenz System with the use of more powerful models, a possible option would be to implement Physics Informed Neural Networks (PINNs). These structures can be trained to numerically solve differential equations and have already been stated as fruitful techniques in a wide variety of applications. A brilliant review on the state of the art of such Neural Networks was published by Karniadakis *et al.* in [52].

# Appendices

# A. Lagrangian derivative

The mathematical definition of the Lagrangian derivative is presented in (A.1). The decomposition in components of the divergence term is detailed in (A.2), where $i$ can represent, for example, each cartesian variable.

$$\frac{D\mathbf{a}}{Dt} = \frac{\partial \mathbf{a}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{a} \tag{A.1}$$

$$[(\mathbf{v} \cdot \nabla)\mathbf{a}]_i = \sum_{i=1}^{n} \mathbf{v} \cdot \nabla a_i \tag{A.2}$$

# B. Understanding Lorenz63 spectrum of solutions.

## B.1 Chaos

In a few sentences, *chaos* can be understood as a long-term aperiodic behaviour expressed by a deterministic system that also shows sensitivity to initial conditions. The roots of this behaviour and characteristics are mainly in the non-linearity of the system. Usually chaos is misunderstood as instability, but the last one usually does not fulfil the three features mentioned before.

## B.2 Attractor and Strange Attractor

A rigorous definition can be difficult to find, but vaguely, an *attractor* is a set of points to which all trajectories in a neighbourhood of the points converge. In mathematical terms, an *attractor* should be a closed set $A$ with the properties below:

1. Any trajectory starting in $A$, stays in $A$. This is, $A$ is an invariant set.
2. $A$ attracts an open set $B$ of initial conditions that are within a certain distance. This means that if a point $x(0) \in B$, when $t \Longrightarrow \infty$, its distance with $A$ will go to zero. The set $B$ is called the Basin of the attractor $A$.
3. There is no subset of $A$ that fulfils conditions 1 and 2

Finally, if there is an attractor —satisfying then the three conditions— that also exhibits sensitive dependence on initial conditions, it is called *strange attractor.*

# C. Complementary Figures

## C.1 Chapter 1: Introduction

**Lorenz63 for** $(\sigma = 16.0, \rho = 45.92, \beta = 4.0)$

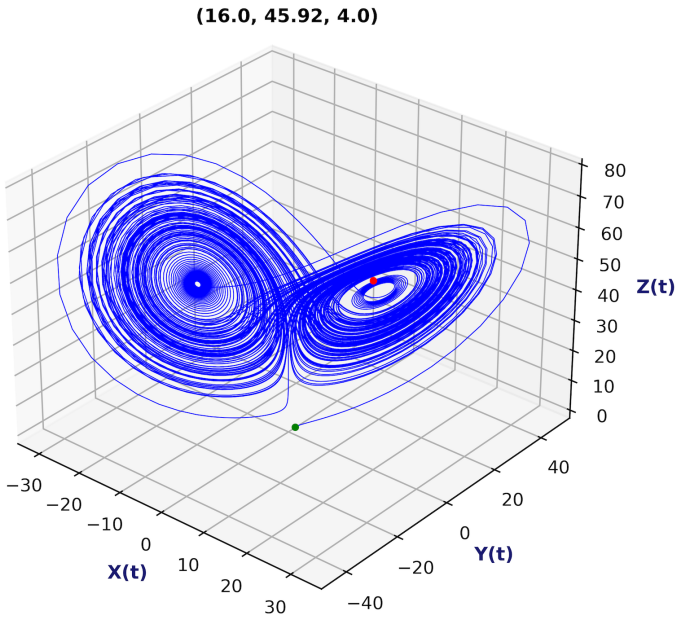



Figure C.1: Phase Space for $(\sigma = 16.0, \rho = 45.92, \beta = 4.0)$

## C.2 Chapter 3: Results

### Short-Term Simulation Figures

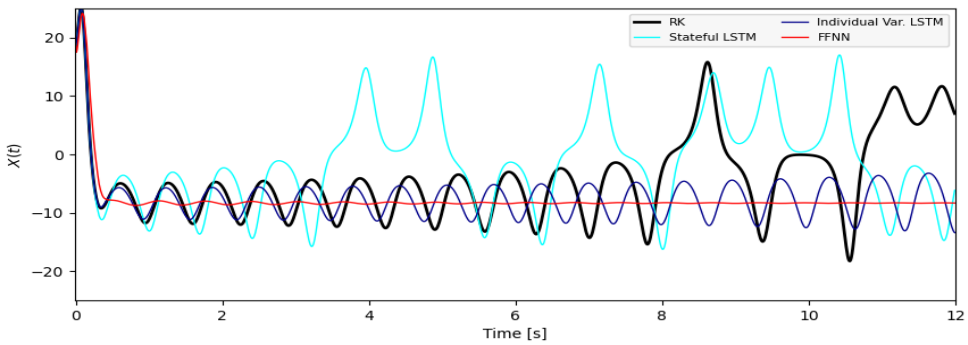



Figure C.2: Example 1 of bad performance by Stateful and Individual Variables LSTMs, as well as FFNN.

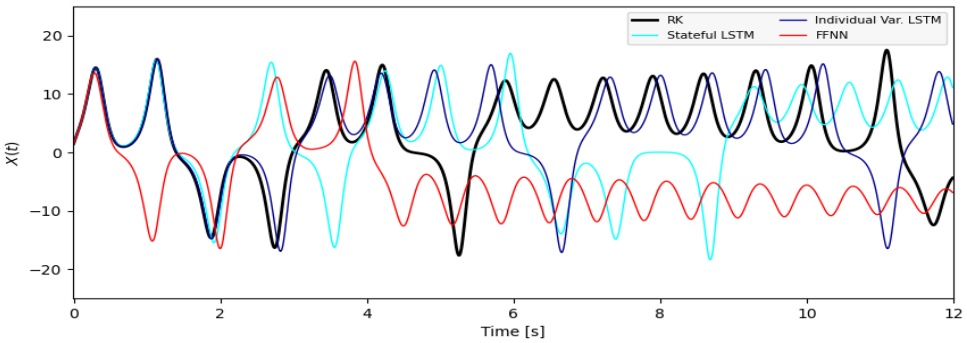



Figure C.3: Example 3 of bad performance by Stateful and Individual Variables LSTMs, as well as FFNN.

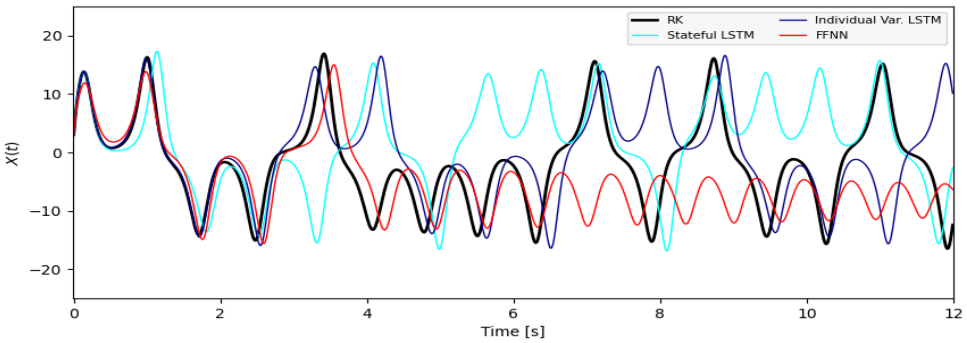



Figure C.4: Example 3 of bad performance by Stateful and Individual Variables LSTMs, as well as FFNN.

## Lyapunov Exponents

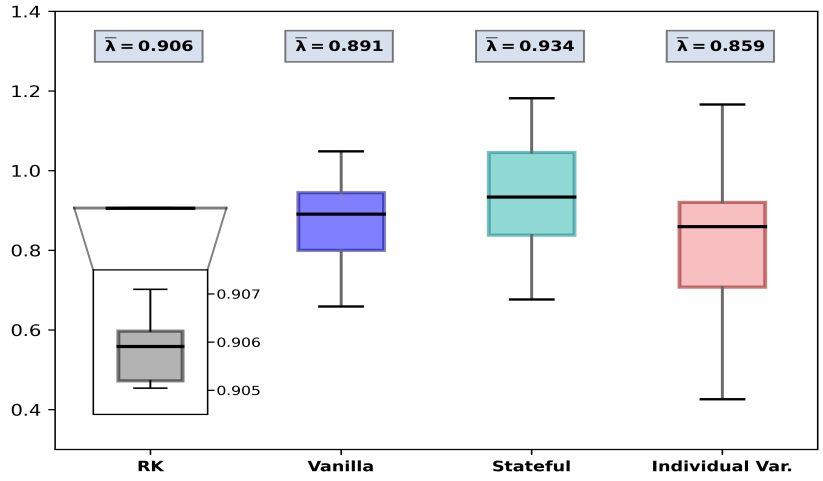



Figure C.5: Lyapunov Exponent additional boxplot Nº.1 comprising Runge-Kutta and LSTMs simulations

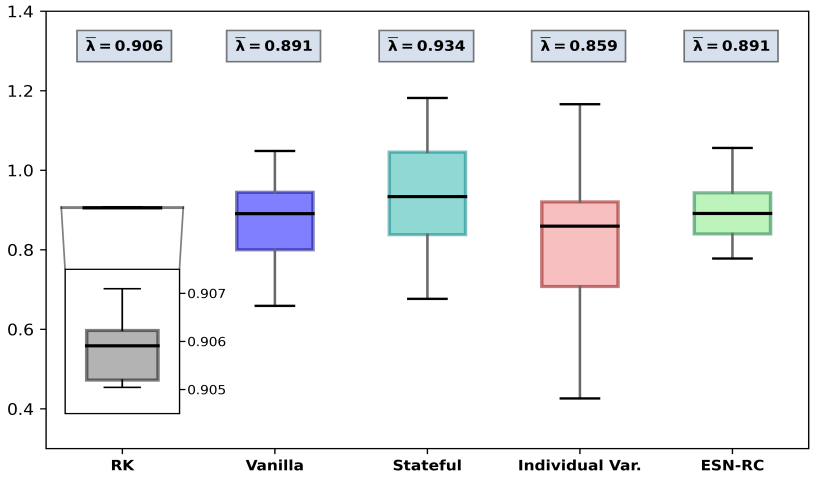



Figure C.6: Lyapunov Exponent additional boxplot Nº.2 comprising Runge-Kutta, LSTMs and ESN-RC simulations.

## Correlation Dimension

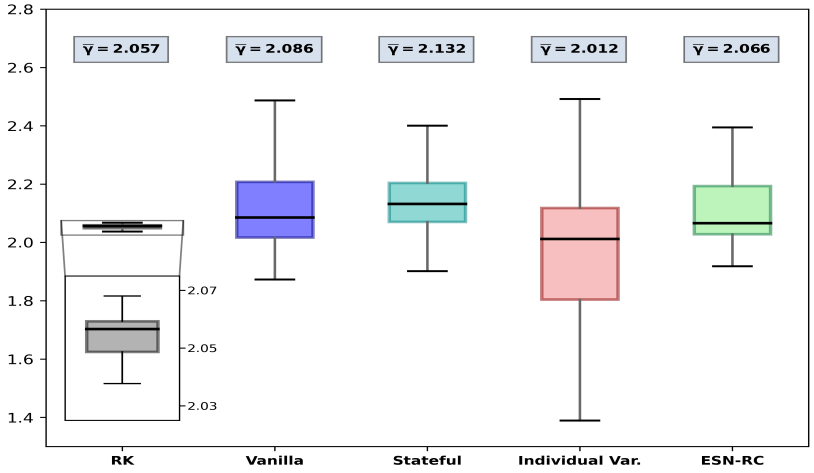



Figure C.7: Correlation Dimension additional boxplot Nº.1 comprising Runge-Kutta, LSTMs and ESN-RC simulations.

## Recurrence Time

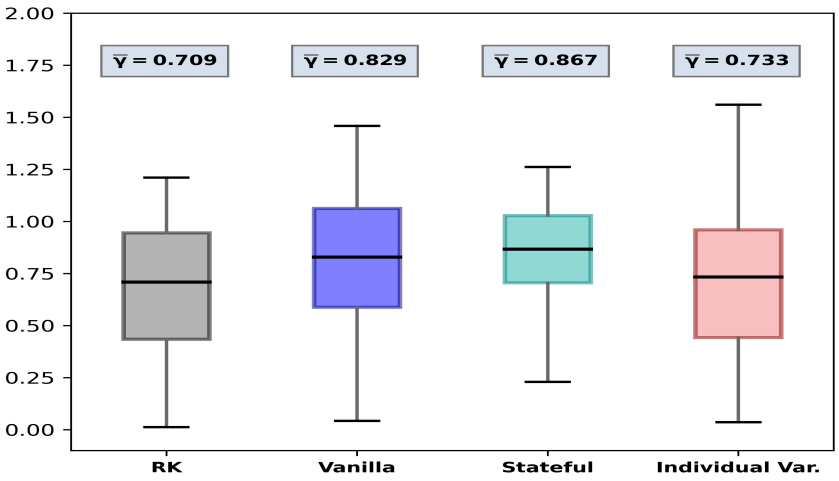



Figure C.8: Recurrence Time additional boxplot Nº.1 comprising Runge-Kutta and LSTMs simulations

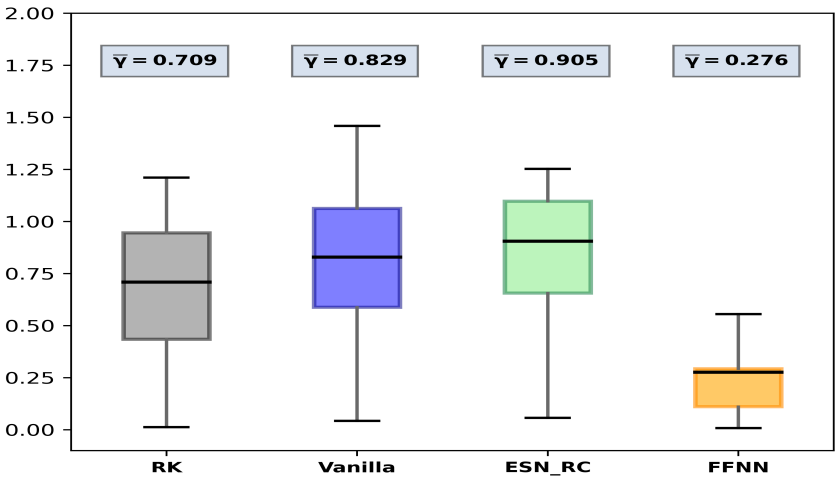



Figure C.9: Recurrence Time boxplot Nº.2 comprising Runge-Kutta, Vanilla LSTM, ESN-RC and FFNN simulations.

# References


[1] E.N. Lorenz. Deterministic nonperiodic flow. *Journal of atmospheric sciences*, 20(2):130–141, 1963.

[2] A Chattopadhyay, P. Hassanzadeh, K.V Palem, and D. Subramanian. Data-driven prediction of a multi-scale Lorenz 96 chaotic system using a hierarchy of deep learning methods: Reservoir computing, ANN, and RNN-LSTM. *CoRR*, abs/1906.08829, 2019.

[3] Artificial Intelligence (AI) vs. Machine Learning. University of Columbia, NY, USA. https://ai.engineering.columbia.edu/ai-vs-machine-learning/. Accessed: April 23rd 2023.

[4] W.S McCulloch and W. Pitt. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[5] D.O. Hebb. *The Organization of Behavior. A Neuropsychological Theory.* John Wiley and Sons., New York, USA., 1949.

[6] A.L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3:210–229, 1959.

[7] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[8] S. Linnainmaa. *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors.* PhD thesis, Master's Thesis (in Finnish), Univ. Helsinki, 1970.

[9] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, Cambridge, 1974.

[10] D.E. Rumelhart, G.E Hinton, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[11] S. Hochreiter and J. Schmidhuber. Long Short-term Memory. *Neural computation*, 9:1735–1780, 12 1997.

[12] ECMWF MOOC on Machine Learning in Weather and Climate. https://www.ecmwf.int/en/newsletter/173/news/ecmwf-launches-massive-open-online-course-machine-learning-weather-and. Accessed: March 1st 2023.

[13] A Gentle Introduction to Neural Networks Series Part 1. https://towardsdatascience.com/a-gentle-introduction-to-neural-networks-series-part-1-2b90b87795bc. Accessed: April 24th 2023.

[14] Machine Learning for Beginners: an Introduction to Neural Networks. https://towardsdatascience.com/machine-learning-for-beginners-an-introduction-to-neural-networks-d49f22d238f9. Accessed: April 24th 2023.

[15] T.E Quantrille and Y.A. Liu. *Artificial intelligence in chemical engineering*. Elsevier, 2012.

[16] StatQuest with Josh Starmer. Youtube Channel https://www.youtube.com/channel/UCtYLUTtgS3k1Fg4y5tAhLbw. Accessed: April 25th 2023.

[17] C. Arrizabalaga. Artificial Neural Networks and their use in simulation-based inference with the CAMELS dataset. Bachelor's thesis, University of La Laguna, San Cristóbal de La Laguna., 2022.

[18] Tensorflow Core. https://www.tensorflow.org/overview. Accessed: April 25th 2023.

[19] Keras API. https://keras.io/. Accessed: April 25th 2023.

[20] TensorFlow 2.0 Complete Course - Python Neural Networks for Beginners Tutorial. freeCodeCamp.org Youtube Channel. https://www.tensorflow.org/overview. Accessed: April 25th 2023.

[21] What are Recurrent Neural Networks. IBM. https://www.ibm.com/topics/recurrent-neural-networks. Accessed: April 25th 2023.

[22] H. Jaeger. Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach. *GMD-Forschungszentrum Informationstechnik, 2002.*, 5, 01 2002.

[23] N. Trouvain, L. Pedrelli, and T.T. Dinh. ReservoirPy: an Efficient and User-Friendly Library to Design Echo State Networks. *ICANN 2020 - 29th International Conference on Artificial Neural Networks, Sep 2020, Bratislava, Slovakia.*, 5, 09 2020.

[24] Predictability: a problem partly solved. Seminar on Predictability, 4-8 September 1995, vol. 1. ECMWF. Shinfield Park. Reading: ECMWF (1995). p. 1–18.

[25] Bo-Wen Shen. Aggregated Negative Feedback in a Generalized Lorenz Model. *International journal of bifurcation and chaos in applied sciences and engineering*, 29(3), 2019.

[26] R.C. Hilborn. *Chaos and Nonlinear Dynamics. An Introduction for Scientists and Engineers*. Oxford University Press, 2nd edition edition, 2000.

[27] Hateley, J. *Lorenz System*. Special Lecture, 2016. UCSB. http://web.math.ucsb.edu/~jhateley/paper/lorenz.pdf. Accessed: March 17th 2023.

[28] Z. Chen and H. Zhao. Cascade solutions of the Lorenz system. 2021.

[29] R.F. Williams. The structure of Lorenz attractors. *Publications Mathématiques de L'Institut des Hautes Scientifiques*, 50:73–99, 2021.

[30] L. F. Richardson. The Approximate Arithmetical Solution by Finite Differences of Physical Problems Involving Differential Equations, with an Application to the Stresses in a Masonry Dam. *Philosophical transactions of the Royal Society of London. Series A, Containing papers of a mathematical or physical character*, 210:307–357, 1911.

[31] J.R Pérez. *Métodos numéricos para la física y la ingeniería*. McGraw-Hill, Madrid, 2009.

[32] Stateful and Stateless LSTMs for Time Series Forecasting with Python. J. Brownlee. Machine Learning Mastery. https://machinelearningmastery.com/stateful-stateless-lstm-time-series-forecasting-python/. Accessed: March 19th 2023.

[33] ReservoirPy Documentation. https://reservoirpy.readthedocs.io/en/latest/index.html. Accessed: May 3rd 2023.

[34] ReservoirPy. GitHubhttps://github.com/reservoirpy/reservoirpy. Accessed: May 3rd 2023.

[35] N. Vandeput. Forecast kpi. In *Data Science for Supply Chain Forecasting*, chapter 2, pages 10–27. De Gruyter, Berlin, Boston, 2021.

[36] Pearson Correlation Coefficient. Scribbr. https://www.scribbr.com/statistics/pearson-correlation-coefficient. Accessed: April 19th 2023.

[37] R2. Investopedia. https://www.investopedia.com/terms/r/r-squared.asp. Accessed: April 19th 2023.

[38] B. Mandelbrot. *The Fractal Geometry of Nature*. W.H Freeman and Co., San Francisco, 1982.

[39] P. Grassberger and I. Procaccia. Characterization of Strange Attractors. *Phys. Rev. Lett.*, 50:346–349, Jan 1983.

[40] Nonlinear Dynamics. Series of Youtube lectures by Professor Liz Bradley, University of Colorado. https://www.youtube.com/watch?v=MizhVorgywY&list=PLF0b3ThojznQ9xUDm-EbgFAnzdbeDVuSz. Accessed: April 1st 2023.

[41] H.D Abarbanel. *Analysis of Observed Chaotic Data*. Springer, 1996. Institute for Nonlinear Science. University of California, San Diego.

[42] V.I. Oseledets. A multiplicative ergodic theorem. Characteristic Ljapunov, exponents of dynamical systems. *Trans. Moscow Math. Soc.*, 19:197–231, 1968.

[43] J.-P. Eckmann, S.O. Kamphorst, D. Ruelle, and S. Ciliberto. Lyapunov exponents from time series. *Phys. Rev. A*, 34:4971–4979, Dec 1986.

[44] A. Wolf, J.B. Swift, H.L. Swinney, and J.A. Vastano. Determining Lyapunov exponents from a time series. *Physica D: Nonlinear Phenomena*, 16(3):285–317, 1985.

[45] M.T. Rosenstein, J.J. Collins, and C.J. De Luca. A practical method for calculating largest Lyapunov exponents from small data sets. *Physica D: Nonlinear Phenomena*, 65(1):117–134, 1993.

[46] J.C. Sprott. Lyapunov Exponent Spectrum Software. *Department of Physics, University of Wisconsin, Madison, USA*, 2005. Revised November 2010.

[47] M.D. Hartl. Lyapunov exponents in constrained and unconstrained ordinary differential equations. *Department of Physics, California Institute of Technology, Pasadena CA, USA*, 2003.

[48] R. Hegger, H. Kantz, and T. Schreiber. Practical implementation of nonlinear time series methods: The TISEAN package, CHAOS 9, 413. 1999.

[49] X. Chen, T. Weng, C. Li, and H. Yang. Equivalence of machine learning models in modeling chaos. *Chaos, Solitons & Fractals*, 165:112831, 2022.

[50] J.C. Sprott. Lyapunov Exponent and Dimension of the Lorenz Attractor. *Department of Physics, University of Wisconsin, Madison, USA*, 1997. Revised May 2005.

[51] P. Grassberger and I. Procaccia. Measuring the strangeness of strange attractors. *Physica D Nonlinear Phenomena*, 9(1-2):189–208, October 1983.

[52] G. Karniadakis, Y. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. pages 1–19, 05 2021.