UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Master Degree in Physics

Final Dissertation

Reconstruction and Parameter Estimation of Dynamical Systems using Neural Networks

Thesis supervisor

Prof. Amos Maritan

Thesis co-supervisors

Prof. Valerio Lucarini

Dr. Varun Ojha

Candidate

Alberto Bassi

Academic Year 2021/2022

Abstract

Dynamical systems can be loosely regarded as systems whose dynamics is entirely determined by en evolution function and an initial condition, being therefore completely deterministic and a priori predictable. Nevertheless, their phenomenology is surprisingly rich, including intriguing phenomena such as chaotic dynamics, fractal dimensions and entropy production. In Climate Science for example, the emergence of chaos forbids us to have meteorological forecasts going beyond fourteen days in the future in the current epoch and therefore building predictive systems that overcome this limitation, at least partially, are of the extreme importance since we live in fast-changing climate world, as proven by the recent not-so-extreme-anymore climate phenomena.

At the same time, Machine Learning techniques have been widely applied to practically every field of human knowledge starting from approximately ten years ago, when essentially two factors contributed to the so-called rebirth of Deep Learning: the availability of larger datasets, putting us in the era of Big Data, and the improvement of computational power. However, the possibility to apply Neural Networks to chaotic systems have been widely debated, since these models are very data hungry and rely thus on the availability of large datasets, whereas often Climate data are rare and sparse. Moreover, chaotic dynamics should not rely much on past statistics, which these models are built on.

In this thesis, we explore the possibility to study dynamical systems, seen as simple proxies of Climate models, by using Neural Networks, possibly adding prior knowledge on the underlying physical processes in the spirit of Physics Informed Neural Networks, aiming to the reconstruction of the Weather (short term dynamics) and Climate (long term dynamics) of these dynamical systems as well as the estimation of unknown parameters from Data.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

The importance of Numerical Weather and Climate forecasting [Lynch, 2008] in our society is unquestionable. Though the Atmosphere is a very complex system, dissipative, chaotic and out of equilibrium, there are several and well studied toy models that capture the most important processes in a coarse grain way in low dimensional spaces. It is for example the case of [Lorenz, 1963], which simplifies the description of the convective motions of a fluid in a gravitational field between two horizontal plates kept at a certain distance and with a certain temperature difference. Another paradigmatic model is given by [Lorenz, 1995], a coarse grained model defined in a ring that describes three important processes for a generic atmospheric quantity: advection, convection and forcing. Thus, in order to better understand the Atmosphere, it is natural to start studying these simplified chaotic systems first.

Chaotic dynamics has been pioneered in the 19th Century by Henri Poincaré, who was mainly interested in the motion of three celestial bodies under mutual gravitational attraction (the so-called three body problem). Poincaré, rather than studying single trajectories, considered the behaviour of may trajectories starting from an ensemble of initial conditions, showing that strange and complicated orbits, that we now call chaotic, were possible.
A deeper comprehension of Chaos, however, came only in the 20th Century, even if we had two wait the second half of the Century to appreciate chaotic systems by using computer simulations, as done by [Lorenz, 1963]. Roughly speaking, a system is chaotic when it exhibits the so-called sensitive dependence of initial conditions. This effect is reported in popular culture also as "butterfly effect": a butterfly flapping its winds in Japan can cause a hurricane in the United States. Moreover, for a lucky coincidence, [Lorenz, 1963] attractor, one of the most famous examples of chaotic systems, is loosely resembling a butterfly. The exponential rates of growth of perturbation of neighbouring orbits are called characteristic or Lyapunov exponents. Therefore, a system exhibiting sensitive dependence on initial conditions has at least one of the Lyapunov exponents positive and in general we can also say that autonomous continuous dynamical systems have the intermediate characteristic exponents zero, since it is the growth rate along the flux of the system.

The theory beyond chaos theory is the one of dynamical systems, which can be referred as mathematical models in which the future evolution is entirely determined by the initial state. This formulation is very important, because dynamical system are indeed deterministic systems and the future evolution is perfectly known if the initial conditions is perfectly known. The problem in this assertion is that in real world application the state of the a system is never perfectly known, because inevitably measurements errors are committed, leading to diverging behaviour, in the sense of sensitive dependence, of trajectories. Even though weather prediction equations are deterministic, every weather forecast suffer from

this sensitivity which ultimately prevents us from having good forecasting beyond 14 days in the future, with the current computational power. For this reasons, we have to abandon the idea to have definite trajectories and instead focus on a Statistical description.

Nowadays, Machine Learning techniques are used in practically every field of human knowledge. Even if the first Machine Learning models date back to the late 1950s, when Perceptron was introduced [Rosenblatt, 1958], it was only at the beginning of the 2010s that we observed an exponential increasing of the usage of Neural Networks. This is essentially due to two factors. First, the progress in computational power the last year has been impressive and nowadays everyone on his own domestic computer can train sufficiently deep networks for a great variety of tasks. Secondly, we live in the era of Big Data, where large datasets are pubblically available on the Internet. Deep Learning models especially are very data hungry and require a large amount of data to have good performances and not lead to overfitting. The more our computers improve, the deeper are the networks we can train, which will create the need of larger datasets to have better performance. However, the question whether Machine Learning is able to replicate dynamical systems and being therefore useful in Weather and Climate forecasting has been long debated, since Machine Learning models are data driven models that look only on past statistics and therefore it may fails in reproducing characteristic not previously seen by the algorithm, which is especially true in a contest in which measures are rare and sparse, as it often happens with Climate data. See [Schultz et al., 2021] for an overview of the problem.

From the methodological point of view, it would be thus much appreciate if Neural Networks were able to learn a chaotic dynamics. Therefore, in this thesis we intend to fill this gap by studying the applications of Machine Learning techniques, in particular Neural Networks, to Chaotic Dynamical systems, with particular focus on dynamics reconstruction and parameter estimation.

1.2 Research Hypothesis and Objectives

In thesis we focus on the following three questions:

1. Are Neural Networks capable of reproducing the short term dynamics, also known as Weather, at least in a temporal horizon dictated by the inverse of first Lyapunov exponent?

2. Are Neural Networks as well capable of learning and reproducing the statistical properties of the long term dynamics, known as Climate, of such systems?

3. Are Neural Networks able to retrieve the unknown parameters of a supposed underlying model?

It is clear that if the answer to the first question is affirmative, then it would in principle be possible to improve the current horizon of Weather predictions, keeping anyway in mind that since we are talking about chaotic systems, trajectories will eventually diverge.
The answer to the second question would lead to modelling long term behaviour of these systems, which could be applied to estimate for example the probability of extreme events.
Lastly, Neural Networks can be used to infer hidden properties of the system, which are not directly observable. To be more precise, our goal is to see if the it is possible to learn the parameters of some paradigmatic models, such as [Lorenz, 1963], not comparing with the true values, with which the dynamics is supposed to be generated, but relying only on a dynamical ansatz equation, which loosely resembles Phsyics Informed Neural Networks (PINNs), see [Raissi et al., 2019].

1.3 Solution Approach

To answer to the questions above mentioned in section 1.2, we try different neural network architectures and methodology, applied to some well known and studied low dimensional, such as [Lorenz, 1963] and [Rössler, 1976], and high dimensional, like [Lorenz, 1995], chaotic systems.

1.3.1 Dynamics Reconstruction

To address the first two questions, we study Recurrent Neural Networks (RNN) and Feed-Forward networks to learn low dimensional models in the sense of Weather and Climate. The robustness of this approach derives from the fact that RNN are indeed dynamical systems, whose dynamics takes place in a higher dimensional space and so its effectiveness is guaranteed by classic embedding theorems, see [Takens, 1979]. We will make use of Long Short Term Memory (LSTM) networks ([Hochreiter and Schmidhuber, 1997]) and Reservoir Computers ([Maass et al., 2002, Jaeger, 2001]), showing that the two first questions can be successfully answered.
While recurrent networks are already dynamical systems, standard Feed-Forward Networks [Haykin, 1994] are efficient learners of functions [Cybenko, 1989], while Convolutional Networks [LeCun et al., 1989] can impressively learn complex spatial correlations. Therefore these models are not made to use a direct Data Driven loss, but instead they can applied to model its evolution function by comparing it to the ground truth in a similar way to how PINNs work [Raissi et al., 2019]. This approach could in principle allow us to try to answer the three questions together, since PINNs can learn the equations and the parameters of the model at the same time.

With regards to Dynamics Reconstruction, we will thus show that:

- The Weather and the Climate of low dimensional chaotic dynamical system can successfully be learned by LSTM, Reservoir Computers and Feed-Forward, even though for the last two architectures the results are heavily dependent on 1) the system that has been used and 2) on manual fine-tuning of hyper-parameters.

1.3.2 Parameters Estimation

To retrieve models' parameters, we will implement Autoencoders [Kramer, 1991] in order to learn the parameters of the model in the encoded space. Autoencoders, as the name suggests, learn to encode high-dimensional data in a low dimensional space while maintaining the maximal information content. Not by chance, they were first used by [Kramer, 1991] to perform non-linear Principal Components Analysis (PCA) [Pearson, 1901].

With regards to Parameters Estimation, we will thus show that:

- Autoencoders can learn the parameters of low dimensional chaotic models with regularized with both a purely Data-Driven and a Physics-Informed loss. Even though less efficient, the latter regularization loss is more robust since in experimental situations the true parameters are not known and one makes only guess about the dynamical equation.

- Autoencoders can successfully learn a chaotic phase transition in high dimensional models in a complete unsupervised way (no regularization used).

1.4 Related Work

1.4.1 Dynamics Reconstruction

In the recent years, many applications to dynamics reconstruction have been performed starting from [Pathak et al., 2017], who first used Reservoir Computers [Maass et al., 2002, Jaeger, 2001] to 1) predict the short term dynamics of [Lorenz, 1963] further than the timescale given by the first Lyapunov exponent and 2) and replicate the chaotic nature of the system. Later Reservoir computer were applied to Multi Variate Time series [Bianchi et al., 2018] and with a multi reservoir approach [Freiberger et al., 2020]. Moreover, in the contest of Data Assimilation [Park and Xu, 2016] Reservoir Networks have been applied with rare and sparse updates in enlarge the predictive horizon [Fan et al., 2020] and [Haluszczynski and Räth, 2021] used this approach to integrate out perturbation and controlling dynamical systems. Reviews on methods that combine data assimilation techniques with Machine Learning can be found in [Sonnewald et al., 2021] and [Gottwald and Reich, 2021].
However, these models are very sensible to the choice of the hyper-parameters and it is not clear if [Pathak et al., 2017] are able to give consistent results in the whole attractor, since [Lorenz, 1963] is notoriously heterogeneous, i.e. some regions are more chaotic than others.
Long Short Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] have achieved impressive results in the field of Natural Language Processing [Wang and Jiang, 2016, Murthy et al., 2020] and they were the state-of-the-art model for this task till the advent of Transformers [Vaswani et al., 2017]. In literature, we have found some works on the applications of LSTM to dynamics reconstruction; for example [Vlachas et al., 2018] use LSTM to forecast high dimensional chaotic systems and [Barzegar et al., 2022] implement stacks of LSTM to analyse multi-variate fast changing time series. Instead [Yeo and Melnyk, 2019] worked on the pdf approximation of by using LSTM.
Another interesting approach is to train the network to reproduce the evolution function by means of an efficient integrator scheme, such as Runge-Kutta methods, like in [Raissi et al., 2018], or to use directly Residual Networks, which are networks in which connections may skip some layers, to build cells that resemble those integration schemes, as done by [Fablet et al., 2018]. [Raissi et al., 2018] approach has been further developed by [Teng and Zhang, 2019] to include Convolutional networks and LSTM.
Autoencoders [Kramer, 1991] have been used to: deep reconstruction of strange attractors from lower dimensional time-series by [Gilpin, 2020]; variational reconstruction of systems from noisy and sparse data by [Nguyen et al., 2020]; recognize chaos from the recurrence plot of the Lyapunov exponents by [Nam and Kang, 2021].

1.4.2 Parameters Estimation

For what regards parameter estimation, [Brunton et al., 2016] published *SInDy*, a general algorithm that allows to retrieve the correct equations, and so the correct parameters, directly from data by using a suitable dictionary of candidate functions, whose weights are learnable. Following this, [Vortmeyer-Kley et al., 2021] proposed a new loss function consisting in the sum of the the norm of the difference of the predicted dynamics with the true value, plus a term that takes into account the angular differences between vectors.
We are not aware of direct applications of PINNs to dynamical systems, meaning results in which the dynamics equation is learned as only function of time by exploiting the power of automatic differentiating packages [Baydin et al., 2017] to compute derivatives. However, an indirect approach has been implemented by [Rackauckas et al., 2020], who used Neural Networks as Universal Differential Equation (UDE) approximator. This method permits to insert some prior physical knowledge in the equations by leaving some uncertain part to be learned by a Neural Network.
In this contest, Autoencoders have been exploited to: directly analyse single parameters time-

series by [Almazova et al., 2021]; recognize phase transitions in a completely unsupervised way by [Wetzel, 2017]; learning summary statistics to enhance Approximate Bayesan Computation (ABC) by [Albert et al., 2022].
We conclude by citing an comprehensive work the replication of the Local Lyapunov Exponents (LLE) time-series by using different neural networks architectures in [Ayers et al., 2021].

1.5 Structure of this Thesis and Code Availability

This thesis is structured as follow:

- In chapter 2 we present the basic theory on dynamical systems, exposing the most interesting properties that could be useful when analysing such models with Neural Networks.
- In chapter 3 we present the learning framework under which Neural Network operates, discussing also the details of the architectures that we will make use of.
- In chapter 4 we present the results of our experiments, trying to address the three questions mentioned earlier in section 1.2.

All the code can be found at `https://github.com/lupoalberto98/Machine-Learning-Dynamical-Systems.git`.

Chapter 2

Dynamical Systems

2.1 Introduction

In this chapter, we will discuss the fundamental features of dynamical systems. We first start giving some basic definitions and properties in section 2.2. Then we will discuss, giving a precise definition, the concept of attractor in section 2.3 and we will proceed by discussing in more detail some important dynamical system in section 2.4, such as the Logistic Map, defined in Equation 2.17, the [Lorenz, 1963] and the [Lorenz, 1995] systems. In section 2.5 we will expose the statistical theory applied to chaotic systems, which is necessary due to the fast forgetting times that such systems undergo. In section 2.6 we discuss the concept of dimension by giving some examples that are important to deal with systems that have non-integer dimensions. In section 2.7 we define what the delays method is and why dynamics embeddings are important. Thus, in section 2.8 we will define what Characteristic (or Lyapunov) exponents are, presenting an algorithm for numerical computation. Then we briefly discuss the concepts of Stable and Unstable Manifolds in section 2.9. In section 2.10 we will present two definitions of entropies in the context of dynamical systems and we will then conclude by giving two definitions of metrics that will be later used to asses the reconstruction power of Neural Networks of the Weather and the Climate of paradigmatic models in section 2.11.

2.2 Basic Definitions

With dynamical systems we mean every mathematical system or model whose future evolution is characterized by the initial value. These systems may be be continuous in time, and we will call them continuous dynamical systems, or may be dependent on a discretized time, and in that case we will call them *maps*. In case of continuous time, we will focus our attention on autonomous dynamical systems, characterized by the dynamic equation

$$\dot{\mathbf{x}}_t = \mathbf{g}(\mathbf{x}_t) \,, \tag{2.1}$$

where $\mathbf{g} : \mathbb{R}^N \to \mathbb{R}^N$ is the evolution function and $\mathbf{x}_t = (x_t^{(1)}, x_t^{(2)}, \ldots, x_t^{(N)}) \in \mathbb{R}^N$. The space of all points $\mathbf{x}$ is called *phase space* Γ and can be either $\mathbb{R}^N$, as we have set, or more generally a Hilbert space $\mathbb{E}$ or more commonly a compact manifold $\mathbb{M}$. The path followed by the system in phase space is called *orbit* or *trajectory*.
In this thesis restrict our attention to *smooth* dynamical systems, i.e. the ones for which the stability matrix

$$\mathbb{L}_{ij} := \frac{\partial g^{(i)}(\mathbf{x})}{\partial x^{(j)}} \tag{2.2}$$

exists for any i, j and any $\mathbf{x}$ in the phase space. For such systems, the theorem of existence and uniqueness of the solution holds, corroborating then the statement that the dynamics

is known by fixing the initial condition.
For continuous dynamical systems, it is useful to introduce the *flow* $f^t(\mathbf{x}) : \mathbb{R} \times \mathbb{R}^N \to \mathbb{R}^N$, in analogy with fluids, defined as

$$f^t(\mathbf{x}_0) \equiv \mathbf{x}_t : t \geq 0, \quad \dot{\mathbf{x}}_t = \mathbf{g}(\mathbf{x}_t) \, , \tag{2.3}$$

i.e. $f^t(\mathbf{x}_0)$ is the trajectory $\mathbf{x}_t$ at time t, knowing that the initial point $\mathbf{x}_0$. We will call it also integrator operator, since it integrates a trajectory starting at x_0 for a time t.

The evolution of the properly normalized probability density function (pdf) $\rho_t : \Gamma \to [0, 1]$ in phase space is determined by the equation [Lasota and Mackey, 1985]:

$$\frac{\partial \rho_t(\mathbf{x})}{\partial t} + \nabla \cdot \big(\rho_t(\mathbf{x}) \, \mathbf{g}(\mathbf{x})\big) = 0 \, , \tag{2.4}$$

which can be expanded as

$$\frac{\partial \rho_t(\mathbf{x})}{\partial t} + \rho_t(\mathbf{x}) \, \nabla \cdot \mathbf{g}(\mathbf{x}) + \mathbf{g}(\mathbf{x}) \cdot \nabla \rho_t(\mathbf{x}) = 0 \, . \tag{2.5}$$

It follows that we can distinguish two kind of systems.
If $\nabla \cdot \mathbf{g}(\mathbf{x}) = 0$, Equation 2.1 describes the evolution of an ensemble of points advected by an incompressible velocity field $\mathbf{g}(\mathbf{x})$ and therefore the volume in phase space is conserved. We say them *conservative* dynamical systems.
If instead If $\nabla \cdot \mathbf{g}(\mathbf{x}) < 0$, the phase space volume contracts and we speak of *dissipative* dynamical systems. The set of points of a dissipative system evolves in a space whose dimension is $d < n$, smaller than the original dimension n, and it is called *attractor*. Instead, the dynamics of conservative systems occupy all the phase space and they do not have an attractor. We do not consider systems for which $\nabla \cdot \mathbf{g}(\mathbf{x}) > 0$ since it leads to unbounded and uninteresting trajectories.
A class of conservative dynamical systems is given by Hamiltonian systems, see [Goldstein et al., 2002]. Given the $2N$ generalized position-momentum coordinates $(q^{(i)}, p^{(i)})$ for $i = 1, 2, \ldots, N$ and the Hamiltonian function $H(\mathbf{q}, \mathbf{p}, t)$, the equation of motion (Hamilton equations) read

$$\begin{aligned} \dot{q}^{(i)} &= \frac{\partial H}{\partial p^{(i)}} \, , \\ \dot{p}^{(i)} &= -\frac{\partial H}{\partial q^{(i)}} \, . \end{aligned} \tag{2.6}$$

By the identification: $x^{(i)} = q^{(i)}$; $x^{(i+N)} = p^{(i)}$ and $g^{(i)} = \partial H / \partial p^{(i)}$; $g^{(i+N)} = -\partial H / \partial q^{(i)}$ for $i = 1, 2, \ldots, N$, Hamilton equations can be rewritten as

$$\dot{\mathbf{x}} = \mathbb{J} \, \nabla H(\mathbf{x}, t) \tag{2.7}$$

where

$$\mathbb{J} = \begin{pmatrix} \mathbf{0} & \mathbb{I}_N \\ -\mathbb{I}_N & \mathbf{0} \end{pmatrix} \tag{2.8}$$

is the *symplettic* unit.
It is clear that

$$\nabla \cdot \mathbf{g} = \sum_{i=1}^{2N} \frac{\partial g^{(i)}}{\partial x^{(i)}} = \sum_{i=1}^{N} \left(\frac{\partial}{\partial q^{(i)}} \frac{\partial H}{\partial p^{(i)}} - \frac{\partial}{\partial p^{(i)}} \frac{\partial H}{\partial q^{(i)}} \right) = 0 \, , \tag{2.9}$$

and thus the system is conservative. Their attractor is the energy shell.

In case that the evolution depends explicitly on time, then the system is non-autonomous and can be written as

$$\dot{\mathbf{x}}_t = g(\mathbf{x}_t, t) \ . \tag{2.10}$$

An example of non-autonomous dynamical system is given by the forced damped pendulum equation

$$\frac{d^2\theta}{dt^2} + v\frac{d\theta}{dt} + \sin\theta = T\sin(2\pi f t) \ , \tag{2.11}$$

where θ is the angle with respect to the vertical, the pivot friction is represented by the second term controlled by v (friction coefficient), the second term represent gravity and the RHS represent a sinusoidal forcing of frequency f. The system can be seen either as a 2d non-autonomous dynamical system by defining the variables $x^{(1)} = d\theta/dt$ and $x^{(2)} = \theta$, or a 3d autonomous system by setting also $x^{(3)} = 2\pi f t$. In the latter case, Equation 2.11 can be rewritten as

$$\begin{aligned} dx^{(1)}/dt &= T\sin x^{(3)} - \sin x^{(2)} - v x^{(1)} \ , \\ dx^{(2)}/dt &= x^{(1)} \ , \\ dx^{(3)}/dt &= 2\pi f \ , \end{aligned} \tag{2.12}$$

which, for some choice of the parameters (T, f, v) exhibits the so-called *sensitive dependence on initial condition* characterizing chaos, for which two bounded trajectory that slightly differ at some time t, will eventually be very far apart as time goes on, even though the system is deterministic and can be thus completely reconstructed given the initial condition, see Figure 2.1. A trajectory is said to be bounded if there exists a ball in phase space, such that $\mathbf{x}(t) < R < \infty$, $\forall t \in \mathbb{R}$. This condition is required in order to exclude trivial example in definition of chaos. For instance, the system $dx/dt = x$ will trivially exhibit divergence of nearby orbits if we let them go to infinity, even if is not chaotic.

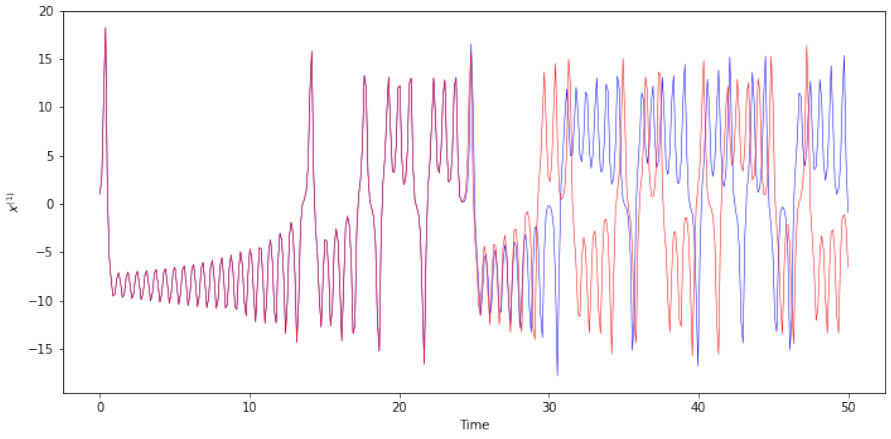


Figure 2.1: Separation of two orbits generated with initial conditions differing of $\Delta x^{(1)} = 10^{-6}$ for Lorenz 63 system, see later subsection 2.4.2 where the model is discussed in detail. We can appreciate how trajectories negligibly separable at the beginning, eventually will exhibit a divergence of the same order of magnitude of the dynamics.

Notice that any N-dimensional non-autonomous dynamical system can be rewritten as a $N+1$ dimensional dynamical system by defining $\mathbf{x}^{(N+1)} = t$ and $g^{(N+1)} = 1$. Therefore, for the sake of simplicity, we will restrict our analysis to autonomous dynamical systems. A natural question arise: what is the minimum phase space dimension n for which chaos will come out? We state that the answer [Ott, 2002] is

$$N \geq 3 \tag{2.13}$$

for continuous time dynamical systems.
Discrete time dynamical systems are also called *maps* and are defined by evolution equation

$$\mathbf{x}_{t+1} = \mathbf{m}(\mathbf{x}_t) \ , \tag{2.14}$$

where again we assume that the evolution function $\mathbf{m}$ has continuous first order derivatives. Also in this case, the dynamic is fully known once the evolution map and the initial condition $\mathbf{x}_0$ are given, meaning that the system is deterministic. In this case we recursively write

$$\mathbf{x}_t = \mathbf{m}(\mathbf{x}_{t-1})) = \mathbf{m}(\mathbf{m}(\mathbf{x}_{t-2})) = \underbrace{(\mathbf{m} \circ \mathbf{m} \circ \cdots \circ \mathbf{m})}_{t \text{ times}}(\mathbf{x}_0) \ . \tag{2.15}$$

Again, we can define a discrete flow $f^t(\mathbf{x}) : \mathbb{N} \times \mathbb{R}^N \to \mathbb{R}^N$ as

$$f^t(\mathbf{x}_0) \equiv \mathbf{x}_t : t \geq 0, \ \mathbf{x}_{t+1} = \mathbf{m}(\mathbf{x}_t) \ . \tag{2.16}$$

In case of maps, the minimum dimension required to have chaos depends on whether the map is invertible or non-invertible. We say that a map is invertible if, given $\mathbf{x}_{t+1}$, we can write $\mathbf{x}_t = \mathbf{m}^{-1}(\mathbf{x}_{t+1})$ and we call $\mathbf{m}^{-1}$ the inverse of $\mathbf{m}$. For example, the 1d logistic map, defined as

$$x_{t+1} = r x_t (1 - x_t) \ , \tag{2.17}$$

where r is a positive constant, is not invertible, because there are two possible values of $\mathbf{x_t}$ given $\mathbf{x}_{t+1}$ being the evolution function quadratic. On the other hand, the 2 dimensional map defined by

$$\begin{aligned} x_{t+1}^{(1)} &= f(x_t^{(1)}) - J x_t^{(2)} \ , \\ x_{t+1}^{(2)} &= x_t^{(1)} \ , \end{aligned} \tag{2.18}$$

which is clearly invertible as long as $J \neq 0$ [Ott, 2002]. The dimensionality requirement for invertible map to have chaos [Ott, 2002] is

$$N \geq 2 \ , \tag{2.19}$$

while if the map is non-invertible we can have chaos also for $N = 1$, as what happen for the logistic map. Thus we have seen that dimensionality plays an important role when talking about chaos.

A N-dimensional continuous time dynamical system can be reduced to $N - 1$ dimensional map by using *Poincaré sections*, which map a N dimensional flow to a $N - 1$ dimensional one. The construction is done as follow: for a generic N-dimensional system, take a generic $N - 1$ dimensional surface in the phase space and observe the intersection of the orbit with the surface, as shown in Figure 2.2 for $N = 3$.

Points A and B represents two consecutive crossing points. We point out that B is distinctively determined by A, since it is the evolved dynamics by fixing A as initial condition. Viceversa, A is uniquely determined by B by reversing the time flow. Therefore, Poincarè sections allow us to build $N - 1$ dimensional invertible maps, in the example of Figure 2.2 building a map that transform the coordinates $(x_t^{(1)}, x_t^{(2)})$ to $(x_{t+1}^{(1)}, x_{t+1}^{(2)})$. In this way, chaos dimensionality requirement stated by Equation 2.13 for continuous dynamical system is linked to invertible maps by Equation 2.19, [Ott, 2002]
Another way to build maps from continuous system is to sample a continuous trajectory $\mathbf{x}_t$ and discrete time steps and consider the forward integrator

$$\mathbf{x}_{t+1} = \int_t^{t+\delta t} \mathbf{g}(\mathbf{x}_\tau) \, d\tau \equiv \mathbf{m}(\mathbf{x}_t) \ , \tag{2.20}$$

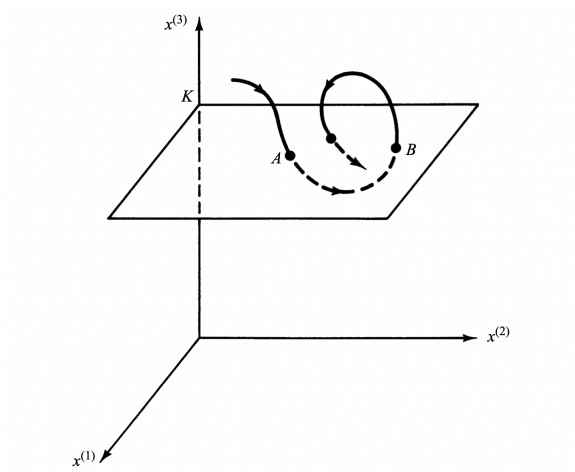


Figure 2.2: 2 dimensional Poincarè section of a 3d continuous dynamical system (figure taken from [Ott, 2002])

which from a N dimensional continuous system yields a N dimensional invertible map, since the Equation 2.1 can be integrated backwards in time.

2.3 Attractors

In many cases the evolution of a dynamical system seems to converge asymptotically to a certain a set in the phase space, called *attractor*. As an example of attractor, let us consider the damped harmonic oscillator

$$\frac{d^2x}{dt^2} + v\frac{dx}{dt} + \omega^2 t = 0 \,, \tag{2.21}$$

where v controlo the friction term and ω is the frequency. It can be considered a 2d dynamical system with the choice $x^{(1)} = x, x^{(2)} = dx/dt$. As the dynamics goes on, the system asymptotically reaches the point $x^{(1)} = 0, x^{(2)} = 0$, and therefore the attractor is constituted by only one point and thus have dimension zero.

An example of an attractor of dimension one is given by Van der Pol equation [van der Pol, 1920], describing simple vacuum oscillator circuits:

$$\frac{d^2x}{dt^2} + (x^2 - \eta)\frac{dx}{dt} + \omega^2 t = 0 \,, \tag{2.22}$$

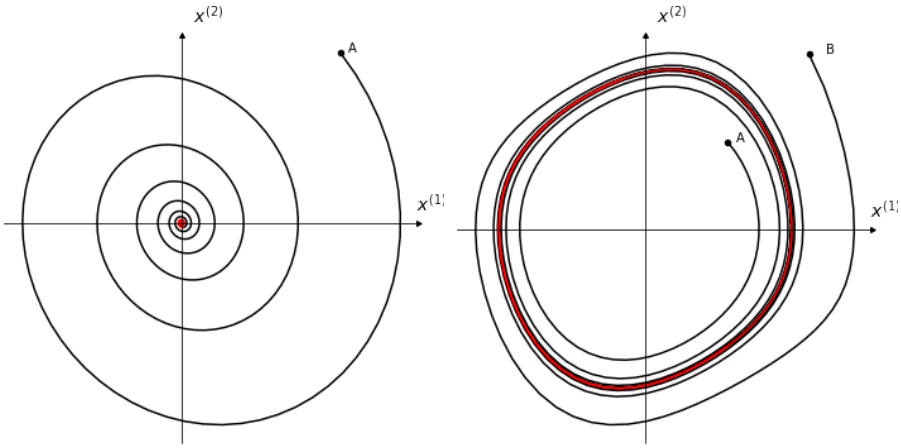
where η is a constant. In this case, the dynamics does not asymptotically tend to a limiting point, but rather to *limiting cycle*, as showed in Figure 2.3, where the trajectory converges both staring from the inside (point A) and the outside (point B).
More formally, we can define an attractor as follow:

Definition 1 (Attractor)**.** *Given a continuous or discrete dynamical system, with flow $f^t(\mathbf{x})$, the attractor is the smallest subset $\mathbb{A}$ of the space phase such that:*

- *It is invariant under the forward dynamics, i.e.*

$$\mathbb{A} = f^t(\mathbb{A}) \,, \qquad \forall t \geq 0 \,. \tag{2.23}$$

- *There exists a neighbour set of $\mathbb{A}$, called basin of attraction $\mathbb{B}$, such that for every open set $O \supset \mathbb{A}$, $\exists\, T > 0$ such that $\forall t > T$, $f^t(\mathbb{B}) \subset O$.*



(a) (b)

Figure 2.3: Generated Phase diagram for the damped harmonic oscillator (a) and the Van der Pol equation (b), starting respectively from points A and B. The attractor is colored in red.

We have shown examples of systems with having attractors of dimension 0 (a point) and 1 (limiting cycle). However, in many situations the shape of the attractor is very complicated and its dimension is not even integer. They are called *fractals*, a definition first given by Mandelbrot for objects that look similarly no matter the scale at which they are observed. Some author refer to fractal attractor as *strange attractors* [Ott, 2002], while following [Eckmann and Ruelle, 1985, Ruelle, 1989] we call strange attractors only those that display a chaotic behaviour, hence paying more attention on the strangeness of the trajectory, rather than on that of its shape.

An example of 3d strange attractor is given by [Rössler, 1976] system:

$$\begin{aligned} \dot{x}^{(1)} &= -x^{(2)} - x^{(3)} \,, \\ \dot{x}^{(2)} &= x^{(1)} + a x^{(2)} \,, \\ \dot{x}^{(3)} &= b + x^{(3)}(x^{(1)} - c) \,, \end{aligned} \tag{2.24}$$

where a, b, c are positive constant. By choosing the constants $a = 0.37$, $b = 0.2$ and $c = 5.7$ the attractor, shown in Figure 2.4, is fractal. However, we remark that this model does not have a direct physical interpretation unlike [Lorenz, 1963] and it was introduced just to study chaos with one circulation roll.

Another interesting example in 2d is given by the Hénon map [Hénon, 1976]:

$$\begin{aligned} x^{(1)}_{t+1} &= 1 + x^{(2)}_t - a(x^{(1)}_t)^2 \,, \\ x^{(2)}_{t+1} &= b x^{(1)}_t \,, \end{aligned} \tag{2.25}$$

where a, b are constants for which the attractor is both fractal (Figure 2.5) and sensitive dependent on initial conditions for the choice $a = 1.4$, $b = 0.3$. In Figure 2.5 we can appreciate how Hénon attractor is composed by many different lines that can look singular from a large scale, but reveal to be divided in many other lines as the scale becomes arbitrarily small [Ott, 2002].

A similar behaviour occurs in the force damped pendolum Equation 2.12, by choosing $f = 1/2\pi$, $v = 0.22$ and $T = 2.7$, as shown in Figure 2.6.

However, there may be for instance systems which have a fractal attractor which is thus strange regarding its shape, but they do not manifest sensitive dependence on initial

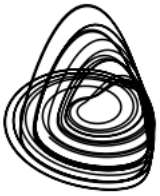


Figure 2.4: Rössler 76 attractor.

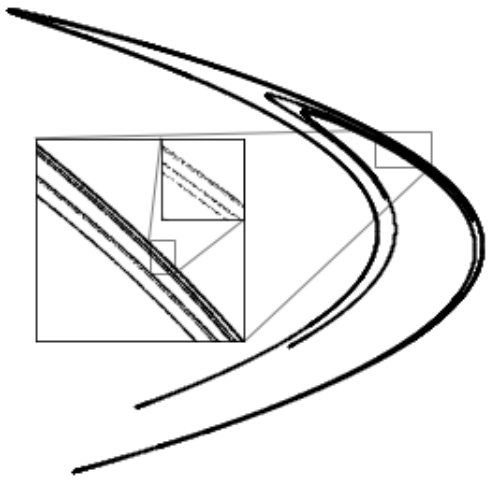


Figure 2.5: Hénon attractor. The fractal nature of this object (which however does not exhibit a perfect self-similarity) is evident as we zoom in different portion of the dynamics, revealing more details the more we zoom.

conditions. This is the case of Feigenbaum attractor, resulting from the logistic map Equation 2.17, which for $0 \leq r \leq 4$ maps the interval $[0, 1]$ into itself. As r varies, the behaviour of logistic map undergoes a variety of regimes, converging to a fixed point for $r < 3$, which as r increases becomes unstable until the system manifest a sequence of period doubling, or bifurcations (on which we will come later), eventually leading to chaos. Then number of unstable periodic orbits becomes infinite for $r \approx 3.57$, for which value the attractor, shown in Figure 2.7, is fractal but not chaotic.
We conclude by noting that mathematically is very difficult to prove that some attractors are strange. In case of the [Lorenz, 1963], it was proven only very recently by [Tucker, 2002].

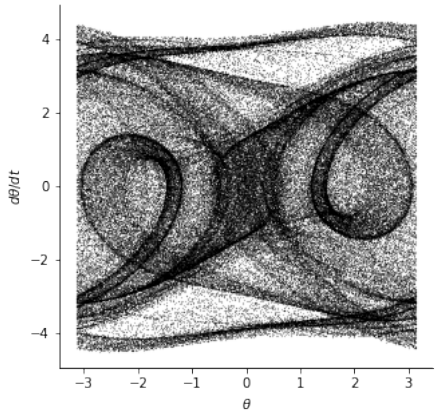


Figure 2.6: Generated stroboscopic plot of the forced damped pendolum in phase space taken at snapshot time $t = 0, 2\pi, 4\pi, 6\pi, \ldots$ for $v = 0.22$ and $T = 2.7$.

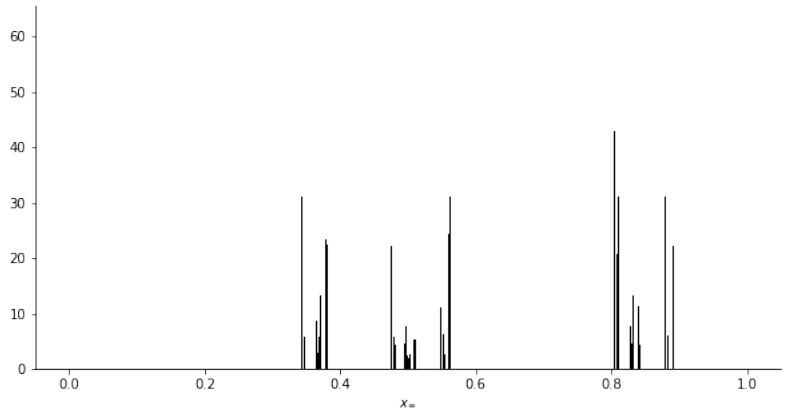


Figure 2.7: Fractal attractor (Feigenbaum attractor) for the logistic map when $r \approx 3.57$. The plot is generated by computing a long trajectory 10000 with a generic initial condition and neglecting the first 1000 values.

2.4 Examples of Dynamical Systems

In this section, we will go deeper into the details of some important and pragmatic dynamical systems, like the 1d logistic map in Equation 2.17 and the model introduced by [Lorenz, 1963], Equation 2.43.

2.4.1 The Logistic Map

Let us come back to the logistic map Equation 2.17. A continuous version was first proposed by Verhulst in 1838, which inspired by the work of Malthus on population dynamics (*An essay on the Principle of Population*, 1798). Malthus first proposed a simplistic exponential growth $x_t = x_0 e^{rt}$, which is unreasonable for $r > 0$ since it would lead to an unbounded number of individuals for a finite-resource environment.

Verhulst generalization instead came by the introduction of a *carrying capacity* term, accounting for the fact that a limited environment can supporting only a (finite) maximum number of individuals: $dx/dt = rx(1 - x)$. This system possesses two fixed points: $x^* = 0$, which is unstable for $r > 0$ and $x^* = 1$, which is the stable limit that the system reaches asymptotically. However, this continuous equation does not sustain chaos, as already seen

for one dimensional continuous systems.
On the other hand, the discretized version Equation 2.17, which is still a reasonable model, may generate chaotic trajectories, being a one-dimensional non-invertible map.
Its erratic behaviour was first understood by [Ulam and von Neumann, 1947], who proposed to use the logistic map for $r = 4$, called Ulam point, to generate random numbers, even if a complete understanding came later thank to the work of [Stein and Ulam, 1964] and [Ricker, 1954], later summarized by [May, 1976].
We will now present qualitatively the behaviour of the logistic map varying r, understandable in the framework of linear stability theory, not developed in this thesis. A detailed analysis can be found in any dynamical systems theory book, such as [Ott, 2002] or [Cencini et al., 2009].
For $r < 1$, there exists only one fixed point $x^* = 0$, which is stable, while for $1 < r < 3$, $x_1^* = 0$ becomes unstable and another stable fixed point appears at $x_2^* = 1 - \frac{1}{r}$. Therefore, in this regime all the orbits starting in a generic x_0 will asymptotically converge to x_2^*, meaning that the system support a finite positive number of individuals.
As $r > r_1 = 3$ further increases, the system undergoes the so-called period *bifurcations*. For $r = 3.2$, also x_2^* becomes unstable and two other stable fixed points appear: $x_{3,4}^* = \frac{r+1 \pm \sqrt{(r+1)(r-3)}}{2r}$, which is understood by studying the second iteration $f_r^{(2)}$ of $f_r(x) = rx(1-x)$. As a consequence, all the trajectories are attracted by a period-2 orbit bouncing back and forth between x_3^* and x_4^*, which is the discrete version of a limiting-cycle. This orbit is stable for $r_1 < r < r_2 \approx 3.448$.
Once $r > r_2$ a period-4 orbit becomes attractive and stable. As r increases, a sequence of r_k values appear, such that for $r_k < r < r_{k+1}$ the dynamics converge to a stable periodic orbit of period 2^k [May, 1976]. Interestingly, this sequence has a limit $\lim_{k\to\infty} r_k \approx 3.57 = r_\infty$, where no stable periodic orbits exist. At this point, the limiting dynamics becomes fractal (see Figure 2.7) and the invariant measures becomes singular with respect to Lebesgue measure. However, there is no sensitive dependence on initial conditions. Instead, there are an infinite number of unstable periodic orbits, eventually giving raise to chaos at the Ulam point $r = 4$, corroborating Ulam's intuition that at this point the system can be used to generate pseudo-random numbers [Ulam and von Neumann, 1947].
The behaviour of the logistic map for $r > r_\infty$ is a bit more complicate than it may look, as it is evident from the *bifurcation* diagram in Figure 2.8, made by generating a long trajectory discarding the first 10000 iterations, and plot the successive 200 points, for 1000 r values in $[2.5, 4]$. Clearly, this allows only periodic orbits up to a period of $P = 200$ to be identified. Remarkably, for $r > r_\infty$, there are several windows of regular period separated by chaotic regions. For instance, it is clearly visible a period-3 stable orbit at $r \approx 3.828$, which eventually bifurcates into period-$6, 12, \ldots$ orbits. Thus behaviour can be understood by studying the graphs of $f_r^{(3)}$, $f_r^{(6)}$ and so on.

How the logistic map give birth to sensitive dependence on initial condition at the Ulam point can be studied by the *topological conjugation* of the logistic map at $r = 4$ with the tent map, which is the map (Figure 2.9) mapping $[0, 1]$ in $[0, 1]$ defined as

$$y_{t+1} = \begin{cases} 2y_t & 0 \le y_t \le 1/2 \\ 2(1 - y_t) & 1/2 \le y_t \le 1 \, , \end{cases} \tag{2.26}$$

which can be written also as $y_{t+1} = m'(y_t) = 1 - 2|y_t - 1/2|$.

Definition 2. *(**Topological conjugacy**) Let us consider for the sake of simplicity a 1d map $x_{t+1} = m(x_t)$, being the evolved dynamics given by*

$$x_t = f^t \, x_0 \, . \tag{2.27}$$

Let consider the invertible change of variables $x \to y = h(x)$, where dh/dx does not change sign. The evolution of y can be written as $y_t = \tilde{f}^t y_0$, where the dynamics of y is given by

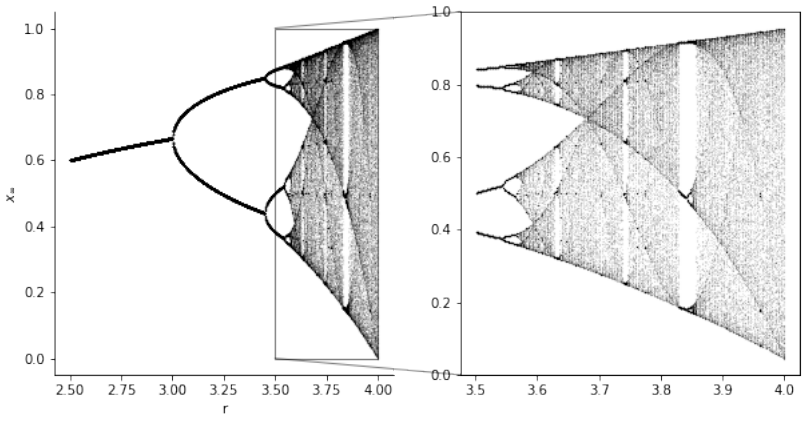


Figure 2.8: Bifurcation diagram for logistic map, obtained by generating orbits of $P = 200$ iterations after a transient of $D = 10000$ is discarded, for 1000 points in the interval $[2.5, 4]$.

$y_{t+1} = m'(y_t)$, *where the function* m *and* m' *are related by* $h(x)$ *as*

$$m'(y) = h(g(h^{(-1)}(y))) \ , \tag{2.28}$$

where $h^{(-1)}$ *is the inverse of* h. *In such case, we say the the maps* x *and* y *are topologically conjugated and there is a one-to-one correspondence between the properties of the two systems [Eckmann and Ruelle, 1985].*

First, it easy to build the change of variable $h(x)$ from the logistic map to the tent map, by setting

$$x = h^{(-1)}(y) = \sin^2(\pi y/2) = [1 - \cos(\pi y)]/2 \ . \tag{2.29}$$

By substituting it in Equation 2.17 with $r = 4$, we obtain $\sin^2(\pi y_{t+1}/2) = \sin^2(\pi y_t)$, which yields

$$\pi y_{t+1}/2 = \pm \pi y_t + k\pi \ , \tag{2.30}$$

where k is an integer. By setting $y_t \in [0, 1]$, we retrieve the tent map.Therefore, by topological conjugation, if the tent map exhibits sensitive dependence on initial condition, so must the logistic map do.
To understand how it works for the tent map, let us consider as a warm up the Bernoulli shift map, which is topologically conjugated with the tent map through a complicated non-differentiable function [Beck and Schlögl, 1997]:

$$y_{t+1} = \begin{cases} 2y_t & 0 \leq y_t \leq 1/2 \\ 2y_t - 1 & 1/2 \leq y_t \leq 1 \ , \end{cases} \tag{2.31}$$

or also $y_{t+1} = 2y_t \mod 1$, see Figure 2.9.

Let us indicate points in binary notation, i.e. the dynamics starts at

$$y_0 = \sum_{i=1}^{\infty} \frac{a_i}{2^i} = 0.a_1 \ a_2 \ a_3 \dots \ , \tag{2.32}$$

where $a_i = 0, 1$. Bernoulli shift map translate the entire sequence one digit to the left at each iteration, removing the most significant, like

$$y_0 = 0.a_1 \ a_2 \ a_3 \dots \quad \rightarrow \quad y_1 = 0.a_2 \ a_3 \ a_4 \dots \quad \rightarrow \quad y_2 = 0.a_3 \ a_4 \ a_5 \dots \ , \tag{2.33}$$

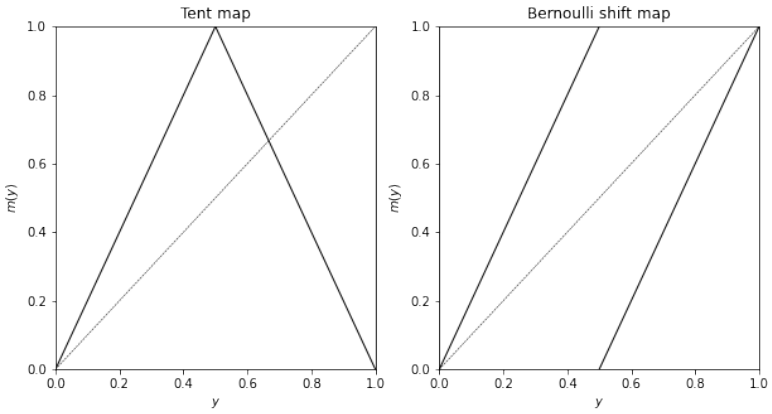


Figure 2.9: Tent (left) and Bernoulli shift (right) maps. The diagonal is denoted with the lighter dotted line. Since the derivative of these maps are always larger that 1, due to linear stability theory there cannot be fixed stable points, see [Cencini et al., 2009].

such that y_t is simply y_0 with the first $t-1$ digits removed. It is clear that small perturbations are amplified by a factor 2 at each iteration, so that any difference in the most insignificant digits will be amplified as

$$\delta y_t = \delta y_0 \, 2^t = \delta y_0 \, e^{t \, \ln 2} \, , \tag{2.34}$$

being the Lyapunov exponent $\ln 2$ (see section 2.8).
In case of the Tent map, it acts as a shift when $y_t < 1/2$ and as a bit negation when $y_t > 1/2$. We indicate the negation operator as $\neg(a_t)$, i.e. $\neg(0) = 1$ and $\neg(1) = 0$. Since the sequence y_t is invariant if $y_t < 1/2$, it is negated if $y_t > 1/2$ and $\neg^2$ is the identity operator, the action of the Tent map can be written as

$$y_t = m^{(n)}(y_0) = 0.\neg^{a_1+a_2+\cdots+a_n}(a_{n+1}) \, \neg^{a_1+a_2+\cdots+a_n}(a_{n+2}) \ldots \, , \tag{2.35}$$

being therefore Equation 2.34 still valid and so the same holds for the logistic map.

2.4.2 The Lorenz 63 Model

[Lorenz, 1963] introduced his famous model by a simplification of the equations involved in Rayleigh-Bénard convection, a problem of fluid mechanics that was first posed by Bénard (1900) and later developed by Rayleigh (1916).
Consider a fluid in a gravitational field g directed along the z-axis and contained between two infinite horizontal plates perpendicular to the z-axis and separated by a distance H. The density of the fluid is a function of the temperature and thus if the temperature in the upper plate T_U equals the temperature in the bottom plate T_B, the density is constant. If $T_U > T_B$ the fluid stratifies and temperatures varies linearly with the altitude from being cold at the bottom and hot at the top [Monin and Yaglom, 1975]:

$$T(z) = T_B + z\frac{T_U - T_B}{H} \, . \tag{2.36}$$

This is the so-called *conduction* state.
On the other hand, if $T_U < T_B$ the fluid is unstable due to buoyancy, since light hot fluid is pushed to the top, while heavy cold fluid sinks due to the gravity field. The process is contrasted by viscous forces and if the temperature difference $\Delta T = T_B - T_U$ exceeds a certain threshold the conduction state is broken and the fluid enters in a *convective* state, in which the fluid is organized in counter-rotating vortices (or rolls) that bring up the light hot fluid and push down the heavy cold fluid. This state, called Rayleigh-Bénard convection, is

controlled by the *Rayleigh number*

$$Ra = \frac{\rho_0 g \alpha H^3 |T_U - T_B|}{k\nu} , \tag{2.37}$$

where k is the coefficient of thermal diffusivity and ν is the viscosity. The average density is ρ_0, while α is the thermal dilation coefficient, relating the density with the temperature as $\rho(T) = \rho(T_0)[1 - \alpha(T - T_0)]$, which is valid when the temperature differences are not too large.
If the Rayleigh number exceeds the critical value Ra_c separating the stable conduction state with the convection state for too much, the fluid becomes unstable and undergoes an erratic and irregular convective motion: we are in presence of chaos.
If the temperature difference $T_B - T_U$ is not too large, the equation of motions under the Boussinesq approximation [Monin and Yaglom, 1975] are:

$$\begin{aligned} \partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} &= -\frac{\nabla p}{\rho_0} + \nu \nabla^2 \mathbf{u} + g\alpha\Theta , \\ \partial_t \Theta + (\mathbf{u} \cdot \nabla)\Theta &= k\nabla^2 \Theta + \frac{T_U - T_B}{H} \, u_z , \end{aligned} \tag{2.38}$$

where $\mathbf{u}$ is the velocity field supplemented with the incompressibility condition $\nabla \cdot \mathbf{u} = 0$ and ∇^2 denotes the Laplacian. The first is the Navier-Stokes equation where p is the pressure field. The second is the advection diffusion equation for the deviation Θ of the temperature field $T(\mathbf{r}, t)$ from the conduction state in Equation 2.36, i.e. $\Theta(\mathbf{r}, t) = T(\mathbf{r}, t) - T_B - z(T_U - T_B)/H$.
A first simplification consist to study the 2-dimensional problem in the plane $x - z$, by mean the so-called *stream* function $\psi(x, z, t)$, defined by

$$u_x = \frac{\psi}{\partial z} , \qquad u_z = -\frac{\psi}{\partial x} , \tag{2.39}$$

in order to ensure the incompressibility assumption. Already Rayleigh found solutions of the form

$$\begin{aligned} \psi &= \psi_0 \sin\left(\frac{\pi a x}{H}\right) \sin\left(\frac{\pi z}{H}\right) , \\ \Theta &= \Theta_0 \cos\left(\frac{\pi a x}{H}\right) \sin\left(\frac{\pi z}{H}\right) , \end{aligned} \tag{2.40}$$

where ψ_0 and Θ_0 are constants and a is the horizontal wave-length of the rolls. However, this solutions become unstable if the Rayleigh number exceeds the critical value

$$Ra_c = \frac{\pi^4 (1 + a^2)^3}{a^2} , \tag{2.41}$$

making them hard to be studied analytically.
A possible overcome is to consider a Fourier expansion in which the time dependency is put on the coefficients, i.e.

$$\begin{aligned} \psi(x, z, t) &= \sum_{mn, n=1}^{\infty} \psi_{mn}(t) \sin\left(\frac{m\pi a x}{H}\right) \sin\left(\frac{n\pi z}{H}\right) , \\ \Theta(x, z, t) &= \sum_{mn, n=1}^{\infty} \Theta_{mn}(t) \cos\left(\frac{m\pi a x}{H}\right) \sin\left(\frac{n\pi z}{H}\right) . \end{aligned} \tag{2.42}$$

However, this substitution into the original PDE give raise to an infinite number of ODEs. The assumption made by [Lorenz, 1963] was to consider the simplest possible

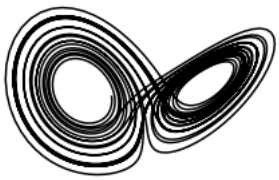


Figure 2.10: Lorenz 63 attractor.

truncation with retains only three coefficients, namely the amplitude of the convection motion $\psi_{11}(t) = x_t^{(1)}$, the temperature difference between the ascending and descending fluid currents $\Theta_{11}(t) = x_t^{(2)}$ and the deviation from the linear temperature profile $\Theta_{02}(t) = x_t^{(3)}$, ending up with:

$$\begin{aligned} \dot{x}^{(1)} &= \rho(x^{(2)} - x^{(1)}) \ , \\ \dot{x}^{(2)} &= x^{(1)}(\sigma - x^{(3)}) - x^{(2)} \ , \\ \dot{x}^{(3)} &= x^{(1)}x^{(2)} - \beta x^{(3)} \ , \end{aligned} \tag{2.43}$$

where $\sigma = \frac{Ra}{Ra_c}$ is the normalized Rayleigh number, $\rho = \nu/k$ is the Prandtl number and $\beta = 4(1 + a^2)^{-1}$ a generic geometric coefficient related to the vortices wave length.The physical time unit is $\pi^2 H^{-2}(1 + a^2)k$. For the common choice of the constants $\rho = 10$, $\sigma = 28$ and $\beta = 8/3$ the system exhibits both a chaotic behaviour and a fractal attractor with dimension with the typical butterfly shape in Figure 2.10.

The Jacobian of the system or stability matrix of Equation 2.43 is

$$\mathbb{L} = \begin{pmatrix} -\rho & \rho & 0 \\ \sigma - x^{(3)} & -1 & -x^{(1)} \\ x^{(2)} & x^{(1)} & -\beta \end{pmatrix} \ . \tag{2.44}$$

We observe that

$$\nabla \cdot \mathbf{g} = \sum_{i=1}^{3} \frac{\partial}{\partial x^{(i)}} \dot{x}^{(i)} = Tr(\mathbb{L}) = -(\rho + \beta + 1) < 0 \ , \tag{2.45}$$

meaning that the phase space volume exponentially converges to a subset having zero measure. The system is thus dissipative. Moreover, trajectories do not explore the whole space, but they remain confined in a bounded region. To show that, let us perform the change of variable $X^{(1)} = x^{(1)}$, $X^{(2)} = x^{(2)}$ and $X^{(3)} = x^{(3)} - \sigma - \rho$. Equation 2.43 can be rewritten as

$$\dot{X}^{(i)} = \sum_{jk} a_{ijk} \ X^{(j)} \ X^{(k)} + \sum_{j} b_{ij} \ X^{(j)} + c_i \ , \tag{2.46}$$

where a_{ijk}, b_{ij} and c_i are constants. Furthermore, we notice that $\sum_{ijk} a_{ijk} \ X^{(i)} \ X^{(j)} \ X^{(k)} = 0$ and $\sum_{ij} X^{(i)} \ X^{(j)} > 0$. If we define the energy function as $Q = \frac{1}{2} \sum_{ij} \ (X^{(i)})^2$ and denote with e_i the roots of the linear equation $\sum_j (b_{ij} + b_{ji}) e_j = c_i$, from the equations of motion we have

$$\dot{Q} = \sum_{ij} b_{ij} e_i e_j - \sum_{ij} b_{ij} (X^{(i)} - e_i)(X^{(j)} - e_j) < 0 \ , \tag{2.47}$$

and therefore trajectories are bounded.

Let us come back to the study of stability of the system. The fixed points are simply the solutions of $\mathbf{g}(\mathbf{x}^*) = 0$, i.e.

$$\mathbf{x}_0^* = (0,0,0)\ , \qquad \mathbf{x}_\pm^* = (\pm\sqrt{\beta(\sigma-1)}, \pm\sqrt{\beta(\sigma-1)}, \sigma-1)\ , \tag{2.48}$$

where the first represents the conduction state, while $\mathbf{x}_\pm^*$, which are real for $\sigma \geq 1$, correspond to the counter-clockwise and clockwise of the convective vortices. The solution of the secular equation $\det(\mathbb{L}(\mathbf{x}^*) - \lambda\mathbf{I})$, yields [Cencini et al., 2009]:

- For $0 < \sigma < 1$, $\mathbf{x}_0^*$ is the only fixed point, which is stable being all the eigenvalues negative.
- For $\sigma > 1$ one of the eigenvalues of $\mathbf{x}_0^*$ becomes positive, while $\mathbf{x}_\pm^*$ have one real negative and two complex conjugated eigenvalues corresponding to convection. If σ is greater than a critical value

$$\sigma_c = \frac{\rho(\rho+\beta+3)}{(\rho-\beta-1)}\ , \tag{2.49}$$

  steady convection is unstable. Since to have a physical meaning ρ, σ and β must be positive number, then Equation 2.49 yields a relevant condition only when $\rho > (\beta+1)$.

Linear stability regime fails when $\rho > (\beta+1)$ and $\sigma > \sigma_c$, therefore a numerical analysis of Equation 2.43, as did by Lorenz himself, is necessary in this domain. Following him, in this thesis we use the combination of parameters $\rho = 10$, $\beta = 8/3$ and $\sigma = 28 > \sigma_c \approx 24.74$. In this regime, trajectories converge to the strange attractor seen in Figure 2.10, where orbits with similar random duration circulate around the two unstable steady convection states $\mathbf{x}_\pm^* = (\pm 6\sqrt{(2)}, \pm 6\sqrt{(2)}, 27)$. Physically, it means that convection switches irregularly from clockwise to counter-clockwise rotation.

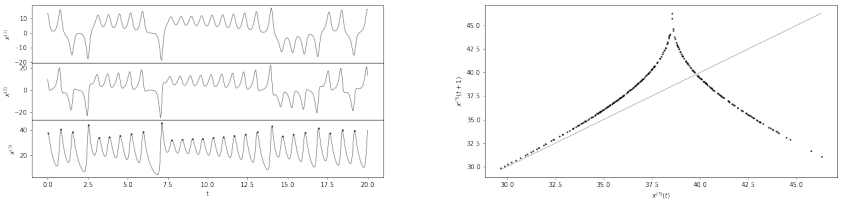
As already noted by [Lorenz, 1963], the chaotic behaviour of Equation 2.43 cane be understood by deriving one dimensional chaotic map, called *return map*. By comparing the evolution of the time sequences $x^{(1)}$ and $x^{(2)}$ versus $x^{(3)}$, we can appreciate how the random switches between clockwise and counter-clockwise rotation seem to appear when $x^{(3)}$ reaches local maxima. Lorenz thus supposed that the chaotic dynamics of the system may be encoded on the sequence of local maxima of $x^{(3)}$. Therefore, he plotted $x_{t+1}^{(3)}$ against $x_t^{(3)}$ and the pattern shown in Figure 2.11 is very interesting. We can notice that the points are not randomly distributed, but they are organized in a smooth one-dimensional curve. Moreover, such a curve is not invertible, so that chaos is possible, and its derivative is always larger than 1, which in linear stability theory means that there cannot be stable fixed points neither for the map itself, nor for its k-th iterations.

2.4.3 The Lorenz 96 Model

We briefly describe the system introduced by [Lorenz, 1995], a simple toy model describing three typical processes happening in the Atmosphere for a generic atmospheric quantity: transfer, advection and external forcing. Consider N sites dislocated on a ring with periodic boundary conditions. The model is described by the system of differential equations in terms of the generic atmospheric quantities $\{x^{(i)}\}_{i=1}^N$

$$\dot{x}^{(i)} = x^{(i-1)}(x^{(i+1)} - x^{(i-1)}) - x^{(i)} + F\ , \tag{2.50}$$

where F is the external force.
It is immediate to see that $x_n = F$ is a constant solution of the system. Moreover, it is



(a) (b)

Figure 2.11: a) Dynamics of Lorenz 63 system up to time 20, generated with $\delta t = 0.002$ from a random point inside the attractor. Maxima of $x^{(3)}$ are highlighted. b) Return map of maxima of $x^{(3)}$, i.e. $x^{(3)}_{t+1}$ against $x^{(3)}_t$. The diagonal is denoted with the lighter dotted line.

known that solutions are stable for $\frac{1}{2} < F < \frac{8}{9}$, while as F increase periodic solutions begin to appear, as for $F = 2$. By increasing again F, the system enters in a chaotic regime, where regular waves break and the trajectories are irregular, see Figure 2.12 where we plot the Hovmöller plot [Persson, 2017] for different values of the force F. From the plot, we can notice that if F is small a regular pattern appear, and the system supports waves propagation with spatial period around 9. However, for large F these waves are broken and the system undergoes a chaotic regime.

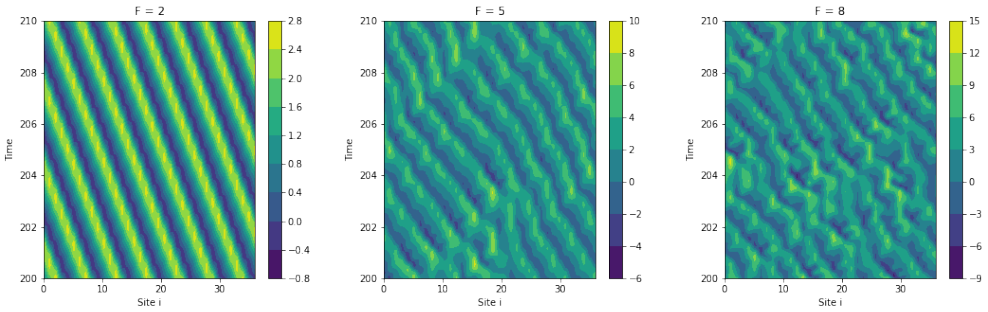


Figure 2.12: Hovmöller plot for Lorenz 96 system for different values of the force F. The plots are generating by intergrating a long trajectory with time step $\delta t = 0.002$ and discarding the first 200 time units of the dynamics.

For detailed analysis on wave propagation see [van Kekem and Sterk, 2018a] and [van Kekem and Sterk, 2018b]. The reader is further referred to [van Kekem and Sterk, 2019] for a study on the symmetries of the model and to [Galias and Tucker, 2008] for an analysis on the short periodic orbits. For some generalization of the model see [Vissio and Lucarini, 2020] and [Kerin and Engler, 2022].

2.5 Probabilistic Approach

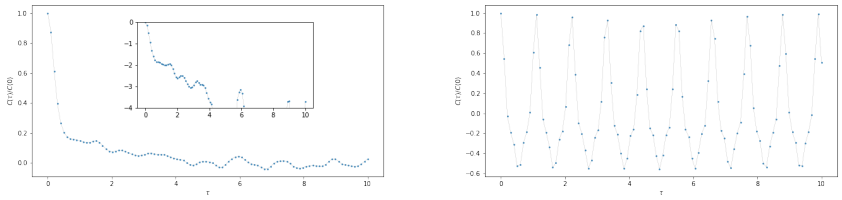
Dynamical systems are fully determined by the choice of initial condition. However, sensitive dependence to initial conditions makes chaotic systems very irregular and therefore a statistical description is needed even if the number of degree of freedom is small. The reason is that chaotic systems loose very quickly *memory* on the initial condition due to the irregularities of a chaotic dynamics. Let us clarify this point by computing the time correlation function, assuming stationarity, defined as

$$C(\tau) = \langle x_{t+\tau}\, x_t \rangle - \langle x_t \rangle^2 \ , \tag{2.51}$$

where $\langle x \rangle = \lim_{T\to\infty} \int_0^T x_t \, dt$, for the Lorenz 63 system Equation 2.43, see Figure 2.13. The correlation function measures how distant points are "related" to each other. In case periodic or quasi-periodic motion, the correlation function cannot relax to zero, because motion repeats itself. However, in case of irregular motion, such as chaotic or stochastic systems, $C(\tau)$ approaches zero for large τ. The characteristic decay time is given by

$$\tau_c := \int_0^\infty d\tau \; C(\tau)/C(0) \; , \tag{2.52}$$

when $0 < A = \int_0^\infty d\tau \; C(\tau) < \infty$, measuring the scale at which the system looses memory from the past. Moreover, it is correlated to the energy density content of the system by Wiener-Kinchin theorem [Wiener, 1930,Khintchine, 1934], being the power spectrum density the Fourier transform of the correlation function.



(a) (b)

Figure 2.13: Rescaled correlation function for the $x^{(1)}$ variable of a) the chaotic Lorenz 63 system with standard parameters $\rho = 10$, $\sigma = 28$ and $\beta = 8/3$. b) Same for non-chaotic Lorenz 63 with $\rho = 10$, $\sigma = 100$ and $\beta = 8/3$. We notice how correlation decades exponentially fast in the chaotic system, while when the system is not chaotic it does not decay.

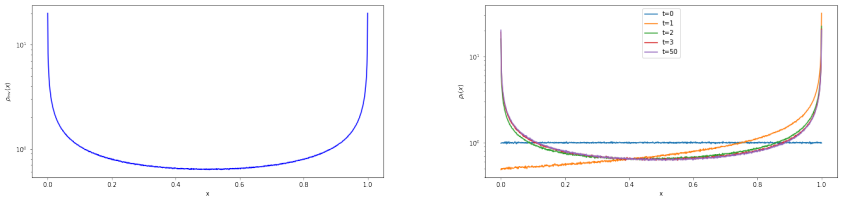
We see that in case of chaos, the past is rapidly forgotten and trajectories appear to be random. Therefore, the idea of trajectory must be abandoned to embrace a *statistical mechanics* description of chaotic systems.
Let us come back to the Logistic map Equation 2.17. Operationally, the probability density is built point-wise considering trajectories in the phase space x_t, starting at x_0

$$\rho(x; x_0) = \lim_{T\to\infty} \frac{1}{T} \int_0^T \delta(x - x_t) \, dt \; , \tag{2.53}$$

which may in principle depend also on the initial condition x_0. In computer simulations, this density is computed by dividing the interval $(0, 1)$ in N bins of equal size $1/N$ and measuring the number of times a long trajectory with duration T passes though the bins, dividing by T and then taking the limit as $T \to \infty$.
Consider the problem to find the conditional probability given the probability of initial condition $\rho_0(x)$, i.e. we are interested in the *time evolution* of the probability density as $\rho_0(x), \rho_1(x), \ldots, \rho_t(x), \ldots$. The natural questions that come up are: 1) does $\rho_t(x)$ have a limiting distribution $\rho_\infty(x) = \lim_{t\to\infty} \rho_t(x)$ and, if so, how fast is it approached? 2) How does the limiting distribution depend on the initial condition?
It is clear that the limiting distribution $\rho_t(x)$, if it exists, should be invariant under the time evolution, i.e. $\rho_\infty(x) = \rho_{inv}(x)$, see Figure 2.14. As an example for continuous systems, in Figure 2.15 we report the projected invariant measures of [Lorenz, 1963].

The problem of assessing if the invariant pdf depends on initial condition is less than trivial. For $r = 4$ the logistic map has infinite number of unstable periodic orbits. Now, let supposed to choose as initial condition x_0 a point belonging to a periodic orbit $(x_0, x_1, \ldots, x_{n-1})$ of period n. Thus, Equation 2.53 becomes

$$\rho(x; x_0) = \frac{\delta(x - x_0) + \delta(x - x_1) + \cdots + \delta(x - x_{n-1})}{n} \; , \tag{2.54}$$



(a) (b)

Figure 2.14: a) Invariant pdf for the logistic map at Ulam point $r = 4$, made by evolving the dynamics for 10^7 step with a generic initial point in $[0, 1]$. b) The evolution of the pdf, created by taking 10^6 points uniformly distributed in $[0, 1]$ at $t = 0$. We notice how the convergence to the invariant pdf is very fast and it is already almost indistinguishable at $t = 3$.

which is trivially invariant, since points just exchange each other as time goes on. This reasoning can be applied to any other unstable periodic orbit and in addition we have that any normalized linear combination of such invariant densities is still an invariant density. However, in physical situation, as in Figure 2.14, the system "chooses" naturally an invariant density, known as *natural* or *physical* measure, which cannot be expressed as linear combination of other invariant densities. Moreover, in many cases we cannot even speak about probability densities, because measures can be singular with respect to Lesbegue measure, as it happens in case of the Feigenbaum attractor (i.e. the attractor of logistic map for $r = 3.57\ldots$.) Figure 2.7, and therefore cannot be written as $d\mu(x) = \rho(x)\, dx$. For that reason, in the following we use the term invariant measure μ_{inv} instead of density. Given a map or a continuous system characterized by the evolution operator f^t, invariant measure is defined by

$$\mu_{inv}(f^{-t}(B)) = \mu_{inv}(B) \quad \forall t \geq 0\,, \tag{2.55}$$

where B is any measurable set. An invariant measure μ is said to be *ergodic* if it cannot be linearly decomposed into invariant measures, i.e. we cannot write

$$\mu = p\mu_1 + (1-p)\mu_2\,, \qquad 0 < p < 1\,, \tag{2.56}$$

where μ_1 and μ_2 are other invariant measures.

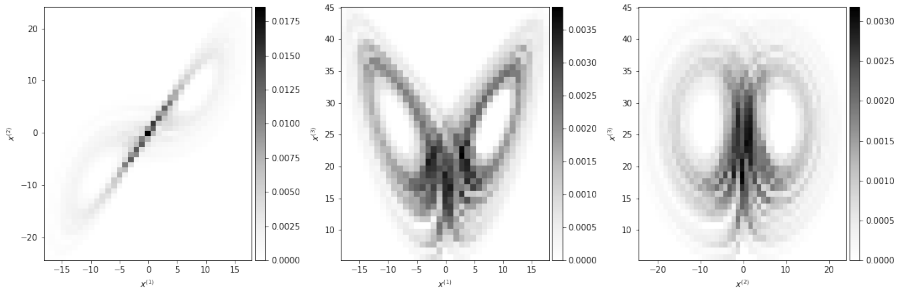


Figure 2.15: Invariant measures of the three projections along coordinates axis for Lorenz 63, generated from a dataset of 100000 samples with $\delta t = 0.002$ by counting the frequency that the trajectory visit squares of side 1.

2.5.1 Time evolution of probability density

The time evolution of the probability density of a map $\mathbf{x}_{t+1} = \mathbf{m}(\mathbf{x}_t)$ is described by the linear Perron-Frobenius transfer operator [Ruelle, 2004, Baladi, 2000]:

$$\rho_{t+1}(\mathbf{x}) = \mathcal{L}_{PF}\rho_t(\mathbf{x}) = \int d\mathbf{y}\ \rho_t(\mathbf{y})\ \delta(\mathbf{x} - \mathbf{m}(\mathbf{y})) = \sum_k \frac{\rho_t(\mathbf{y}_k)}{|\det \mathbb{L}(\mathbf{y}_k)|}\,, \tag{2.57}$$

where $\mathbf{m}(\mathbf{y}_k) = \mathbf{x}$ and $\mathbb{L}_{ij} = \partial m_i / \partial x_j$.
For a continuous time dynamical system $\dot{\mathbf{x}} = \mathbf{g}(\mathbf{x})$, we have

$$\rho_t(\mathbf{x}) = \mathcal{L}_{PF}^t \, \rho_0(\mathbf{x}) \; , \tag{2.58}$$

where the family $\{\mathcal{L}_{PF}^t\}_{t \geq 0}$ satisfies the semi-group properties, i.e. $\mathcal{L}_{PF}^{t+s} = \mathcal{L}_{PF}^t + \mathcal{L}_{PF}^s$ and $\mathcal{L}_{PF}^0 = \mathbb{I}$.
Perron-Frobenius operator is the adjoint of the evolution operator $f^t = \left(\mathcal{L}_{PF}^t\right)^\dagger$ or flow, such that $\left\langle f^t \, O \right\rangle_{\rho_0} = \langle O \rangle_{\rho_t}$, for any observable O.
Assuming strong continuity and boundedness of the semi-group family $\{\mathcal{L}_{PF}^t\}_{t \geq 0}$, we can introduce the generator $\mathcal{L}_L$ of the unperturbed Perron-Frobenius operator $\mathcal{L}_{PF} = e^{t \, \mathcal{L}_L}$, which allows us to rewrite Equation 2.4 as

$$\partial_t \rho = \mathcal{L}_L \rho = -\nabla(\mathbf{g}\rho) \; . \tag{2.59}$$

The connection with dynamical systems with stochastic processes is established by the introduction of a noise term in Equation 2.1, see [Feller, 1968], getting the Langevin equation:

$$\dot{\mathbf{x}} = \mathbf{g}(\mathbf{x}) + \boldsymbol{\eta}_t \; , \tag{2.60}$$

where $\boldsymbol{\eta}_t$ is white Gaussian noise, i.e. $\langle \boldsymbol{\eta}_t \rangle = \mathbf{0}$ and $\langle \eta_t \, \eta_{t'} \rangle = Q\delta(t - t')$, where the covariant matrix Q is positive-definite [Chandrasekhar, 1943]. In this case the evolution equation of the probability density is substituted by Fokker-Planck equation [Fokker, 1914, Planck, 1917](in Ito prescription):

$$\frac{\partial \rho}{\partial t} = \mathcal{L}_L \, \rho + \frac{1}{2} \sum_{ij} Q_{ij} \frac{\partial^2 \rho}{\partial x^{(i)} \partial x^{(j)}} \; , \tag{2.61}$$

where Q is independent of $\mathbf{x}$. The Liouville operator is thus substituted by Fokker-Planck operator [Gardiner, 2004]:

$$\mathcal{L}_{FP} = \mathcal{L}_L + \frac{1}{2} \sum_{ij} Q_{ij} \frac{\partial^2}{\partial x^{(i)} \partial x^{(j)}} \; . \tag{2.62}$$

As we have seen, there may be invariant measures in the dynamics, but not all are physically relevant. In common situation, it is the system itself that "choose" the most suitable physical measure. Indeed, in real world scenarios, the dynamics given by Equation 2.1 is not reasonable, since we have to deal with noise and instead the dynamics may be described by Equation 2.60. This approach remains valid also in computer simulation, since round off errors due to computer operations are source of noise. Therefore, assuming that it exists, we say that the physical measure chosen by the system is the limit

$$\rho = \lim_{\boldsymbol{\eta} \to \mathbf{0}} \rho_{\boldsymbol{\eta}} \; , \tag{2.63}$$

where $\rho_{\boldsymbol{\eta}}$ is the measure of the stochastic dynamics, and it is typically the one computed by running a long trajectory and counting the number of points lying in each boxes by which space is divided.

It is clear that the invariant density satisfies

$$\mathcal{L}_{PF} \rho_{inv}(\mathbf{x}) = \rho_{inv}(\mathbf{x}) \; . \tag{2.64}$$

More generally, Perron-Frobenius operator admits an infinite number of eigen-functions $\Psi^k(\mathbf{x})$, i.e.

$$\mathcal{L}_{PF} \Psi^k(\mathbf{x}) = \alpha_k \Psi^k(\mathbf{x}) \; , \tag{2.65}$$

where the eigenvalues α_k are in general complex.
A generalization of Perron-Frobenius theorem asserts the existence of a real eigenvalue equal to unity $\alpha_1 = 1$, while the other eigenvalues are $|\alpha_k| \leq 1,\ k \geq 2$. Therefore, all its eigenvalues belong to unitary circle in the complex plane.
If the eigenfunctions $\{\Psi^k\}_{k=1}^{\infty}$, any initial density can be factorised as

$$\rho_0(\mathbf{x}) = \rho_{inv}(\mathbf{x}) + \sum_{k=2}^{\infty} A_k\ \Psi^k(\mathbf{x})\ , \tag{2.66}$$

where A_k are such that ρ_0 is real and non-negative in all the domain. The time evolution is thus:

$$\rho_t(\mathbf{x}) = \rho_{inv}(\mathbf{x}) + \sum_{k=2}^{\infty} A_k\ \alpha_k\ \Psi^k(\mathbf{x}) = \rho_{inv}(\mathbf{x}) + O\left(e^{-t\log\left|\frac{1}{\alpha_2}\right|}\right)\ , \tag{2.67}$$

therefore being the rate of convergence to the invariant measure determined by the second largest eigenvalue of the Perron-Frobenius operator.
In case of the logistic map, for $r < 3$ there is a unique fixed point x^*, and thus

$$\rho_t(x) \to \delta(x - x^*)\ . \tag{2.68}$$

For $r_{n-1} < r_n$, the trajectories are attracted by periodic orbit $(x_1, x_2, \ldots, x_{2^n})$ of period 2^n, such that after a transient time

$$\rho_t(x) = \sum_{k=1}^{2^n} c_k(t)\ \delta(x - x_k)\ , \tag{2.69}$$

where $c_k(t)$ are coefficient evolving in a cyclic way, i.e. $c_1(t+1) = c_{2^n}(t)$, $c_2(t+1) = c_1(t)$ and so on. For $n \to \infty$, we come back to the Feigenbaum attractor case.

We conclude by giving the exact invariant probability density for the logistic map Equation 2.17 at the Ulam point $r = 4$, by exploiting the property of *topological conjugacy* with the tent map, see 2. It easy to argue, see [Cencini et al., 2009], that evolution of the pdf density of the tent map satisfies

$$\rho_{t+1}(y) = \frac{1}{2}\rho_t\left(\frac{y}{2}\right) + \frac{1}{2}\rho_t\left(1 - \frac{y}{2}\right)\ , \tag{2.70}$$

because small intervals centered in y, whose measure is the probability to lie in that interval, are the pre-images of two intervals whose centers are $y/2$ and $1 - y/2$.
Since the tent map and the logistic map are topologically conjugated, the invariant pdf are related by the change of variable in Equation 2.29 as

$$\rho_{inv}^{(y)}(y) = \left|\frac{dh}{dx}\right|^{-1} \rho_{inv}^{(x)}(x = h^{(-1)}(y))\ , \tag{2.71}$$

and since $\rho_{inv}^{(y)}(y) = 1$, we get

$$\rho_{inv}^{(x)}(x) = \frac{1}{\pi\sqrt{x(1-x)}}\ , \tag{2.72}$$

which is exactly the pdf numerically found in Figure 2.14-(a).

2.5.2 Ergodicity

In Figure 2.14 we have obtained the pdf of the logistic map Equation 2.17 by iterating the Perron-Frobenius operator. Such systems, for which the time evolution visit all the phase-space, are called *ergodic*. Ergodic systems are very interesting, because ensemble averages can be computed with time averages, therefore simplifying a lot the computation. The problem to asses if a system satisfy the ergodic hypothesis goes under the name of the modern ergodic problem.
In abstract terms, a continuous time or a discrete dynamical system can be defined by the triad (Γ, f^t, μ), where Γ is the phase-space, $f^t : \Gamma \to \Gamma$ is the time evolution operator, such that

$$\mathbf{x}_0 \to \mathbf{x}_t = f^t \, \mathbf{x}_0 \ , \tag{2.73}$$

where for maps it assumes discrete time steps, and μ is the invariant measure under the evolution of the time evolving operator f^t , i.e.

$$\mu(B) = \mu(f^{-t}B) \ , \tag{2.74}$$

for any measurable set $B \subset \Gamma$.
Notice that we speak of measure and not invariant pdf, because for dissipative systems the invariant measure is typically singular with respect to the Lebesgue measure, see Figure 2.7.

A system (Γ, f^t, μ) is said *ergodic*, with respect to the (ergodic) invariant measure μ, if for any integrable (measurable) function $O(\mathbf{x})$, we have

$$\bar{O} := \lim_{T\to\infty} \frac{1}{T} \int_{t_0}^{t_0+T} dt \; O(\mathbf{x}_t) = \int_\Gamma d\mu(\mathbf{x}) \; O(\mathbf{x}) =: \langle O \rangle_\mu \ , \tag{2.75}$$

where $\mathbf{x}_t = f^{t-t_0}\mathbf{x}_0$ for almost all (with respect to μ) the initial conditions $\mathbf{x}_0$.

We first remark that all the definitions are done with respect to the measure μ, thus possibly failing for sets of 0 μ measure, which are not null with respect to another measure.
Secondly, ergodicity is not a peculiar property of chaos. Let consider the map on the torus $[0,1] \times [0,1]$

$$\begin{cases} x_t^{(1)} & = x^{(1)}(0) + \omega_1 \, t \qquad \mod 1 \\ x_t^{(2)} & = x^{(2)}(0) + \omega_2 \, t \qquad \mod 1 \ , \end{cases} \tag{2.76}$$

for which the Lebesgue measure $d\mu(\mathbf{x}) = dx^{(1)}dx^{(2)}$ is invariant. It can be proven, see [Cencini et al., 2009] for example, that if the fraction ω_1/ω_2 is rational, the motion of Equation 2.76 is periodic and non-ergodic with respect of the Lebesgue measure; however for ω_1/ω_2 irrational the motion is quasi-periodic and ergodic with respect to the Lebesgue measure.

2.6 Dimensions

Roughly speaking, the dimension of a geometrical object is the information needed to specify its points accurately [Eckmann and Ruelle, 1985, Ott, 2002]. We have seen examples of attractors constituted by single points (so zero dimensional) and limiting cycles (curves in space are one dimensional). The peculiarity of dynamical systems, however, is the fact the typically their attractors have a dimension that cannot be described by an integer value: they are thus *fractals*, a term first coined by [Mandelbrot, 1982] who first started systematically studying the geometric properties of those objects. Fractals exhibits a a fine structure at arbitrary small scale and the determination of their dimension can provide information on the mathematical space needed to describe such objects.
In this section, we address this question by giving the definition of the most used concept of dimensions in dynamical systems.

2.6.1 Box Counting Dimension

Let us first introduce the so-called *counting box dimension*, also known as *capacity*. Let $S \in \mathbb{R}^N$ be a compact set. We say that a set of cubes of edge ϵ, $\{B_i(\epsilon) : i \in \mathcal{I}\}$, where $\mathcal{I}$ is countable indexes set, covers the set S if $S \subset \bigcup_{i\in\mathcal{I}} B_i(\epsilon)$. Let $\tilde{N}(\epsilon) = |\{B_i(\epsilon) : i \in \mathcal{I}\}|$ the the numbers of cubes needed to cover the compact set S. The capacity of S is defined as

$$D_0 = \lim_{\epsilon\to 0} \frac{\tilde{N}(\epsilon)}{\ln(1/\epsilon)} \, . \tag{2.77}$$

This definition is very simple and indeed straightforward to apply in numerical implementations. It is clear that, no matter how large ϵ is, only one cube is needed to cover a single point and therefore its capacity is 0. More generally, the capacity of a finite discrete set of points is still zero, because for a sufficiently small ϵ the number of cubes needed to cover the set is the number of points. For a line, like in Figure 2.3(b), the number of boxes scale generally with $\sim l/\epsilon$, where l is the length of the curve, and therefore Equation 2.77 yields $D_0 = 1$. Likewise, for a surface we would have $\tilde{N}(\epsilon) \sim A/\epsilon^2$, where A is the surface, and therefore $D_0 = 2$. However, this definition requires only a rough estimation of the number of boxes to cover the set, since $\forall K \in \mathbb{R}$, if $\tilde{N}(\epsilon) = K\epsilon^{-d}$, then $D_0 = d$ independently of K. We have seen how box counting dimension can reproduce the expected intuitive dimension of common geometrical object. However, we have seen how there are objects that manifest a fine structure at arbitrarily small scale and therefore its dimension cannot be trivially inferred by the dimension of the space in which they are embedded. For example, Lorenz butterfly Figure 2.10 seems to be a surface, but detailed analysis show that its fractal dimension is $D_0 \simeq 2.06$, meaning that it is more than a surface but less than a volume and this is a consequence of its fine structure.
An interesting example of 1d fractal, in which the capacity happens to be not integer, is given by Cantor set, defined as follow: from the closed interval $[0, 1]$ remove the middle open interval $(\frac{1}{3}, \frac{2}{3})$ of length 1/3, leaving the two closed intervals $[0, \frac{1}{3}]$, $[\frac{2}{3}, 1]$ From the two remaining parts, remove again their third middle open intervals $(\frac{1}{9}, \frac{2}{9})$ and $(\frac{7}{9}, \frac{8}{9})$, which leave four closed intervals, namely $[0, \frac{1}{9}]$, $[\frac{2}{9}, \frac{1}{3}]$, $[\frac{2}{3}, \frac{7}{9}]$ and $[\frac{8}{9}, 1]$. If we repeat this operation to infinity, we are left with the Cantor set $\mathcal{C}$. At step n, staring from $n = 0$, we remove 2^n intervals of length $1/3^{n+1}$ and therefore Cantor set has a null Lebesgue measure, since the total length removed is $\sum_{n=0}^{\infty} \frac{2^n}{3^{n+1}} = 1$. Topologically $\mathcal{C}$ is a closed, since the complement of a countable union of open sets, subset of $\mathbb{R}$ (and bounded) and therefore is complete due to Heine-Borel theorem. Moreover, $\mathcal{C}$ is uncountable. To see that, let us build a bijective function with real numbers. At each interaction of the construction of Cantor set, we are left with a subset that either goes to the left or to the right. Thus, we associate a 1 if a point lies in the right interval and 0 if it is the left part. Therefore, in the limiting process we can associate each point with a distinct sequence in $\{0, 1\}^{\mathbb{N}}$ and since these sequence are in bijective correspondence, through binary representation, with the numbers on the interval $[0, 1]$, we conclude that its cardinality is non numerable. Cantor set has the same number of points of the original interval in a sense, with zero measure though.
Let us compute its box counting dimension. Since at each iteration we are left with 2^n intervals of length $1/3^n$, if we choose the sequence $\epsilon_n = 1/3^n$, which obviously yields $\lim_{n\to\infty} \epsilon_n = 0$, then we need $\tilde{N}(\epsilon_n)$ boxes to cover the set and therefore

$$D_0 = \frac{\ln 2}{\ln 3} = 0.63\ldots \, , \tag{2.78}$$

meaning that Cantor set is "something" more that a discrete set of points (and indeed is uncountable as already seen), but not enough to be counted as full interval in the real line, with dimension 1. Indeed, it is easy to see that Cantor set have self-similarity, since every little interval of length $1/3^n$ can put in bijective correspondence with $[0, 1]$ through an affine function (it looks exactly equal no matter the "lens" we use to observe it). However,

we point out that self similarity does is not always manifest in fractal objects [Ott, 2002] (that are merely defined with a non integer dimension) and indeed it does not appear in force damped pendolum attractor in Figure 2.6.

2.6.2 D_q Dimension

Box counting dimension is build by looking at the number of cubes that cover an attractor. However, it is not obvious that all the cubes should count equally to the dimension counting, since a trajectory can spend more time in a box than in another, and this is particularly true for fractal attractors, and therefore we would like to look at the frequencies at which trajectories visit the the various cubes covering the attractor. We say that these frequencies are natural if they are independent on the initial condition, expect at most for a set of zero Lebesgue measure, and they are defined as

$$\mu_i = \lim_{T\to\infty} \frac{\eta(C_i, \mathbf{x}_0, T)}{T} \ , \tag{2.79}$$

where $\eta(C_i, \mathbf{x}_0, T)$ is the time time spent by an orbit originating at $\mathbf{x}_0$ and lasting a time T in the cube C_i. We call typical points the ones for which the limit Equation 2.79 gives the same results expect at most for a set of zero Lebesgue measure.
By keeping that definition in mind, one may define the D_q dimension as [Hentschel and Procaccia, 1983, Grassberger, 1983]

$$D_q = \frac{1}{1-q} \lim_{\epsilon\to 0} \frac{\ln I(q,\epsilon)}{\ln(1/\epsilon)} \ , \tag{2.80}$$

where

$$I(q,\epsilon) = \sum_{i=1}^{\tilde{N}(\epsilon)} \mu_i^q \ , \tag{2.81}$$

where $\tilde{N}(\epsilon)$ are again the number of cubes of side ϵ needed to cover the attractor. This definition depends on a continuous parameter q, which measures the weight of the visiting frequencies: for $q > 0$ cubes with larger frequencies count more. For $q = 0$, $I(q,\epsilon) = \tilde{N}(\epsilon)$ and therefore we retrieve the box counting dimension. Moreover, it is easy to see that we get again D_0 independently of q also in the case that all frequencies are equal, since in that case $I(q,\epsilon) = (1-q)\tilde{N}(\epsilon)$. By defining $D_1 = \lim_{q\to 1} D_q$, from L'Hospital's rule we get

$$D_1 = \lim_{\epsilon\to 0} \frac{\sum_{i=1}^{\tilde{N}(\epsilon)} \mu_i \ln \mu_i}{\ln \epsilon} \ , \tag{2.82}$$

which is nothing than the *information dimension* [Eckmann and Ruelle, 1985] and [Balatoni and Renyi, 1956].
In general D_q decreases when q increase, expect for the cases when $\mu_i \sim 1/(\epsilon)$ for which $D_q = D_0, \forall q$, and it can be proved that

$$D_{q_1} \leq D_{q_2} \qquad q_1 > q_2 \ . \tag{2.83}$$

Numerically, D_q can be estimated by running a long trajectory and computing the time the trajectory spend in each cube varying ϵ, as it has been done for several fractal attractors in [Russell et al., 1980] by plotting $I(q,\epsilon)$ against $\ln \epsilon$ which can be linearly interpolate for a wide range of ϵ values. However, this method gives a good estimate only in the case that the total time of the orbit T is small than the typical roundoff induced computer period, since due to finite machine precision, computer orbits repeat them self after a certain time.

2.6.3 Hausdorff Dimension

We briefly mention a more difficult to define and yet useful definition of dimension is given by Hausdorff dimension [Gneiting et al., 2012, Hausdorff, 1919]. Let us first introduce the concept of *Hausdorff measure*. The diameter of a metric space A is defined as the maximum distance between its points

$$|A| = \sup_{\mathbf{x},\mathbf{y}\in A} d(\mathbf{x},\mathbf{y}) \ , \tag{2.84}$$

where $d : A \times A \to \mathbb{R}^+$ is the metric. Let us consider a countable sequence of metric spaces S_i with diameter ϵ_i less or equal a certain threshold δ, such that they cover A, i.e. $A \in \bigcup_i S_i$. Hausdorff measure is defined as

$$\Gamma_H^d = \lim_{\delta\to 0} \inf_{S_i} \sum_{i=1}^{\infty} \epsilon_i^d \ . \tag{2.85}$$

If for example A is a surface in an euclidean space, Γ_H^2 is just the area of the set, while for $d < 2$, $\Gamma_H^d = \infty$ and $\Gamma_H^d = 0$ for $d > 2$. More generally, it can be proved that Hausdorff measure is infinite if d is smaller than some critical value and 0 if it is greater. Hausdorff dimension is defined as this critical value

$$D_H = \min\{d : 0 < \Gamma_H^d < \infty\} \ . \tag{2.86}$$

2.6.4 The Pointwise Dimension

Another important concept useful to study strange attractors and invariant sets is the pointwise dimension $D_p(\mathbf{x})$. Let $B_\epsilon(\mathbf{x})$ an N dimensional ball centered in $\mathbf{x}$ with radius ϵ. The pointwise dimension of a probability measure μ at $\mathbf{x}$ is defined as

$$D_p(\mathbf{x}) := \lim_{\epsilon\to 0} \frac{\ln \mu(B_\epsilon(\mathbf{x}))}{\ln \epsilon} \ . \tag{2.87}$$

If the invariant measure is ergodic, then $D_p(\mathbf{x})$ assumes a single common value $\bar{D}_p(\mathbf{x})$ for all points $\mathbf{x}$ except a set of zero μ- measure. As proved by [Young, 1982], in this case the pointwise dimension is connected to the Information dimension ([Eckmann and Ruelle, 1985] and [Balatoni and Renyi, 1956]) and the single common value is

$$\bar{D}_p(\mathbf{x}) = D_1 \ . \tag{2.88}$$

Let us consider for simplicity the case of invertible maps $\mathbf{x}' = \mathbf{m}(\mathbf{x})$. To show that the pointwise dimension assumes the same common value, we want to first prove that

$$D_p(\mathbf{x}) = D_p(\mathbf{x}') \ . \tag{2.89}$$

Since the measure is invariant, then $\mu(B_\epsilon(\mathbf{x})) = \mu(\mathbf{m}(B_\epsilon(\mathbf{x})))$. Assuming that the map is smooth and ϵ sufficiently small, then the ball $B_\epsilon(\mathbf{x})$ is mapped to an elipsoidal region centered in $\mathbf{x}' = \mathbf{m}(\mathbf{x})$. Thus, we can find constants $r_1 > r_2$ such that the ball $B_{r_1\epsilon}(\mathbf{x}')$ contains $\mathbf{m}(B_\epsilon(\mathbf{x}))$ that contains $B_{r_2\epsilon}(\mathbf{x}')$ (see [Cencini et al., 2009]). Therefore

$$\mu(B_{r_1\epsilon}(\mathbf{x}')) \geq \mu(\mathbf{m}(B_\epsilon(\mathbf{x}))) = \mu(B_\epsilon(\mathbf{x})) \geq \mu(B_{r_2\epsilon}(\mathbf{x}')) \ . \tag{2.90}$$

From Equation 2.87, we have

$$D_p(\mathbf{x}') = \lim_{\epsilon\to 0} \frac{\ln \mu(B_{r_{1,2}\epsilon}(\mathbf{x}'))}{\ln r_{1,2}\epsilon} = \lim_{\epsilon\to 0} \frac{\ln \mu(B_{r_{1,2}\epsilon}(\mathbf{x}'))}{\ln \epsilon} \ , \tag{2.91}$$

and from Equation 2.90 it immediately follows that $D_p(\mathbf{x}') \geq D_p(\mathbf{x}) \geq D_p(\mathbf{x}')$, i.e. $D_p(\mathbf{x}) = D_p(\mathbf{x}')$.

Now, suppose by absurd that $D_p(\mathbf{x})$ is not the same for almost every $\mathbf{x}$ with respect to the ergodic measure μ. Thus, there exists some value d_p and a set S_-, such that $D_p(\mathbf{x}) \leq d_p$ for almost every $\mathbf{x} \in S_-$ and another disjoint set S_+ such that $D_p(\mathbf{x}) > d_p$ for almost every $\mathbf{x} \in S_+$. Moreover, $\mu(S_\pm) > 0$. Since $D_p(\mathbf{x}) = D_p(\mathbf{x}')$, the two sets $S_\pm$ are invariant. But this is not possible because the measure is ergodic and would be otherwise be divided into two parts, one never visiting the other.

2.7 Delay's Method, Embeddings and Data Assimilation

In real world applications we usually have sparse and rare measurements, often a scalar sequence which can be either one of the coordinates of a higher dimensional system or more generally a function of the state vector

$$g_t = G(\mathbf{x}_t) \ . \tag{2.92}$$

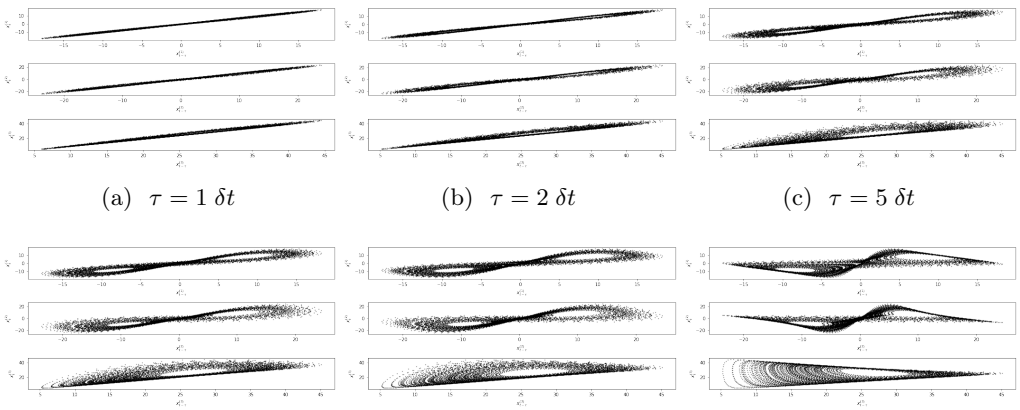
Even though the information provided by the scalar sequence is incomplete, we could learn properties of the system from which is was generated by using M dimensional *delayed coordinates* [Takens, 1979] $\mathbf{y} = (g_t, g_{t-\tau}, g_{t-2\tau}, \ldots, g_{t-(M-1)\tau})$.
Since in principle we can integrate backward Equation 2.1, $\mathbf{x}_{t-j\tau}$ can be seen as a function of $\mathbf{x}_t$

$$\mathbf{x}_{t-j\tau} = f^{-j}(\mathbf{x}_t) \ , \tag{2.93}$$

therefore also the vector $\mathbf{y}_t$ can be thought as function of $\mathbf{x}_t$, though an operator as

$$\mathbf{y}_t = \mathbf{H}(\mathbf{x}_t) \ . \tag{2.94}$$

It can be shown that if M is sufficiently large, then delayed method reveals more and more details of the underlying dynamical system, as it is shown in Figure 2.16 for Lorenz 63.



(a) $\tau = 1\ \delta t$ (b) $\tau = 2\ \delta t$ (c) $\tau = 5\ \delta t$

(d) $\tau = 7\ \delta t$ (e) $\tau = 10\ \delta t$ (f) $\tau = 20\ \delta t$

Figure 2.16: Delayed plots ($x_t^{(i)}$ vs $x_{t-\tau}^{(i)}$) of coordinates of Lorenz 63 system, corresponding to different multiples of sampling time step $\delta t = 0.01$.

To give a meaningful result, Equation 2.94 must be such that for $\mathbf{x} \neq \mathbf{x}'$, we have

$$\mathbf{H}(\mathbf{x}) \neq \mathbf{H}(\mathbf{x}') \ , \tag{2.95}$$

and in such case we say the $\mathbf{H}$ is an *embedding* of the N-dimensional space $\mathbf{x}$ into the M-dimensional space $\mathbf{y}$.
So a natural question arizes: how large M should be in order to reproduce correctly the unknown dynamical system? For example, let us consider the system $dx/dt = \omega$, where

x is considered an angle, i.e. module 2π. If we consider a 3-dimensional embedding $\mathbf{y} = (G(x_t), G(x_{t-\tau}), G(x_{t-2\tau}))$, the orbit would result in a limiting cycle as shown in Figure 2.17-(a). Instead, if we build a 2-dimensional embedding $\mathbf{y} = (G(x_t), G(x_{t-\tau}))$, the trajectory would intersect itself, as shown in Figure 2.17-(b), being not a dynamical system, because the knowledge of $\mathbf{y}$ would not allow us to know the future evolution.

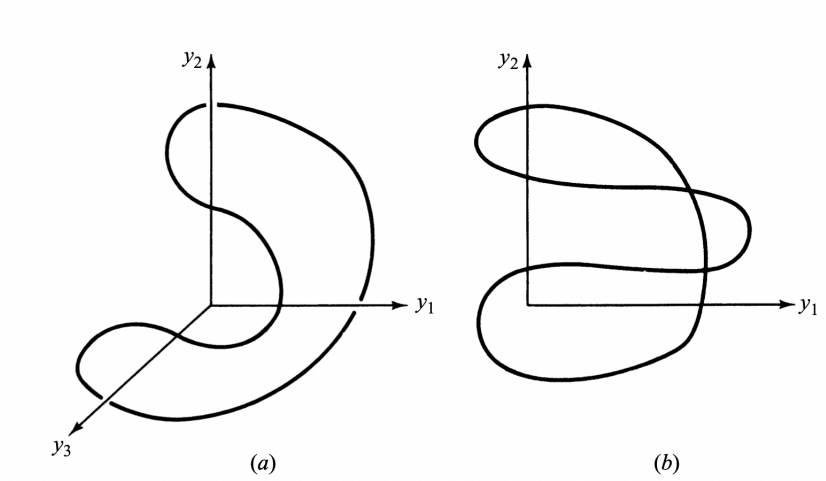


Figure 2.17: a) 3-dimensional versus b) 2-dimensional embedding of $\dot{x} = \omega$, with x module 2π. We can notice how the dimension of the embedding should be $m \geq 3$ to ensure to have a dynamical system. Image taken from [Ott, 2002].

Coming back to the minimum dimension required to have an embedding, [Takens, 1979] showed that

$$M \geq 2N + 1 \tag{2.96}$$

is sufficient.

The operator $\mathbf{H}$ can be seen more generally as the operator that acts to the real state vector $\mathbf{x}_t$, supposed to be unknown, in the act of measure the state of the system and therefore it goes under the name of *data assimilation*. More generally, a measure is also characterized by noise and thus we would have

$$\mathbf{y}_t = \mathbf{H}(\mathbf{x}_t) + \eta_t \ , \tag{2.97}$$

where η_t is zero mean Gaussian noise, i.e. $\mathbb{E}(\eta_t) = 0$, $\forall t$ and the correlation matrix $\mathbf{C}_t$ is defined as

$$\mathbb{E}[\eta_t \eta_{t'}^{\intercal}] = \delta(t - t') \ \mathbf{C}_t{}^{1} \ . \tag{2.98}$$

The goal of Data Assimilation (DA) is to combine theory with experiments in order to retrieve the most of information coming from usually sparse and rare measures. Moreover, since the dynamics is chaotic, it is necessary to continuously correct the trajectories with new data coming from observations. For a fundamental review on DA methods the reader is referred to [Park and Xu, 2016].

2.8 Characteristic Exponents

Characteristic exponents, also known as Lyapunov exponents, answer to the question of what is the response of the system when a slightly perturbation is added to the dynamics,

[1] Dirac delta is replaced with Kroeneker delta in case of maps.

which is useful in real world scenarios since measurement errors always arise. We will review the basic theory following [Ayers et al., 2021]. We refer to [Pikovsky and Politi, 2016] for a more exhaustive treatment. Let us consider the time evolution of a tiny perturbation $\delta\mathbf{x}_t$, where $\delta\mathbf{x}_0$ is small compared to the dynamical variable, $||\delta\mathbf{x}_0|| \ll ||\mathbf{x}_0||$.
The idea behind Lyapunov exponents is to look for the exponential growth factor of the perturbation, hence

$$\frac{||\delta\mathbf{x}_t||}{||\delta\mathbf{x}_0||} = e^{t\lambda_t} \qquad \Rightarrow \qquad \lambda_t \equiv \frac{1}{t}\ln\frac{||\delta\mathbf{x}_t||}{||\delta\mathbf{x}_0||}\,. \tag{2.99}$$

However, this definition yields only the characteristic exponents only along a particular direction dictated by $\delta\mathbf{x}$, it is explicitly dependent on time and the initial condition. Lyapunov exponents generalize the. previous notion to perturbation along any direction, by looking at the eigenvalues of some matrix of interest. Operationally, they determine the growth rate of spheres in the phase space. Indeed, if we look how a N-dimensional sphere (resembling for N independent perturbations) evolves in the phase space according to the dynamical system, some directions may eventually shrink, other diverge or remain neutral resulting in an elipsoid and therefore in a sense Lyapunov exponents are simply the fraction between the axis of this elipsoid with respect to the radius of the sphere. This idea lies also behind the notion of stable, unstable and neutral manifolds, on which we will com back later.
Geometrically, Lyapunov exponents thus measure the time evolution of linear, surface and volume elements [Eckmann and Ruelle, 1985]. The growth rate of the perturbation $\delta\mathbf{x}$ is given by the largest Lyapunov exponent $\lambda^{(1)}$. The growth rate of a surface element $\delta\sigma_t = \delta_1\mathbf{x}_t\delta_2\mathbf{x}_t$ is given by the sum of the two largest Lyapunov exponents $\lambda^{(1)} + \lambda^{(2)}$ and in general the evolution of the k-volume element is given by the sum of the largest k characteristic exponents. For instance, for a dynamical system in $\mathbb{R}^N$, the growth rate of the N-dimensional volume is given by the determinant of the Jacobian $|J| = |\frac{\partial\mathbf{g}}{\partial\mathbf{x}}|$ and is it thus $\lambda^{(1)} + \lambda^{(2)} + \cdots + \lambda^{(N)}$. If the map $\mathbf{g}$ has a constant jacobian, we have $\ln|J| = \lambda^{(1)} + \lambda^{(2)} + \cdots + \lambda^{(N)}$. Thus, in dissipative systems the sum of Lyapunov exponents is negative. For example, for the Lorenz 63 system [Lorenz, 1963], we have $d|J(t)|/dt = -(\rho + 1 + b) = \lambda^{(1)} + \lambda^{(2)} + \lambda^{(3)}$ and since $\lambda^{(2)} = 0$, which is generally true (the middle Lyapunov exponents of continuous and smooth dynamical systems is always zero, since it measures the evolution of perturbations along the flow of the system), we can determine $\lambda^{(3)}$ from $\lambda^{(1)}$ and viceversa.
The analysis of characteristic exponents relies on the Multiplicative Ergodic Theorem proved by Oseledet in 1968 [Ruelle, 1979, Ruelle, 1989, Eckmann and Ruelle, 1985, Oseledets, 1968, Pikovsky and Politi, 2016], which states:

Theorem 1 (**Multiplicative Ergodic**). *Let ρ be a probability measure on space $\mathbb{M}$ of dimension N (either $\mathbb{R}^N$ or a Hilbert space or a compact manifold) and $f : \mathbb{M} \to \mathbb{M}$ a measure preserving map such that ρ is ergodic* [2] *. Moreover, let $\mathbf{T} : \mathbb{M} \to Mat(N, \mathbb{R})$ a measurable map such that*

$$\int \rho(dx)\ln^+||\mathbf{T}(\mathbf{x})|| < \infty\,, \tag{2.100}$$

where $\ln^+(\mathbf{u}) = \max(0, \ln\mathbf{u})$ is the positive part.
By defining the matrix $\mathbf{T}^t_{\mathbf{x}} \equiv \prod_{k=0}^{t-1}\mathbf{T}(f^k(\mathbf{x}))$, for almost all $\mathbf{x}$ with respect to ρ, the following limit exists:

$$\lim_{t\to\infty}\left(\mathbf{T}^{t\,\intercal}_{\mathbf{x}}\mathbf{T}^t_{\mathbf{x}}\right)^{1/2t} = \mathbf{P}_{\mathbf{x}}\,\mathbf{D}\mathbf{P}^{\intercal}_{\mathbf{x}}\,, \tag{2.101}$$

where $\mathbf{P}_{\mathbf{x}}$ is the eigenvector matrix and the diagonal matrix $\mathbf{D}$ does not depend on the initial condition $\mathbf{x}$ and the natural logarithms of its elements are the Lyapunov exponents $\lambda^{(1)} \geq \lambda^{(2)} \geq \cdots \geq \lambda^{(N)}$ (possibly repeated with multiplicity), ρ-almost everywhere constant.

[2] An invariant measure is ergodic is if it cannot be linearly decomposed by other invariant measures.

Now, let us assume that the characteristic exponents are no longer repeated with multiplicity and and we call m_i the multiplicity of $\lambda^{(i)}$. Let us define

$$\mathbb{E}^i_{\mathbf{x}} \equiv \{\lambda \in \mathbf{D} : \lambda \leq e^{\lambda^{(i)}}\} \, . \tag{2.102}$$

Then we have $\mathbb{R}^N = \mathbb{E}^1_{\mathbf{x}} \supset \mathbb{E}^2_{\mathbf{x}} \supset \dots$ and the following

Theorem 2. *For almost all* $\mathbf{x}$ *with respect the measure* ρ*, the following limit holds:*

$$\lim_{t\to\infty} \frac{1}{t} \ln ||\mathbf{T}^t_{\mathbf{x}} \mathbf{u}|| = \lambda^{(i)} \, , \qquad \mathbf{u} \in \mathbb{E}^i_{\mathbf{x}} / \mathbb{E}^{i+1}_{\mathbf{x}} \, . \tag{2.103}$$

2.8.1 Continuous Systems

The evolution of the perturbation vector $\delta\mathbf{x}$ takes place in the tangent boundle of the mainifold and thus is determined by the equation

$$\frac{d\, \delta\mathbf{x}}{dt} = J_{\mathbf{g}(\mathbf{x_t})} \delta\mathbf{x} \, , \tag{2.104}$$

where $J_{\mathbf{g}(\mathbf{x_t})}$ is the Jacobian of the evolution function $\mathbf{g}$ computed at the point $\mathbf{x}_t = f^t(\mathbf{x}_0)$, since in principle it can be dependent also on the initial condition. This definition can be generalised further [Pikovsky and Politi, 2016], even though here we consider only smooth and continuous systems.
The solution of Equation 2.104 can be found using a fundamental matrix $\mathbf{T}$ [Ayers et al., 2021], denoted coherently with the hypothesis of Oseledets theorem, defined as a solution of the fundamental equation

$$\dot{\mathbf{T}}^t = J_{\mathbf{g}(\mathbf{x_t})} \mathbf{T}^t \, . \tag{2.105}$$

Given a solution of the fundamental matrix $\mathbf{T}^t$, any solution of Equation 2.104 can be written as $\delta\mathbf{x}_t = \mathbf{T}^t\mathbf{c}$, for some vector $\mathbf{c} \in \mathbb{R}^N$. Without loss of generality, we can assume that $\det \mathbf{T}^0 \neq 0$, since for any constant matrix $\mathbf{P}$, we can write another fundamental matrix $\mathbf{A}^t = \mathbf{T}^t(\mathbf{T}^t)^{-1}\mathbf{P}$ such that $\mathbf{A}^0 = \mathbf{P}$. The solution of Equation 2.104 is given by $\delta\mathbf{x}_t = \mathbf{A}^t\mathbf{c}'$, where $\mathbf{c}' = \mathbf{P}^{-1}\mathbf{T}^0\mathbf{c}$. For simplicity, we set $\mathbf{T}^0 = \mathbf{I}_N$ so that $\mathbf{c} = \delta\mathbf{x}_0$. Coming back again to the example of the largest perturbation only, Equation 2.99 can be rewritten as

$$e^{\lambda_t} = \left(\frac{||\delta\mathbf{x}_t||}{||\delta\mathbf{x}_0||}\right)^{1/t} = \left(\frac{\sqrt{\delta\mathbf{x}_0^\intercal (\mathbf{T}^t)^\intercal \mathbf{T}^t \delta\mathbf{x}_0}}{\sqrt{\delta\mathbf{x}_0^\intercal \delta\mathbf{x}_0}}\right)^{1/t} . \tag{2.106}$$

We are interested in the limit:

$$\lim_{t\to\infty} \ln\left(\frac{\sqrt{\delta\mathbf{x}_0^\intercal (\mathbf{T}^t)^\intercal \mathbf{T}^t \delta\mathbf{x}_0}}{\sqrt{\delta\mathbf{x}_0^\intercal \delta\mathbf{x}_0}}\right)^{1/t} = \lim_{t\to\infty} \ln\left(\delta\mathbf{x}_0^\intercal (\mathbf{T}^t)^\intercal \mathbf{T}^t \delta\mathbf{x}_0\right)^{1/2t} . \tag{2.107}$$

The analysis of the latter limit relies on the Multiplicative Ergodic Theorem, for which the limit

$$\mathbf{W}_{\mathbf{x}_0} \equiv \lim_{t\to\infty} \left[(\mathbf{T}^t_{\mathbf{x}_0})^\intercal \mathbf{T}^t_{\mathbf{x}_0}\right]^{1/2t} \tag{2.108}$$

exists and depends on the initial condition $\mathbf{x}_0$. However, $\mathbf{W}_{\mathbf{x}_0}$ can be diagonalized as

$$\mathbf{W}_{\mathbf{x}_0} = \mathbf{P}_{\mathbf{x}_0}\, \mathbf{D} \mathbf{P}^\intercal_{\mathbf{x}_0} \, , \tag{2.109}$$

where $\mathbf{P}_{\mathbf{x}_0}$ is the matrix of the orthonormal eigenvectors. The eigenvalue matrix $\mathbf{D}$ is not dependent anymore on the initial condition and the Lyapunov exponents in descending order $\lambda^{(1)} \geq \lambda^{(2)} \geq \dots \lambda^{(N)}$ are the natural logarithms of its diagonal elements

$$\lambda^{(i)} = \ln \mathbf{D}_{ii} \, . \tag{2.110}$$

2.8.2 Discrete Maps

In case of discrete maps, the perturbation $\delta\mathbf{x}_t$ obeys the equation

$$\delta\mathbf{x}_{t+1} = J_{\mathbf{m}(\mathbf{x}_t)}\delta\mathbf{x}_t \ . \tag{2.111}$$

The perturbation can be obtained from the initial point $\mathbf{x}_0$ as

$$\begin{aligned}\delta\mathbf{x}_t = J_{\mathbf{m}(\mathbf{x}_{t-1})}\delta\mathbf{x}_{t-1} = J_{\mathbf{m}(\mathbf{x}_{t-1})}J_{\mathbf{m}(\mathbf{x}_{t-2})}\delta\mathbf{x}_{t-2} = \\ = J_{\mathbf{m}(\mathbf{x}_{t-1})}J_{\mathbf{m}(\mathbf{x}_{t-2})}\dots J_{\mathbf{m}(\mathbf{x}_0)}\delta\mathbf{x}_0 = \prod_{k=0}^{t-1} J_{\mathbf{m}(\mathbf{x}_k)}\delta\mathbf{x}_0 \ .\end{aligned} \tag{2.112}$$

Therefore, we write the map $\mathbf{T}$ in the hypothesis of Theorem 1 as

$$\mathbf{T}^t_{\mathbf{x}_0} = \prod_{k=0}^{t-1} J_{\mathbf{m}(\mathbf{x}_k)} \ . \tag{2.113}$$

Again the characteristic exponents are defined as the natural logarithm of the eigenvalues of the limit there defined.

2.8.3 An Algorithm for Numerical Computation

The computation of the fundamental matrix for both continuous systems Equation 2.8.1 and discrete maps Equation 2.113 accumulates numerical errors and instability in the asymptotic limit, in which we are interested [Pikovsky and Politi, 2016]. Instead of computing a large time limit, we propagate perturbation for a short time and then we re-normalize orthogonal perturbations, computing the Local Lyapunov Exponents (LLE). If then the system is ergodic, the aritmetic mean of local Lyapunov exponents converge to the true characteristic exponents. We present the method developed by [Benettin et al., 1980] based on QR decomposition [Gottwald and Reich, 2021] and recently used by [Ayers et al., 2021] to learn Local Lyapunov exponents with supervised machine learning.
Given a trajectory $\{\mathbf{x}_t : t \in [0, T]\}$, we initialize a perturbation matrix $\mathbf{Q}_0$, such that its columns are orthogonal and normalized even though the latter assumption is not strictly necessary). Then we repeat the following operations for M iterations, staring with iteration $j = 1$:

- At step $j \geq 1$, we propagate perturbation along the trajectory $\{\mathbf{x}_t : t \in [(j-1)\tau, j\tau]\}$. where $\delta t << \tau << T$. At each iteration j, the algorithm generates N local exponents $\lambda^{(i)}_j$, $i = 1, 2, \dots, N$:
  - First, we numerically integrate Equation 2.8.1 or propagate Equation 2.113 to get the fundamental matrix $\mathbf{T}^{\tau j}$, by setting the initial condition as the identity $\mathbf{T}^{(j-1)\tau} = \mathbf{I}_N$.
  - We propagate the perturbation as $\mathbf{V}_j = \mathbf{T}^{\tau j}\mathbf{Q}_{j-1}$.
  - We orthogonalize $\mathbf{V}_j$, a crucial operation since the dynamical evolution mixes perturbation along different directions, by means of QR decomposition [Gottwald and Reich, 2021]:

    $$\mathbf{Q}_j\mathbf{R}_j = \mathbf{V}_j \ , \tag{2.114}$$

    such that $\mathbf{R}_j$ is an upper triangular matrix with positive coefficients on the diagonal and $\mathbf{Q}$ is an orthogonal matrix.

– The diagonal elements $r_j^{(i)}$ of $\mathbf{R}_j$ are the norm of the orthogonalized $v_j^{(i)}$ and since the columns $\mathbf{q}_{j-1}$ of $\mathbf{Q}_{j-1}$ are normalized, those are just the ratios that we need. The N Lyapunov Exponents (LE) at time $(j-1)\tau$ are thus

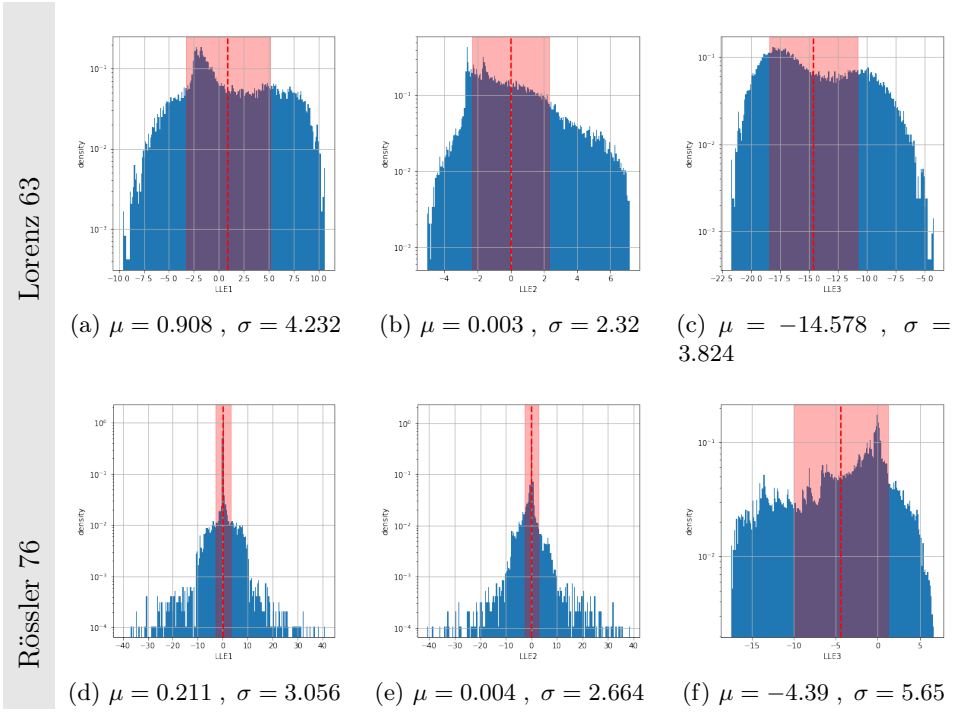
$$\lambda_j^{(i)} = \frac{1}{\tau} \ln r_j^{(i)} \ , \tag{2.115}$$

where the order of LLE must be chosen once and for all (possibly discarding the first k entries in order to achieve convergence).

- In then end, global characteristic exponents are computed as

$$\lambda^{(i)} = \frac{1}{m-k} \sum_{j=k}^{m} \lambda_j^{(i)} \ . \tag{2.116}$$

In general, this algorithm relies on the idea to compute Lyapunov exponents by calculating the ration between the axis of a elipsoid perturbation trough the tangent operator given by $\mathbf{M}$ with respect the initialized sphere of infinitesimal perturbations [Ayers et al., 2021]. However, is difficult that the leading perturbation $\mathbf{q}_0$ is mapped exactly in the leading axis of the elipsoid, raising to a poor estimate of the Lyapunov exponents in the first iterations, therefore needing to discard the first k local exponents in order to achieve better estimation. The Local Lyapiunov exponents distribution for [Lorenz, 1963] and [Rössler, 1976], computed from a dataset of 100000 points generated starting from a generic initial condition and a time step $\delta t = 0.002$ and with $\tau = 4$, is reported in Figure 2.18, while the estimated Lyapunov Exponents are reported in Table 2.1. For a complete analysis on the computational cost of the algorithm see [Ayers et al., 2021].



(a) $\mu = 0.908$, $\sigma = 4.232$ (b) $\mu = 0.003$, $\sigma = 2.32$ (c) $\mu = -14.578$, $\sigma = 3.824$

(d) $\mu = 0.211$, $\sigma = 3.056$ (e) $\mu = 0.004$, $\sigma = 2.664$ (f) $\mu = -4.39$, $\sigma = 5.65$

Figure 2.18: Distributions of Local Lyapunov Exponents (LLE) with mean μ (dashed line) and standard deviation σ (shaded area is between a σ deviation from the mean), computed from a dataset of 100000 samples with $\delta t = 0.002$ and $\tau = 4$ and discarding the first 1000 values. We do not assist to significant different results for τ in the range $[1, 40]$.

2.8.4 Lyapunov Dimension

Lyapunov exponents are also useful to compute along the dimension of the attractor. Let K be the largest integer such that the sum of the K largest Lyapunov exponents is greater

	$\lambda^{(1)}$	$\lambda^{(2)}$	$\lambda^{(3)}$
Lorenz 63	0.908 ± 0.046	0.003 ± 0.020	-14.578 ± 0.026
Rössler 76	0.211 ± 0.013	0.004 ± 0.031	-4.39 ± 0.22

Table 2.1: Computed Lyapunov exponents (LE) from LLE. They are estimated by dividing LLE into four equal parts and computing the mean as the average of the mean values and error as the standard deviation of the mean values. For both system, the middle Lyapunov exponent should be theoretically zero, since it corresponds to perturbations along the flow of the system for which we do not expect neither decay nor explosion.

than zero

$$\sum_{i=1}^{K} \lambda^{(i)} \geq 0 \, . \tag{2.117}$$

Kaplan and York [Kaplanand and Yorke, 1979] conjectured that we can use Lyapunov exponents to give a dimension for typical fractal attractors. The Lyapunov dimension is defined as

$$D_L \equiv K + \frac{1}{|\lambda^{(K+1)}|} \sum_{i=1}^{K} \lambda^{(i)} \, . \tag{2.118}$$

The conjecture states that Lyapunov dimension is equal to information dimension, at least for typical attractors

$$D_1 = D_L \, , \tag{2.119}$$

even though is has not been proved yet.
For example, in for the Hénon map we have $\lambda^{(1)} \simeq 0.603$ and $\lambda^{(2)} \simeq -2.34$ and thus $D_L = 1 + \frac{0.603}{2.34} \simeq 1.26$.

2.8.5 Time Reversal Dynamics

Let us briefly examine the properties of a system whose dynamics reverses in time. If such a system is such that its evolution map (or flow) is defined also for negative times, we call $\hat{f}^t = f^{-t}$ the time reversal flow. If ρ is an invariant (ergodic) map for the forward time evolution (discrete or continuous), it is invariant (ergodic) also for the time reversal dynamics. Moreover, its Lyapunov exponents are the same of the original system, but with opposite sign (which is intuitive, since trajectories that diverge in the forward dynamics will collide backward in time and viceversa). Accordingly, we have a sequence of subsets $\hat{\mathbb{E}}^1_{\mathbf{x}} \subset \hat{\mathbb{E}}^2_{\mathbf{x}} \subset \dots$ for almost all $\mathbf{x}$, such that

$$\lim_{t \to -\infty} \frac{1}{|t|} \ln ||T^t_{\mathbf{x}} \mathbf{u}|| = -\lambda^{(i)} \, , \qquad \text{if } \mathbf{u} \in \hat{\mathbb{E}}^i_{\mathbf{x}} / \hat{\mathbb{E}}^{i-1}_{\mathbf{x}} \, . \tag{2.120}$$

2.9 Stable and Unstable Manifolds

From Multiplicative Ergodic theorem Theorem 1 we know that there exist linear subspaces $\mathbb{R}^N = \mathbb{E}^1_{\mathbf{x}} \supset \mathbb{E}^2_{\mathbf{x}} \supset \dots$, such that

$$\lim_{t \to \infty} \frac{1}{t} \ln ||\mathbf{T}^t_{\mathbf{x}} \mathbf{u}|| = \lambda^{(i)} \, , \qquad \mathbf{u} \in \mathbb{E}^i_{\mathbf{x}} / \mathbb{E}^{i+1}_{\mathbf{x}} \, , \tag{2.121}$$

meaning that vectors in $\mathbb{E}^i_{\mathbf{x}} / \mathbb{E}^{i+1}_{\mathbf{x}}$ are expanded exponentially with rate given by $\lambda^{(i)}$ [Eckmann and Ruelle, 1985].
By looking to a non-linear analogous of these subspaces, we can define a *local stable manifold* tangent to the subspace $\mathbb{E}^i_{\mathbf{x}}$ at $\mathbf{x}$ and with the same dimension. Consider contraction $\lambda < 0$ and $\epsilon > 0$, then we define

$$\mathbb{V}^s_{\mathbf{x}}(\lambda, \epsilon) = \{\mathbf{y} : d(f^t(\mathbf{x}), f^t(\mathbf{y})) \leq \epsilon e^{\lambda t}), \forall t \geq 0\} \qquad (\lambda^{(i-1)} > \lambda > \lambda^{(i)}) \, , \tag{2.122}$$

where d is the metric of the space on which the system is defined.
If the dynamical system is defined also for negative times, we can define *global stable manifolds* as

$$\mathbb{V}^{s(i)}_{\mathbf{x}} \equiv \{\mathbf{y} : \lim_{t\to\infty} \frac{1}{t} \ln d(f^t(\mathbf{x}), f^t(\mathbf{y})) \leq \lambda^{(i)}\} = \bigcup_{t>0} f^{-t}(\mathbb{V}^s_{\mathbf{x}}(\lambda, \epsilon)) \ , \tag{2.123}$$

with $\lambda < 0$ and $\lambda^{(i)} < \lambda < \lambda^{(i-1)}$. Therefore we have a set of stable manifold corresponding to the negative characteristic exponents. By taking the largest of those, we can define the *global stable manifold* as

$$\mathbb{V}^s_{\mathbf{x}} \equiv \{\mathbf{y} : \lim_{t\to\infty} \frac{1}{t} \ln d(f^t(\mathbf{x}), f^t(\mathbf{y})) \leq 0\} \ . \tag{2.124}$$

The analogous of the stable manifolds for positive Lyapunov exponents are *unsatble mainifolds* and they can be either obtained for a system whose dynamics is defined also for negative times, by replacing t with $-t$ in the above definitions (since Lyapunov exponents of the time reversal dynamics are minus the Lyapunov exponents of the original system) or one can assume that the flow $f^t(\mathbf{x})$ and its space derivative $\partial f^t(\mathbf{x})/\partial \mathbf{x}$ are both injective, meaning that

$$f^t(\mathbf{x}) = f^t(\mathbf{y}) \left(\frac{\partial f^t(\mathbf{x})}{\partial \mathbf{x}} = \frac{f^t(\mathbf{y})}{\partial \mathbf{y}} \right) \qquad \Rightarrow \qquad \mathbf{x} = \mathbf{y} \ . \tag{2.125}$$

In the latter case, the definition of local unstable manifold, global unstable manifold and global unstable manifold are respectively ($\lambda > 0$ and $\epsilon > 0$):

$$\begin{aligned}
\mathbb{V}^u_{\mathbf{x}}(\lambda, \epsilon) &\equiv \{\mathbf{y} : \exists\ \mathbf{y}_{-t} \mid f^t(\mathbf{y}_{-t}) = \mathbf{y}, \text{ and } d(\mathbf{x}_{-t},\ \mathbf{y}_{-t}) \leq \epsilon e^{\lambda t},\ \forall t \geq 0\} \ , \\
\mathbb{V}^{u(i)}_{\mathbf{x}} &\equiv \{\mathbf{y} : \exists\ \mathbf{y}_{-t} \mid f^t(\mathbf{y}_{-t}) = \mathbf{y}, \text{ and } \lim_{t\to\infty} \frac{1}{t} \ln d(\mathbf{x}_{-t},\ \mathbf{y}_{-t}) \leq -\lambda^{(i)}\} = \\
&= \bigcup_{t>0} f^t(\mathbb{V}^u_{\mathbf{x}}(\lambda, \epsilon)) \ , \\
\mathbb{V}^u_{\mathbf{x}} &\equiv \{\mathbf{y} : \exists\ \mathbf{y}_{-t} \mid f^t(\mathbf{y}_{-t}) = \mathbf{y}, \text{ and } \lim_{t\to\infty} \frac{1}{t} \ln d(\mathbf{x}_{-t},\ \mathbf{y}_{-t}) \leq 0\} \ .
\end{aligned} \tag{2.126}$$

2.10 Entropies

Another way to describe chaotic systems is to look at the information production rate. Since infinitesimal neighbouring trajectories diverge due to the positiveness of the largest Lyapunov exponent, entropy is produced in the system. The reason is that if two trajectory cannot be distinguished at time 0 with finite precision, they will eventually become sufficiently distinct to be told apart. Similarly, the insignificant digits in decimal representation of a point will become more and more relevant as the orbit moves forward.
Following [Ott, 2002], we will give two definition of entropy, *metric entropy*, also called Kolmogorov-Sinai entropy [Kolmogorov, 1958, Sinai, 1959, Sinai, 1959], and topological entropy.

2.10.1 Kolmogorov-Sinai Entropy

Metric entropy is based on Shannon entropy, which measures the amount of information contained in a probabilistic event. Let $p_1, p_2, \ldots, p_m$ be a discrete probability distribution, i.e. $p_i \geq 0$, $\forall i$ and $\sum_{i=1}^m p_i = 1$. Shannon entropy is defined as

$$H(p_1, p_2, \ldots, p_m) = -\sum_{i=1}^m p_i \ln p_i \ , \tag{2.127}$$

where as usual we define $0 \ln 0 = 0$. For example, if one of the event was certain, i.e. $p_j = 1$ for a certain j and $p_i = 0$ for $i \neq j$, thus the information contained in that event would be zero. On the other hand, the greatest lack of information on the system, if we did not have other prior knowledge, would be the case of a uniform distribution, i.e. $p_i = 1/m$, $\forall i = 1, 2, \ldots, m$ and Shannon entropy would be $\ln n$. More generally, we have $0 \leq H(p_1, p_2, \ldots, p_m) \leq \ln m$.
Let ρ an invariant measure for the dynamical system with compact support. Let $\mathbb{W} = \{\mathbb{W}_1, \mathbb{W}_2, \ldots, \mathbb{W}_r\}$ be a finite partition of the support of ρ. We can define the entropy of the partition as

$$H(\{\mathbb{W}_i\}) = -\sum_{i=1}^{r} \rho(\mathbb{W}_i) \ln[\rho(\mathbb{W}_i)] \, . \tag{2.128}$$

Now, let us build a sequence $\{\mathbb{W}_i^{(n)}\}$ of smaller and smaller partitions by considering the sets $f^{-t}(\mathbb{W}_i)$, i.e. the backward evolution of steps $t = 1, 2, \ldots, n$ of the partition set $\mathbb{W}_i$. At the first step, for each pair $j, k = 1, 2, \ldots, r$, we form the r^2 intersections

$$\mathbb{W}_j \bigcap f^{-1}(\mathbb{W}_k) \, , \tag{2.129}$$

and we say that all of those non empty intersection are the second partition $\{\mathbb{W}_i^{(2)}\}$. The next partition $\{\mathbb{W}_i^{(3)}\}$ is built by taking all the non empty intersections among the r^3 intersections, which for indexes $j, k, l = 1, 2, \ldots, r$ are

$$\mathbb{W}_j \bigcap f^{-1}(\mathbb{W}_k) \bigcap f^{-2}(\mathbb{W}_l) \, . \tag{2.130}$$

Therefore, at stage n, we have

$$\begin{aligned} \left\{\mathbb{W}_i^{(n)}\right\} \equiv \Bigg\{ & \mathbb{W}_{i_1} \bigcap f^{-1}(\mathbb{W}_{i_2}) \bigcap f^{-2}(\mathbb{W}_{i_3}) \bigcap \cdots \bigcap f^{-(n-1)}(\mathbb{W}_{i_n}) \neq \emptyset : \\ & : (i_1, i_2, \ldots, i_n) \in \{1, 2, \ldots, r\}^n \Bigg\} \, . \end{aligned} \tag{2.131}$$

By taking the limit for $n \to \infty$ and the sup over all possible partitions of the support of ρ, the metric entropy (density) of the invariant measure ρ is defined as

$$h(\rho) = \sup_{\{\mathbb{W}_i\}} \lim_{n \to \infty} \frac{1}{n} H\left(\left\{\mathbb{W}_i^{(n)}\right\}\right) \, . \tag{2.132}$$

We stress that this definition holds for discrete maps as well as continuous systems. In the latter case, f^{-1} is simply the unit time reversed map.

In general, there is an interesting bound proved by [Ruelle, 1978] connecting the Kolmogorov-Sinai entropy with the sum of positive Lyapunov exponents.

Theorem 3. *Let f a differentiable map on a finite dimensional manifold and ρ an ergodic measure with compact support, then the following inequality holds:*

$$h(\rho) \leq \sum_{\lambda^{(i)} > 0} \lambda^{(i)} \, . \tag{2.133}$$

In many situations, the latter inequality is often an equality, as it was proven to hold more generally by Pesin for typical Hamiltonian systems [Pesin, 1976]:

$$h(\rho) = \sum_{\lambda^{(i)} > 0} \lambda^{(i)} \, , \tag{2.134}$$

which was later generalized to *Axiom A* systems [Ruelle, 1989](which are systems whose attractor is hyperbolic and with dense periodic orbits).
[Young, 1982] proved that metric entropy for an ergodic invariant measure ρ on a smooth two dimensional invertible map with Lyapunov exponents $\lambda^{(1)} > 0 > \lambda^{(2)}$ is related to information dimension as

$$D_1 = h(\rho)\left(\frac{1}{\lambda^{(1)}} + \frac{1}{|\lambda^{(2)}|}\right) , \tag{2.135}$$

where $\lambda^{(1)}, \lambda^{(2)}$ are the Lyapunov exponents for almost all $\mathbf{x}$ with respect to ρ.
Given Equation 2.135, for this kind of maps [Kaplanand and Yorke, 1979] conjecture is equivalent to

$$h(\rho) = \lambda^{(1)} , \tag{2.136}$$

as experiments for example on Hénon map confirm.

2.10.2 Topological Entropy

Topological entropy, introduced by [Adler et al., 1965], is built similarly to metric entorpy. Given a map $\mathbf{m}$, we build the same finer and finer partitions that we did to define $h(\rho)$, starting from a partition $\{\mathbb{W}_i\}$ and constructing the succession $\{\mathbb{W}_i^{(n)}\}$ as before. We define $N^{(n)}(\{\mathbb{W}_i\})$ as the number of non-empty components of the partition $\{\mathbb{W}_i^{(n)}\}$ derived from $\{\mathbb{W}_i\}$ and let

$$h_T(\mathbf{m}) = \sup_{\{\mathbb{W}_i\}} \lim_{n\to\infty} \left[\frac{1}{n} \ln N^{(n)}(\{\mathbb{W}_i\})\right] . \tag{2.137}$$

In general we have that

$$h_T \geq h(\rho) , \tag{2.138}$$

and that the topological entropies of $\mathbf{m}$ and its inverse $\mathbf{m}^{-1}$ are the same

$$h_T(\mathbf{m}) = h_T(\mathbf{m}^{-1}) . \tag{2.139}$$

The same is true for any map derived from $\mathbf{m}$ thorugh any continuous, invertible change of phase space variable.

2.11 Metrics to Asses Performances

Whereas there are several metrics to asses the skills of dynamical system, in this thesis we will make use of the Wasserstein Distance [Kantorovich, 1939] for the Climate and the R2 score [Heinisch et al., 1962] for the Weather assessment. The aim is to assess the goodness of Machine Learning predictions against the ground truth dynamics.

2.11.1 Long Term (Climate) Dynamics: the Wasserstein Distance

Wasserstein distance is distance between probability density functions that measures what is the cost to move a cloud of points with a certain mass to another cloud and for that reason it is included in the optimal transport problems, first formalized by [Monge, 1781]. The term was first attributed to [Vaserstein, 1969], even though the first that defined it was [Kantorovich, 1939]. Recently, it has been used by [Vissio et al., 2020] to assess the performance of certain climate models against ground truth references and by [Vissio and Lucarini, 2018] to evaluate the stochastic parametrization for fast-slow systems.
The power of Wasserstein distance above other measures of distances between probability distributions comes from the fact that Wasserstein distances take into account all the momenta of these measures, while for example the simple $L2$ of the difference of two measurable functions in $\mathbb{R}^N$ is not a suitable metric to asses distances between probability measures. Therefore, we will use Wasserstein Distance to asses the power of Machine

Learning models to reproduce the long term dynamics of dynamical systems, also known as Climate. Numerical method to compute it can be found in [Santambrogio, 2015].
The optimal cost is defined as the minimum effort to move a set of mass points into another [Villani, 2019]. Let us consider to point mass distributions of n_1, n_2 points respectively in a M-dimensional space as

$$\mu = \sum_{i=1}^{n_1} \mu_i \, \delta_{\mathbf{x}_i} \, , \qquad \nu = \sum_{i=1}^{n_2} \nu_i \, \delta_{\mathbf{y}_i} \, , \tag{2.140}$$

where $\delta_{\mathbf{x}_i}, \delta_{\mathbf{y}_i}$ are the Dirac delta measure associate to points $\mathbf{x}_i, \mathbf{y}_i$ and (μ_i, ν_i) are the respective masses, subject to the normalization condition $\sum_{i=1}^{n_1} \mu_i = \sum_{i=1}^{n_2} \nu_i = 1$.
The quadratic Wasserstein distance is defined as

$$W_2(\mu, \nu) = \left(\inf_{\gamma} \sum_{i,j} \gamma_{i,j} \, d(\mathbf{x}_i, \mathbf{y}_j)^2 \right)^{1/2} , \tag{2.141}$$

where γ_{ij} is the transfer protocol specifying how mass is transported form μ to ν and $d(\mathbf{x}, \mathbf{y})$ is the Euclidean distance in $\mathbb{R}^N$.
The transport couplings γ_{ij} are subject to the conditions:

$$\sum_j \gamma_{ij} = \mu_i \ \ \forall i \, , \qquad \sum_i \gamma_{ij} = \nu_j \ \ \forall j \, . \tag{2.142}$$

In case we deal with multidimensional probability distributions, such as those arising from dynamical systems, we first have to compute the invariant measures by discretizing the space with small boxes of side ϵ small and counting the fraction of points lying in those boxes, giving us the mass weights. The distances are taken as the distances of the centroids of those cubes. Finally, following [Vissio et al., 2020], we divide Equation 2.141 by the geometric mean of discretization points $\sqrt{n_1 n_2}$ such that the Wasserstein distance in a m-dimensional space is bounded by $\sqrt{m}$. Our numerical implementation is based on the work of G. Peyré, available at the link: `https://nbviewer.org/github/gpeyre/numerical-tours/blob/master/python/optimaltransp_1_linprog.ipynb`.

2.11.2 Short Term (Weather) Dynamics: the R2 Score

To asses the performance of our model in reproducing the short term dynamics (Weather) we compute point-wise accuracy of a ML algorithm as the R2 score between the ground truth dynamics and reproduced dynamics starting at the same point. This metric asses the short term predictive power of our model. To do so, R2 scores will be computed only for a time comparable with the inverse of the largest Lyapunov exponent of the model, because for larger times trajectories are anyway expected to diverge due to the chaotic dynamics. R2 score, also known as coefficient of determination, between a target output $\{y_i\}_{i=1,\dots,d}$ and a model prediction $\{\hat{y}_i\}_{i=1,\dots,d}$, is defined as

$$R2\big[(y_i, \hat{y}_i), i = 1, \dots, d\big] := 1 - \frac{\sum_{i=1}^{d} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{d} (y_i - \bar{y})^2} \in (-\infty, 1] \, , \tag{2.143}$$

where $\bar{y}$ is the mean of target outputs. A perfect score is 1, while a score of 0 means that the predictor is as good as predicting the mean of the target sequence every time. We notice that computing the $R2$ score is more meaningful than simply calculating the mean square displacement of the prediction from the target, which is the numerator of Equation 2.143, because it has also to balance the intrinsic variability of the target sequence, the denominator of Equation 2.143. Therefore a low mean square displacement of the prediction from the target in a region where the target has a very low variance, is not an indication of a good prediction. On the other hand, if the model can reproduce the target with some error and this error is still small compared to the variance of the target in the region of interest, then the prediction shall be considered good.

Chapter 3

Machine Learning

3.1 Introduction

In this chapter we will expose the basic Machine Learning theory and the fundamental architectures. In section 3.2 we encapsulate the learning problem into the realm of Statistical Learning in the contest of supervised learning. In section 3.3 we will discuss the concept of Uniform Convergence and its connections with the learning framework. We will proceed by presenting the Bias-Complexity trade-off in section 3.4, which is essential to understand the concept of overfitting, and how to pick the best learning algorithm in section 3.5. In section 3.6 we will discuss the importance of Regularization techniques to reduce the complexity of networks in order to avoid overfitting. In section 3.7 we start analysing the most used neural networks architectures, from Multi Layer Perceptron (or Feed-Forward networks) and we will present the most used training algorithm in section 3.8. Furthermore, ee will present and give detailed information of Recurrent Neural Networks in section 3.9, Autoencoders in section 3.10 and Convolutional Neural Networks in section 3.11. We will then conclcude with a brief exposition on Physics Informed Neural Networks in section 3.12.

3.2 A Formal Learning Framework

Machine Learning is a set of methods that allow computers to "learn" from data. The are three kind of Machine Learning domains:

- **Supervised Learning**
  Labels are provided to the learning algorithm, such that the task is usually to classify inputs to have the correct labels.
- **Unsupervised Learning**
  Data are not labelled, the learning algorithm is asked to capture the probability distribution of data without any human supervision.
- **Reinforcement Learning**
  While in the first two types of learning, the data are provided before the learning phase and the environment is fixed, in Reinforcement learning framework the learning algorithm, also known as *agent*, is asked to learn from the environment by interacting and modify it and adapt its *action* according to a *reward* that is gained for each action. In this way, the agent is able to find causal relations in data, which would otherwise be hidden, by actively modify the environment.

While Reinforcement learning is more similar to how human learn and it is a filed of intense research, it has not achieved the same level of maturity of supervised and unsupervised learning and most framework are still strongly dependent on human supervision in fine-

tuning.

Since out goal is to analyse dynamics systems, in this thesis we will focus primarily on unsupervised learning. We will now state the learning framework in the case of supervised tasks, which is better formalised and offers many tools to understand the best practices to make machines learn. We will follow [Shalev-Shwartz and Ben-David, 2014].
In case of unsupervised tasks, in which we do not have a labels set, but we can see the input dataset itself as the label set and therefore most of the definitions are similar. In case of sequences, our goal is to predict the next element of the sequence by the current state. Since element of the sequence are not statistically independent, we have to feed into the models sequences of length greater than 1 in order to learn the dynamical equations. However, we may assume that different sequences are statistically independent and this is approximately true if the sequence lengths are greater that the typical correlation time. In this case, the label can be consider the same input set and the goal is to minimize the distance between prediction and true values. In case of autoencoders, as we will see, the goal is to reconstruct the trajectory and therefore the expected output would be the exact same sequence fed into the input.

In the next sections, we will develop the basic theory on Statistical Learning, a framework that allows establish the statistical background that allows machine to learn from the data. An comprehensive textbook on statistical learning basis is [James et al., 2013]. Standard book focusing on general Machine Learning are [Bishop, 2006, Murphy, 2013]. While Machine Learning is referred to any problem that is learnable by computers, especially in the domain of Neural Networks the state of the art is Deep Learning. Deep Neural Networks are simply Neural Networks with many layers one after another. The bible reference book on Deep Learning is [Bengio et al., 2015]. A great review for physicist on deep learning techniques can be found in [Mehta et al., 2019].

In the basic statistical learning framework, the learner has access to the following elements:

- **Domain set** $\mathcal{X}$
  The set of all points to make predictions about. They can be everything, from images in case of the famous MNIST dataset [Deng, 2012], to sequences of real features, such financial data or dynamical system as we are focusing on this thesis.

- **Labels set** $\mathcal{Y}$
  The set of all possible classes in which the domain dataset is divided. In case of unsupervised learning, notice that we do not have classes, but instead we may think to the domain set itself as the label set. In case of binary classification, we would have $\mathcal{Y} = \{0, 1\}$.

- **Training set**
  A finite sequence of labelled (in case of supervised learning) domain points in $\mathcal{X} \times \mathcal{Y}$ $S = ((x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m))$. Elements of the training set are often referred as *training samples*.

- **Prediction rule** $\hat{f} : \mathcal{X} \to \mathcal{Y}$
  It is the learner's output, also called hypothesis or classifier. We call $A(S)$ the hypothesis that a learning algorithm A returns when trained on a training dataset S.

- **Data generation model**
  We assume that instances are generate according to a probability distribution $\mathcal{D}$ over $\mathcal{X}$, which is not known by the learning algorithm. Labelled are supposed to be generated by a true function $f : \mathcal{X} \to \mathcal{Y}$, not known by the learning algorithm, such that $\forall x_i \in S$, $y_i = f(x_i)$.

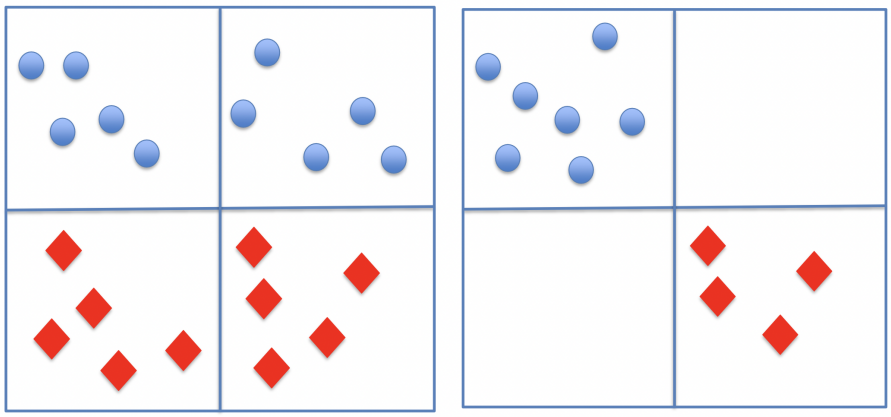
- **Measure of success**
  We define the prediction error of a classifier the probability that it does not predict the correct label over random generated points according to the above mentioned distribution:
  $$\mathcal{L}_{\mathcal{D},f}(\hat{f}) := \mathbb{P}_{x\sim\mathcal{D}}[\hat{f}(x) \neq f(x)] \,, \tag{3.1}$$
  which is often also called generalization error, true loss or true risk and depends on both the data distribution $\mathcal{D}$ and the labelling function f.

3.2.1 Empirical Risk Minimization

The goal of the learning algorithm is to output an algorithm $\hat{f}_S : \mathcal{X} \to \mathcal{Y}$ once it receives as input a training dataset S generated according to $\mathcal{D}$ and labelled with f. Since they are not directly available to the learner, the generalization error is not calculable. The fundamental question of Machine Learning is thus: how can a learning algorithm minimize the true error when it has the availability of only a small fraction of possible data in the training dataset, which. can be considered a window though the true world? We are in the so-called Empirical Risk Minimization (ERM) framework.
The answer to this question is not trivial. Since the learner has available only a snapshot of the real world, it is natural for it to minimize the *training error*, i.e. the error that the classifier encounters in the training dataset:

$$\mathcal{L}_S(\hat{f}) := \frac{|\{i : \hat{f}(x_i) \neq f(x_i),\ 1 \leq i \leq m\}}{m} = \frac{\#\text{ wrong predictions}}{\#\text{ training samples}} \,. \tag{3.2}$$

Even if Empirical Risk Minimization is a good strategy, yet it can lead to bad learning algorithm in real situation. It may happen that the training set is not representative of true data and therefore the algorithm could perform very well on the training dataset, but very bad on generalization data: this condition goes under the name of *overfitting*.
For example, let us suppose to have binary classify the dataset reported in Figure 3.1-(a), where instances are randomly generated on the square and labelled 0 if lie in the upper plane and 1 if lie below (red vs blue).



(a) (b)

Figure 3.1: Balanced dataset (a) vs. unbalanced dataset (b) that can lead to overfitting. Figure taken from the slides of the "Machine Learning" course of the master degree in "Physics of Data" given by Prof. Pietro Zanuttigh at University of Padua in the a.y. 2021/2022, `https://elearning.unipd.it/dfa/enrol/index.php?id=1106`.

Let us consider the predictor

$$\hat{f}_S(x) = \begin{cases} 0 \text{ if x in left side} \\ 1 \text{ if x in right side} \,, \end{cases} \tag{3.3}$$

applied to the training dataset in Figure 3.1-(b). In this case we would have $\mathcal{L}_S(\hat{f}) = 0$, meaning that the classification would be perfect on the training dataset, but the true error would be $\mathcal{L}_{\mathcal{D},f}(\hat{f}) = 1/2$, exactly the same as random guess!

This example show it is important to look for the condition under which the learning algorithm perform well on the training set, then it performs well also on the underlying data distribution. To do so, it is useful to restrict ERM over a limited search space, called *hypothesis class* $\mathcal{H}$. Each $\hat{f} \in \mathcal{H}$ is a function mapping $\mathcal{X}$ to $\mathcal{Y}$. The restriction on the hypothesis class is equivalent to make assumption (prior) on the problem at hand. For a given hypothesis class $\mathcal{H}$ and a training set S, the ERM procedure outputs a prediction $\hat{f}$ that has the lowest possible error over the training datasets, more formally:

$$ERM_{\mathcal{H}} \in \arg\min_{\hat{f}\in\mathcal{H}} \mathcal{L}_S(\hat{f}) \,, \tag{3.4}$$

since there can be multiple optimal solution, without loss of generality.

3.2.2 Probably Approximate Correct Learning

The question is then become which hypothesis class does not lead to overfitting. Let us suppose that the hypothesis class is finite, i.e. $|\mathcal{H}| < \infty$ and let $\hat{f}_S$ the output of $ERM_{\mathcal{H}}$, i.e. $\hat{f}_S \in \arg\min_{\hat{f}\in\mathcal{H}} \mathcal{L}_S(\hat{f})$.
Let us further assume the following two properties:

1. **Realizability**: there exist $\hat{f}^* \in \mathcal{H}$, such that $\mathcal{L}_{\mathcal{D},f}(\hat{f}^*) = 0$, i.e. there exist an optimal solution with zero generalization error.
2. **i.i.d.**: all samples in the training dataset are identically and independently distributed accorded to $\mathcal{D}$, i.e. $x_i \sim \mathcal{D}$, $\forall\, x_i \in S$.

With this two assumption we can only have predictors that are *approximately* correct with a certain *probability*, i.e. Probably Approximate Correct (PAC). The following theorem holds:

Theorem 4. ***(PAC learning)*** *Let $\mathcal{H}$ be a finite hypothesis class. Let $\delta \in (0,1)$, $\epsilon \in (0,1)$ and $m \in \mathbb{N}$ such that:*

$$m \geq \frac{\log\left(\frac{|\mathcal{H}|}{\delta}\right)}{\epsilon} \,. \tag{3.5}$$

Then, for any f and any $\mathcal{D}$ for which the realizability assumption holds, with probability $\geq 1-\delta$ we have that for every ERM hypothesis $\hat{f}_S$, computed on a training dataset S of m samples generated i.i.d. according to $\mathcal{D}$, we have:

$$\mathcal{L}_{\mathcal{D},f}(\hat{f}_S) \leq \epsilon \,. \tag{3.6}$$

The message of the theorem is clear. Issues arise only with critical datasets, for which $\mathcal{L}_S(\hat{f}) = 0$, but $\mathcal{L}_{\mathcal{D},f} > \epsilon$. Moreover, if $\mathcal{H}$ is a finite hypothesis class, ERM will not lead to overfitting, if the training dataset is sufficiently big. How big it should be depends on the grade of accuracy and with which probability we want to get that solution.
The definition of PAC learnability is then:

Definition 3. ***(PAC learnability)*** *An hypothesis class $\mathcal{H}$ is PAC learnable if there exists a function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm such that $\forall\, \delta, \epsilon \in (0,1)$, for every distribution $\mathcal{D}$ over $\mathcal{X}$ and for every labelling function $f : \mathcal{X} \to \mathcal{Y}$, if the realizability assumption holds with respect $\mathcal{D}, \mathcal{H}, f$ when running an algorithm on $m \geq m_{\mathcal{H}}(\epsilon, \delta)$ i.i.d samples generated with $\mathcal{D}$ and labelled with f, the algorithm returns an hypothesis $\hat{f}$ such that, with probability $\geq 1-\delta$ over the choice of m samples:*

$$\mathcal{L}_{\mathcal{D},f}(\hat{f}) \leq \epsilon \,. \tag{3.7}$$

From this definition and the previous Theorem 4, we have the following:

Corollary 1. *Every finite hypothesis class $\mathcal{H}$ is PAC learnable with sample complexity*

$$m_{\mathcal{H}}(\epsilon, \delta) \leq \left\lceil \frac{\log\left(\frac{|\mathcal{H}|}{\delta}\right)}{\epsilon} \right\rceil . \tag{3.8}$$

3.2.3 Agnostic PAC Learning

In practical situations, the realizability assumption, which states the existence, in the hypothesis class, of perfect learners, is too strong. Besides, it is not realistic to assume the existence of a true labelling function f and thus we assume that $\mathcal{D}$ is now a probability distribution over $\mathcal{X} \times \mathcal{Y}$, i.e. $z = (x, y) \sim \mathcal{D}$. The true risk for classification becomes

$$\mathcal{L}_{\mathcal{D}}(\hat{f}) := \mathbb{P}_{z\sim\mathcal{D}}[\hat{f}(x) \neq y] . \tag{3.9}$$

Since again the data generating distribution is not known to the algorithm, the predictor computes the training error over the training dataset as

$$\mathcal{L}(\hat{f}) = \frac{|\{i : \hat{f}(x_i) \neq y_i, 1 \leq i \leq m\}|}{m} . \tag{3.10}$$

The idea of Agnostic PAC learning is drop the requirement of having a perfect predictor, but not going too far from it:

Definition 4. ***(Agnostic PAC learning)*** *An hypothesis class $\mathcal{H}$ is Agnostic PAC learnable if there exists a function $m_{\mathcal{H}} : (0, 1)^2 \to \mathbb{N}$ and a learning algorithm such that $\forall\ \delta, \epsilon \in (0, 1)$ and for every distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, when running an algorithm on $m \geq m_{\mathcal{H}}(\epsilon, \delta)$ i.i.d samples generated with $\mathcal{D}$, the algorithm returns an hypothesis $\hat{f}$ such that, with probability $\geq 1 - \delta$ over the choice of m samples:*

$$\mathcal{L}_{\mathcal{D}}(\hat{f}) \leq \min_{\hat{f}'\in\mathcal{H}} \mathcal{L}_{\mathcal{D}}(\hat{f}') + \epsilon . \tag{3.11}$$

3.2.4 Generalized Loss Function

So far we have had in mind the case of binary or multi-class classification, defining the loss function as the fraction of misclassified samples. However, according to the problem at hand different loss function are possible. Given an hypothesis class $\mathcal{H}$ and a domain $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, a (generalised) loss function is any function $l : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}^+$.
The true risk is defined as the expectation value of the loss of an hypothesis $\mathcal{H}$ with respect a distribution $\mathcal{D}$ over $\mathcal{Z}$:

$$\mathcal{L}_{\mathcal{D}}(\hat{f}) := \mathbb{E}_{z\sim\mathcal{D}}[l(\hat{f}, z)] . \tag{3.12}$$

The predictor is trained to minimize the error over the training dataset $S = (z_1, z_2, \ldots, z_m)$:

$$\mathcal{L}_S(\hat{f}) := \frac{1}{m} \sum_{i=1}^{m} l(\hat{f}, z_i) . \tag{3.13}$$

There are several types of labelling functions and the choice depends on the kind of problem taken at hand. Sometimes, multiple choices are possible. For multi-class classification, the natural choice is to use cross entropy loss, even if L2 loss is also possible. In case of sequences, we make use of L2 loss. The most used loss functions are:

- ◇ **0-1 loss**
  Commonly used in binary or multi-class classification, it is defined as

$$l_{0-1}(\hat{f}, (x, y)) = \begin{cases} 0 \; if \; \hat{f}(x) = y \\ 1 \; if \; \hat{f}(x) \neq y . \end{cases} \tag{3.14}$$

◇ **Cross entropy loss**
Also used in classification. Loss function reads

$$l_{CE}(\hat{f}, (x, y)) := -\log \hat{f}(x) \ , \tag{3.15}$$

so that the predictor is trained to output a probability distribution condition to the data to have certain values. It is easy (and we do not report it) to prove that Cross Entropy minimization is equivalent to Maximum Likelihood Estimation (MLE), see [Bengio et al., 2015].

◇ **L2 loss**
It can be used for regression, but it is generally used for regression and other tasks. It is defined as:

$$l_{L2}(\hat{f}, (x, y)) := |\hat{f}(x) - y|^2 \ . \tag{3.16}$$

◇ **L1 loss**
Used in regression, it penalized small errors:

$$l_{L1}(\hat{f}, (x, y) := |\hat{f}(x) - y| \ . \tag{3.17}$$

There are of course many more, see [Wang et al., 2022] for a systematic comparison.
With a generalized loss function. the definition of Agnostic PAC learnability becomes:

Definition 5. *(**General Agnostic PAC learning**) An hypothesis class $\mathcal{H}$ is Agnostic PAC learnable respect to a set $\mathcal{Z}$ and a loss function $l : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}^+$, if there exists a function $m_{\mathcal{H}} : (0, 1)^2 \to \mathbb{N}$ and a learning algorithm such that $\forall\ \delta, \epsilon \in (0, 1)$ and for every distribution $\mathcal{D}$ over $\mathcal{Z}$, when running an algorithm on $m \geq m_{\mathcal{H}}(\epsilon, \delta)$ i.i.d samples generated with $\mathcal{D}$, the algorithm returns an hypothesis $\hat{f}$ such that, with probability $\geq 1 - \delta$ over the choice of m samples:*

$$\mathcal{L}_{\mathcal{D}}(\hat{f}) \leq \min_{\hat{f}' \in \mathcal{H}} \mathcal{L}_{\mathcal{D}}(\hat{f}') + \epsilon \ , \tag{3.18}$$

where $\mathcal{L}_{\mathcal{D}}(\hat{f}) = \mathbb{E}_{z \sim \mathcal{D}}[l(\hat{f}, z)]$.

3.3 Learning from Uniform Convergence

We have seen that a learning algorithm receives as input a training set S, the learning algorithm evaluate the error of each element of the hypothesis class $\mathcal{H}$ and selects the one with the lowest possible empirical error. The question was under which conditions to asses if the predictor with the best empirical error minimized also the true error. A sufficient conditions would be to require that all members of $\mathcal{H}$ give a good approximation of the true risk. It is useful to introduce the concept of *ϵ-representativeness.*

Definition 6. *A training set is called ϵ-representative with respect a domain $\mathcal{Z}$, a hypothesis class $\mathcal{H}$, a loss function l and a distribution $\mathcal{D}$ if*

$$\forall\ \hat{f} \in \mathcal{H} : \ |\mathcal{L}_{\mathcal{D}}(\hat{f}) - \mathcal{L}_S(\hat{f})| \leq \epsilon \ . \tag{3.19}$$

For $\epsilon/2$-representative sets we have the following:

Theorem 5. *Let S be a $\epsilon/2$-representative set with respect a domain $\mathcal{Z}$, a hypothesis class $\mathcal{H}$, a loss function l and a distribution $\mathcal{D}$. Then, any output of $ERM_{\mathcal{H}}(S)$, i.e. any $\hat{f}_S \in \arg\min_{\hat{f} \in \mathcal{H}} \mathcal{L}_S(\hat{f})$, satisfies:*

$$\mathcal{L}_{\mathcal{D}}(\hat{f}_S) \leq \min_{\hat{f} \in \mathcal{H}} \mathcal{L}_{\mathcal{D}}(\hat{f}) + \epsilon \ . \tag{3.20}$$

We are now in position to state the definition of *Uniform Convergence*.

Definition 7. ***(Uniform Convergence)*** *An hypothesis class $\mathcal{H}$ has the Uniform Convergence property with respect to a set $\mathcal{Z}$ and a loss function $l : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}^+$, if there exists a function $m^{UC}_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ such that $\forall\, \delta, \epsilon \in (0,1)$ and for every distribution $\mathcal{D}$ over $\mathcal{Z}$, if S is a set of $m \geq m^{UC}_{\mathcal{H}}(\epsilon, \delta)$ i.i.d samples generated from $\mathcal{D}$, then with probability $\geq 1 - \delta$, S is ϵ-representative.*

The connection of Uniform Convergence with PAC learning is expressed by the following

Theorem 6. *If a class $\mathcal{H}$ has the Uniform Convergence property with a function $m^{UC}_{\mathcal{H}}$, then:*

1. *The class is Agnostic PAC learnable with sample complexity $m_{\mathcal{H}}(\epsilon, \delta) \leq m^{UC}_{\mathcal{H}}(\epsilon/2, \delta)$.*
2. *The $ERM_{\mathcal{H}}$ paradigm is a successful Agnostic PAC learner for $\mathcal{H}$.*

In case of *finite* hypothesis classes, we can sate the following

Proposition 1. *Let $\mathcal{H}$ be a finite hypothesis class, $\mathcal{Z}$ a domain set and $l : \mathcal{H} \times \mathcal{Z} \to [0,1]$ be a loss function. Then:*

- *$\mathcal{H}$ has the Uniform Convergence property with sample complexity*

$$m^{UC}_{\mathcal{H}}(\epsilon, \delta) \leq \left\lceil \frac{\log\left(\frac{2|\mathcal{H}|}{\delta}\right)}{2\epsilon^2} \right\rceil . \tag{3.21}$$

- *$\mathcal{H}$ is agnostic PAC learnable using ERM algorithm with sample complexity*

$$m_{\mathcal{H}}(\epsilon, \delta) \leq m^{UC}_{\mathcal{H}}(\epsilon/2, \delta) \leq \left\lceil \frac{2\log\left(\frac{2|\mathcal{H}|}{\delta}\right)}{\epsilon^2} \right\rceil . \tag{3.22}$$

3.4 Bias-Complexity Trade-Off

The learning problem was formulated to find a function $\hat{f}$, given a training set S and a loss function l, for which the generalization error $\mathcal{L}_{\mathcal{D}}(\hat{f})$ is small. In general, a training algorithm A that given S produces a good predictor $\hat{f}$ need two components:

1. A hypothesis class $\mathcal{H}$.
2. A procedure to pick $\hat{f}$ from $\mathcal{H}$.

Given that, we might ask if there exists an *universal learner*, i.e. an algorithm A that predicts the best output $\hat{f}$ for every distribution $\mathcal{D}$. Moreover, what happens if for hypothesis class we pick all the function from $\mathcal{X}$ to $\mathcal{Y}$?

The two questions are answered by the following theorem and its corollary in case of binary classification.

Theorem 7. ***(No-Free Lunch)*** *Let A any algorithm for the task of binary classification with respect to the 0-1 loss over the domain $\mathcal{X}$. Let $m < \frac{|\mathcal{X}|}{2}$, representing a training set S. Then, there exists a distribution $\mathcal{D}$ over $\mathcal{X} \times \{0,1\}$ such that:*

1. *There exists a function $f : \mathcal{X} \to \{0,1\}$, such that $\mathcal{L}_{\mathcal{D}}(f) = 0$.*
2. *With probability $\geq 1/7$ over the choice of S we have*

$$\mathcal{L}_{\mathcal{D}}(A(S)) \geq \frac{1}{8} . \tag{3.23}$$

This means that, roughly speaking, for every ML algorithm there exists a task in which it fails, even if there is another algorithm that solves it. The key idea, similarly to what seen in the example of Figure 3.1, is that we do not have information on half of domain, so that there exists a function that works well on the other half and contradicts our estimated labels.
In case of infinite domain set, we have the following:

Corollary 2. *(**No-Free Lunch**) Let $\mathcal{X}$ an infinite domain set and let $\mathcal{H}$ the set of all functions from $\mathcal{X}$ to $\{0,1\}$. Then, $\mathcal{H}$ is not PAC learnable.*

The demonstration idea is that since $\mathcal{X}$ is infinite, $|\mathcal{X}| > 2m$, $\forall m$.

Therefore, even if in principle the hypothesis class $\mathcal{H}$ should be infinite, it is not a good idea to pick a too big one, because of the risk of overfitting. Let us think at the case of polynomial regression, in which data are generated with a polynomial of degree two with some noise and we aim to learn that function by putting a polynomial prior on it, see Figure 3.2. Our hypothesis class correspond to the degree of polynomial used. If we allow it to be to large, i.e. we pick infinitely large polynomial degree, we will end up for sure with overfitting. That is because given $n+1$ points, there a polynomial of grade n fitting them. In then end, by choosing the hypothesis class too large, we will fit the noise, which the worst case scenario, even when the true model is simply quadratic.
On the other hand, picking a hypothesis class too small will lead to a poor approximation probabilities. This problem is expressed as *bias-complexity trade-off*. $\mathcal{H}$ should be chosen in between to have good approximation ($\mathcal{L}_S$ small) and generalization ($\mathcal{L}_{\mathcal{D}}$ small) capabilities.

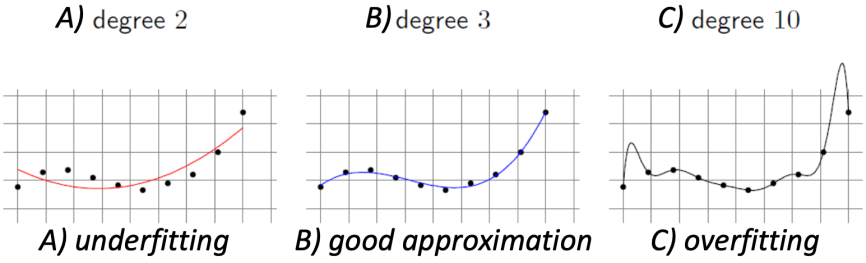


Figure 3.2: The typical scenario encountered in polynomial regression. By allowing $\mathcal{H}$ to be too large, we seriously risk to learn noise. Figure taken from the slides of the "Machine Learning" course of the master degree in "Physics of Data" given by Prof. Pietro Zanuttigh at University of Padua in the a.y. 2021/2022, `https://elearning.unipd.it/dfa/enrol/index.php?id=1106`.

Given an $ERM_{\mathcal{H}}$ hypothesis $\hat{f}_S$, the generalization error, our unknown minimization target, can be decomposed as

$$\mathcal{L}_{\mathcal{D}}(\hat{f}_S) = \epsilon_{appr} + \epsilon_{est} \ , \tag{3.24}$$

where:

- $\epsilon_{appr} = \min_{\hat{f}\in\mathcal{H}} \mathcal{L}(\hat{f})$ is the *approximation error*, which is the minimum true risk reachable by the predictor in $\mathcal{H}$. It thus depends on $\mathcal{H}$, but not on the training algorithm S and it is zero when the realizability assumption holds.

- $\epsilon_{est} = \mathcal{L}_{\mathcal{D}}(\hat{f}_S) - \min_{\hat{f}\in\mathcal{H}} \mathcal{L}(\hat{f})$ is the *estimation error*, i.e. the difference between the true error of ERM and the best error achievable in $\mathcal{H}$. It depends in S and decreases when the hypothesis class is small.

A visualization of bias-complexity trade-off can be found in Figure 3.3.

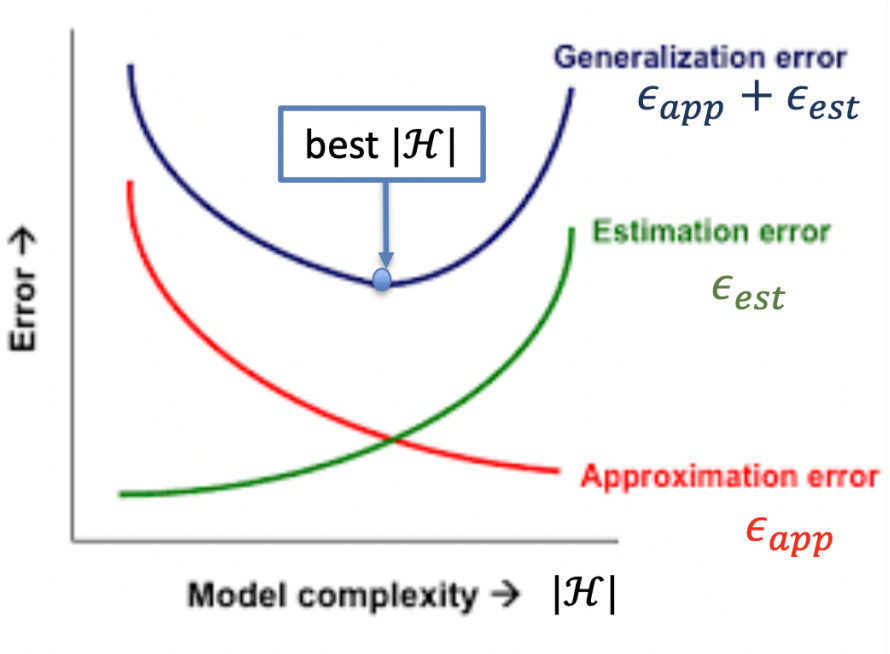


Figure 3.3: Illustration of bias-complexity trade-off. The best complexity of the hypothesis class $\mathcal{H}$ should be chosen in between to have a good approximation error (small inductive bias) and a still small estimation error (low complexity). Figure taken from the slides of the "Machine Learning" course of the master degree in "Physics of Data" given by Prof. Pietro Zanuttigh at University of Padua in the a.y. 2021/2022, `https://elearning.unipd.it/dfa/enrol/index.php?id=1106`.

3.5 Model Selection and Validation

To give a more reliable estimate of the generalization error, it is often used a *test dataset*, which must be different from the training dataset. However, we must be careful at not to look at the final estimate until the final hypothesis is picked. In practical situations, the training dataset is further divided into a training set and a validation set and the latter is then used either to pick the best hyper-parameters of an algorithm, or to evaluate the error during training. Hyper-parameter of the model are those which are not directly trained, but anyway influence the finale performance. Examples of hyper-parameters are the polynomial degree in case of polynomial regression, the umber of layers in a Neural Network (on which we will come back later) and the learning rate of Gradient Descent.
With validation set, the training procedure is then:

1. Select Hyper-parameters values.
2. Train the model on the training dataset.
3. Evaluate the performance on the validation dataset.
4. Repeat from step (1).
5. Pick hyper-parameters combination with the smallest validation error.
6. Evaluate the generalization error of the chosen configuration on the test dataset.

With validation set, the following theorem holds.

Theorem 8. *Let V be a validation dataset with m_V samples and let $\hat{f}$ be a predictor, for example chosen with ERM rule on a hypothesis class $\mathcal{H}$. Then $\forall \delta \in (0,1)$, with probability $\geq 1-\delta$ over the choice of V, the validation loss $\mathcal{L}_V(\hat{f})$ satisfies:*

$$|\mathcal{L}_{\mathcal{D}}(\hat{f}) - \mathcal{L}_V(\hat{f})| \leq \sqrt{\frac{\log\left(\frac{2}{\delta}\right)}{2m_V}} \,. \tag{3.25}$$

The use of a validation set on different combinations of hyper-parameters gives us a general procedure for model selection in case of different candidates in the hypothesis class, that are in a certain way "augmented" by considering different set of hyper-parameters for each one of them:

1. For each hyper-parameter set or algorithm, there is a different hypothesis class
$$\mathcal{H}_i = \{\hat{f}_{i1}, \hat{f}_{i2}, \dots, \hat{f}_{iI}\},\ I = |\mathcal{H}_i| \ . \tag{3.26}$$
2. Train the algorithm in each hypothesis class independently and call $\hat{f}_i^{ERM}$ the optimal predictor found.
3. Collect the solutions $\hat{f}_i^{ERM}$ on a new super hypothesis class
$$\mathcal{H}' = \{\hat{f}_1^{ERM}, \hat{f}_2^{ERM}, \dots, \hat{f}_r^{ERM}\} \ . \tag{3.27}$$
4. Pick the best predictor in $\mathcal{H}'$ as the one $\hat{f}^*$ which minimizes the validaition error.

The use of an output predictor set $\mathcal{H}'$ is similar to a case in which the hypothesis class is not fixed ahead, but depends on the training set. In this case we have:

Proposition 2. *Let $\mathcal{H}' = \{\hat{f}_1^{ERM}, \hat{f}_2^{ERM}, \dots, \hat{f}_r^{ERM}\}$ be an arbitrary set of predictors with loss in $[0, 1]$. Assume that the validation dataset V, of size m_V is sampled independently of the training set. Then, with probability $\geq 1 - \delta$ over the choice of V, we have*
$$|\mathcal{L}_{\mathcal{D}}(\hat{f}) - \mathcal{L}_V(\hat{f})| \leq \sqrt{\frac{\log\left(\frac{2|\mathcal{H}'|}{\delta}\right)}{2m_V}} \ , \qquad \forall \hat{f} \in \mathcal{H}' \ . \tag{3.28}$$

The estimate on the true risk is given by the error on the test set, which must not be seen by the training procedure and so it is no involved in the best predictor $\hat{f}^*$. The guarantees thus come from the proposition on $\mathcal{L}_{\mathcal{D}}(\hat{f}^*)$ for one class.

K-fold Cross Validation

In case we have a small amount of data, we cannot afford to drop a part of the training dataset creating a validation set. Therefore, it is useful to perform K-fold Cross Validation [Stone, 1974, Mosteller and Tukey, 1968]. The idea is to divide the training dataset into k folds, and at each iteration per discard the first fold and perform the training on the remaining $k - 1$, using the excluded one as validation set. Then, we discard the second, we perform the training on the remaining ones and we estimate the error on the selected fold. We repeat the procedure for all the folds. Finally, the estimate error is the average of the errors evaluated on each fold. A graphical explanation is given in Figure 3.4.

In this thesis, we will use Cross Validation to perform Hyper-parameter selection using the Bayesan optimization library [Akiba et al., 2019]. Once the hyper-parameters are chosen, training will be performed on large datasets using an independent validation dataset consisting of 20 % of the training samples, which it will be used as an estimation of the true error. Training is performed either by looking visually to the convergence to a plateau, or by using Early Stopping [Zhang and Yu, 2005], which consists in stopping training when the validation error does not improve for a a consecutive number of epochs called *patience*.

3.6 Regularization

Often Machine Learning models tend to overfit data because are too complex. Let our hypothesis $\hat{f}$ be characterized by a parameter vector $\boldsymbol{\theta} \in \mathbb{R}^d$. In case of neural networks, the parameters vector includes all weights and all biases of all layers; it is just a short hand

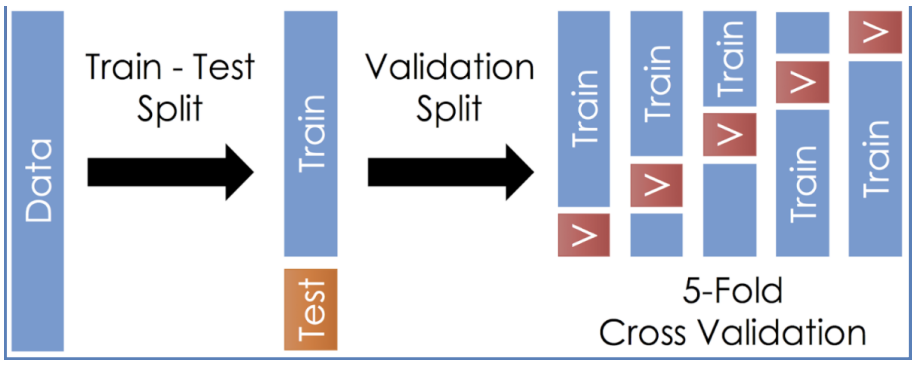


Figure 3.4: Splitting of the training dataset into k-fold to perform Cross Validation in case of rare data. Figure taken from the slides of the "Machine Learning" course of the master degree in "Physics of Data" given by Prof. Pietro Zanuttigh at University of Padua in the a.y. 2021/2022, `https://elearning.unipd.it/dfa/enrol/index.php?id=1106`.

notation to easily refer to all of them.
The idea of *regularization* is to jointly minimize the training error with a function $R : \mathbb{R}^d \to \mathbb{R}$ of the parameters, which measures in some sense the "complexity" of the model. Therefore, the regularised ML problem becomes

$$\arg\min_{\boldsymbol{\theta}} \big(\mathcal{L}_S(\hat{f}) + R(\boldsymbol{\theta})\big) \ . \tag{3.29}$$

One of the most used regularization techniques in Deep Learning is $L2$ regularization [Nowlan and Hinton, 1992, Krogh and Hertz, 1991], also known as Tikhonov regularization, where

$$R(\boldsymbol{\theta}) = \lambda ||\boldsymbol{\theta}||^2 = \lambda \sum_{i=1}^{d} \theta_i^2 \ , \tag{3.30}$$

where λ is a parameter controlling the strength of regularization. In Neural Networks, L2 regularization will push weights towards zero, improving stability and statistical efficiency, being multiplied at each iteration of Stochastic gradient Descent by $1 - \lambda$. That is the reason why it is also known as *weight decay*. Another used regularization technique is L1 [Nowlan and Hinton, 1992], where

$$R(\boldsymbol{\theta}) = \lambda \sum_{i=1}^{d} |\theta_i| \ , \tag{3.31}$$

which promotes sparsity instead.
Nowadays, the most used form of regularization of deep networks, among L2, is *dropout* [Srivastava et al., 2014], which consists in randomly drop (set to zero) some neurons according to a probability p, in order to avoid that they "co-adapt" much and that the output is too dependent on some neuron in particular. Since dropout removes randomly neurons during training, it practically behaves as L2 regularization because it is all averaged and it is preferred over it in modern applications. Another regularization used, which reduces the need for dropout, is Batch Normalization [Ioffe and Szegedy, 2015], that consists in normalizing batches at each layer and then perform a (learnable) affine transformation in order to fasten convergence and improve stability.

The importance of weight decay regularization lies in the fact that it makes the algorithm *stable* with respect small changes on the training data. In this section we focus on the *On-Average-Replace-One-Stable* (OAROS) algorithms, based on the replacement of one element z_i with z' in the training datasets, i.e. the new dataset is$S^{(i)} = \{z_1, \dots, z_{i-1}, z', z_{i+1}, \dots, z_m\}$.

Definition 8. ***(OAROS)*** *Let $\epsilon : \mathbb{N} \to \mathbb{R}$ a monotonically decreasing function. We say that a learning algorithm A is On-Average-One-Replace-Stable with rate $\epsilon(m)$ if for every distribution $\mathcal{D}$, we have*

$$\mathbb{E}_{(S,z')\sim\mathcal{D}^{m+1},i\sim U(m)}\left[l(A(S^{(i)}),z_i) - l(A(S),z_i)\right] \leq \epsilon(m) \ , \tag{3.32}$$

where $U(m)$ is the Uniform distribution over m integers elements.
OAROS stable algorithms do not overfit, due to the following:

Theorem 9. *If an algorithm A is OAROS with rate $\epsilon(m)$, then:*

$$\mathbb{E}_{S\sim\mathcal{D}^m}\left[\mathcal{L}_{\mathcal{D}}(A(S)) - \mathcal{L}_S(A(S))\right] \leq \epsilon(m) \ . \tag{3.33}$$

It can be proved that Tikhonov regularization is a stabiliser for ρ-Lipschitz continuous loss function.

Definition 9. ***(Lipschitzness)*** *Let $C \in \mathbb{R}^d$. A function $f : \mathbb{R}^d \to \mathbb{R}^k$ is said to be ρ-Lipschitz over C if $\forall \boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in C$, we have*

$$||f(\boldsymbol{\theta}_1) - f(\boldsymbol{\theta}_2)|| \leq \rho||\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2|| \ . \tag{3.34}$$

Of course, if f is differentiable and its derivative is bounded by ρ in C, then it is ρ-Lipschity continuous.

It follows the followeing theorem for convex and ρ-Lipschitz continuous loss functions.

Theorem 10. *Assume that the loss function is convex and ρ-Lipschitz continuous. Then, the regularized ML rule with regularizer $\lambda||\boldsymbol{\theta}||^2$ is OAROS with rate $\frac{2\rho}{\lambda m}$. It follows from Theorem 9 that*

$$\mathbb{E}_{S\sim\mathcal{D}^m}\left[\mathcal{L}_{\mathcal{D}}(A(S)) - \mathcal{L}_S(A(S))\right] \leq \frac{2\rho}{\lambda m} \ . \tag{3.35}$$

The consequences are intuitive. On the one hand, the size of the training dataset acts as regularizer. On the other, also the strength of Tikhonov regularization will lead to not overfit data.

To see how regularization influence the learning algorithm, let us decompose the true error into two parts, as:

$$\mathcal{L}_{\mathcal{D}}(A(S)) = \mathcal{L}_s(A(S)) + \left[\mathcal{L}_{\mathcal{D}}(A(S)) - \mathcal{L}_S(A(S))\right] \ , \tag{3.36}$$

where $\mathcal{L}_S$ is the training error and the difference $\mathcal{L}_{\mathcal{D}} - \mathcal{L}_S$ in square parenthesis controls measures overfitting, see Figure 3.5.

3.7 Neural Networks

Even if there are several Machine Learning models, spacing from Support Vector Machines (SVM) [Cortes and Vapnik, 1995] to Random Forests [Breiman, 2001], in this thesis we focus on deep Neural Network models only, which have become the predominant and most powerful Machine Learning tools in the last ten years.
Neural Networks have a history much more ancient than people usually think, being the first model ever published by [Rosenblatt, 1958]. *Perceptron* neuron loosely resemble a brain neuron and it was designed to find separating hyperplanes in linearly separable data. In our brain dendrites bring different output to the neuron body, called *soma*, which integrates them to produce an output. If the *activation* of the neuron is greater than a certain threshold, the potential action start and the neuron sends its output thorough the *axon*. Perceptron is the correspondent artificial of a neuron. It receives some inputs, it

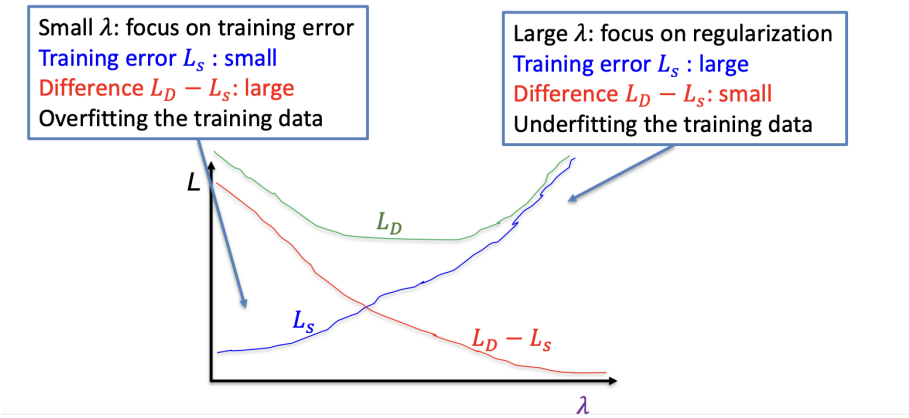


Figure 3.5: The qualitative impact of the regularization constant λ on training curves. Figure taken from the slides of the "Machine Learning" course of the master degree in "Physics of Data" given by Prof. Pietro Zanuttigh at University of Padua in the a.y. 2021/2022, `https://elearning.unipd.it/dfa/enrol/index.php?id=1106`.

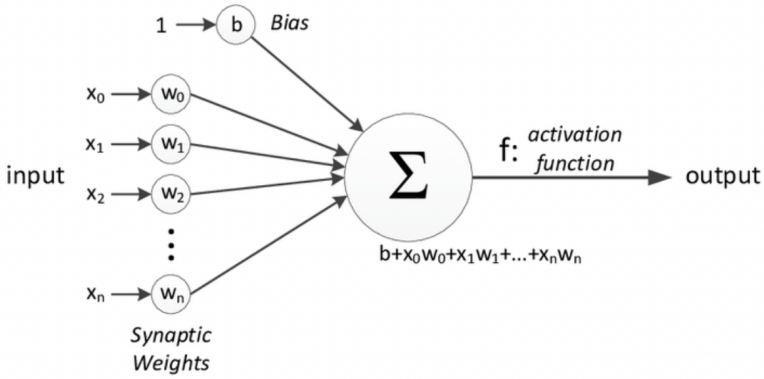


Figure 3.6: Schematic view of Perceptron neuron.

sums then together, possibly adding a bias, and then it returns an output after passing the post-sinaptic signal to an activation function, Figure 3.6.

Perceptron offers good performance on simple datasets, while to deal with more complex data we had to wait the second Renaissance of Deep Learning in the early 2010s. The recent improvement Deep Learning models, which are essentially Neural Networks with many layers, has been driven by the availability of large datasets, for which is more difficult to encounter overfitting issues, and the improvement of computational power. We will now present the basic theory regarding neural networks, in particular fully connected ones, along with the basic function of Stochastic Gradient Descent (SGD) [Robbins and Monro, 1951] and Backpropagation (BackProp) [Rumelhart et al., 1986] to perform training error minimization. In the next sections, we will take into consideration more advanced models which will be used later on in this thesis, but for now we focus on standard FeedForward networks.

FeedForward Neural Networks (FFNN) are essentially built putting together many Perceptron neurons to build a layer and many layers one after another. For that reason, they are also known as Multi Layer Perceptrons (MPL). The first applications date back to 1990s, but it was only in the last ten years that the started giving impressive performances. A historical review can be found in [Schmidhuber, 2015].
A FFNN is defined by means of graph $G = (V, E)$ and a function $\mathbf{w} : E \to \mathbb{R}$, where V are the neurons of the network, E the edges and $\mathbf{w}$ the function that maps the edges to weights

in $\mathbb{R}$. The neurons can be seen as the union of $R+1$ disjoint layers as $V = \bigcup_{r=0}^{R} V_r$, where V_r are the $d_r + 1$ neurons in the r-th layer [1]. The input layer is V_0, while the output is V_R. We indicate the i-th neuron in the r-th layer as $v_i^{(r)}$ and therefore in compact homogeneous notation $\mathbf{v}^{(r)} = \{1, v_1^{(r)}, v_2^{(r)}, \dots, v_{d_t}^{(r)}\}$. The activation function of layer r is denoted with $\sigma^{(r)}$.

There exists several possible activation functions and the use of one with respect to the other could depend on the problem at hand. Systematic reviews on the possible activation functions are [Lederer, 2021] and [Szandała, 2021]. The most common ones are reported in Figure 3.7.

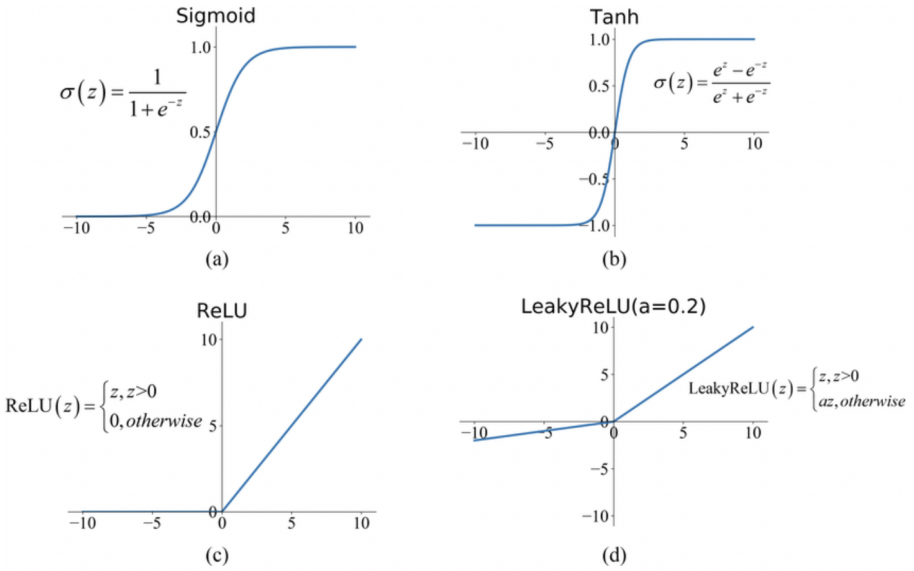


Figure 3.7: The most used activation functions with graphical visualization.

Another important activation function, which seems to have been introduced in ML by [Bridle, 1989], is the Softmax:

$$\mathrm{Softmax}(a_j) = \frac{e^{a_j}}{\sum_i e^{a_i}} , \tag{3.37}$$

which is often used in classification problems in the output layer to convert linear activation in probabilities to learn conditional distributions by mean of Cross Entropy Loss.

The output of a FFNN is computed recursively starting from the first layer by first multiplying each layer r by the weight matrix $\mathbf{w}^{(r+1)}$ (so that it links layer r with layer $r+1$) whose rows are

$$\mathbf{w}_j^{(r+1)} = \left(w_{0,j}^{(r+1)}, w_{1,j}^{(r+1)}, \dots, w_{d_r-1,j}^{(r+1)}\right)^\intercal , \qquad j = 0, 1, \dots, d_r . \tag{3.38}$$

This produces the *activation* of the next layer $a_j^{(r+1)} = \mathbf{w}_j^{(r+1)} \cdot \mathbf{v}^{(r)}$. The activation is then passe through the activation function to computed the neurons in the next layer as

$$v_j^{(r+1)} = \sigma^{(r+1)}\left(a_j^{(r+1)}\right) . \tag{3.39}$$

A typical FFNN is reported in Figure 3.8.

From now on we indicate all the weights and biases of a network with $\boldsymbol{\theta}$ and therefore the Machine Learning ansatz predictor will be denoted with $\hat{f}_{\boldsymbol{\theta}}$. In case of Neural Network,

[1] We incorporate the bias into a constant neuron to simply the notation, also known as homogeneous notation.

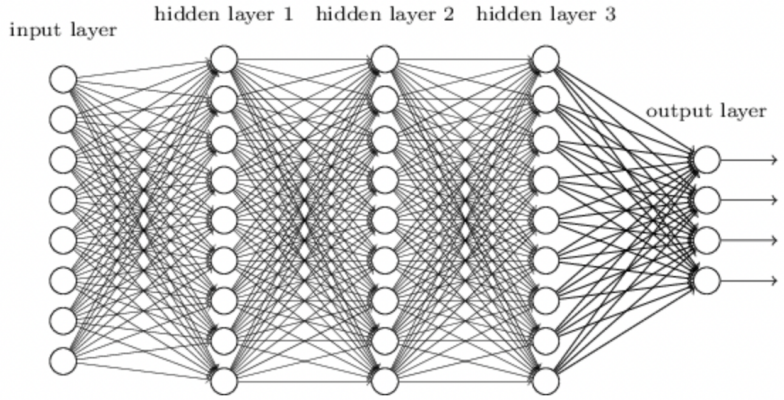


Figure 3.8: Typical FFNN with three hidden layers.

the hypothesis class $\mathcal{H}_{G,\sigma}$ becomes its architecture (comprehensive of weights, biases and activation functions). The forward pass pseudo Algorithm is 1.

Algorithm 1 Forward pass of a standard FFNN.

Input: $\mathbf{x} = (x_1, x_2, \dots, x_d)^\intercal$
Output: prediction $\hat{f}_{\boldsymbol{\theta}}$

$\mathbf{v}^{(0)} = (1, x_1, \dots, x_d)^\intercal$

for r=1 **to** R **do**
  $\mathbf{a}^{(r)} = (\mathbf{w}^{(r)})^\intercal \mathbf{v}^{(r-1)}$

  $\mathbf{v}^{(r)} = (1, \sigma^{(r+1)}(\mathbf{a}^{(r)}))^\intercal$
end for

return $\mathbf{v}^{(R)}$

The power of FF Neural Networks, due to strong approximation theorems [Kolmogorov, 1961, Hornik, 1991]. In case of Lipschitz functions, an interesting result is due to [Cybenko, 1989] in case of sigmoid activation function.

Proposition 3. *For every fixed $\epsilon > 0$ and every Lipschitz function $f : [-1,1]^d \to [-1,1]$ it is possible to construct a (FF) Neural Network such that for every input $\mathbf{x} \in [-1,1]^d$, the output of the Neural Network satisfies:*

$$\hat{f}_\theta(\mathbf{x}) \in [f(\mathbf{x}) - \epsilon, f(\mathbf{x}) + \epsilon] \,. \tag{3.40}$$

This beautiful result however comes to a cost on the number of neurons needed, namely:

Proposition 4. *Let $\epsilon \in (0,1)$ fixed. For every d, let $s(d)$ be the minimum integer such that there exists a graph $G = (E,V)$ with $|V| = s(d)$, such that $\mathcal{H}_{G,\sigma}$, with $\sigma = \text{sigmoid}$, can approximate with precision ϵ every 1-Lipschitz function $f : [-1,1]^d \to [-1,1]$. Then, $s(d)$ grows exponentially with d.*

3.8 Stochastic Gradient Descent and Backpropagation

The aim of learning for a Neural Network is to minimize a cost function, i.e. the training loss $\mathcal{L}_S(\hat{f}_{\boldsymbol{\theta}})$ with respect to the parameters $\boldsymbol{\theta}$ of the model. In this sense, ML with Neural

Network can be seen as a parameter regression over a very complex parameter space, which can be rich of local minima and saddle points and almost never a convex optimization problem.
Gradient Descent (GD) and its variants are the most important optimization algorithm in modern Deep Learning, performing surprisingly well in many different tasks. GD is not the only possibility, though. For example, Boltzmann Machines [Hinton and Ackley, 1985], a popular unsupervised model often used in Recommendation Systems [Salakhutdinov et al., 2007], can be trained with GD, but the best performance are obtained with Contrastive Divergence [Hinton, 2002]. A review on optimization techniques in ML can be found in [Bottou et al., 2018].
The idea of GD is to update the weights along the direction of maximum variation given by the gradient of the loss function with respect to the weights, namely Algorithm 2.

Algorithm 2 Gradient Descent

Initialize $\boldsymbol{\theta}^{(0)}$
for t=0 **to** T-1 **do**
$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \frac{\partial \mathcal{L}_S(\hat{f}_{\boldsymbol{\theta}^{(t)}})}{\partial \boldsymbol{\theta}^{(t)}}$,
end for

where η is a hyper-parameter controlling the convergence velocity called *learning rate*. The algorithm is not stopped until convergence. The parameter initialization is usually performed with small random number, to improve convergence and break the symmetry of weights. There are several more sophisticated schemes, such as Xavier (Glorot) [Glorot and Bengio, 2010] or He [He et al., 2015] initialization methods. A review on various techniques can be found in [Narkhede et al., 2022].
Computing the entire gradient, i.e. over the entire training dataset at each iteration, is computationally demanding and therefore it is preferred to perform *Stochastic Gradient Descent* (SGD), whose godfathers can be considered [Robbins and Monro, 1951]. Instead to compute the gradient along the entire dataset, we randomly sample an element $\mathbf{v}S$ at iteration such that its expectation value given the actual weights is the gradients. More precisely Algorithm 3.

Algorithm 3 Stochastic Gradient Descent

Initialize $\boldsymbol{\theta}^{(0)}$
for t=0 **to** T-1 **do**
Choose randomly a vector $\mathbf{v}^{(t)}$, such that $\mathbb{E}(\mathbf{v}^{(t)}|\boldsymbol{\theta}^{(t)}) = \frac{\partial \mathcal{L}_S(\hat{f}_{\boldsymbol{\theta}^{(t)}})}{\partial \boldsymbol{\theta}^{(t)}}$
$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \mathbf{v}^{(t)}$,
end for

A choice for $\mathbf{v}^{(t)}$ that satisfies the requirement is

$$\mathbf{v}^{(t)} = \frac{\partial l(\hat{f}_{\boldsymbol{\theta}^{(t)}}, z_i)}{\partial \boldsymbol{\theta}^{(t)}} \ , \tag{3.41}$$

where $z_i = (x_i, y_i)$ is uniformly sampled from the training dataset. Its consistency relies on the fact that the gradient of the loss function $l(\boldsymbol{\theta}^{(t)}, z)$ is an unbiased estimate of the gradient of the true risk:

$$\mathbb{E}[\mathbf{v}^{(t)}|\boldsymbol{\theta}^{(t)}] = \mathbb{E}_{z\sim\mathcal{D}}[\nabla l(\boldsymbol{\theta}^{(t)}, z)] = \nabla \mathbb{E}_{z\sim\mathcal{D}}[l(\boldsymbol{\theta}^{(t)}, z)] = \nabla \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}^{(t)}) \ . \tag{3.42}$$

The advantage of SGD compared to normal GD is certainly the speed. However, it is more noisy at can lead to local minima, since gradients are computed considering one sample at a time. Therefore, it is often preferred *Mini-Batch Gradient Descent*, where the gradient is average over a set of B samples, called *batch size*, of the training dataset, being thus half

way between GD ($B = m$) and SGD ($B = 1$).
There several ways to improve stability and speed convergence up. One way could be to design adapting learning rate schedulers, in which learning rate changes according to a determined schedule, for example $\eta_t = \frac{1}{\lambda t}$. See for example [Xu et al., 2019], where they proposed a Reinforcement Learning strategy.
The most used approach nowadays is to make use of *momentum*, see Figure 3.9. Let us think at our optimization problem as a ball that runs down the hill made by training loss. As the ball moves down, it accumulates speed, converging faster towards the minimum. So, SGD with momentum instead of updating the weights according to the current gradient, it calculates an average of previous gradients, weighting more the most recent ones, through *moving averages*. Gradient is updated as:

$$\begin{aligned} \mathbf{v}^{(t)} &= \gamma \mathbf{v}^{(t-1)} + (1-\gamma)\frac{\partial \mathcal{L}_S(\hat{f}_{\boldsymbol{\theta}^{(t)}})}{\partial \boldsymbol{\theta}^{(t)}} , \\ \boldsymbol{\theta}^{(t+1)} &= \boldsymbol{\theta}^{(t)} - \eta \mathbf{v}^{(t)} . \end{aligned} \tag{3.43}$$

In this way, the statistical fluctuations of gradient are averaged out and the components towards the minima are preferred.

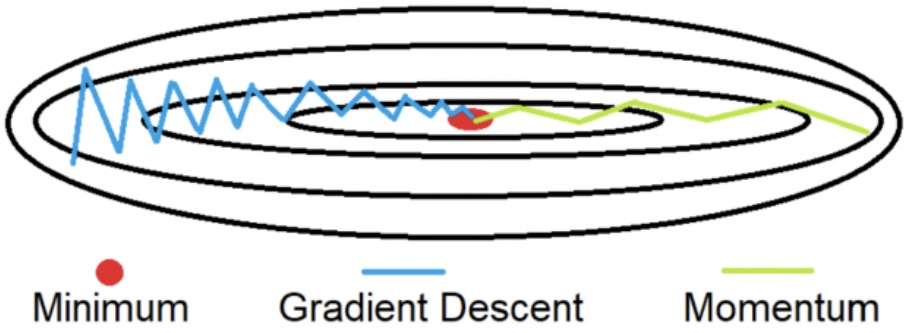


Figure 3.9: SGD with or without momentum. Figure taken from `https://cedar.buffalo.edu/~srihari/CSE676`.

Another approach is too adapt the learning rate independently for each parameter with *Adagrad*, see in [Duchi et al., 2011] also for a systematic review of SGD variants. Improved versions of Adagrad are *Adadelta* [Zeiler, 2012] and *RMSProp*. The most efficient and more utilized algorithm available nowadays, which adapts learning rate for each parameter combining Adagrad with momentum, is *Adam* [Kingma and Ba, 2014]. We refer to [Ruder, 2016] for another review on SGD variants.

So far we have seen algorithms to update the weights of a Neural Network, but how is this updating performed? Since we have to compute the gradient of the loss function with respect all the parameters in the network, we have to save all the activation during the forward phase and then *backpropagate* the gradients from the last layer to the first using the chain rule. For this reason the training algorithm is known as *Backpropagation*. We now see how it works in case of MLP.
The updating rule of the weight element i, j of layer r, at iteration t is:

$$w_{ij}^{(r)[t+1]} = w_{ij}^{(r)[t]} - \eta \frac{\partial \mathcal{L}_S}{\partial w_{ij}^{(r)[t]}} , \tag{3.44}$$

where the loss is the average over a mini-batch (Mini-batch GD), a single example (SGD) or the average over the entire training set (GD).
Let us call $\boldsymbol{\delta}^{(r)}$ the gradient of the loss function wit respect to the activation, i.e. $\boldsymbol{\delta}^{(r)} = \frac{\partial \mathcal{L}_S}{\partial \mathbf{a}^{(r)}}$.

By recalling that

$$a_j^{(r)} = \sum_{k=0}^{d_{r-1}} w_{kj}^{(r)}\ v_k^{(r-1)}\ , \tag{3.45}$$

we have

$$\begin{aligned} \frac{\partial \mathcal{L}_S}{\partial w_{ij}^{(r)}} &= \frac{\partial \mathcal{L}_S}{\partial a_j^{(r)}} \frac{\partial a_j^{(r)}}{w_{ij}^{(r)}} = \delta_j^{(r)} \frac{\partial}{w_{ij}^{(r)}} \left[\sum_{k=0}^{d_{r-1}} w_{kj}^{(r)}\ v_k^{(r-1)} \right] = \delta_j^{(r)} v_i^{(r-1)}\ , \\ \delta_j^{(r)} &= \frac{\partial \mathcal{L}_S}{\partial v_j^{(r)}} \frac{\partial v_j^{(r)}}{\partial a_j^{(r)}} = \frac{\partial \mathcal{L}_S}{\partial v_j^{(r)}}\ \sigma^{(r)'}\big(a_j^{(r)}\big)\ . \end{aligned} \tag{3.46}$$

where

$$\frac{\partial \mathcal{L}_S}{\partial v_j^{(r)}} = \sum_{k=0}^{d_{r+1}} \frac{\partial \mathcal{L}_S}{\partial a_k^{(r+1)}} \frac{\partial a_k^{(r+1)}}{\partial v_j^{(r)}} = \sum_{k=0}^{d_{r+1}} w_{jk}^{(r+1)}\ \delta_k^{(r+1)}\ , \tag{3.47}$$

and therefore

$$\delta_j^{(r)} = \sigma^{(r)'}\big(a_j^{(r)}\big) \sum_{k=0}^{d_{r+1}} w_{jk}^{(r+1)}\ \delta_k^{(r+1)}\ . \tag{3.48}$$

So it. is clear that to compute all the gradient, one has to start from the last layer, where $\boldsymbol{\delta}^{(R)}$ can be compute fro the loss, and then error is backpropagated along the chain as:

$$\mathbf{v}^{(0)} \leftarrow \mathbf{v}^{(1)} \leftarrow \mathbf{v}^{(2)} \leftarrow \cdots \leftarrow \mathbf{v}^{(R)}\ . \tag{3.49}$$

We conclude mentioning some of the many Machine Learning frameworks available nowadays. For standard model, such as SVM and Random Forests, *Sci-kit Learn* [Pedregosa et al., 2011] is still the most popular choice. For Neural Networks, the most common library in research is *Pytorch* [Paszke et al., 2019], while *Tensorflow/Keras* [Abadi et al., 2015,Chollet, 2015] and *Theano* [Al-Rfou et al., 2016] are mostly used for industrial applications.

3.9 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a kind of networks in which the dependencies of time series of data a carried through the computational graph. In this way, they are suitable to analyse temporal series. They have obtained a great success in the field of Natural Language Processing (NPL) and time series forecasting. Good explanatory works can be found in [Goodfellow et al., 2016, Sherstinsky, 2020].
While standard machine learning techniques require that the input data are independent sampling from the same distribution, which then is aimed to be approximated in the learning phase, recurrent neural networks allow the analysis of data which are correlated, for example by means of an ordinary differential equation. The typical example, which is also the framework in which we have been working in this thesis, is given by autonomous dynamical systems in Equation 2.1.
At time t we feed into the network not only the input vector at time t, but also a hidden state at previous time h_{t-1} supposed to carry on the information on the sequence from previous time-steps. Notice that this hidden state dimension of the feature dimension, but according to the problems can be smaller, if we want to compress information, or larger if a small hidden space is not sufficient to correctly reproduce the data. How this hidden state is computed depends on which information fro the past we want to retrieve. The hidden state evolves according to a dynamical equation as

$$\mathbf{h}_t = \tilde{\mathbf{g}}(\mathbf{h}_{t-1}, \mathbf{x}_t; \boldsymbol{\theta})\ , \tag{3.50}$$

where $\mathbf{x}_t$ is the current input and $\boldsymbol{\theta}$ are the networks parameters. Notice that the RNN network itself a dynamical system.
Once the current hidden state is computed from the previous one and the current input vector of the sequence, the network output is defined to be a function of this hidden state, whose dimension m can be different from both the input and the hidden state dimension

$$\hat{\mathbf{y}}_t = \mathbf{o}(\mathbf{h}_t) \ . \tag{3.51}$$

In the language of neural networks, the functions $\tilde{\mathbf{g}}$ and $\mathbf{o}$, which map the input to the hidden state and the hidden state to the output respectively, consist of linear feed-forward layers . The model equations read, for suitable activation functions σ_h and σ_o, as

$$\begin{aligned} \mathbf{h}_t &= \sigma_h(\mathbf{W}_h[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_h) \ , \\ \hat{\mathbf{y}}_t &= \sigma_o(\mathbf{W}_o\mathbf{h}_t + \mathbf{b}_o) \ , \end{aligned} \tag{3.52}$$

where $[\cdot, \cdot]$ stands for the concatenation of vectors, while $\mathbf{W}$ are the weight matrices and $\mathbf{b}$ the bias vectors.

The computational graph and the unfolded version in case of no outputs is shown in Figure 3.10.

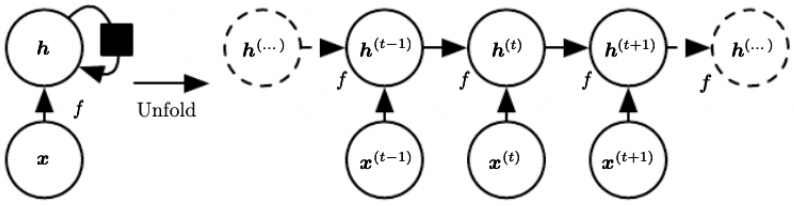


Figure 3.10: The computational graph of a recurrent neural network without outputs. On the left there is the compact representation, while on the right the unfolded one. Figure taken from [Bengio et al., 2015].

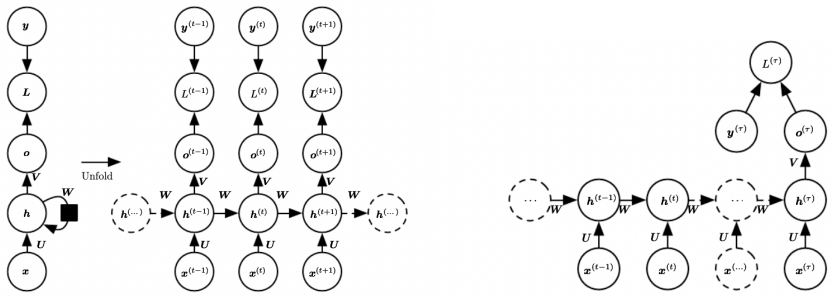
There are two main advantages of this kind of neural networks:

- The input size is always the same, regardless of the sequence length
- The function $\tilde{\mathbf{g}}$ is always the same for every time step, meaning that the parameters $\boldsymbol{\theta}$ are shared through the computational graph, allowing faster training.

In general, the output sequence length L_y does not need to be equal to the input sequence length L_x, therefore we can distinguish different kind of networks, according to the differences in the input and output sequence length: many to many, one to many, many to one and so on.

The most common cases are many to many, when $L_x = L_y$ and the network is trained to predict the next element of a sequence and many to one, in which examples are fed to the network and only after that a prediction is made. In both cases, it is the hidden vector that is passed as input in the next step, as well as the next input vector, as show in Figure 3.11. However, we could also have architectures in which the predicted output is fed as hidden state in the next step [Goodfellow et al., 2016], but we will not treat them here.

Training is performed by minimizing the discrepancy between the predicted output $\hat{\mathbf{y}}_t$ and the ground truth $\mathbf{y}_t$. In the context of Natural Language Processing (NPL) for example, the input vector is a word that has to be translated (after being encoded in a useful representation through embeddings), while the ground truth is the corresponding translated word. Since in machine translation words in a phrase do not correspond in different



(a) a (b) b

Figure 3.11: Two types of recurrent neural networks. Many to many (a), with $L_x = L_y$ and many to one (b), where L_y. In both cases, the recurrent state h is passed to the next step. Figure taken from [Bengio et al., 2015].

languages, it is necessary that the predicted sequence (translated phrase) is presented to the network and only after that the network is run without input in order to reproduce the translated phrase, which thus does not necessarily have the same number of elements. On the other hand, in the case of temporal sequences coming from weather forecasting or financial series, the aim of the network is to predict the next element of the sequence and therefore the output is computed at each time step and then compared with the next input element, which becomes the ground truth. Training is minimizing the total loss over time steps, which is $\mathcal{L} = \sum_{t=1}^{T_x} \mathcal{L}_t(\mathbf{y_t}, \hat{\mathbf{y}}_t)$ in the case $L_x = L_y$. If a Mean Square Error (MSE) is used, the each terms read

$$\mathcal{L}_t(\mathbf{y_t}, \hat{\mathbf{y}}_t) = ||\mathbf{y_t} - \hat{\mathbf{y}}_t||_2^2 \, . \tag{3.53}$$

Stochastic Gradient Descent is performed over batch sample of length B, so that by exploiting the parallel computational power of tensor, tensor of size (B, L_x, N) are fed to the network at each training step. However, the Backpropagation step called **Backpropagation through time** needed to compute gradients cannot run in parallel through time, because it must be completed sequentially for each time step.
One of the main problem that arises with standard recurrent neural network is that of vanishing or exploding gradient. Since in order to analyze long sequences Backpropagation must run very deep in the past, eventually the information coming from the first elements of the sequence become more and more irrelevant or powerful since its gradient either tends to diverge or go to zero. Therefore, standard RNN fail to pay sufficient attention to the first elements of the sequence they are analyzing, making difficult to predict long term behaviours.
Once training is completed, the network is used to make sequence prediction. By starting with a initial condition $\mathbf{x}_0$, the output of the first prediction will be used as input in the next step, until a stopping condition is met.
Training can be performed either by feeding the ground truth into the network at every time steps or using the predicted output as a next input. The first case scenario, called **teacher forcing**, the current example will be shown at every time steps, making difficult to make long previsions, since we use predictions as inputs in the feed-forward phase.On the other hand, to make better predictions, we could either compute the error after $m > 1$ forward steps in the training phase, or implement some **curriculum strategies** [Bengio et al., 2015].

3.9.1 Reservoir Computers: Echo State Networks

Reservoir computers [Lukoeviius and Jaeger, 2009] originates from an idea born independently in two papers [Maass et al., 2002, Jaeger, 2001]. Recent summary reviews can be found in [Nakajima, 2020, Tanaka et al., 2019]. The key point is to couple a recurrent neural network with a physical reservoir, usually a large and sparse random matrix, while the

training is performed only on the last layer through linear regression, possibly with Tikhonov or noise regularization. In this way, the the dynamics of a lower dimensional system is mapped into a higher dimensional space on which linear regression perform better, allowing to training a reduced number of parameters. However, the gaining in performance due to the fact that only the output layer is trained, makes reservoir networks highly sensitive to hyper-parameters, being necessary therefore a heavy fine-tuning phase. Different kind of reservoirs are possible and the power of this kind of networks has been proved in the years for mechanical [Coulombe et al., 2017, Hauser et al., 2011], memristive [Tran and Teuscher, 2019, Demis et al., 2015, Donahue et al., 2015], spintronic [Torrejon et al., 2017, Nakane et al., 2018], biological [Jones et al., 2007] and photonic resevoirs [Larger et al., 2012, Smerieri et al., 2012]. Moreover, reservoir networks have been successfully applied a to dynamical system in predicting the evolution of the state 5%6 Lyapunov times in certain regions of the attractor [Pathak et al., 2017], as well as with sparse and random input updates [Fan et al., 2020], multi deep reservoir trained with Backpropagation [Freiberger et al., 2020] and multi-variate time series [Bianchi et al., 2018].
At each iteration, the hidden state of the reservoir is update taking into account the previous hidden state, the current input and the current target output, while the update is performed through a linear layer applied to the current hidden state, possibly extended with the current input. While many types or reservoirs are possible, we can choose it to be a large random square matrix W of dimension $n_{reservoir}$ with fixed spectral radius ρ in analogy with [Maass et al., 2002, Jaeger, 2001] or an Erdòs-Reny graph consisting only of 0 and 1. Possibly, we allow the reservoir to be sparse, meaning that a fraction s of connections are removed. These sparse Reservoir Networks are called Echo State Networks (ESN). Time is discretized every δt time for continuous systems, while for discrete maps we simply put $\delta t = 1$. The input sequence at time t is denoted with $\mathbf{x}(t) \in \mathbb{R}^{n_{in}}$, while we call the output sequence $\mathbf{y}(t) \in \mathbb{R}^{n_{out}}$ Reservoir hidden state is update according to

$$\mathbf{h}(t) = \tanh\big[W\ \mathbf{h}(t-\delta t) + W_{in}\ \mathbf{f}_{in}(\mathbf{x}(t)) + W_{out}\ \mathbf{f}_{out}(\mathbf{y}(t-\delta t))\big] + \eta\ , \tag{3.54}$$

where W_{in} is a $n_{reservoir} \times n_{in}$ matrix coupling the input with the reservoir, W_{out} is a $n_{reservoir} \times n_{out}$ coupling the output with the reservoir and $\mathbf{f}_{in}$, $\mathbf{f}_{out}$ are scaling function acting on input and output respectively, while η is a regularization noise. By inserting also the wanted output in computed the evolution of the hidden state, we insert also the true information which then differentiate significantly the training phase from the predictive one and comes under the name of *teacher forcing*, allowing during training to "anticipate" the solution. Here we choose a standard Gaussian noise with zero mean and specified standard deviation as regularizer, while other authors preferred instead to use Tikhonov regularization [Tikhonov and Arsenin, 1977], for example [Pathak et al., 2017].
Once the hidden state is update, the output is computed from

$$\hat{\mathbf{y}}(t) = W_{out}\ \mathbf{z}(t)\ , \tag{3.55}$$

where we extend the hidden state with the input state and therefore

$$\mathbf{z}(t) = \begin{bmatrix} \mathbf{h}(t) \\ \mathbf{x}(t) \end{bmatrix}\ . \tag{3.56}$$

The output matrix W_{out}, which is thus a $n_{reservoir} \times n_{reservoir}\ n_{in}$ matrix, is optimized with linear regression as

$$W^*_{out} = (S^\dagger D)^\intercal\ , \tag{3.57}$$

where the symbol † indicates the Moore-Penrose pseudo-inverse [Moore, , Penrose, , Bjerhammar, 1951]. S is the collection matrix of extended states, meaning the i-th row of S is $\mathbf{z}(i\ \delta t)$. In the same manner, D is the collection matrix of desired outputs. Training is performed by the collection of a dynamics of T seconds, possibly discarding the first 10%

entries in order get rid of the effects of random initialization. The optimization formula Equation 3.57 is equivalent to the minimization of the objective function

$$\sum_{t_0 \leq t \leq 0} ||W_{out}\ \mathbf{z}(t) - \mathbf{y}(t)||^2 \ , \tag{3.58}$$

where $t_0 = \min(T/10, 100)$.
On the other, in the case of Tikhonov regularization, which is not considered in this thesis, the objective function would be

$$\sum_{t_0 \leq t \leq 0} ||W_{out}\ \mathbf{z}(t) - \mathbf{y}(t)||^2 + \beta\ ||W_{out}||^2 \ , \tag{3.59}$$

where $||W_{out}||^2$ is the sum of the squares of the elements of W_{out} and $\beta > 0$ acts as the regularization constant instead of the variance of the Gaussian noise η.Equation 3.59 can be solved with standard linear regression techniques [Puntanen, 2010].

Once trained, we want to use our network to make predictions. Instead of filling as input the true input at each prediction step, we want to feed directly the predict output. Therefore, in prediction phase the network differ significantly from the training epoch, since no new information is put into the network.

3.9.2 Long Short Time Memory

Long Short Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] networks were first introduced to limit the problem of exploding/vanishing gradient in long sequences. Instead of having only information encoding the previous time steps in the hidden state $\mathbf{h}_t$, they have an additional state, called cell state $\mathbf{c}_t$ which encode the long term information of the network. Because its ability to analyze deeper sequences, LSTM networks have had a great success in NLP [Sak et al., 2014], until at least the advent of Transformers [Vaswani et al., 2017].
How long and short term information are propagated through time depends on a series of "gates", called *forget gate*, *update gate* and *output gate*.
At each time the network perform different operations:

- **Forget gate**
  The current element of the sequence $\mathbf{x}_t$ and the previous time step hidden state $\mathbf{h}_{t-1}$ are concatenated and the passed through a linear layer and a sigmoid activation function σ to determine the forget gate, with elements in $[0, 1]$, which will determine how much long term information coming from the cell state $\mathbf{c}_{t-1}$ will be forget:

  $$\Gamma_f = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \ . \tag{3.60}$$

- **Input gate**
  This gate determines how much information coming from long term dependencies is worth to be remembered and passed through the next step:

  $$\Gamma_u = \sigma(\mathbf{W}_u[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_u) \ . \tag{3.61}$$

- **Proposed cell state**
  This steps computes the proposed long term information to pass at next step, selecting relevant features and neglecting less useful ones:

  $$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c) \ , \tag{3.62}$$

  where we use the tanh activation function in order to mitigate the impact of bad components in the next cell state.

- **New cell state**
  The update cell state at time step t will be a weighted balance between the previous cell state through the forget gate and the proposed one through the input gate:
  $$\mathbf{c}_t = \Gamma_u . * \tilde{\mathbf{c}}_t + \Gamma_f \mathbf{c}_t \ , \tag{3.63}$$
  where $*$ is the Hadamard product [2]

- **Output gate**
  This gate determines the new hidden state and as before is given by a sigmoid activated linear layer
  $$\Gamma_o = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \ . \tag{3.64}$$

- **New hidden state**
  The next hidden state is simply the Hadamard product of the output gate with the new cell state:
  $$\mathbf{h}_t = \Gamma_o * \mathbf{c}_t \ , \tag{3.65}$$
  which will then be used to compute the predicted output of the network $\hat{\mathbf{y}}_t$ at time step t (e.g. through another learnable linear layer or directly through an activation function like a softmax in case we want to classify with probabilities).

The complete computational scheme of one block is reported in Figure 3.12.

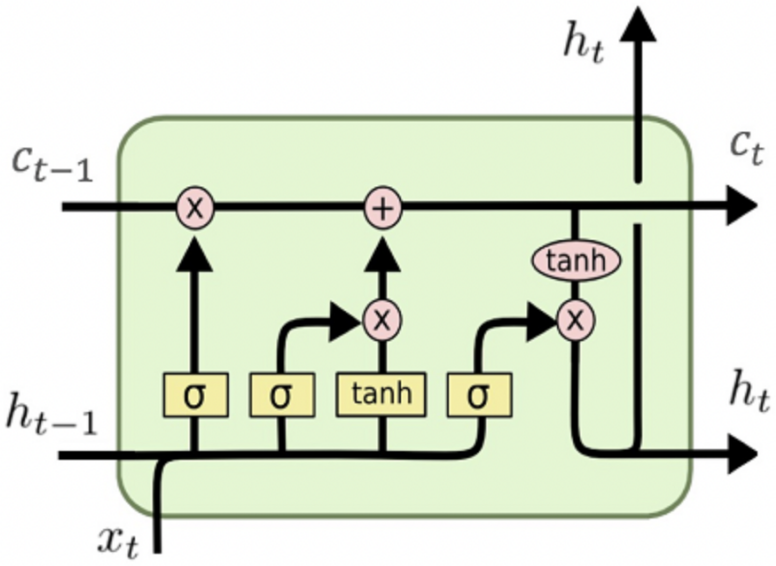


Figure 3.12: The schematic representation of operation involved in a block of a LSTM network. Intersecating lines stand for concatenation, while operations to be performed are represented with circles. Figure taken from `https://amr-khalil.medium.com/what-is-long-short-term-memory-lstm-cdd8c669a73e`.

3.10 Autoencoders

Autoencoders are a class of neural networks for unsupervised learning which aim to compress the data and then reconstruct it minimizing the information loss. They are composed by two parts: encoder and a decoder. Input data $\mathbf{x}$ are fed into the encoder part, which transform

[2] When dealing with RNNs, we usually work with tensor of shape $(batch\_size, sequence\_length, feature\_dim)$ and the Hadamard product is simply the element-wise product between tensors of equal shape.

data and return an embedded representation $e = E(\mathbf{x})$, usually living lower dimensional space called *hidden space*. The encoded representation is then fed to the decoder, which tries to reconstruct data by minimizing the loss between input and reconstructed data, usually given by L2 loss $\hat{\mathbf{x}} = D(e) = D(E(\mathbf{x}))$, as shown in Figure 3.13:

$$\mathcal{L}_{rec}(\theta) = \mathbb{E}(|\mathbf{x} - \hat{\mathbf{x}}|^2) \ . \tag{3.66}$$

In some applications, however, also a binary cross entropy.
Since we do not use labels, these models go under the class of unsupervised learning algorithms, which aim to capture the probability distribution of the environment without having labelled it before and compress the high dimensional features of input data, of which many of them may be useless, in a low dimensional representation that forces only relevant information to pass.
Once the Autoencoder is trained, it could be either used to compress input data for dimensionality reduction or to generate new data starting from samples in the hidden space. In this sense, it goes under the category of generative models.

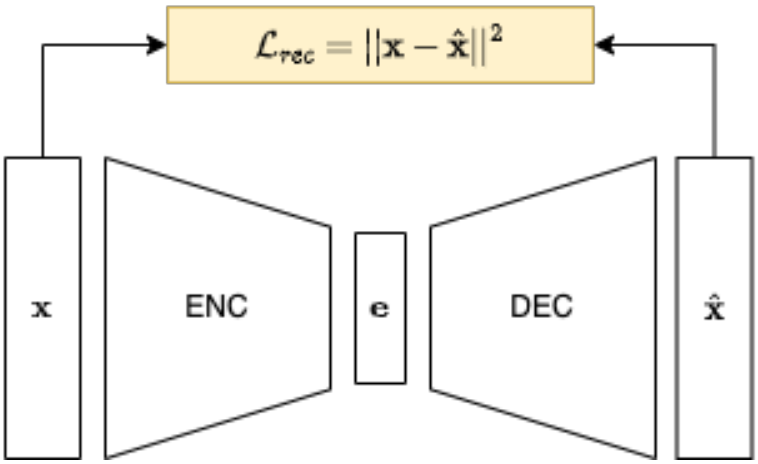


Figure 3.13: General structure of an autoencoder. The information is first compressed into a latent space, also known as hidden space, that carries the relevant information and then reconstructed to minimized the information loss.

The information compressed, however, should be sufficient to reconstruct the input in the most precise manner, unless we use some *regularization* for the encoded space, on which we will come back later.
The Encoder and Decoder network can be any type of network, from Multi Layer Perceptron (MLP), Convolutional Networks, LSTMs or Boltzamann Machines. In many applications, the Decoder is simply the symmetric image of the Encoder network, even though this requirement is not strictly necessary and the overall structure would learn anyway, even with more freedom on its parameters.
In case that both the Encoder and the Decoder are MLP (a.k.a. Feed-forward networks), the Autoencoder would simply perform Principal Component Analysis (PCA) [Pearson, 1901], when the activation functions would be linear. The structure of the Encoder and the Decoder will depend on the kind of data that we aim to analyse. If we want to find a lower dimensional codification of images, for example, we would rather use Convolutional networks, as shown in [Deng, 2012].

In standard autoencoders, the hidden space is not regularized and the networks has only the constraint to minimize the loss between input and reconstructed samples. This means that encoded samples in the hidden space can be organized arbitrarily and not in the needed way. Passing from linearity, when the autoencoder behaves similarly to PCA, to non linear architectures, an autoencoder becomes more and more powerful eventually being able to compress data into a hidden space of dimension 1. However, this reconstruction

power comes at the cost of interpretability and structure of the hidden space. Moreover, without regularizing the hidden space an autoencoder could in principle bring to zero the reconstruction loss resulting in overfitting. For example, in the case of MNIST dataset [Deng, 2012], the Autoencoder cold learn well how to reconstruct images, but their hidden representation could be very messy. It could happen that two 1s are represented very far apart, even though we would expect that similar digits are compressed in points close to each other since they have similar features. On the other hand, it could happen that between two 1s, we could find a 6 in the hidden representation, meaning that similar samples do not transform continuously between each other. Therefore, we would need to regularize the hidden space in order to have:

- **Continuity**: two close input samples should be compressed into closed point in the hidden space.
- **Completeness**: a chosen point in the hidden space should give a meaningful content once decoded.

Therefore, instead of compress information into single points in the hidden space, we would better encode information into probability distributions over the encoded space. By requiring that the encoded probability distributions are independent standard Gaussian, we build (Gaussian) Variational Autoencoders. The output of the encoder are the mean $\boldsymbol{\mu}$ and the standard deviation $\boldsymbol{\sigma}$ vectors of the Gaussian of hidden space variables. In order to build the hidden representation $\mathbf{e}$, we need to sample each component according to a Gaussian with mean μ_i and standard deviation σ_i, $e_i \sim \mathcal{N}(\mu_i, \sigma_i)$. However, if we sampled directly $\mathbf{e}$, we could not calculate the gradients since they are determined stochastically. Instead, we make use of the so-called *reparametrization trick*, by sampling $\boldsymbol{\epsilon}$ according to a normal Gaussian with zero mean and unitary standard deviation $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{O}, \mathbf{I})$, then computing

$$\mathbf{e} = \boldsymbol{\mu} + \boldsymbol{\sigma} * \boldsymbol{\epsilon} \,, \tag{3.67}$$

where $*$ is Hadamard element-wise product.
In this way, we can compute gradients and make flow backpropagation through the encoder/decoder bottleneck.
The error is the sum of two terms. The first term is given by the reconstruction loss $\mathcal{L}_{rec}(\boldsymbol{\theta})$, which is again the L2 loss between the reconstructed and the input sample; minimizing it would push the network to optimize as much as we can the reconstruction capabilities. The regularization loss is given by the *Kullback-Leibler divergence* (KL divergence) [Kullback and Leibler, 1951] between the encoded probability distribution and the target probability distribution, which in case of Gaussian depends only on the means and standard deviations. The Kullback-Leibler divergence between distributions P and Q is a pseudo-metric (it is not symmetric and does not satisfy triangular inequality) measuring the excess of surprise that we have when approximating Q with P. In case of Gaussian targets, KL divergence loss is given by

$$\mathcal{L}_{KL}(\boldsymbol{\theta}) = -\frac{1}{2} \sum_{i=1}^{d_{enc}} \left[1 + \ln \sigma_i^2 - \mu_i^2 - \sigma_i^2 \right] . \tag{3.68}$$

By minimizing the KL divergence between the encoded probability distribution and a standard Gaussian, we enforce the covariance matrix to be an identity, such that the encoded samples lie close to each other and *overlap* as much as possible, satisfying the requirement of continuity and completeness, as shown schematically in Figure 3.14.

3.11 Convolutional Neural Networks

Convolutional Neural Networks (CNN), first introduced by [LeCun et al., 1989], are neural networks that employ a *convolutional* operation instead of standard matrix vector

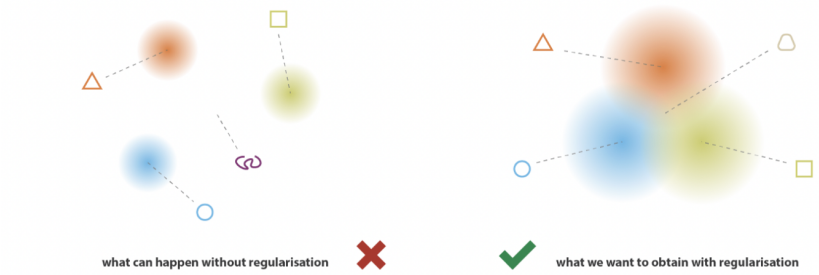


Figure 3.14: A schematic view on how KL regularization help to organize the hidden space in order. Figure taken from `https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73`.

multiplication used in standard (Feed-Forward) networks [Bengio et al., 2015, Mallat, 2016]. They are the state of the art models for a great variety tasks, such as image classification with thousands of complex classes [Krizhevsky et al., 2012], speech recognition [Hinton et al., 2012], biomedical applications [Leung et al., 2014], natural language understanding [Sutskever et al., 2014] and many other fields.
Let supposed to have a continuous signal $x(t)$, the convolution operation is simply an integral that weights that signal for different temporal regions according to a weight function $w(t)$

$$s(t) = (x * w)(t) = \int x(a)\, w(t-a)\, da\,, \tag{3.69}$$

which in Fourier space becomes simply the product of the two Fourier transforms. In order for w to be a valid density function, it should be 0 for all negative arguments in order not to look at the future statistics and preserve causality. The signal $x(t)$ is often referred as the *input*, the weight function $w(t)$ as the *kernel* and the output as *feature map*.
In Machine Learning applications, we work with discrete data and so the convolution operation becomes

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a) w(t-a)\,. \tag{3.70}$$

The latter formula can be further generalized to multi dimensional inputs, such as images, as

$$S(i,j) = (I * K)(i,j) = \sum_{m,n} I(m,n) K(i-m, j-n)\,, \tag{3.71}$$

which satisfy the commutation relation by performing a simple *flipping* of the kernel

$$S(i,j) = (K * I)(i,j) = \sum_{m,n} I(i-m, j-n) K(m,n)\,, \tag{3.72}$$

which is simpler to implement because of the less variation of the indices m, n. However, usually machine learning libraries implement $cross - correlation$ [Bengio et al., 2015], as simple as

$$S(i,j) = (K * I)(i,j) = \sum_{m,n} I(i+m, j+n) K(m,n)\,, \tag{3.73}$$

which is the same as convolution, but without flipping the kernel. Since then in machine learning application the kernel filters will be learnable functions, the learned convolution flipping filter will learn a kernel that is flipped with respect cross correlation, but usually convolutional networks perform many other operations to extract features from data and therefore it is not relevant which of the two is used and we will reference both as convolution, following [Bengio et al., 2015].
Usually, indexes run over a restricted set of possibilities in order to capture local information

of the so-called *receptive fields*, which may be defined as the input neurons that determine the output of a given neuron in the next layer. In case that we have an image, we might expect that the pixels are correlated in a small range and so the kernel filters would be small (compared to the size of the image) rectangles of sides m, n. This property goes under the name of *sparse interaction* and it is useful to detect small properties, like tiny edges, of an image, which may have thousands of pixels, helping to reduce memory storage and improve statistical efficiency. Therefore these filters are usually (much) smaller than input. Of course, we might apply different kernels to the same image in order to extract different information and thus have multiple *channels*.
Roughly speaking, a kernel is simply a (learnable) matrix that that "slides" along the input in its different dimension, each pixel of the window "covered" by the kernel is multiplied by the corresponding image pixel and all are summed together, possibly adding a bias, to give the pixel output. Due to convolution, the resulting image is shrinked with respect to the original one, as shown in Figure 3.15. Doing so, pixels near the edges of the images will count less with respect central pixels, since they will have a weight in less pixels of the output. Therefore, it is often useful to extend the input on the border by adding some *padding* pixels, consisting of 0s. In this way, border pixels will have a greater impact on the final output, which will be at the same time less shrinked.
The way in which kernels are move across the input in the various direction is another control parameter that goes under the name of *stride*. For example, if the stride of the convolutional filter is $(1, 2)$ it means that the kernel moves of one step at each time on the x direction, and of 2 at the y direction.
In the end, dimensionality reduction of a $H_{in} \times W_{in}$ image after applying a convolutional filter will be

$$\begin{aligned} H_{out} &= \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel}[0] - 1) - 1}{\text{stride}[0]} + 1 \ , \\ W_{out} &= \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel}[1] - 1) - 1}{\text{stride}[1]} + 1 \ , \end{aligned} \tag{3.74}$$

where *dilation* is another hyper-parameter controlling the spacing between kernels.

Let us come back to the image example and try to see better how convolution there works in order to extract spatial information. If we wanted to analyze pictures by using standard Feed-Forward neural networks, we would need to flatten the input. However, in this way we would loose all the information coming from the fact that some pixels are "near" to others and therefore might be correlated (the same reasoning apply to time series, in which the "image" would simply be the sequence length times the number of features). Instead, convolution use the information coming from the proximity of some pixel to the other on order to useful classification for other tasks, such recognizing edges efficiently, as shown in Figure 3.16.

Another important aspect of convolutional networks is *parameter sharing*. In standard Feed Forward networks, each layer is multiplied by a weight matrix and thus each element of the matrix is used only one time when computing the output. On the other hand, convolutional networks have filters that run over the entire image and thus they are reused at every position of the input, trying to detect the same patters in different parts of the input. Moreover, the use of the same parameters reduce the memory storage and the statistical efficiency, even though it does not gain computational speed.
Convolutional networks are also characterized by *equivariance* to translation, meaning that the output change in the same way as the input. A function $f(x)$ is equivariant to a function $g(x)$ if $f(g(x)) = g(f(x))$. For example if I is the function measuring the image brightness at each integer coordinate and g a generic image mapping function, such that $I' = g(I)$ with $I'(i, j) = I(i - 1, j)$, meaning that shift each pixel of I one unit to the right. In this

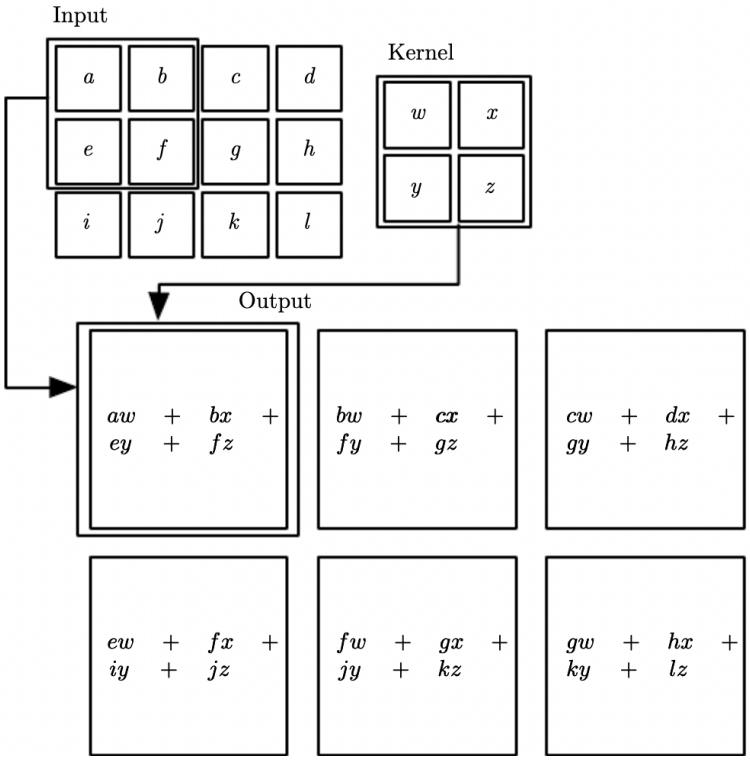


Figure 3.15: An example of application of a 2×2 convolutional filter to a 3×4 images without flipping. We notice how the kernel. slides across the input image, being the resulting output pixels the sum of the element-wise product of the kernel matrix with the window currently "observed" by the kernel. The figure is taken from [Bengio et al., 2015].

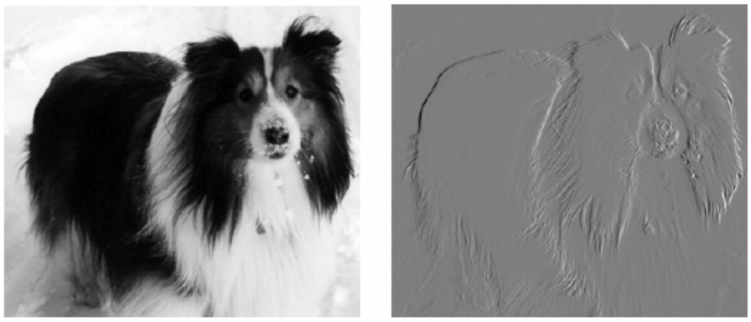


Figure 3.16: The figure is made by subtracting to each pixel of the left picture its neighbouring left pixel, thus applying the convolutional filter $[-1, 1]$, making evident the impact of vertical oriented edges. Figure taken from [Bengio et al., 2015].

case equivariance to translation means that if we applied this transformation to I, then applying the convolution, the result would be the same as first applying the convolution to I' and then transformation g to the output. Convolution build a map showing where certain features appear in the input. If the input is translated, then the convolution output map will be translated of the same amount. However, convolution is not naturally equivariant to other kind of transformation, such as rotation or dilation and therefore other means of data handling are required.

Usually, a convolutional layer can be seen as a more complex layer performing other operations. The first of course is a set of convolution filters applied in parallel to get a linear activation. The linear activation is then passed to a non-linear activation function, such as the Rectified Linear Unit (ReLU), in a process also known as *detector stage*. The last process is to apply a *pooling* layer.

A pooling operation simply replies the output with a summary statistics of the nearby

outputs. A commonly used layer is *maxpool*, in which the output is replaced with the maximum output in a rectangular window centered in the output [Zhou and Chellappa, 1988]. Another popular choice is *average* pooling, in which the output is replaced with the average of nearby outputs. A theoretical work guiding on the usage of different kind of pooling can be found in [Boureau et al., 2010]. In both cases, the final result is further shrinked with respect to the original input, again reducing memory requirements and computational power. Moreover, pooling layers act as noise suppressing.
In all cases, pooling makes the representation approximately invariant with respect translation, because we sometimes are more interested in the presence of some feature and not maybe exactly where this feature appear in the input. When detecting eyes, for example, we do not need to know exactly where to find an eye, but we may want only to know that there is an eye in the upper left side of the image and another in the upper right part.
The pooling operation can be seen as putting a strong prior on the fact that learned functions must be invariant to translation of the input, thus improving statistical efficiency. But this is not limited to spatial translations, which arise when we pool over spatial regions. Indeed, if we perform pooling over the outputs of different convolutional filters, the features can learn to which transformation to be invariant to.

As we have seen, convolutional layers, seen as complex layers with many operations (convolutional filters, non-linear activations and pooling) are very efficient in extracting local information, learn invariant features, reducing memory storage and increasing statistical efficiency. Standard convolutional networks are composed by many of these complex layers, in order to learn different features at different depth levels. For example, if we want to classify complex images coming from real worlds, such as planes, trees, cars etc., we might need to detect vertical or horizontal edges in the deepest layers, while the surface one may be learning more complex features, such as the presence of wheels, useful to tell apart vehicles for examples, or plane wings. Therefore, after convolutional layers it is common practice to insert fully connected layer in order to learn high level functions helping classifying the various classes. In this case, having various classes the common choice would be to apply a softmax activation in the last layer in order to make the network output probabilities and then use cross-entropy loss as loss function. In this case, a trained network would output the probability that a given input belong to the different possible classes. A typical example of full convolutional network for classification is reported Figure 3.17.

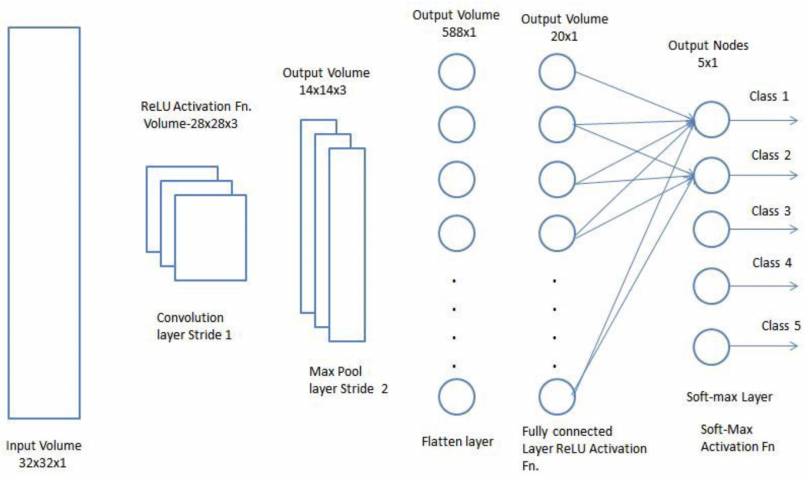


Figure 3.17: A typical convolutional networks used to classify one channel inputs into 5 classes, consisting of one convolutional layer, max pooling and 2 fully connected classification layers. Figure taken from `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`.

3.12 Physics Informed Neural Networks

We briefly describe Physical Informed Neural Networks (PINN) [Raissi et al., 2019, Zhang et al., 2020], a kind of networks designed to solve a general partial differential equation by using also the physical information that comes with the model. In this way, PINNs can approximate a generic function by exploiting the approximation power of neural networks [Kolmogorov, 1961, Hornik, 1991] and the computational tools for automatic derivation [Baydin et al., 2017] using the prior knowledge directly available with the equation that one wants to solve, such the partial differential equation constraint, boundary loss, initial condition loss etc.
Let us consider a general partial differential equation of the function $u(\mathbf{x}, t)$, depending on time and position:

$$\begin{aligned} \frac{\partial u(\mathbf{x},t)}{\partial t} + \mathcal{N}(u,\lambda) = 0 \ , \quad \mathbf{x} \in \Omega, t \in [0,T] \ , \\ u(\mathbf{x},0) = h(\mathbf{x}) \ , \quad \mathbf{x} \in \Omega \ , \\ u(\mathbf{x},t) = g(\mathbf{x},t) \ , \quad \mathbf{x} \in \partial\Omega, t \in [0,T] \ , \end{aligned} \tag{3.75}$$

where $\Omega \subset \mathbb{R}^D$ and $\mathcal{N}$ is a generic differential operators acting on u, the true hidden solution of the problem, and depending parametrically on λ, which can be a generic vector of parameters. Equation 3.75 encapsulate many problems from physic and mathematics, like the Schrödinger equation and Navier-Stockes equation. For example, in the case of Shcrödinger equation, we would simply have $\mathcal{N}(u,\lambda) = \frac{i}{\hbar}\hat{H}u$, where u is the wave function and $\hat{H}$ the Hamiltonian operator.
Let us suppose to build a neural network that aims to approximate the true hidden solution $u(\mathbf{x}, t)$ with its output $f_\theta(\mathbf{x}, t)$, which depends parametrically by θ, the network's parameters. Let us define the residual function

$$r_\theta(\mathbf{x},t) \equiv \frac{\partial f_\theta(\mathbf{x},t)}{\partial t} + \mathcal{N}(f_\theta(\mathbf{x},t),\lambda) \ , \tag{3.76}$$

where derivatives with respect to space and temporal coordinates an be computed with high precision using automatic differentiation techniques available with modern deep learning softwares [Baydin et al., 2017].
Trained is performed with Stochastic Gradient Descent by minimizing the composite loss function

$$\mathcal{L}(\theta) \equiv \mathcal{L}_r(\theta) + \sum_i \lambda_i \mathcal{L}_i(\theta) \ , \tag{3.77}$$

where the residual loss is computed as

$$\mathcal{L}_r(\theta) = \frac{1}{N_r}\sum_{i=1}^{N_r} r_\theta(\mathbf{x}_i, t_i)^2 \ , \tag{3.78}$$

for a set of collocation points $\{(\mathbf{x}_i, t_i); \mathbf{0}\}_{i=1}^{N_r}$ placed randomly inside the domain Ω.
The terms $\mathcal{L}_i(\theta)$, which are weighted for certain factors λ_i, can have different nature, like they can be boundary loss, initial condition loss or training loss if we had also example of the true hidden solution to show to the network. In the case of a initial condition problem with only physical driven loss, we would have two terms

$$\begin{aligned} \mathcal{L}_{u_b} = \frac{1}{N_b}\sum_{i=1}^{N_b} (f_\theta(\mathbf{x}_i^b, t_i^b) - g_i^b)^2 \ , \\ \mathcal{L}_{u0} = \frac{1}{N_0}\sum_{i=1}^{N_0} (f_\theta(\mathbf{x}_i^0, 0) - h_i^0)^2 \ , \end{aligned} \tag{3.79}$$

for a set $\{\mathbf{x}_i^0, h_i^0\}_{i=1}^{N_0}$ of initial points and $\{\mathbf{x}_i^b, g_i^b\}_{i=1}^{N_b}$ of boundary points.

Chapter 4

Experiments

4.1 Introduction

In this chapter we perform several experiments with Machine Learning techniques applied to the paradigmatic models of [Lorenz, 1963], [Rössler, 1976] and [Lorenz, 1995].
In section 4.2 we will make use of LSTM, ESN and Feed-Forward neural networks to learn the dynamics of low dimensional systems, i.e. [Lorenz, 1963] and [Rössler, 1976] by using a purely Data-Driven and a Physics-Informed loss. We will assess the performance of the model in Weather and Climate reconstruction against the ground truth given by an independent test dataset, thus trying to answer to question 1 and 2 of section 1.2. At the end of the section, we will also try to infer the model parameters with the Physics Informed kind loss, but without getting good results. In any case, we will see that the best model to reconstruct both the Weather and the Climate are LSTM, even though results are a little bit worse for [Rössler, 1976] system, which is notoriously difficult due to the rare and random jumps that $x^{(3)}$ undergoes. For both models analysed with LSTM, we assist to a decisive improvement when using optimized hyper-parameters and the Physics Informed loss. In case of ESN, we will get always worse results, being the model able to reconstruct both Weather and Climate only using sparse Random Matrices and only for Lorenz 96. The use of Erdos-Reny graphs worsen the performance.
In section 4.3 we will try to address to the 3rd question in section 1.2 by learning the parameters of Lorenz 63 and Rössler 76 with regularized Convolutional and Feed-Forward Autoencoders. Single dynamics (single model parameters combinations) will be fed to the architectures and the regularization loss will be wither purely Data-Driven (i.e. pretending to know the true parameters) or Physics-Informed (i.e. relying only on the dynamical equations). However, the results are worse for Physics-Informed loss kind.
Finally, in section 4.4 we will change prospective in addressing question 3 of section 1.2, by trying to learn the forcing of [Lorenz, 1995] in a completely unsupervised way by feeding to Convolutional Networks not dynamics inputs, but directly images with varying forcing. We will see that the model is able to capture the chaotic phase transition for both layer normalized and non-normalized inputs. However, experiments with Variational Autoencoders did not give any result and so we decided not to show them.
The supporting figures are reported in Appendix A.
All the models that we will present are trained either on a home computer or on the free Google Colab account. For practical reasons, considered the number of models trained and the many experiments that were performed, we could train for a maximum of 2%3 hours each model, paying however attention to make the loss to reach a plateau.

4.2 Dynamics Reconstruction

4.2.1 Problem Statement and Datasets Specification

In this section, we explore the power of neural networks to replicate the dynamics of Lorenz 63 [Lorenz, 1963] and [Rössler, 1976] systems. The models are given by the set of ordinary differential equations in Equation 2.43 and Equation 2.24. As already mentioned, by the standard choice of the parameters ($\sigma = 28, \rho = 10, \beta = 8/3$) and ($a = 0.37, b = 0.2, c = 5.7$) the systems exhibits chaotic behaviour, meaning that two trajectories that starts with very tiny different initial conditions will eventually be very distant apart. In other terms, at least one of the Lyapunov exponents, defined as the exponential growth rate of small perturbations, is positive. Moreover, in this setting both systems are dissipative and have a strange attractor of fractal dimension. More generally, let us consider to have and autonomous dynamical system, defined by the differential equation

$$\dot{\mathbf{x}}_t = \mathbf{g}(\mathbf{x}_t) \ , \tag{4.1}$$

where $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{g} : \mathbf{R}^N \to \mathbb{R}^N$ is the evolution function. Given an initial condition $\mathbf{x}_0 \in \mathbb{R}^N$, we define the flow of the system as $f : \mathbb{R} \to \mathbb{R}^N$, returning the trajectory point at time t that started at $\mathbf{x}_0$ at time 0. Therefore we indicate trajectories as the set of points $\{f^t(\mathbf{x}_0) : t \in \mathbb{R}_{\geq 0}\}$.
It has already been proved that recurrent neural networks, in like Reservoir Networks [Pathak et al., 2017], can effectively replicate the short term dynamics (weather) of the systems much further than the time scale given by the inverse of the largest Lyapunov exponent and the long term dynamics (climate).
We design a network $\hat{f}_\theta$ that will be trained to replicate the time step integrator

$$\mathbf{x}_{t+\delta t} = \mathbf{x}_t + \int_t^{t+\delta t} \mathbf{g}(\mathbf{x}_\tau) \ d\tau \tag{4.2}$$

as $\hat{\mathbf{x}}_{i+1} = \hat{f}_\theta(\mathbf{x}_i)$, where $\mathbf{x}_i \equiv \mathbf{x}(i\delta t)$.
First, we numerically integrate the systems Equation 2.43 and Equation 2.24 to build train, test and validation datasets, as specified in Table 4.1. We also build a optimization datasets which will be used later to perform hyperparameter optimization.

δt	Train	Validation	Test	Optimization	Discarded
0.002	100000	20000	100000	5000	5000

Table 4.1: Dataset specification

Train, validation and optimization datasets are generated starting from a random point and neglecting the first 5000 entries, while test datasets is generated starting from the last point of the training dataset [1].
Loss is simply the $L2$ between the predicted output and the true state:

$$\mathcal{L}(\theta) = \mathbb{E}\big(||\mathbf{x}_{i+1} - \hat{f}_\theta(\mathbf{x}_i)||^2\big) \ , \tag{4.3}$$

where averages are done over sequences and batches.

For models trainable with SGD, training is performed with Adam optimizer [Kingma and Ba, 2014] until convergence, which is evaluated with Early Stopping [Zhang and Yu, 2005] with patience of either 100 or 200 iterations. For Echo State Networks, training is performed as specified in subsection 3.9.1.

[1] This is done to better analyze Reservoir Networks, which require a continuation state. However, it does not change anything for the other models, since trajectories are still attractors.

To assess the performances of our models, we compute the Wasserstein Distance (WD) of the 2d projections between the reproduced dynamic and the test dataset for the long term dynamics, while to asses the performance in short term, we compute the $R2$ scores (see section 2.11) between the predicted dynamics and the test dataset for 500 time steps, i.e. $t = 1$ for the Lorenz 63 system and for 2500 steps ($t = 5$) for Rössler 76, which are the typical times dictated by the inverse of the first Lyapunov exponents. This is done by sliding a window along the predicted dynamics by starting the prediction every 10 steps in order to computed the computational time, which takes about ≈ 6 minutes for Lorenz 63 and ≈ 50 minutes for Rössler 76.

We conclude remarking that the training phase significantly differ from the prediction phase, because of *teacher forcing*. In all the model that we will explore, during training batches of sequences long from 1 to 2500 time steps are fed into the network at each iteration, and all are randomly sampled from the training dataset. The network is trained to predict only the next step, in case it acts an integrator, or the evolution function at that time step. On the other hand, in the prediction phase, the output is fed into the input in the next time step, starting from the last point of the training dataset.

4.2.2 LSTM

LSTM have achieved great results in sentiment analysis [Murthy et al., 2020] and natural language processing [Wang and Jiang, 2016], even if we are only aware of applications to dynamical systems in [Vlachas et al., 2018], [Barzegar et al., 2022] and [Yeo and Melnyk, 2019].
In this thesis, we first build a LSTM (see subsection 3.9.2) with standard configuration for both Equation 2.43 and Equation 2.24 systems, as specified in Table 4.2. We came up with this architecture after several trials of manual fine-tuning of the hyper-parameters. We refer at this architectures as Standard models. We notice that at least 2 LSTM layers are needed to replicate correctly the dynamics, but with more we do not increase sufficiently the performance to justify the computational cost. In line with reservoir networks, the recurrent dimension of the LSTM layers must be chosen much higher than the dimension of the dynamics, in order to properly embed the system [Takens, 1979]. Here we choose 100 hidden units. We prefer Dropout with probability $p = 0.3$ as regularizer [Srivastava et al., 2014], instead of $L2$ regularization [Krogh and Hertz, 1991] and [Nowlan and Hinton, 1992], following the latest trend in Deep Learning community. However, we have also tried also experiments with a standard weight decay of 0.1, but gaining worse results than using dropout.

	Hidden units	Number of layers	Dropout	Sequence length
LSTM	100	2	0.3	500

Table 4.2: LSTM standard architectures.

For each train epoch, the network is fed with input tensors $\mathbf{x}$ of size (B, S, N), where we choose the batch size $B = 20$, the sequence length of $S = 500$, while $N = 3$ is the dimension of the system. We choose 500 time steps because is at the order of magnitude of the fisrt Lyapunov time, defined as the inverse of the greatest Lyapunov exponent [2], which is the time at which we expect to observe a chaotic behaviour. The statistical results are reported in Table 4.3, while losses in Figure A.4 in Appendix A. We report also the predicted attractors colored with Local Lyapunov Exponents (LLE) in Figure 4.1 and the distribution of LLE computed from the dynamics in Figure A.1. Lorenz return map is shown in Figure 4.2. Examples of short reconstructed trajectories are reported in Figure A.5.

From Figure 4.1 we can see that LSTM succeed in reproducing the attractor for both Lorenz

[2]Remember that $\lambda_1 \approx 0.9$ for Lorenz 63 and ≈ 0.2 for Rössler system.

System		Lorenz 63	Rössler 76
WD ($\cdot 10^{-4}$)	$x^{(1)} - x^{(2)}$	25.8919	49.9143
	$x^{(1)} - x^{(3)}$	27.65010	28.3855
	$x^{(2)} - x^{(3)}$	18.2429	32.5581
R2	$x^{(1)}$	0.24 ± 1.89	0.74 ± 0.26
	$x^{(2)}$	0.31 ± 1.41	0.65 ± 0.37
	$x^{(3)}$	0.63 ± 0.69	-24 ± 190
	< 0	14.24 %	17.47 %

Table 4.3: Statistics of standard LSTM models. We report the Wasserstein Distance (WD) of the 2d projections of the invariant measure against the test dataset, the R2 scores for the short term dynamics along with the global percentage of those smaller than zero, see text.

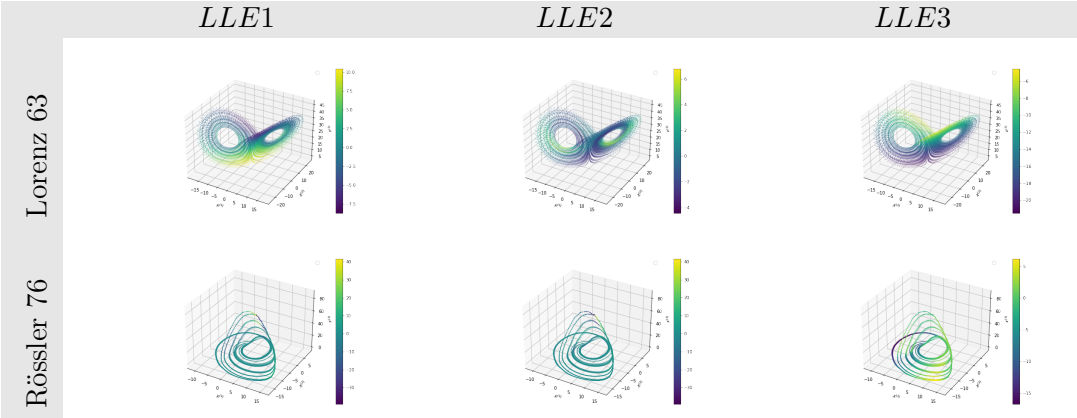
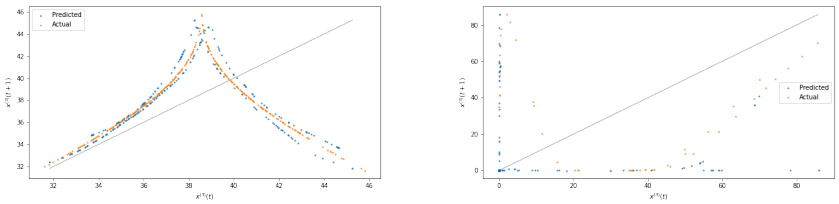


Figure 4.1: Standard LSTM models. Predicted attractor colored with Local Lyapunov Exponents.



(a) Lorenz 63 (b) Rössler 76

Figure 4.2: Standard LSTM. Lorenz return map of maxima of $x^{(3)}$ for Lorenz 63 (a) and Rössler 76 (b). Values computed from test dataset are shown in orange, while the LSTM results are shown in blue. The dashed line is the principal diagonal.

and Rössler systems. From Table 4.3 we see that the long term dynamics is better replicated for Lorenz 63, while the R2 scores are better for Rössler 76 in the first two variables ($x^{(1)}$ and $x^{(2)}$). The networks evidently fail to reproduce the short term behaviour of $x^{(3)}$ for Rössler 76. This model is notoriously difficult, due to the very erratic and unpredictable "jumps" on the plane $x^{(3)}$ that Equation 2.24 undergoes. However, in both models the percentage of R2 scores below zero, which correspond to very bad predictive performance, are relatively small. From Figure 4.2 we can appreciate that the chaotic behaviour of $x^{(3)}$ encoded is correctly reproduced for Lorenz 63, but not quite well for Rössler 76.

Hyper-Parameters Optimization

We perform a Bayesan optimization procedure with Optuna [Akiba et al., 2019]. We make use of a independently generated dataset of 5000, see items Table 4.1, corresponding to a dynamics of 10 seconds ($\delta t = 0.002$). The optimization target are the learning rate of

Adam optimizer [Kingma and Ba, 2014] and the sequence length with which dataset is divided. Since the dataset is small, at each epoch we split the dataset into 3 folds and thus conduct a 3-fold cross-validation [Stone, 1974, Mosteller and Tukey, 1968], see section 3.5, for 2000 epochs with Early Stopping [Zhang and Yu, 2005] with patience of 100 epochs. Found hyper-parameters are reported in Table 4.4, while the parallel coordinate plot is reported in Appendix A in Figure A.6. Training losses are in Figure A.4 in Appendix A. The other hyper-parameters is the same as reported in Table 4.2. As done before, we report the Wassersetein Distance (WD)and the R2 scores (see section 2.11) in Table 4.5. Again, we report the reproduced attractors colored with Local Lyapunov exponents in Figure 4.3 and the complete distribution of predicted LLE from the reproduced dynamics in Figure A.2. Lorenz return map is shown in Figure 4.4. Examples of short reconstructed trajectories are reported in Figure A.5.

System	Lorenz 63	Rössler 76
Learning rate	0.00036	0.0018
Sequence length	117	893

Table 4.4: Best hyper-parameters found for LSTM.

System		Lorenz 63	Rössler 76
WD($\cdot 10^{-4}$)	$x^{(1)} - x^{(2)}$	8.3805	31.38937
	$x^{(1)} - x^{(3)}$	11.6156	17.53575
	$x^{(2)} - x^{(3)}$	9.5591	20.0740
R2	$x^{(1)}$	0.81 $\pm$ 0.81	0.85 $\pm$ 0.27
	$x^{(2)}$	0.77 $\pm$ 0.83	0.76 $\pm$ 0.38
	$x^{(3)}$	0.89 $\pm$ 0.29	-47 $\pm$ 216
	< 0	3.76 %	11.55 %

Table 4.5: Statistics of optimized LSTM models. We report the Wasserstein Distance (WD) of the 2d projections of the invariant measure against the test dataset, the R2 scores for the short term dynamics along with the global percentage of those smaller than zero, see text.

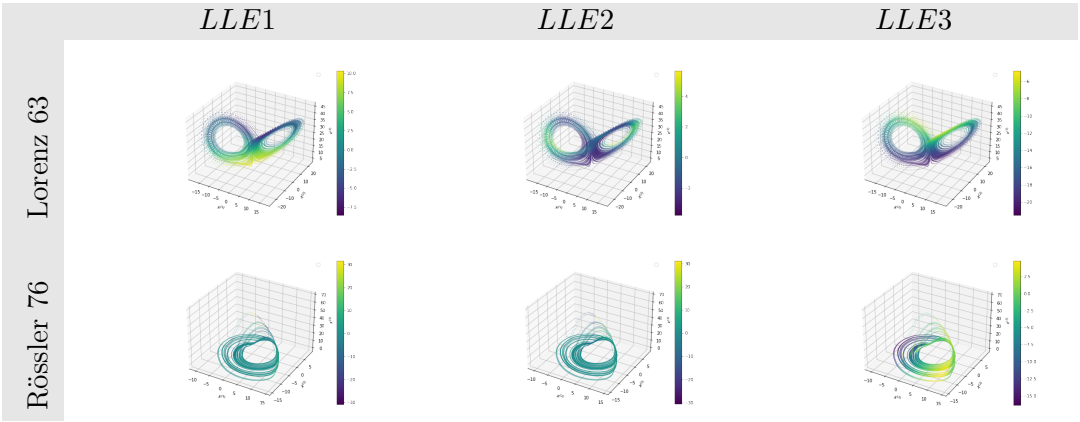
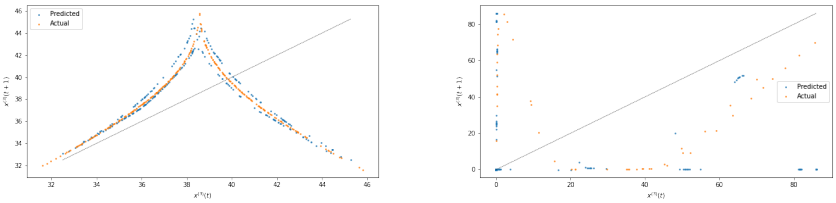


Figure 4.3: Optimized LSTM models. Predicted attractor colored with Local Lyapunov Exponents.

Also for the optimized LSTM models, as we expect, we successfully succeed in replication the chaotic attractor of both the model taken into consideration, as it is evident from Figure 4.3. Comparing Table 4.5 with Table 4.3, we can appreciate the better performance of the optimized LSTM models. We can see that for both models, the Wasserstein Distances decrease, symptom that the Climate is better replicated, but also the R2 scores (along with the percentage of scores below zero) are significantly better. This is expected, but also surprising in a certain measure, because the optimization procedure was 1) done on a very small dataset, in which the chaotic properties of the system could not evidently



(a) Lorenz 63 (b) Rössler 76

Figure 4.4: Optimized LSTM. Lorenz return map of maxima of $x^{(3)}$ for Lorenz 63 (a) and Rössler 76 (b). Values computed from test dataset are shown in orange, while the LSTM results are shown in blue. The dashed line is the principal diagonal.

emerge) and 2) the target of the optimization is the L2 losses between the true and the replicated trajectories of a given length. Indeed, as we can see from the optimized value of these lengths in Table 4.4, we can notice that the best value for Lorenz 63 is well below the typical chaotic time scale (≈ 1), while for Rössler it increases. In both cases, the fraction between the found value and the chaotic time scale is significantly greater than 1, which is a symptom that the LSTM internally encode somehow the dynamics of the systems and thus require less time steps to be fed of, being able to replicate the Climate as well. In line with this, we can notice from Figure 4.4 that the return map is successfully reproduced for Lorenz 63, but again not quite well for Rössler 76.

Physics Informed Loss

In this section, we analyse the case in which the network is trained to learn the evolution function $\mathbf{g}$ if the dynamical system in Equation 2.1, similarly to what done by [Raissi et al., 2018] and loosely resembling Physics Informed Networks [Raissi et al., 2019]. The loss is thus

$$\mathcal{L}(\theta) = \mathbb{E}(||RK_4[\mathbf{g}(\mathbf{x}_t)] - RK_4[\hat{\mathbf{g}}_\theta(\mathbf{x}_t)]||) \ , \tag{4.4}$$

where $\mathbf{g}(\mathbf{x}_t)$ is the true evolution function evaluated on the training sample, while $\hat{\mathbf{g}}_\theta(\mathbf{x}_t)$ is the neural network ansatz. In both cases, the Runge-Kutta 4 (RK_4) operator acts as

$$\begin{aligned} \mathbf{k}_1 &= \hat{\mathbf{g}}_\theta(\mathbf{x}_t) \ , \\ \mathbf{k}_2 &= \hat{\mathbf{g}}_\theta(\mathbf{x}_t + \delta t \ \mathbf{k}_1/2) \ , \\ \mathbf{k}_3 &= \hat{\mathbf{g}}_\theta(\mathbf{x}_t + \delta t \ \mathbf{k}_2/2) \ , \\ \mathbf{k}_4 &= \hat{\mathbf{g}}_\theta(\mathbf{x}_t + \delta t \ \mathbf{k}_3) \ , \\ RK_4[\hat{\mathbf{g}}_\theta(\mathbf{x}_t)] &= (\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)/6 \ . \end{aligned} \tag{4.5}$$

The latter Equation 4.5 operator can be directly substituted by a residual network, as done by [Fablet et al., 2018].

Training is performed again with Adam optimizer [Kingma and Ba, 2014] with learning rate 0.001. The dataset is the same as before in Table 4.1. We compare LSTM and Feed-Forward. Results for the latter model are reported only for Lorenz 63, because we did not succeed in obtaining any good result to show for Rössler 76 with Feed-Forward networks.

However, we point out that this is not the canonical Physics Informed Neural Network (PINN), since they are usually trained by only relying on the dynamical equations and not see data. Our approach only resemble PINN because we make use of the true evolution function. However, a true PINN would learn the dynamical equation by only having the time as input, thus learning the map $t \to \mathbf{x}_t$ of the dynamical system.

We build a LSTM model equal to the standard case analyses in the previous sections, i.e. Table 4.2. Statistical results are reported in Table 4.6, reproduced attractor colored with Local Laypunov Exponents (LLEs) in Figure 4.5, while Figure 4.6 the return maps for the maxima of $x^{(3)}$. Losses are reported in the last row of Figure A.4, while the complete distribution of LLE is reported in Figure A.3.

System		Lorenz 63	Rössler 76
WD($\cdot 10^{-4}$)	$x^{(1)} - x^{(2)}$	6.6412	21.2067
	$x^{(1)} - x^{(3)}$	7.5863	11.4657
	$x^{(2)} - x^{(3)}$	5.5518	13.2375
R2	$x^{(1)}$	0.85 $\pm$ 0.55	0.9945 $\pm$ 0.0095
	$x^{(2)}$	0.80 $\pm$ 0.64	0.992 $\pm$ 0.021
	$x^{(3)}$	0.90 $\pm$ 0.25	0.95 $\pm$ 0.10
	< 0	3.32 %	0.06 %

Table 4.6: Physics Informed LSTM models statistics. We report the Wasserstein Distance (WD) of the 2d projections of the invariant measure against the test dataset, the R2 scores for the short term dynamics along with the global percentage of those smaller than zero, see text.

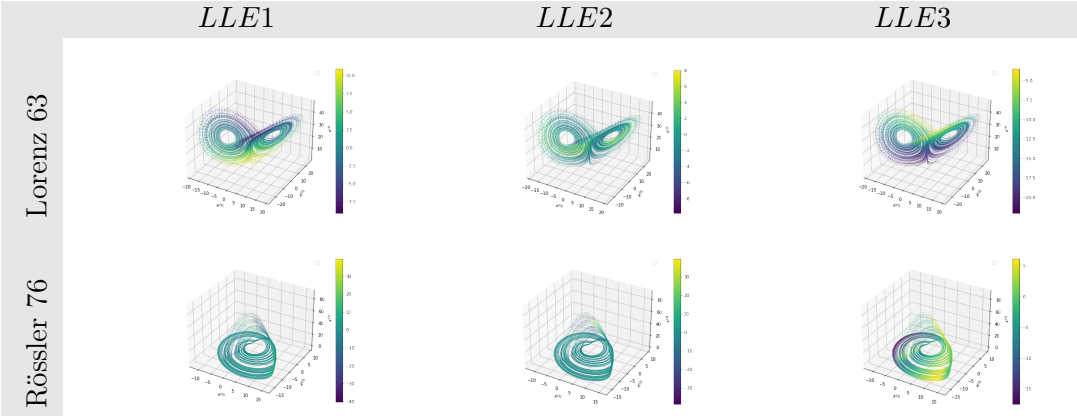
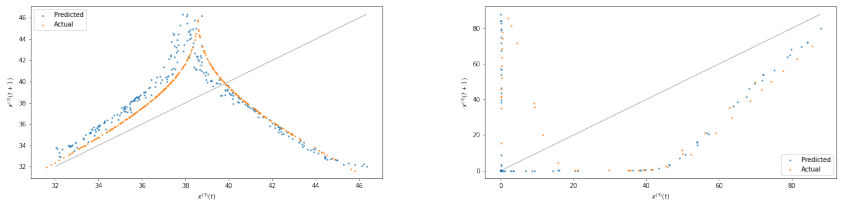


Figure 4.5: Physics Informed LSTM models. Predicted attractor colored with Local Lyapunov Exponents.



(a) Lorenz 63 (b) Rössler 76

Figure 4.6: Physics Informed LSTM. Lorenz return map of maxima of $x^{(3)}$ for Lorenz 63 (a) and Rössler 76 (b). Values computed from test dataset are shown in orange, while the LSTM results are shown in blue. The dashed line is the principal diagonal.

By comparing Table 4.6 with the Standards (Table 4.3) and Optimized (Table 4.5) LSTM models with purely Data-Driven loss, we can appreciate the better results for both Equation 2.43 and Equation 2.24. Not only the model with Physics Informed loss is able to perform better than the corresponding architecture trained with data-driven loss in Equation 4.3, but also better than the optimized data-driven model, in both the long term dynamics (expressed by the Wasserstein Distances), but also for the short term dynamics in the chaotic time scale (R2 scores). Moreover, we can also see how the return map in

Figure 4.6 is better replicated. Examples of short reconstructed trajectories are reported in Figure A.5.

Models Parameters Physics Informed Learning

PINNs can be either used to learn a dynamical equations (more often PDEs) as well as the parameters of such equations, see [Raissi et al., 2019]. This is known as the *inverse problem*. Inspired by this, we try to infer the parameters of Lorenz 63 model by using our Physics Informed loss kind of Equation 4.4, by adding these parameters in the computational graph of the Networks and thus allowing Backpropagation to update them too. However, after some trials the model did not converge as show in Figure 4.7.

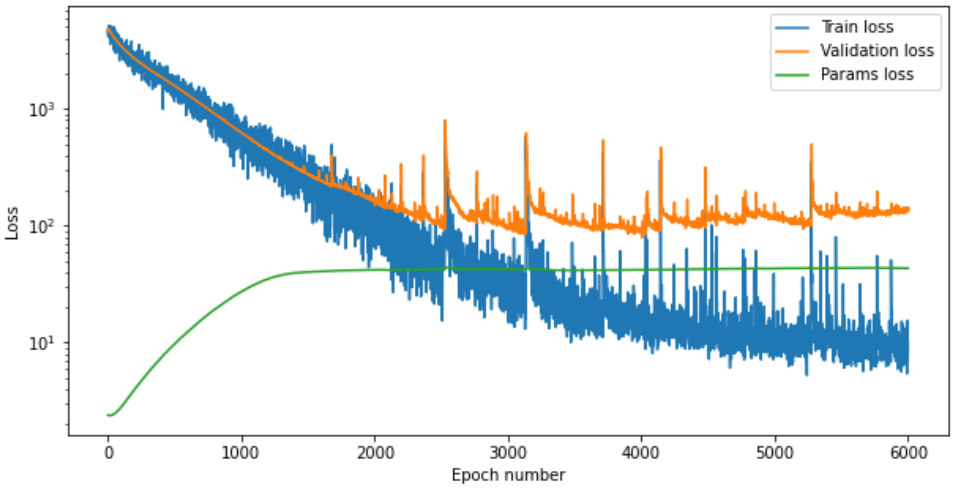


Figure 4.7: Physics Informed learning of the dynamics and parameters of Lorenz 63 system. The architecture is the same as Table 4.2 and the loss is Equation 4.4. We can see that the parameters loss (L2 loss between learned and true parameters) converges but not to 0.

4.2.3 Echo State Networks

In this section, we study the dynamics reproductive power of Echo State Networks (ESN), see subsection 3.9.1. Our architecture is slightly different from that used by [Pathak et al., 2017], which can be considered the pioneer in studying the possibility of chaotic dynamics reconstruction with Recurrent Networks. First, [Pathak et al., 2017] did feed the teacher forcing into the recurrent Equation 3.54. Secondly, regularization is done with Tikhonov procedure like Equation 3.59, while instead we prefer to add some noise, and the prediction output is not computed by extending the hidden state with the input. Moreover, [Ott, 2002] introduced an affine transformation to the recurrent state before applying the output matrix to take into account the $x \to -x$, $y \to -y$ symmetry of Lorenz 63, as discussed in [Lu et al., 2017]. Since this transformation is specifically designed to account for a symmetry in a specified model, we do not use it in order to compare different models with the same architecture. We leave this to further works.
In this thesis, we focus on the impact of sparsity in ESN and we analyze the case in which the recurrent matrix W is substituted with an Erdos-Reny graph, i.e. a sparse matrix consisting of only 0s and 1s. We refer to the first architecture as ESN1 and the second as ESN2, whose hyper-parameters are reported in Table 4.7 and Table 4.9 respectively. Notice that an Erdos-Reny graph does not have a spectral radius, since the elements of W are not rescaled.

Random Graph

In this section, we study the ESN1, i.e. a sparse random matrix with given spectral radius. After several attempts, we succeeded in build a model capable of learning the dynamics of Lorenz 63 system, reported in Table 4.7. However, this model fails in reproducing the attractor of Rössler 76, as visible in Figure 4.8. The summary statistics are reported in Table 4.8. Lorenz return maps are shown in Figure 4.9. Examples of short reconstructed trajectories are reported in Figure A.7.

Reservoir nodes	Spectral radius	Sparsity	Noise
500	1.5	0.9	0.1

Table 4.7: ESN1 (Random matrix W) hyper-parameters.

System		Lorenz 63	Rössler 76
WD($\cdot 10^{-4}$)	$x^{(1)} - x^{(2)}$	27.6676	53.52177
	$x^{(1)} - x^{(3)}$	28.6293	24.3787
	$x^{(2)} - x^{(3)}$	18.9467	26.4659
R2	$x^{(1)}$	0.80 ± 0.71	0.19±0.96
	$x^{(2)}$	0.75 ± 0.74	-0.24± 1.9
	$x^{(3)}$	0.85 ± 0.51	(-6.8 ± 31)·10^3
	< 0	4.18 %	42.27 %

Table 4.8: ESN1 random matrix statistics comparing Lorenz 63 with Rössler 76 systems. We compute as usual the Wasserstein Distance (WD) between 2d projected measures, the $R2$ scores along with the global percentage of those smaller than zero and the global percentage of those smaller than zero.

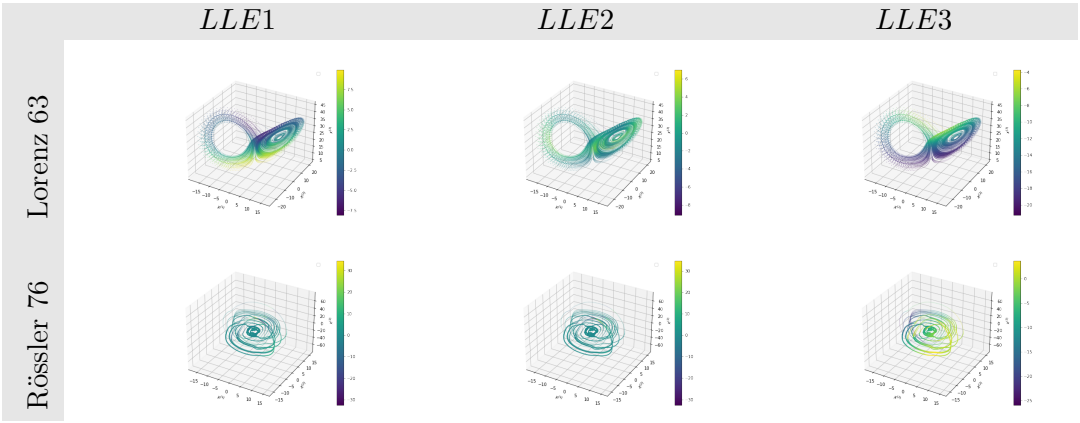
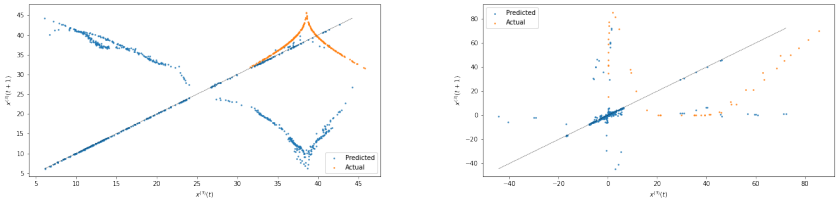


Figure 4.8: ESN1 Random matrix predicted attractor colored with Local Lyapunov Exponents.



(a) Lorenz 63 (b) Rössler 76

Figure 4.9: ESN1 Random matrix. Lorenz return map of maxima of $x^{(3)}$ for Lorenz 63 (a) and Rössler 76 (b). Values computed from test dataset are shown in orange, while the LSTM results are shown in blue. The dashed line is the principal diagonal.

From the statistics in Table 4.8 and the reproduced attractors in Figure 4.8 we can see that the model gives good results only for Lorenz 63, even though they are far worse than those obtained with LSTM in the previous sections. Moreover, it quite fails in reproducing the return map in Figure 4.9. This bad results, worse compared to those obtained by [Pathak et al., 2017] can be explained by considering that we did not perform the aforementioned affine transformation to take into account the $x \to -x$, $y \to -y$ symmetry of the systems. [Pathak et al., 2017] claim to have reproduced the short term behaviour of Lorenz 63, reproducing the dynamics beyond the first Lyapunov time. However, we are not aware of a systematic analysis of the short term prediction of the dynamics of such systems in the whole attractor, as we did.

Erdos-Reny Graph

We compare the two models Equation 2.43 and Equation 2.24 with a ESN network whose random matrix W is substituted by an Erdos-Reny graph [Erdös and Rényi, 1959], i.e. a random matrix consisting of only 1s and 0s, with a given sparsity. We call it ESN2. Now, matrix elements are not rescaled by the maximum eigenvalue, so that the spectral radius does not make sense. The complete architecture is reported in Table 4.9. Notice that the other hyper-parameters are equal to those of ESN1, see Table 4.7.

Reservoir nodes	Sparsity	Noise
500	0.9	0.1

Table 4.9: ESN2 architecture.

Statistical results are reported in Table 4.10 while the reader can find the predicted attractors in Figure 4.10 and the computed distribution of local Lyapunov exponents in Figure A.9. Lorenz return maps are reported in Figure 4.11. Examples of short reconstructed trajectories are reported in Figure A.7.

System		Lorenz 63	Rössler 76
WD($\cdot 10^{-4}$)	$x^{(1)} - x^{(2)}$	6.3884	228.7312
	$x^{(1)} - x^{(3)}$	11.58499	232.1972
	$x^{(2)} - x^{(3)}$	9.5545	255.0022
R2	$x^{(1)}$	-3.0 ± 7.3	-0.4 ± 1.6
	$x^{(2)}$	-2.5 ± 5.5	-2.6 ± 6.2
	$x^{(3)}$	-11 ± 30	-208 ± 600
	< 0	51.25 %	50.41 %

Table 4.10: ESN2 Erdos-Reny graph statistics comparing Lorenz 63 with Rössler 76 systems. We compute as usual the Wasserstein Distance (WD) between 2d projected measures, the $R2$ scores along with the global percentage of those smaller than zero and the global percentage of those smaller than zero.

Again, we can observe that the results are worse than LSTM models. Moreover, even if the Erdos-Reny graphs are able to replicate the attractor in Lorenz 63 system, they completely fail the R2 scores Table 4.10 and the return maps in Figure 4.11.

4.2.4 Feed-Forward Network

We apply a Feed-Forward Neural Network to Equation 2.43 with loss expressed in Equation 4.4, which is more robust since this kind of network are good at predicting functions and they are not dynamical systems. Direct application of these architecture to the problem at hand with loss given by Equation 4.3 does not give good results. The reasons are that these are not dynamical systems and the prediction phase differs substantially from the training phase, being outputs fed into the input in a loop way, while during training the

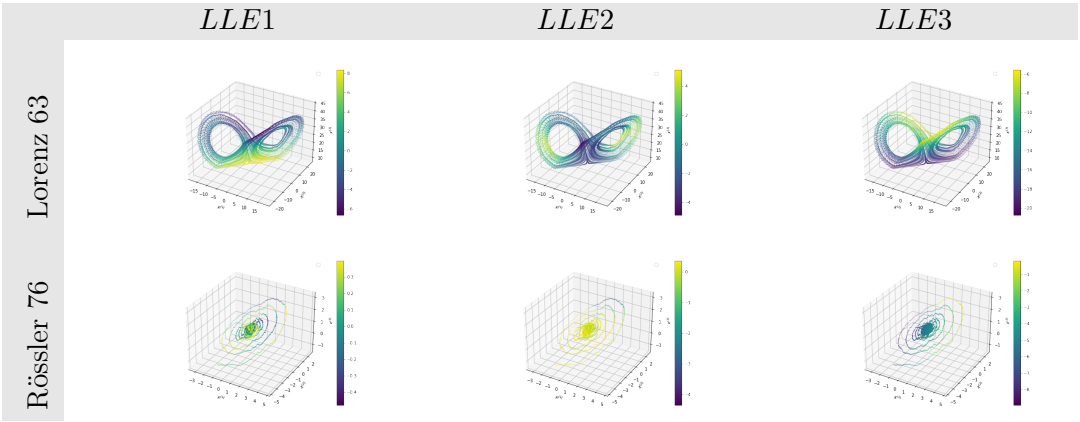
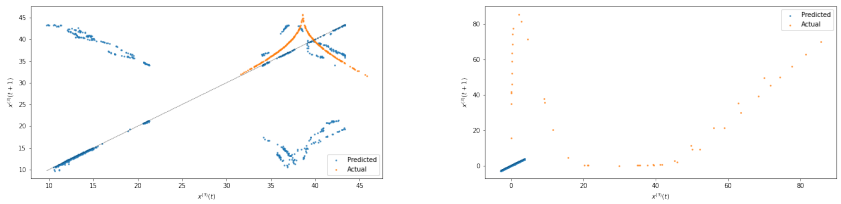


Figure 4.10: ESN2 Erdo-Reny graph. . Predicted attractor colored with Local Lyapunov Exponents.



(a) Lorenz 63 (b) Rössler 76

Figure 4.11: ESN2 Erdos-Reny graph. Lorenz return map of maxima of $x^{(3)}$ for Lorenz 63 (a) and Rössler 76 (b). Values computed from test dataset are shown in orange, while the LSTM results are shown in blue. The dashed line is the principal diagonal.

net learn only how to reproduce correctly the next time step. The architectures details are show in Table 4.11.

Sequence length	Hidden Layers	Neurons	Dropout	Hidden Operations
1	5	64	0.3	BN1d/ReLU/DO(O.3)

Table 4.11: Feed-Forward network architecture with Physics Informed kind loss. Input sequences are chosen to be of length 1 to give better results. BN mean Batch Normalization, while DO stands for DropOut.

We have tried also Convolutional Neural Networks with many different hyper-parameters combinations, but the results were not at all satisfactory. Statistical results are reported in Table 4.12. We decide to show only those for Lorenz 63, since the model does not give anything useful for Rössler 76, which is notoriously harder. In Figure 4.12 we show the reconstructed attractor colored with LLEs and in Figure 4.13 the Lorenz return map.

System		Lorenz 63
WD($\cdot 10^{-4}$)	$x^{(1)} - x^{(2)}$	15.6915
	$x^{(1)} - x^{(3)}$	23.7877
	$x^{(2)} - x^{(3)}$	14.19848
R2	$x^{(1)}$	0.09 $\pm$ 1.0
	$x^{(2)}$	-0.06 $\pm$ 1.1
	$x^{(3)}$	-0.09 $\pm$ 0.66
	< 0	38.05 %

Table 4.12: Physics Informed Feed-Forward network statistics. We report the Wasserstein Distance (WD) of the 2d projections of the invariant measure against the test dataset, the R2 scores for the short term dynamics along with the global percentage of those smaller than zero, see text.

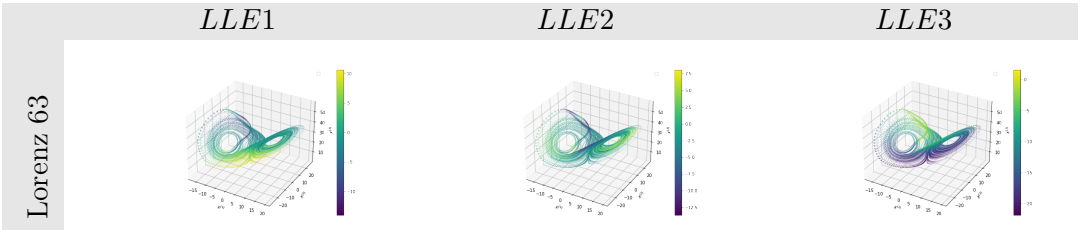


Figure 4.12: Physics Informed Feed-Forward network. Predicted attractor colored with Local Lyapunov Exponents.

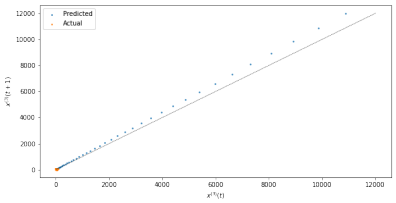


Figure 4.13: Physics Informed Feed-Forward network. Lorenz return map of maxima of $x^{(3)}$ for Lorenz 63. Values computed from test dataset are shown in orange, while the LSTM results are shown in blue. The dashed line is the principal diagonal.

From Table 4.12 and Figure 4.12 we can see that the network learn to reconstruct the attractor, the long term dynamics, but it completely fails in predicting the short term behaviour as well as the chaos of the maxima of $x^{(3)}$, which as pointed out by [Lorenz, 1963] plays an important role in studying the chaotic properties of this system.

4.3 Parameter Estimation with Regularised Autoencoders

In this section, we compare the power of Autoencoders directly applied to the dynamics to reconstruct the attractors and estimate models parameters by regularizing either the encoded space with a Physics Informed (PI) kind loss or a Data Driven (DD) loss, in which parameters are directly learned by the encoded representation. Denoted with $\hat{\mathbf{e}}$ the encoded representation of input (sequence) $\mathbf{x}$, the encoder part will compress the information of this sequence in order to extract the relevant parameters of the model. Therefore there is a one-to-one correspondence between input sequences in the batches fed to the encoder with the encoded parameters.
In the first case, we compute the regularization loss as

$$\mathcal{L}_{reg} = ||\mathbf{x}_{t+1} - \mathbf{x}_t - \delta t \; \mathbf{g}_{\hat{\mathbf{e}}}(\mathbf{x}_t)||^2 \; , \tag{4.6}$$

where $\mathbf{g}_{\hat{\mathbf{e}}}$ is the dynamical system evolution function Equation 2.1 when computed with the output of the decoder part.
In the second case, parameters are learned by direct minimization with the ground truth parameters denoted by $\mathbf{e}$, supposed to be known, with which the training dataset has been generated. Regularization loss is thus:

$$\mathcal{L}_{reg} = ||\hat{\mathbf{e}} - \mathbf{e}||^2 \; . \tag{4.7}$$

In both cases, the total training loss is

$$\mathcal{L}_S = \mathcal{L}_{rec} + \gamma \mathcal{L}_{reg} \; , \tag{4.8}$$

where γ is a hyper-parameter controlling the strength of regularization loss in the spirit of $\beta - VAEs$ [Higgins et al., 2017]. A different γ for Physics Informed and Data Driven loss is needed, since Equation 4.6 is at the order of magnitude δt, while DD regularization loss

is at the order of $\mathcal{L}_{rec}$ and therefore if we trained the autoencoders with the same $\gamma = 1$, the PI regularized would not learn any parameter in the encoded space.
As we have seen in section 3.10 an autoencoder acts as dimensionality reduction and therefore it can be used to learn summary statistics from high dimensional data in the spirit of [Jiang et al., 2017] and [Albert et al., 2022]. Of course a Neural Network could be trained to correctly learn the model parameters in a supervised way, but in this case the autoencoder has to balance the reconstruction loss with parameter learning loss, therefore acting as bottleneck to learn the most relevant features for reconstruction.
The motivation to compare this two regularization losses is clear. In most physical situations we do not know the parameters of the model, but only we can make clever guesses on the nature of the equations that govern the observed data and therefore a procedure that extract this information in a Physics Informed way is highly desirable.

A schematic comparison between the the two regularization procedures are reported in Figure 4.14.

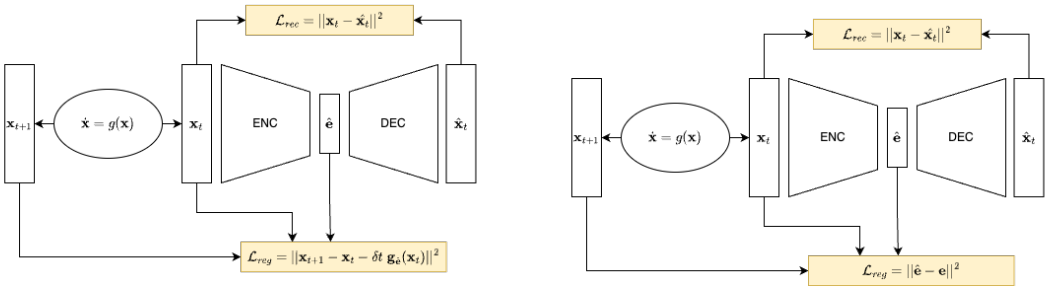


Figure 4.14: Schematic autoencoders architectures with a) Physics-Informed and 2) Data-Driven losses.

We compare two paradigmatic low dimensional chaotic models, namely Equation 2.43 and Equation 2.24, with two different architectures, i.e. a fully symmetric Convolutional Autoencoder and a fully symmetric Feed-Forward Autoencoder. The details of the architectures are reported in Table 4.13 (Feed-Forward autoencoder) and Table 4.14 (symmetric Convolutional autoencoder). After each layer, except output/input and encoded space layer, along with Dropout [Srivastava et al., 2014] and a ReLU activation function (see section 3.7), we perform Batch Normalization [Ioffe and Szegedy, 2015], which consist in normalizing data batch for batch with a given (learnable) mean and standard deviation in order to speed training up.

Input	Layer	Operations	Output Shape
	Input Sequence	/	(B, 500, 3)
Input Sequence	Flatten	/	(B, 1500)
Flatten	FC1-Enc	BN1d-DO(0.1)-ReLU	(B, 512)
FC1-Enc	FC2-Enc	BN1d-DO(0.1)-ReLU	(B, 256)
FC1-Enc	Enc Space	/	(B, 3)
Enc Space	F1C-Dec	BN1d-DO(0.1)-ReLU	(B, 256)
FC1-Dec	FC2-Dec	BN1d-DO(0.1)-ReLU	(B, 512)
FC2-Dec	FC3-Dec	/	(B, 1500)
FC3-Dec	Unflatten	/	(B, 500,3)

Table 4.13: Detailed architecture of the Feed-Forward autoencoder. The batch size B is 20. After each layer, except the input, output and the encoded space, 1d Batch Normalization (BN1d), DropOut (DO) with probability $p = 0.1$ and a ReLU activation function are sequentially applied.

Training is performed with Adam Optimizer [Kingma and Ba, 2014] with learning rate

Input	Layer	Kernels	Operations	Output shape
	Input sequence	/	/	(B, 1, 500, 3)
Input sequence	2dConv1	(5,3)	BN2d-ReLU-DO(0.1)	(B, 16, 496,1)
2dConv1	2dConv2	(5,1)	BN2d-ReLU-DO(0.1)	(B, 16, 492,1)
2dConv2	2dConv3	(5,1)	BN2d-ReLU-DO(0.1)	(B, 32, 488,1)
2dConv3	Flatten	/	/	(B, 15616)
Flatten	FC-Enc	/	BN1d-ReLU-DO(0.1)	(B, 256)
FC-Enc	Enc Space	/	/	(B,3)
Enc Space	FC1-Dec	/	DO(0.1)-ReLU-BN1d	(B, 256)
FC1-Dec	FC2-Dec	/	/	(B, 15616)
FC2-Dec	Unflatten	/	DO(0.1)-ReLU-BN2d	(B, 488,1)
Unflatten	2dUnConv1	(5,1)	DO(0.1)-ReLU-BN2d	(B, 492,1)
2dUnConv1	2dUnConv2	(5,1)	DO(0.1)-ReLU-BN2d	(B, 496,1)
2dUnConv2	2dUnConv3	(5,3)	/	(B, 500,3)

Table 4.14: Detailed architecture of symmetric Convolutional (symm-Conv) autoencoder. Batch size B is again 20, while BN stand for Batch Normalization and DO for DropOut (with probability $p = 0.1$).

$lr = 0.001$ until the reach of a plateau. Training and validation losses (including also regularization losses) are reported in Figure A.10 in Appendix A. The regularization strength for Physic Informed loss (left column) is chosen to be $\gamma = 100$ for Lorenz 63 and $\gamma = 1000$ for Rössler 76 in order to make the regularization loss of the same order of magnitude of the reconstruction loss.

To compare the performances of the models, in Table 4.15 for Lorenz 63 and Table 4.16 for Rössler 76 are reported the mean and the standard deviation of the distribution of the learned parameters, along with the compatibility λ between the predicted value and the true value. In general, given two noisy measures of the same quantity $x_1 = \mu_1 \pm \sigma_1$ and $x_2 = \mu_2 \pm \sigma_2$, the compatibility $\lambda_{1,2}$ between then is defined as

$$\lambda_{1,2} := \frac{|\mu_1 - \mu_2|}{\sqrt{\sigma_1^2 + \sigma_2^2}} \,. \tag{4.9}$$

To asses the reconstruction power, we report the Wasserstein Distance (WD) (see section 2.11) between the 2d projected distributions of the reconstructed attractor and the validation dataset, see Table 4.1.

From the compatibility with the true values in Table 4.15 we can see that in the Data Driven loss (DD) performs better in retrieving the model's parameters in the Convolutional Autoencoder, while for the Feed-Forward autoencoder the results are similar. However, this comes to a cost on the reconstruction loss, as we can see from the Wasserstein Distance (WD), typical for an autoencoder. Comparing the two architectures, we can see that the FeedForward autoencoder performs better in parameter retrieving with PI loss (with however a very spread distribution) and slightly worse in DD loss, while the symm-Conv is generally better in reproducing the attractor.

For Rössler 76 (Table 4.16), we can notice that Physics Informed (PI) regularization loss systematically fails, except for a, in retrieving the parameters, even if the Wassersetein Distances (WD) are substantially good. In this case, the symmetric Convolutional autoencoder seems to perform worse in both tasks. We notice that the Feed-Forward autoencoder is worse in reproducing the attractor and generally better in estimating the parameters.

From Table 4.15 and Table 4.16 we can argue that autoencoders succeed in parameters reconstruction while being able to replicate the attractor, see Figure 4.15. However, a Physics Informed regularization, more credible in research situations, seems to work generally worse.

	System	Lorenz 63	
	Loss	PI	DD
Symm-Conv	PAR σ	22.2 ± 9.1 ($\lambda = 0.64$)	27.6 ± 2.1 ($\lambda = 0.19$)
	PAR ρ	6.3 ± 5.7 ($\lambda = 0.65$)	9.8 ± 1.2 ($\lambda = 0.17$)
	PAR β	2.4 ± 1.6 ($\lambda = 0.17$)	2.72 ± 0.94 ($\lambda = 0.06$)
	WD ($\cdot 10^{-4}$) $x^{(1)} - x^{(2)}$	3.2823	7.0069
	WD ($\cdot 10^{-4}$) $x^{(1)} - x^{(3)}$	4.6435	9.6255
	WD ($\cdot 10^{-4}$) $x^{(2)} - x^{(3)}$	3.4329	8.1743
FeedForward	PAR σ	26.4± 4.8 ($\lambda = 0.33$)	27.5 ± 1.8 ($\lambda = 0.28$)
	PAR ρ	10.3 ± 2.5 ($\lambda = 0.12$)	9.79 ± 0.84 ($\lambda = 0.25$)
	PAR β	2.69 ± 0.88 ($\lambda = 0.03$)	2.60 ± 0.64 ($\lambda = 0.10$)
	WD ($\cdot 10^{-4}$) $x^{(1)} - x^{(2)}$	7.9489	11.3697
	WD ($\cdot 10^{-4}$) $x^{(1)} - x^{(3)}$	12.7950	16.5170
	WD ($\cdot 10^{-4}$) $x^{(2)} - x^{(3)}$	10.4477	13.5823

Table 4.15: Lorenz 63 system. Mean and standard deviations of the distribution for learned parameters (PAR) and the Wasserstein Distances (WD) between the 2d projections of the invariant measure of the reconstructed attractors and the test dataset Table 4.1, comparing Physics Informed (PI) and Data Driven (DD) losses for the symmetric Convolutional (Symm-Conv) and the Feed-Forward autoencoders; in parenthesis we report the compatibility with the true value λ, see text. The true values are $\sigma = 28$, $\rho = 10$ and $\beta = 8/3 \approx 2.667$.

	System	Rössler 76	
	Loss	PI	DD
Symm-Conv	PAR a	0.44 ± 0.67 ($\lambda = 0.10$)	0.30 ± 0.44 ($\lambda = 0.16$)
	PAR b	-0.6 ± 1.9 ($\lambda = 0.42$)	0.23 ± 0.49 ($\lambda = 0.06$)
	PAR c	2.6 ± 1.4 ($\lambda = 5.93$)	5.46 ± 0.83 ($\lambda = 0.29$)
	WD ($\cdot 10^{-4}$) $x^{(1)} - x^{(2)}$	16.5676	15.5188
	WD ($\cdot 10^{-4}$) $x^{(1)} - x^{(3)}$	6.0772	6.1758
	WD ($\cdot 10^{-4}$) $x^{(2)} - x^{(3)}$	7.2552	5.4430
FeedForward	PAR a	0.38 ± 0.53 ($\lambda = 0.02$)	0.36 ± 0.32 ($\lambda = 0.03$)
	PAR b	-1.0 ± 1.0 ($\lambda = 1.20$)	0.18 ± 0.28 ($\lambda = 0.07$)
	PAR c	4.19 ± 0.61 ($\lambda = 2.48$)	5.68 ± 0.35 ($\lambda = 0.06$)
	WD ($\cdot 10^{-4}$) $x^{(1)} - x^{(2)}$	19.5620	27.2989
	WD ($\cdot 10^{-4}$) $x^{(1)} - x^{(3)}$	11.8809	12.1241
	WD ($\cdot 10^{-4}$) $x^{(2)} - x^{(3)}$	13.3956	12.6825

Table 4.16: Rössler 76 system. Mean and standard deviations of the distribution for learned parameters (PAR) and the Wasserstein Distances (WD) between the 2d projections of the invariant measure of the reconstructed attractors and the test dataset Table 4.1, comparing Physics Informed (PI) and Data Driven (DD) losses for the symmetric Convolutional (Symm-Conv) and the Feed-Forward autoencoders; in parenthesis we report the compatibility with the true value λ, see text. The true values are $a = 0.37$, $b = 0.2$ and $c = 5.7$.

Further study will analyse deeper models in order to see if the results are more robust with more powerful methods.

In Figure 4.15 the reconstructed attractors are reported, while we refer to Appendix A for the learned parameters distribution in 3d Figure A.12, the full distributions in Figure A.13 (Lorenz 63) and Figure A.14 (Rössler 76) and the projected 2d invariant measures Figure A.11.

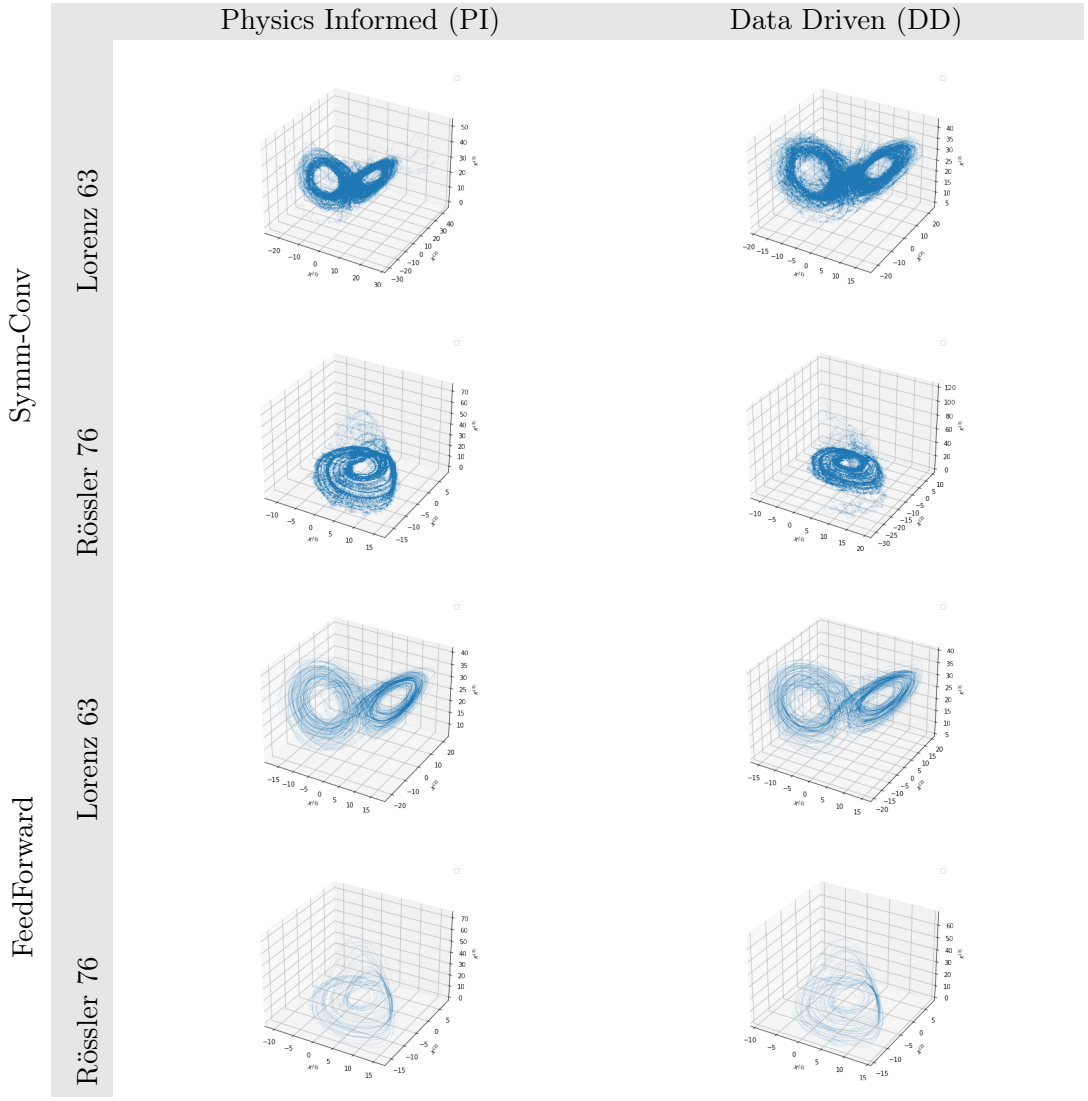


Figure 4.15: Reconstructed attractors for different Autoencoder models and systems

4.4 Unsupervised Learning of Forcing in Lorenz 96

In the previous section, we have addresses question 3 of section 1.2 by inferring the parameters of low dimensional models, i.e. [Lorenz, 1963] and [Rössler, 1976], by adding a regularization term in the encoded space. We have concluded that it is therefore possible to make parameter estimation in such way while maintaining the reconstruction capabilities of the autoencoder, even though we observed that a Physic Informed regularization kind is so far less efficient than the direct comparison with the true values. However, this approach is more meaningful in physical situations, where we do not know the true parameters.
In this section, we want to address question 3 in a complete unsupervised way, by using autoencoders to estimate the forcing term of a simple high dimensional dynamical system in Equation 2.50 that mimic the typical processes in the atmosphere [Lorenz, 1995]. We will see that, similarly to what done by [Wetzel, 2017] for the Ising 2d model [Onsager, 1944], autoencoders are able to learn a chaotic phase transition.

4.4.1 Normalized Input

As first experiment, we study directly the dynamics with normalized inputs. To do so, we generate a dataset of 1455 samples of Equation 2.50 with $n = 36$ sites with forcing equally spaced in the range $F \in [1, 15]$, well beyond in the chaotic regime. We divide the dataset as specified in Table 4.17. The time step to generate the dynamics is $\delta t = 0.002$ and the

first 200 time units (i.e. $200/0.002 = 100000$ samples) are discarded to assure to reach the attractor of the system. We take the successive 10 time units (5000 samples) as the training sample.

Total samples	Training	Validation	Test	δt
1455	1018	218	219	0.002

Table 4.17: Dataset division for experiment 2. The 1455 samples in the forcing interval $[1, 15]$ are randomly divided into a training, validation and test dataset with proportions 70/15/15%.

However, the complete dynamics information coming from samples of shape $(5000, 36)$ is too big to be reasonably fed into home networks. Therefore, data are reduced by a bi-linear interpolation along the time axis to have shape $(56, 36)$ and therefore be treated as images with one channel in the training algorithm. Samples are then normalized in the interval $[0, 1]$ to improve numerical stability, even though the dynamics has different length scale as F increases, being the maximum values roughly proportional to F. Random examples of the training dataset are reported in Figure A.15

We build a symmetric convolutional autoencoder with encoded space dimension of 1 in order to see if the model is able to capture the chaotic phase transition which occurs around $F = 5$. The model architecture is reported in Table 4.18. Training is performed with Adam [Kingma and Ba, 2014] with learning rate of 0.001 till convergence, monitored with Early Stopping with patience 100.

Input	Layer	Kernels	Operations	Output shape
	Input sequence	/	/	(B, 1, 56, 36)
Input sequence	2dConv1	(7,7)	BN2d-ReLU-DO(0.1)	(B, 32, 50, 30)
2dConv1	MaxPool1	(2,2)	/	(B, 32, 25, 15)
MaxPool1	2dConv2	(4,4)	BN2d-ReLU-DO(0.1)	(B, 64, 22, 12)
2dConv2	MaxPool2	(2,2)	/	(B, 64, 11, 6)
MaxPool2	Flatten	/	/	(B, 4224)
Flatten	FC1-Enc	/	BN1d-ReLU-DO(0.1)	(B, 512)
FC1-Enc	FC2-Enc	/	BN1d-ReLU-DO(0.1)	(B,256)
FC2-Enc	Enc Space	/	/	(B, 1)
Enc Space	FC1-Dec	/	DO(0.1)-ReLU-BN1d	(B, 256)
FC1-Dec	FC2-Dec	/	DO(0.1)-ReLU-BN1d	(B, 512)
FC2-Dec	Unflatten	/	DO(0.1)-ReLU-BN2d	(B, 64, 11, 6)
Unflatten	UpSampling1	(2,2)	/	(B,64, 22,12)
UpSampling1	2dUnConv1	(4,4)	DO(0.1)-ReLU-BN2d	(B, 32, 25,15)
2dUnConv1	UpSampling2	(2,2)	/	(B, 32, 50,30)
UpSampling2	2dUnConv2	(7,7)	/	(B, 1, 56,36)

Table 4.18: Detailed architecture of symmetric convolutional autoencoder. Batch size B is 16, while BN stand for Batch Normalization and DO for DropOut (with probability $p = 0.1$).

In Figure 4.16 the encoded representation of the test dataset is reported in function of the forcing F, while in Figure 4.17 the reader can find generated samples from the encoded representation in the interval $[-20, 120]$.

Reconstruction samples taken randomly from the test dataset, see Table 4.17, are reported in Figure 4.18.In Figure A.16 we report the convolutional filters of the first layer.

From Figure 4.16 and Figure 4.17 it is evident how the model is able to separate the ordered phase, in which the model exhibits travelling waves, with the chaotic phase, while maintaining a good reconstruction power, see Figure 4.18. However, the model is not quite able to tell the magnitude of F in the chaotic regime, arguably because data were normalized in the unit interval and thus all the information coming form the different scales

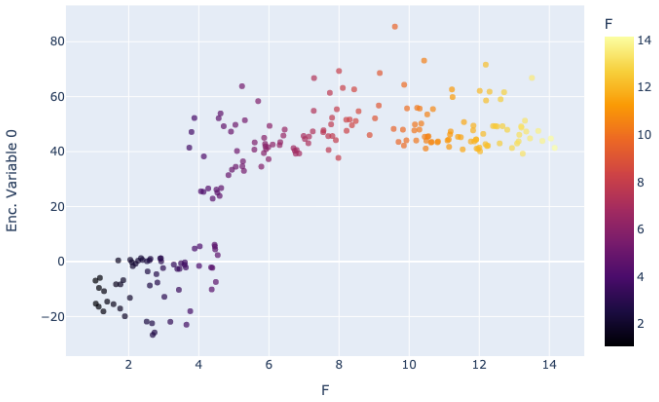


Figure 4.16: Normalized Inputs. Encoded representation of the test dataset in function of the forcing F. Values are colored according to the magnitude of F.

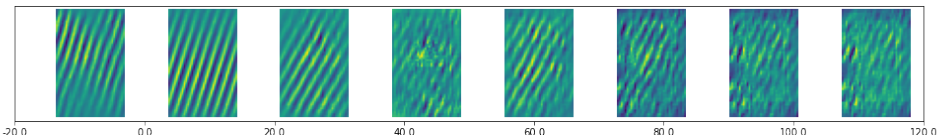


Figure 4.17: Normalized Inputs. Samples generated from the encoded space representation in the interval $[-20, 120]$. We can appreciate a transition between order (left) and chaos (right).

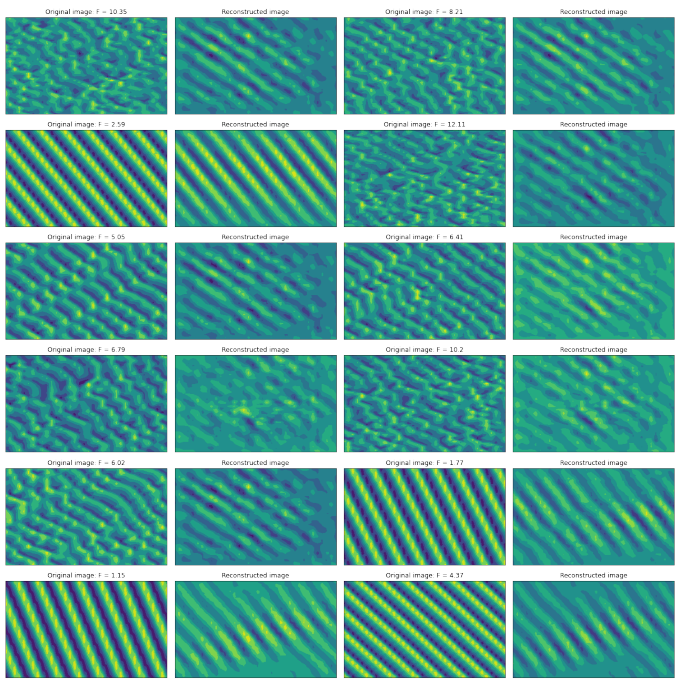


Figure 4.18: Normalized Inputs. Reconstruction of random samples of the test dataset. In the first and third columns the original images are reported, while in columns second and fourth one can find the corresponding reconstructions.

of the dynamics has been lost.

4.4.2 Non-Normalized Inputs

Now, we do not normalized input samples in the interval $[0, 1]$, but rather in the interval $[-0.5, 0.5]$ by scaling every image by the maximum values reached by the various dynamics for $F \in [1, 15]$, such that the relative sizes matter. The architecture is the same as Table 4.18 and the other hyper-parameters specification are the same. As before, we report the encoded space representation in Figure 4.19, the generation of samples starting from the encoded representation in Figure 4.20 and some random reconstructed samples in Figure 4.21. In the appendix we report the first convolutional filters in Figure A.16.

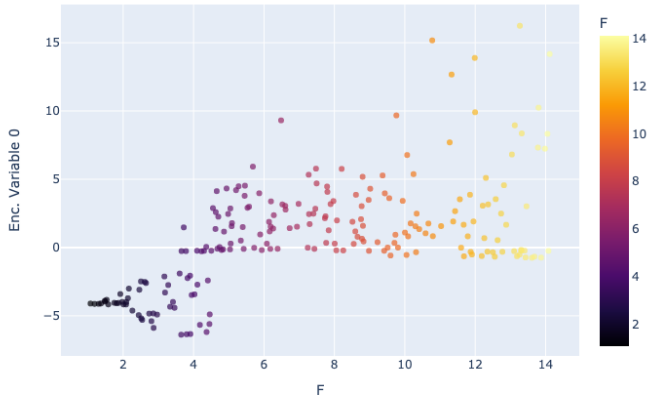


Figure 4.19: Non-normalized Inputs. Encoded representation of the test dataset in function of the forcing F for the dataset rescaled by the maximum values attained by the dynamics in the whole forcing interval. Values are colored according to the magnitude of F.

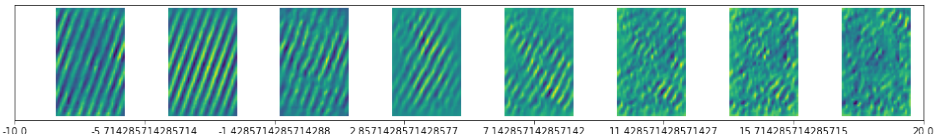


Figure 4.20: Non-normalized Inputs. Samples generated from the encoded space representation in the interval $[-10, 20]$. We can appreciate a transition between order (left) and chaos (right).

From Figure 4.19 and Figure 4.20 we can appreciate a phase separation, with an increased attention towards larger values of the forcing F. However, even though at large F the model seem to have attained a larger variability, we cannot clearly notice a neat proportionality between the encoded variable and the forcing. Furthermore, even the reconstruction in Figure 4.21 seems worse, since numerically there is more instability due to the chosen scaling of the dataset.
We conclude by remarking that experiments with Variational Autoencoders, see section 3.10, did not give meaningful results.

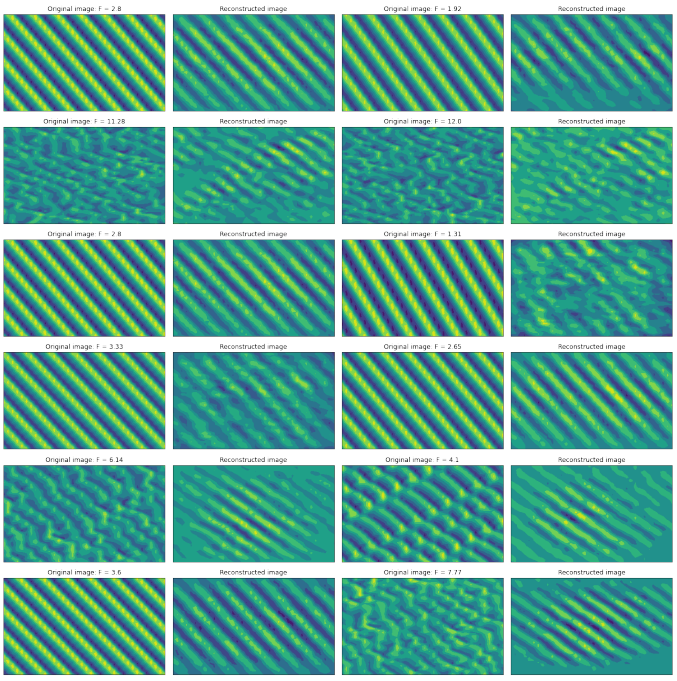


Figure 4.21: Non-normalized Inputs. Reconstruction of random samples of the test dataset for the model with samples rescaled by the maximum values attained by the dynamics in the whole forcing interval. In the first and third columns the original images are reported, while in columns second and fourth one can find the corresponding reconstructions.

Chapter 5

Conclusions and Future Work

In this thesis, we have studied the applications of Neural Networks to the realm of dynamics systems, being particularly interested to answer to the three questions of section 1.2.
In chapter 2 we have exposed the basic theory of dynamical systems, focusing more attention on the chaotic properties of such systems and giving the fundamental tools the one needs to study them. Moreover, we analyse some examples in detail, such the Logistic Map of Equation 2.17 and Lorenz 63 system Equation 2.43.
In chapter 3 we have exposed the basics theory of Machine Learning, with particular attention to develop a formal learning framework. Even if this was done with respect supervised learning, which is not the main target of this thesis, most conclusion remains still valid. In particular, we have seen the importance of dividing the dataset into a training and validation part in order to avoid overfitting, which is one of the main issues in Machine Learning. Moreover, we have presented some typical architectures often used in the Deep Learning community, justifying how they can be used to analyse dynamical sequences.
In chapter 4 we have conducted experiments with different Machine Learning models, such as Recurrent Neural Networks (LSTM and ESN), standard Feed-Forward networks, Convolutional Networks and Autoencoders. We were interested into the three questions mentioned in section 1.2 in the Introduction, which we believe they have been successfully answered. We have shown that Neural Networks can:

1. Reproduce the Weather (short term dynamics) of chaotic dynamical systems.
2. Reproduce the Climate (long term dynamics) of these systems.
3. Be used to retrieve the parameters of the model from data.
4. Learn a chaotic phase transition.

In particular, the main conclusions that we got to are:

- LSTM are efficient learners for both the model taken at hand and are very reliable because they do not need an excessive fine tuning.
- Echo State and Feed-Forward networks can in principle be used to reproduce a chaotic dynamics, but they are very sensitive to the choice of the hyper-parameters.
- A Physics-Informed kind of loss is more efficient and robust that a purely Data-Driven loss for LSTM models for both the low dimensional systems taken into consideration.
- Physics Informed regularization can be used to learn models parameters in Convolutional and Feed-Forward symmetric Autoencoders, maintaining good reproduction capabilities. Even though Physics-Informed regularization is more useful in situations where we cannot access the true parameters, Data-Driven regularization seems to work better.

- Convolutional symmetric Autoencoders can successfully learn a chaotic phase transition in Lorenz 96 model, when trained in a completely unsupervised way. The latent space is organized to separate low forces against larger forcing. However, the model cannot tell the exact magnitude of the forcing.

These results were obtained by training the networks either on a home computer or on the free Colab account, which does not reserve very powerful GPUs. Moreover, training did not take more than 2%3 hours for each model. Therefore, we believe that the first way to improve the results of the thesis is to train the model on more powerful devices or for more time.

Other lines of future research shall include:

- An analysis of ESN with Erdos-Reny graph by taking into account the symmetries of the model as done by [Pathak et al., 2017].
- A purely Physics-Informed approach to learn the chaotic systems, i.e. feeding into the networks only the time step and relying only on the dynamical equations, without seeing any generated data.
- An extension of Autoencoders analysis to the case of sparse and rare data, which is much more common in practical situations.
- An study on the applications of Autoencoders to Linear Response Theory. in the context of dynamical systems.
- Decoupling the effect of slow variable from fast variable in high dimensional models that include more than one atmospheric quantity, like [Vissio and Lucarini, 2020].

References

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., and et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[Adler et al., 1965] Adler, R., Konheimand, A., and McAndrew, M. (1965). Topological entropy. *Trans. Am. Math. Soc.*, 114:309.

[Akiba et al., 2019] Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A Next-generation Hyperparameter Optimization Framework. *arXiv e-prints*, page arXiv:1907.10902.

[Al-Rfou et al., 2016] Al-Rfou, R., Alain, G., Almahairi, A., Angermüller, C., Bahdanau, D., Ballas, N., and et al. (2016). Theano: A python framework for fast computation of mathematical expressions. *CoRR*, abs/1605.02688.

[Albert et al., 2022] Albert, C., Ulzega, S., Ozdemir, F., Perez-Cruz, F., and Mira, A. (2022). Learning summary statistics for bayesian inference with autoencoders. *arXiv:2201.12059*.

[Almazova et al., 2021] Almazova, N., Barmparis, G. D., and Tsironis, G. P. (2021). Analysis of chaotic dynamical systems with autoencoders. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31(10):103109.

[Ayers et al., 2021] Ayers, D., Lau, J., Amezcua, J., Carrassi, A., and Ojha, V. (2021). Supervised machine learning to estimate instabilities in chaotic systems: estimation of local lyapunov exponents. *arXiv:2202.04944*.

[Baladi, 2000] Baladi, V. (2000). *Positive Transfer Operators and Decay of Correlations*. WORLD SCIENTIFIC.

[Balatoni and Renyi, 1956] Balatoni, J. and Renyi, A. (1956). *Pub. Math. Inst. Hungarian Acad. Sci.*, 1:9.

[Barzegar et al., 2022] Barzegar, V., Laflamme, S., Hu, C., and Dodson, J. (2022). Ensemble of recurrent neural networks with long short-term memory cells for high-rate structural health monitoring. *Mechanical Systems and Signal Processing*, 164:108201.

[Baydin et al., 2017] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2017). Automatic differentiation in machine learning: A survey. *J. Mach. Learn. Res.*, 18(1):5595–5637.

[Beck and Schlögl, 1997] Beck, C. and Schlögl, F. (1997). *Thermodynamics of Chaotic System*. Cambridge University Press, Cambridge, UK.

[Benettin et al., 1980] Benettin, G., Galgani, L., Giorgilli, A., and al, e. (1980). Lyapunov characteristic exponents for smooth dynamical systems and for hamiltonian systems; a method for computing all of them. part 1: Theory. *Meccanica*, 15:9–20.

[Bengio et al., 2015] Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 1171–1179, Cambridge, MA, USA. MIT Press.

[Bianchi et al., 2018] Bianchi, F. M., Scardapane, S., Løkse, S., and Jenssen, R. (2018). Reservoir computing approaches for representation and classification of multivariate time series. *IEEE Transactions on Neural Networks and Learning Systems*, PP.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

[Bjerhammar, 1951] Bjerhammar, A. (1951). Application of calculus of matrices to method of least squares; with special references to geodetic calculations. *Trans. Roy. Inst. Tech. Stockholm.*, 49.

[Bottou et al., 2018] Bottou, L., Curtis, F. E., and Nocedal, J. (2018). Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311.

[Boureau et al., 2010] Boureau, Y.-L., Ponce, J., and LeCun, Y. (2010). A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 111–118, Madison, WI, USA. Omnipress.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine Learning*, 45:5–32.

[Bridle, 1989] Bridle, J. S. (1989). Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In *Proceedings of the 2nd International Conference on Neural Information Processing Systems*, NIPS'89, page 211–217, Cambridge, MA, USA. MIT Press.

[Brunton et al., 2016] Brunton, S. L., Proctor, J. L., and Kutz, J. N. (2016). Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937.

[Cencini et al., 2009] Cencini, M., Cecconi, F., and Vulpiani, A. (2009). *Chaos*. World Scientific.

[Chandrasekhar, 1943] Chandrasekhar, S. (1943). Stochastic problems in physics and astronomy. *Rev. Mod. Phys.*, 15:1–89.

[Chollet, 2015] Chollet, F. (2015). keras. `https://github.com/fchollet/keras`.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20:273–297.

[Coulombe et al., 2017] Coulombe, J. C., York, M. C. A., and Sylvestre, J. (2017). Computing with networks of nonlinear mechanical oscillators. *PLOS ONE*, 12(6):1–13.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314.

[Demis et al., 2015] Demis, E., Aguilera, R., Sillin, H., Scharnhorst, K., Sandouk, E., Aono, M., Stieg, A., and Gimzewski, J. (2015). Atomic switch networks—nanoarchitectonic design of a complex system for natural computing. *Nanotechnology*, 26(20):204003.

[Deng, 2012] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142.

[Donahue et al., 2015] Donahue, C., Merkel, C., Saleh, Q., Dolgovs, L., Ooi, Y. K., Kudithipudi, D., and Wysocki, B. (2015). Design and analysis of neuromemristive echo state

networks with limited-precision synapses. In *2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, pages 1–6.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12(null):2121–2159.

[Eckmann and Ruelle, 1985] Eckmann, J. and Ruelle, D. (1985). Ergodic theory of chaos and strange attractors. *Rev. Mod. Phys.*, 57:617.

[Erdös and Rényi, 1959] Erdös, P. and Rényi, A. (1959). On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290.

[Fablet et al., 2018] Fablet, R., Ouala, S., and Herzet, C. (2018). Bilinear residual neural network for the identification and forecasting of geophysical dynamics. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 1477–1481.

[Fan et al., 2020] Fan, H., Jiang, J., Zhang, C., Wang, X., and Lai, Y.-C. (2020). Long-term prediction of chaotic systems with machine learning. *Phys. Rev. Research*, 2:012080.

[Feller, 1968] Feller, W. (1968). *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley.

[Fokker, 1914] Fokker, A. D. (1914). Die mittlere energie rotierender elektrischer dipole im strahlungsfeld. *Ann. Phys.*, 348:810–820.

[Freiberger et al., 2020] Freiberger, M., Bienstman, P., and Dambre, J. (2020). A training algorithm for networks of high-variability reservoirs. *Sci Rep*, 10:14451.

[Galias and Tucker, 2008] Galias, Z. and Tucker, W. (2008). Short periodic orbits for the lorenz system. pages 285 – 288.

[Gardiner, 2004] Gardiner, C. W. (2004). *Handbook of stochastic methods for physics, chemistry and the natural sciences*, volume 13 of *Springer Series in Synergetics*. Springer-Verlag, Berlin, third edition.

[Gilpin, 2020] Gilpin, W. (2020). Deep reconstruction of strange attractors from time series. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA. Curran Associates Inc.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

[Gneiting et al., 2012] Gneiting, T., Ševčíkováand, H., and Percival, Donald, B. (2012). Estimators of fractal dimension: Assessing the roughness of time series and spatial data. *Statistical Science*, 27 (2):247–277.

[Goldstein et al., 2002] Goldstein, H., Poole, C., and Safko, J. (2002). *Classical Mechanics*. Addison- Wesley, 3rd edition.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[Gottwald and Reich, 2021] Gottwald, G. A. and Reich, S. (2021). Supervised learning from noisy observations: Combining machine-learning techniques with data assimilation. *Physica D: Nonlinear Phenomena*, 423:132911.

[Grassberger, 1983] Grassberger, P. (1983). Generalized dimensions of strange attractors. *Phys.Lett.A*, 97:227.

[Haluszczynski and Räth, 2021] Haluszczynski, A. and Räth, C. (2021). Controlling nonlinear dynamical systems into arbitrary states using machine learning. *Sci Rep*, 11:12991.

[Hausdorff, 1919] Hausdorff, F. (1919). Dimension und äußeres maß. *Mathematische Annalen*, 79 (1–2):157–179.

[Hauser et al., 2011] Hauser, H., Ijspeert, A., Füchslin, R., and et al. (2011). Towards a theoretical foundation for morphological computation with compliant bodies. *Biol Cybern*, 105:355–370.

[Haykin, 1994] Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034.

[Heinisch et al., 1962] Heinisch, O., Steel, R. G. D., and Torrie, J. H. (1962). Principles and procedures of statistics. (with special reference to the biological sciences). *Biometrische Zeitschrift*, 4(3):207–208.

[Hentschel and Procaccia, 1983] Hentschel, H. G. E. and Procaccia, I. (1983). The infinite number of generalized dimensions of fractals and strange attractors. *Physica D Nonlinear Phenomena*, 8(3):435–444.

[Higgins et al., 2017] Higgins, I., Matthey, L., Pal, A., Burgess, C. P., Glorot, X., Botvinick, M. M., Mohamed, S., and Lerchner, A. (2017). beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*.

[Hinton et al., 2012] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., and Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97.

[Hinton, 2002] Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800.

[Hinton and Ackley, 1985] Hinton, G. E. and Ackley, D. H. (1985). A learning algorithm for boltzmann machines. *Cognitive Science*, 9:147–169.

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

[Hornik, 1991] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257.

[Hénon, 1976] Hénon, M. (1976). A two-dimensional mapping with a strange attractor. *Commun. Math. Phys*, 50:69.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448–456. JMLR.org.

[Jaeger, 2001] Jaeger, H. (2001). The echo state approach to analysing and training recurrent neural networks.

[James et al., 2013] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R*. Springer.

[Jiang et al., 2017] Jiang, B., Wu, T.-Y., Zheng, C., and Wong, W. H. (2017). Learning summary statistic for approximate bayesian computation via deep neural network. *Statistica Sinica*, 27(4):1595–1618.

[Jones et al., 2007] Jones, B., Stekel, D., Rowe, J., and Fernando, C. (2007). Is there a liquid state machine in the bacterium escherichia coli? In *2007 IEEE Symposium on Artificial Life*, pages 187–191.

[Kantorovich, 1939] Kantorovich, L. (1939). Mathematical methods of organizing and planning production. *Management Science*, 6(4):366–422.

[Kaplanand and Yorke, 1979] Kaplanand, J. and Yorke, J. (1979). Chaotic behavior of multidimensional difference equations. In H.O., P. and H.O., W., editors, *Functional Differential Equations and Approximations of Fixed Points*, volume 730, page 204. Springer (Berlin).

[Kerin and Engler, 2022] Kerin, J. and Engler, H. (2022). On the lorenz '96 model and some generalizations. *Discrete and Continuous Dynamical Systems - B*, 27(2):769–797.

[Khintchine, 1934] Khintchine, A. (1934). Korrelationstheorie der stationären stochastischen prozesse. *Mathematische Annalen*, 109 (1):604–615.

[Kingma and Ba, 2014] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

[Kolmogorov, 1958] Kolmogorov, A. (1958). A new metric invariant of transitive dynamical systems and automorphisms in lebesgue spaces. *Dokl. Acad. Nauk SSSR*, 119:861.

[Kolmogorov, 1961] Kolmogorov, A. (1961). On the representation of continuous functions of several variables by superpositions of continuous functions of fewer variables. *Dokl. Akad. Nauk SSSR*, 108:179–182.

[Kramer, 1991] Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

[Krogh and Hertz, 1991] Krogh, A. and Hertz, J. (1991). A simple weight decay can improve generalization. In Moody, J., Hanson, S., and Lippmann, R., editors, *Advances in Neural Information Processing Systems*, volume 4. Morgan-Kaufmann.

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86.

[Larger et al., 2012] Larger, L., Soriano, M. C., Brunner, D., Appeltant, L., Gutierrez, J., Pesquera, L., Mirasso, C. R., and Fischer, I. (2012). Photonic information processing beyond turing: an optoelectronic implementation of reservoir computing. *Opt. Express*, 20(3):3241–3249.

[Lasota and Mackey, 1985] Lasota, A. and Mackey, M. C. (1985). *Probabilistic Properties of Deterministic Systems*. Cambridge University Press.

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1989). Handwritten digit recognition with a back-propagation network. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann.

[Lederer, 2021] Lederer, J. (2021). Activation Functions in Artificial Neural Networks: A Systematic Overview. *arXiv e-prints*, page arXiv:2101.09957.

[Leung et al., 2014] Leung, M. K. K., Xiong, H. Y., Lee, L. J., and Frey, B. J. (2014). Deep learning of the tissue-regulated splicing code. *Bioinformatics*, 30(12):i121–i129.

[Lorenz, 1995] Lorenz, E. (1995). Predictability: a problem partly solved. In *Seminar on Predictability, 4-8 September 1995*, volume 1, pages 1–18, Shinfield Park, Reading. ECMWF, ECMWF.

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

[Lu et al., 2017] Lu, Z., Pathak, J., Hunt, B., Girvan, M., Brockett, R., and Ott, E. (2017). Reservoir observers: Model-free inference of unmeasured variables in chaotic systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(4):041102.

[Lukoeviius and Jaeger, 2009] Lukoeviius, M. and Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Comput. Sci. Rev.*, 3:127–149.

[Lynch, 2008] Lynch, P. (2008). The origins of computer weather prediction and climate modeling. *Journal of Computational Physics*, 227(7):3431–3444. Predicting weather, climate and extreme events.

[Maass et al., 2002] Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Comput.*, 14(11):2531–60.

[Mallat, 2016] Mallat, S. (2016). Understanding deep convolutional networks. *Trans. R. Soc.*

[Mandelbrot, 1982] Mandelbrot, B. B. (1982). *The fractal geometry of nature*. W. H. Freeman and Co.

[May, 1976] May, R. M. (1976). Simple mathematical models with very complicated dynamics. *Nature*, 261:459.

[Mehta et al., 2019] Mehta, P., Bukov, M., Wang, C.-H., Day, A. G., Richardson, C., Fisher, C. K., and Schwab, D. J. (2019). A high-bias, low-variance introduction to machine learning for physicists. *Physics Reports*, 810:1–124. A high-bias, low-variance introduction to Machine Learning for physicists.

[Monge, 1781] Monge, G. (1781). Mémoire sur la théorie des déblais et des remblais. *Histoire de l'Acadmie Royale des Sciences*, page 666–704.

[Monin and Yaglom, 1975] Monin, A. and Yaglom, A. (1975). *Statistical Fluid Dynamics*. MIT Press, Cambridge MA.

[Moore,] Moore, E. H. On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society*, 26(9):394–95.

[Mosteller and Tukey, 1968] Mosteller, F. and Tukey, J. W. (1968). Data analysis, including statistics. In Lindzey, G. and Aronson, E., editors, *Handbook of Social Psychology, Vol. 2*. Addison-Wesley.

[Murphy, 2013] Murphy, K. P. (2013). *Machine learning : a probabilistic perspective*. MIT Press, Cambridge, Mass. [u.a.].

[Murthy et al., 2020] Murthy, D., Allu, S., Andhavarapu, B., and Bagadi, M. (2020). Text based sentiment analysis using lstm. *International Journal of Engineering Research and*, V9.

[Nakajima, 2020] Nakajima, K. (2020). Physical reservoir computing–an introductory perspective. *Japanese Journal of Applied Physics*, 59.

[Nakane et al., 2018] Nakane, R., Tanaka, G., and Hirose, A. (2018). Reservoir computing with spin waves excited in a garnet film. *IEEE Access*, 6:4462–4469.

[Nam and Kang, 2021] Nam, J. and Kang, J. (2021). Classification of chaotic signals of the recurrence matrix using a convolutional neural network and verification through the lyapunov exponent. *Applied Sciences*, 11(1).

[Narkhede et al., 2022] Narkhede, M. V., Bartakke, P. P., and Sutaone, M. S. (2022). A review on weight initialization strategies for neural networks. *Artificial Intelligence Review*, 55:291–322.

[Nguyen et al., 2020] Nguyen, D., Ouala, S., Drumetz, L., and Fablet, R. (2020). Variational deep learning for the identification and reconstruction of chaotic and stochastic dynamical systems from noisy and partial observations. *arXiv:2009.02296*.

[Nowlan and Hinton, 1992] Nowlan, S. J. and Hinton, G. E. (1992). Simplifying Neural Networks by Soft Weight-Sharing. *Neural Computation*, 4(4):473–493.

[Onsager, 1944] Onsager, L. (1944). Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Phys. Rev.*, 65:117–149.

[Oseledets, 1968] Oseledets, V. (1968). A multiplicative ergodic theorem. lyapunov characteristic numbers for dynamical systems. *Trans. Moscow Math. Soc.*, 19:197–231.

[Ott, 2002] Ott, E. (2002). *Chaos in Dynamical Systems*. Cambridge University Press, 2 edition.

[Park and Xu, 2016] Park, S. and Xu, L. (2016). *Data Assimilation for Atmospheric, Oceanic and Hydrologic Applications (Vol. III)*. Number v. 3. Springer International Publishing.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., and et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[Pathak et al., 2017] Pathak, J., Lu, Z., Hunt, B. R., Girvan, M., and Ott, E. (2017). Using machine learning to replicate chaotic attractors and calculate lyapunov exponents from data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(12):121102.

[Pearson, 1901] Pearson, K. F. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., and et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Penrose,] Penrose, R. A generalized inverse for matrices. *Proceedings of the Cambridge Philosophical Society*, 51 (3):406–13.

[Persson, 2017] Persson, A. (2017). The story of the hovmöller diagram: An (almost) eyewitness account. *Bulletin of the American Meteorological Society*, 98(5):949 – 957.

[Pesin, 1976] Pesin, J. B. (1976). Lyapunov characteristic exponents and ergodic properties of smooth dynamical systems with an invariant measure. *Sov. Math. Doklady*, 17:196.

[Pikovsky and Politi, 2016] Pikovsky, A. and Politi, A. (2016). *Lyapunov Exponents: A Tool to Explore Complex Dynamics*. Cambridge University Press.

[Planck, 1917] Planck, M. (1917). Über einen satz der statistischen dynamik und seine erweiterung in der quantentheorie. *Sitzungsberichte der Preussischen Akademie der Wissenschaften zu Berlin*, 24:324–341.

[Puntanen, 2010] Puntanen, S. (2010). Linear regression analysis: Theory and computing by xin yan, xiao gang su. *International Statistical Review*, 78(1):144–144.

[Rackauckas et al., 2020] Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., Skinner, D., Ramadhan, A., and Edelman, A. (2020). Universal differential equations for scientific machine learning. *PNAS*, XXX:1–6.

[Raissi et al., 2018] Raissi, M., Perdikaris, P., and Karniadakis, G. (2018). Multistep neural networks for data-driven discovery of nonlinear dynamical systems.

[Raissi et al., 2019] Raissi, M., Perdikaris, P., and Karniadakis, G. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707.

[Ricker, 1954] Ricker, W. E. (1954). Stock and recruitment. . *Fish. Res. Canada*, 2:559.

[Robbins and Monro, 1951] Robbins, H. and Monro, S. (1951). A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400 – 407.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

[Rössler, 1976] Rössler, O. E. (1976). An equation for continuous chaos. *Physics Letters A*, 57(5):397–398.

[Ruder, 2016] Ruder, S. (2016). An overview of gradient descent optimization algorithms. cite arxiv:1609.04747Comment: Added derivations of AdaMax and Nadam.

[Ruelle, 1978] Ruelle, D. (1978). An inequality for the entropy of differenti- able maps. *Bol. Soc. Bras. Mat.*, 9:83.

[Ruelle, 1979] Ruelle, D. (1979). Ergodic theory of differentiable dynamical systems. *Publications Mathématiques de L'Institut des Hautes Scientifiques*, 50:27–58.

[Ruelle, 1989] Ruelle, D. (1989). Chaotic evolution and strange attractors : the statistical analysis of time series for deterministic nonlinear systems.

[Ruelle, 2004] Ruelle, D. (2004). *Thermodynamic Formalism: The Mathematical Structure of Equilibrium Statistical Mechanics*. Cambridge Mathematical Library. Cambridge University Press, 2 edition.

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

[Russell et al., 1980] Russell, D., Hansonand, J., and Ott, E. (1980). Dimension of strange attractors. *Phys. Rev.Lett.*, 44:453.

[Sak et al., 2014] Sak, H., Senior, A. W., and Beaufays, F. (2014). Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *ArXiv*, abs/1402.1128.

[Salakhutdinov et al., 2007] Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th International*

Conference on Machine Learning, ICML '07, page 791–798, New York, NY, USA. Association for Computing Machinery.

[Santambrogio, 2015] Santambrogio, F. (2015). *Optimal transport for applied mathematicians: Calculus of variations, PDES, and modeling.* Birkhäuser, Basel.

[Schmidhuber, 2015] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.

[Schultz et al., 2021] Schultz, M. G., Betancourt, C., Gong, B., Kleinert, F., Langguth, M., Leufen, L. H., Mozaffari, A., and Stadtler, S. (2021). Can deep learning beat numerical weather prediction? *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2194):20200097.

[Shalev-Shwartz and Ben-David, 2014] Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding Machine Learning - From Theory to Algorithms.* Cambridge University Press.

[Sherstinsky, 2020] Sherstinsky, A. (2020). Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306.

[Sinai, 1959] Sinai, Y. G. (1959). On the concept of entropy of a dynamical system. *Dokl. Acad. Nauk SSSR*, 124:758.

[Smerieri et al., 2012] Smerieri, A., Duport, F., Paquot, Y., Haelterman, M., Schrauwen, B., and Massar, S. (2012). Towards fully analog hardware reservoir computing for speech recognition. In Simos, T. E., Psihoyios, G., Tsitouras, C., and Anastassi, Z., editors, *Numerical Analysis and Applied Mathematics ICNAAM 2012: International Conference of Numerical Analysis and Applied Mathematics*, volume 1479 of *American Institute of Physics Conference Series*, pages 1892–1895.

[Sonnewald et al., 2021] Sonnewald, M., Lguensat, R., Jones, D. C., Dueben, P. D., Brajard, J., and Balaji, V. (2021). Bridging observations, theory and numerical simulation of the ocean using machine learning. *Environmental Research Letters*, 16(7):073008.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

[Stein and Ulam, 1964] Stein, P. R. and Ulam, S. M. (1964). Non-linear transformation studies on electronic computers. *ozprawy Matematyczne*, 39:1.

[Stone, 1974] Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *J. Royal Stat. Soc.*, 36(2):111–147.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

[Szandała, 2021] Szandała, T. (2021). *Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks*, pages 203–224.

[Takens, 1979] Takens, F. (1979). Detecting strange attractors in turbulence. In D.A., Randand L.S., Y., editor, *Dynamical Systems and Turbulence*, volume 898, page 366. Springer Verlag(New York).

[Tanaka et al., 2019] Tanaka, G., Yamane, T., Héroux, J. B., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., and Hirose, A. (2019). Recent advances in physical reservoir computing: A review. *Neural Networks*.

[Teng and Zhang, 2019] Teng, Q. and Zhang, L. (2019). Data driven nonlinear dynamical systems identification using multi-step cldnn. *AIP Advances*, 9(8):085311.

[Tikhonov and Arsenin, 1977] Tikhonov, A. N. and Arsenin, V. I. (1977). *Solutions of ill-posed problems / Andrey N. Tikhonov and Vasiliy Y. Arsenin ; translation editor, Fritz John.* Winston ; distributed solely by Halsted Press Washington : New York.

[Torrejon et al., 2017] Torrejon, J., Riou, M., Araujo, F., and et al. (2017). Neuromorphic computing with nanoscale spintronic oscillators. *Nature*, 547:428–431.

[Tran and Teuscher, 2019] Tran, S. J. and Teuscher, C. (2019). Hierarchical memcapacitive reservoir computing architecture. In *2019 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–6.

[Tucker, 2002] Tucker, W. (2002). A rigorous ode solver and smale's 14th problem. *Found. Comput. Math.*, 2:53.

[Ulam and von Neumann, 1947] Ulam, S. M. and von Neumann, J. (1947). On combination of stochastic and deterministic processes - preliminary report. *Bull. Am. Math. Soc.*, 53:1120.

[van der Pol, 1920] van der Pol, B. (1920). A theory of the amplitude of free and forced triode vibrations. *Radio Review*, 1:701–710.

[van Kekem and Sterk, 2018a] van Kekem, D. L. and Sterk, A. E. (2018a). Travelling waves and their bifurcations in the lorenz-96 model. *Physica D: Nonlinear Phenomena*, 367:38–60.

[van Kekem and Sterk, 2018b] van Kekem, D. L. and Sterk, A. E. (2018b). Wave propagation in the lorenz-96 model. *Nonlinear Processes in Geophysics*, 25(2):301–314.

[van Kekem and Sterk, 2019] van Kekem, D. L. and Sterk, A. E. (2019). Symmetries in the lorenz-96 model. *International Journal of Bifurcation and Chaos*, 29(01):1950008.

[Vaserstein, 1969] Vaserstein, L. (1969). Markov processes over denumerable products of spaces, describing large systems of automata. *Problemy Peredači Informacii*, 5(3):64–72.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[Villani, 2019] Villani, C. (2019). *Optimal transport: Old and new.* Springer-Verlag, Berlin Heidelberg, Germany.

[Vissio et al., 2020] Vissio, G., Lembo, V., Lucarini, V., and Ghil, M. (2020). Evaluating the performance of climate models based on wasserstein distance. *Geophysical Research Letters*, 47(21):e2020GL089385. e2020GL089385 10.1029/2020GL089385.

[Vissio and Lucarini, 2018] Vissio, G. and Lucarini, V. (2018). Evaluating a stochastic parametrization for a fast–slow system using the wasserstein distance. *Nonlinear Processes in Geophysics*, 25(2):413–427.

[Vissio and Lucarini, 2020] Vissio, G. and Lucarini, V. (2020). Mechanics and thermodynamics of a new minimal model of the atmosphere. *Eur. Phys. J. Plus*, 135:807.

[Vlachas et al., 2018] Vlachas, P. R., Byeon, W., Wan, Z. Y., Sapsis, T. P., and Koumoutsakos, P. (2018). Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2213):20170844.

[Vortmeyer-Kley et al., 2021] Vortmeyer-Kley, R., Nieters, P., and Pipa, G. A. (2021). Trajectory-based loss function to learn missing terms in bifurcating dynamical systems. *Sci Rep*, 11:20394.

[Wang et al., 2022] Wang, Q., Ma, Y., Zhao, K., and Tian, Y. (2022). A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, 9:187–212.

[Wang and Jiang, 2016] Wang, S. and Jiang, J. (2016). Learning natural language inference with LSTM. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1442–1451, San Diego, California. Association for Computational Linguistics.

[Wetzel, 2017] Wetzel, S. J. (2017). Unsupervised learning of phase transitions: From principal component analysis to variational autoencoders. *Phys. Rev. E*, 96:022140.

[Wiener, 1930] Wiener, N. (1930). Generalized harmonic analysis. *Acta Mathematica*, 55:117–258.

[Xu et al., 2019] Xu, Z., Dai, A. M., Kemp, J., and Metz, L. (2019). Learning an adaptive learning rate schedule. *ArXiv*, abs/1909.09712.

[Yeo and Melnyk, 2019] Yeo, K. and Melnyk, I. (2019). Deep learning algorithm for data-driven simulation of noisy dynamical system. *J. Comput. Phys.*, 376:1212–1231.

[Young, 1982] Young, L. (1982). Dimension, entropy and lyapunov exponents. *Ergodic Theory and Dyn. Systems*, 2:109.

[Zeiler, 2012] Zeiler, M. D. (2012). Adadelta: An adaptive learning rate method. *CoRR*, abs/1212.5701.

[Zhang et al., 2020] Zhang, R., Liu, Y., and Sun, H. (2020). Physics-informed multi-lstm networks for metamodeling of nonlinear structures. *Computer Methods in Applied Mechanics and Engineering*, 369:113226.

[Zhang and Yu, 2005] Zhang, T. and Yu, B. (2005). Boosting with early stopping: Convergence and consistency. *The Annals of Statistics*, 33(4):1538 – 1579.

[Zhou and Chellappa, 1988] Zhou, Y.-T. and Chellappa, R. (1988). Computation of optical flow using a neural network. *IEEE 1988 International Conference on Neural Networks*, pages 71–78 vol.2.

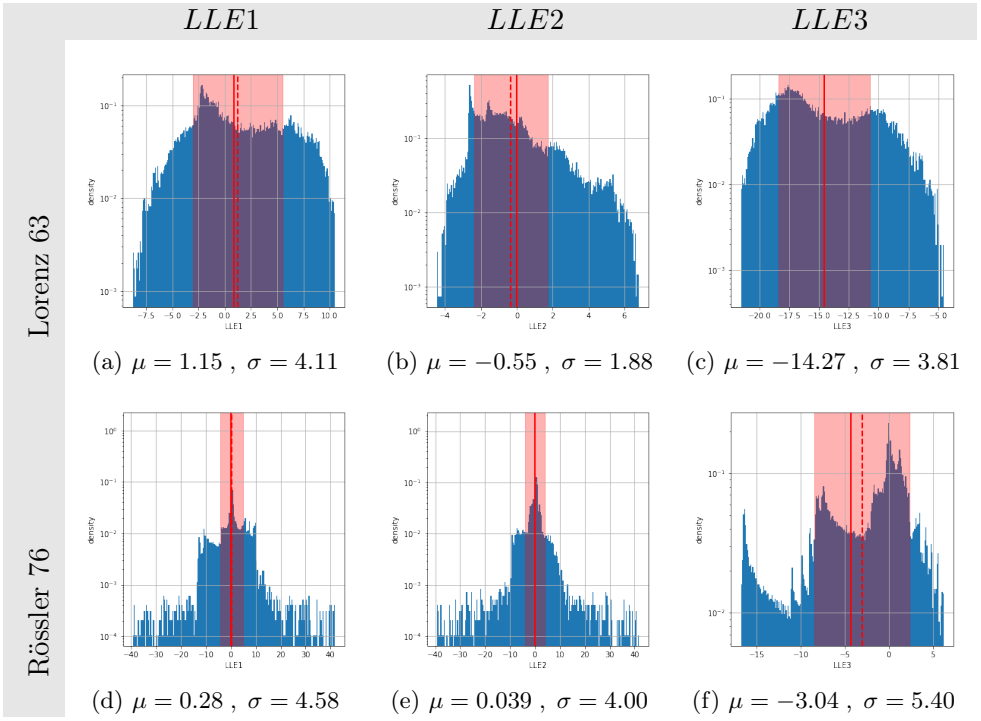
Appendix A

Supporting Figures

This appendix reports the supporting figures for chapter 4. In section A.1 ee report the training losses of LSTM models of subsection 4.2.2, the parallel coordinate plot for Optuna optimization. We report the full distribution of Local Lyapunov Exponents (LLE), along with the mean and the standard deviation, and the short term reconstructed trajectories for LSTM and ESN models. We do not report those for the FFNET, because they have been lost due to a technical error. In section A.2 we report the training losses, the 2d projected learned invariant measures and the full distribution for autoencoders learning problem in section 4.3. Lastly, in section A.3 we report the dataset examples for section 4.4 and the first learn convolutional filters.

A.1 Dynamics Reconstruction

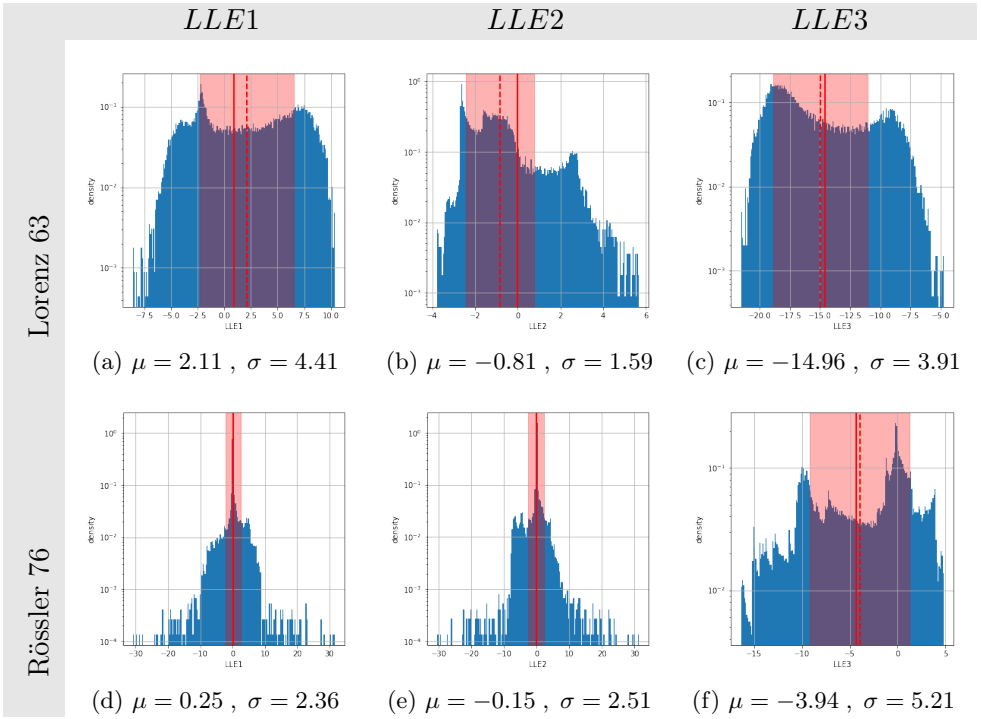
A.1.1 LSTM



(a) $\mu = 1.15$, $\sigma = 4.11$ (b) $\mu = -0.55$, $\sigma = 1.88$ (c) $\mu = -14.27$, $\sigma = 3.81$

(d) $\mu = 0.28$, $\sigma = 4.58$ (e) $\mu = 0.039$, $\sigma = 4.00$ (f) $\mu = -3.04$, $\sigma = 5.40$

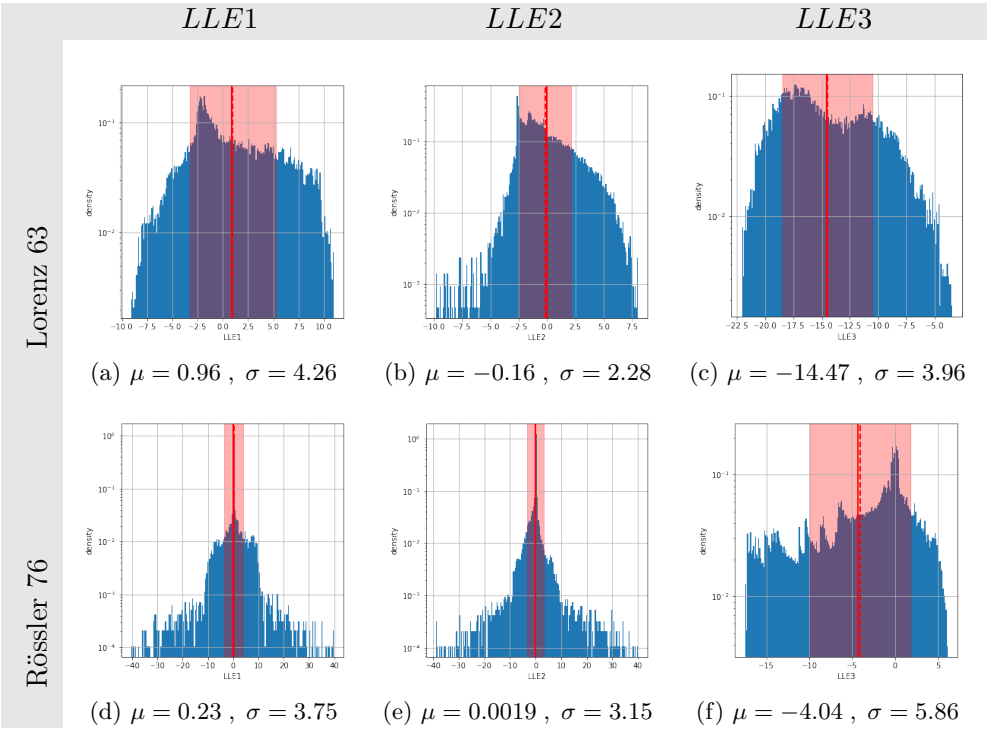
Figure A.1: Standard LSTM models. Distributions of Local Lyapunov Exponents (LLE) with mean μ and standard deviation σ. They are computed following the algorithm reported in subsection 2.8.3 with $\tau = 4$. The continuous red line indicates the true value computed from the test dataset, while the dashed red line is the average of the distribution and the colored red area is the displacement from the mean of 1σ.



(a) $\mu = 2.11$, $\sigma = 4.41$ (b) $\mu = -0.81$, $\sigma = 1.59$ (c) $\mu = -14.96$, $\sigma = 3.91$

(d) $\mu = 0.25$, $\sigma = 2.36$ (e) $\mu = -0.15$, $\sigma = 2.51$ (f) $\mu = -3.94$, $\sigma = 5.21$

Figure A.2: Optimized LSTM models. Distributions of Local Lyapunov Exponents (LLE) with mean μ and standard deviation σ. They are computed following the algorithm reported in subsection 2.8.3 with $\tau = 4$. The continuous red line indicates the true value computed from the test dataset, while the dashed red line is the average of the distribution and the colored red area is the displacement from the mean of 1σ.



(a) $\mu = 0.96$, $\sigma = 4.26$ (b) $\mu = -0.16$, $\sigma = 2.28$ (c) $\mu = -14.47$, $\sigma = 3.96$

(d) $\mu = 0.23$, $\sigma = 3.75$ (e) $\mu = 0.0019$, $\sigma = 3.15$ (f) $\mu = -4.04$, $\sigma = 5.86$

Figure A.3: Physics Informed LSTM models. Distributions of Local Lyapunov Exponents (LLE) with mean μ and standard deviation σ. They are computed following the algorithm reported in subsection 2.8.3 with $\tau = 4$. The continuous red line indicates the true value computed from the test dataset, while the dashed red line is the average of the distribution and the colored red area is the displacement from the mean of 1σ.

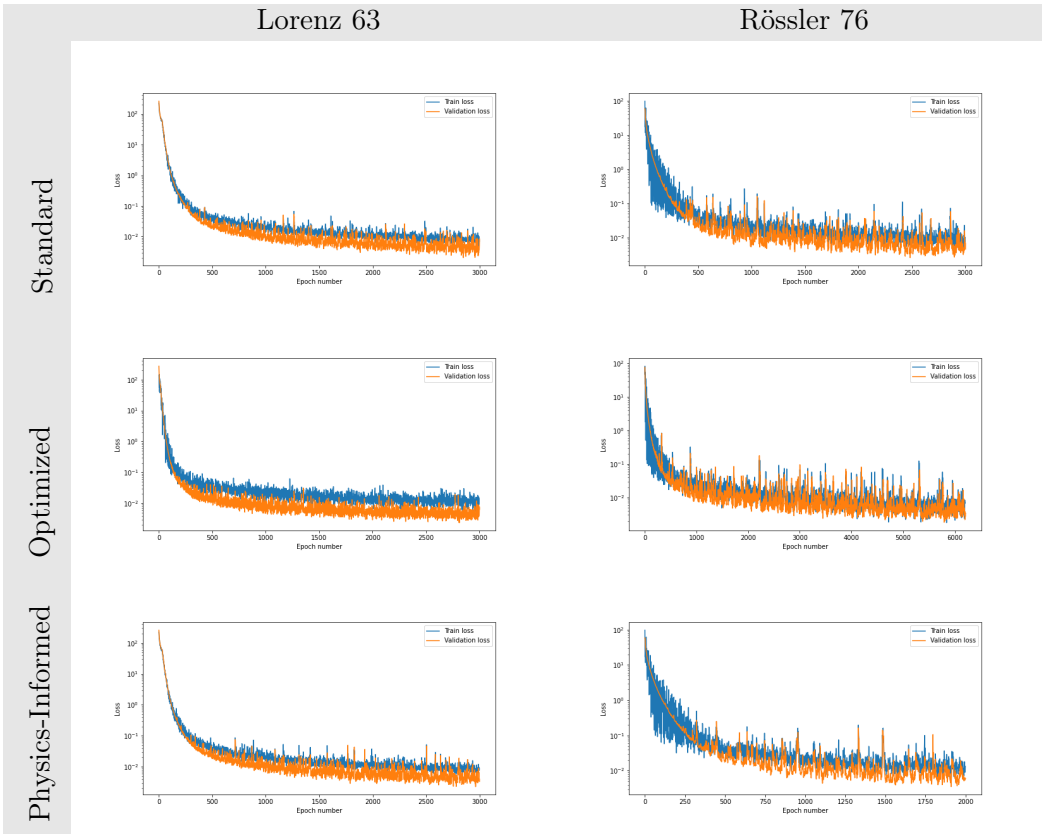


Figure A.4: LSTM training losses for Standard and Optimized Optuna models with Data-Driven loss and for Physics-Informed loss, see section 4.2. Training loss is in blue and validation loss in orange.

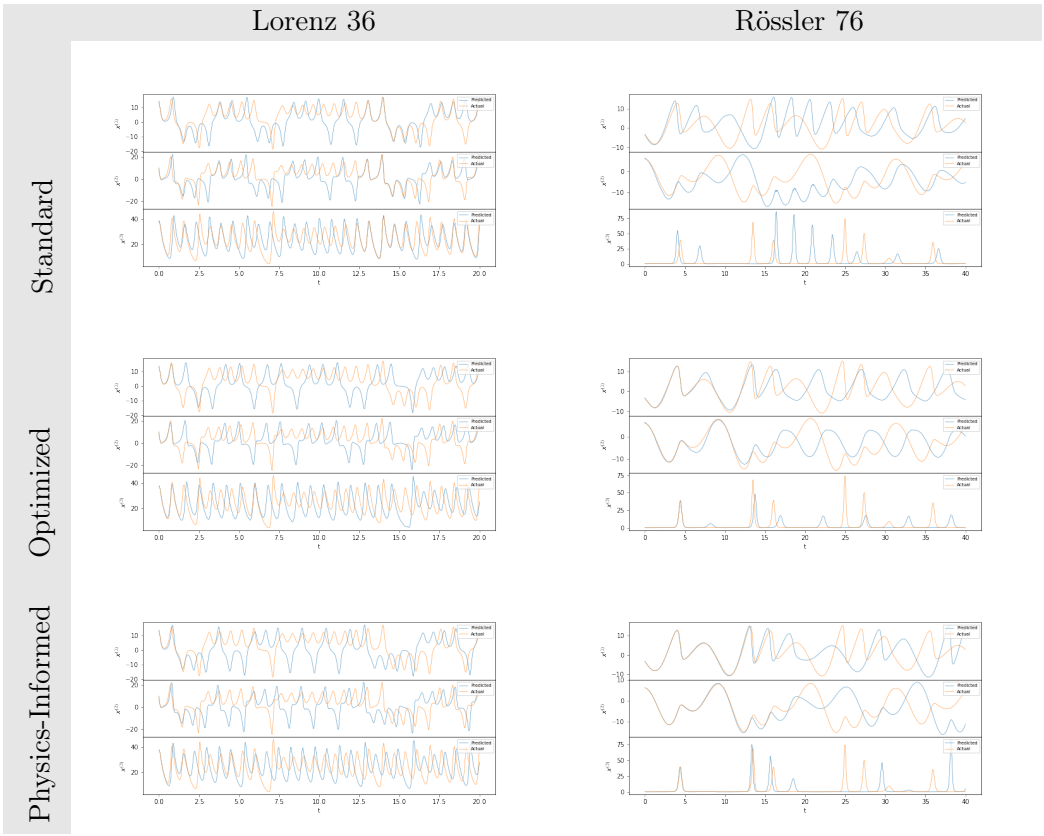
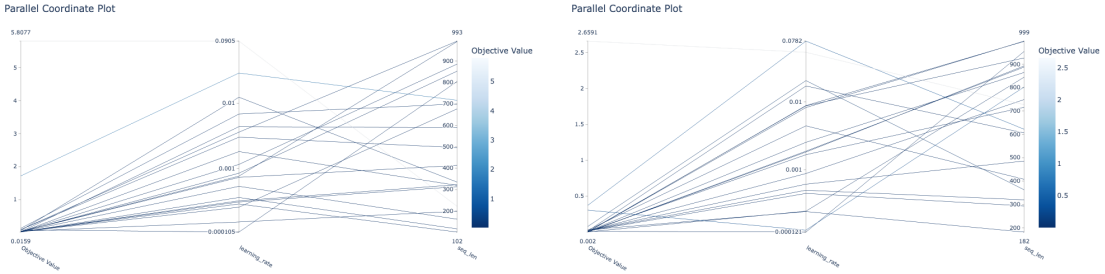


Figure A.5: Short dynamics reconstruction for LSTM models. We compare the two systems, Lorenz 63 and Rössler 76, in the short term by comparing the reconstructed trajectories (blue) with the test dataset (orange) starting at the same initial point.



(a) (b)

Figure A.6: Hyperparameter optimization parallel coordinate plot for LSTM, see section 4.2.

A.1.2 Echo State Networks

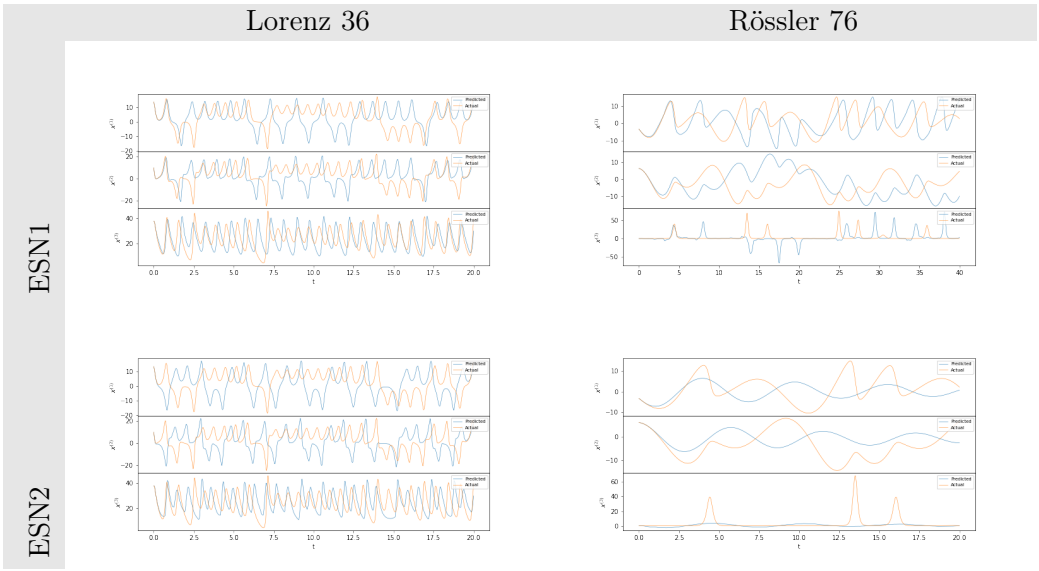
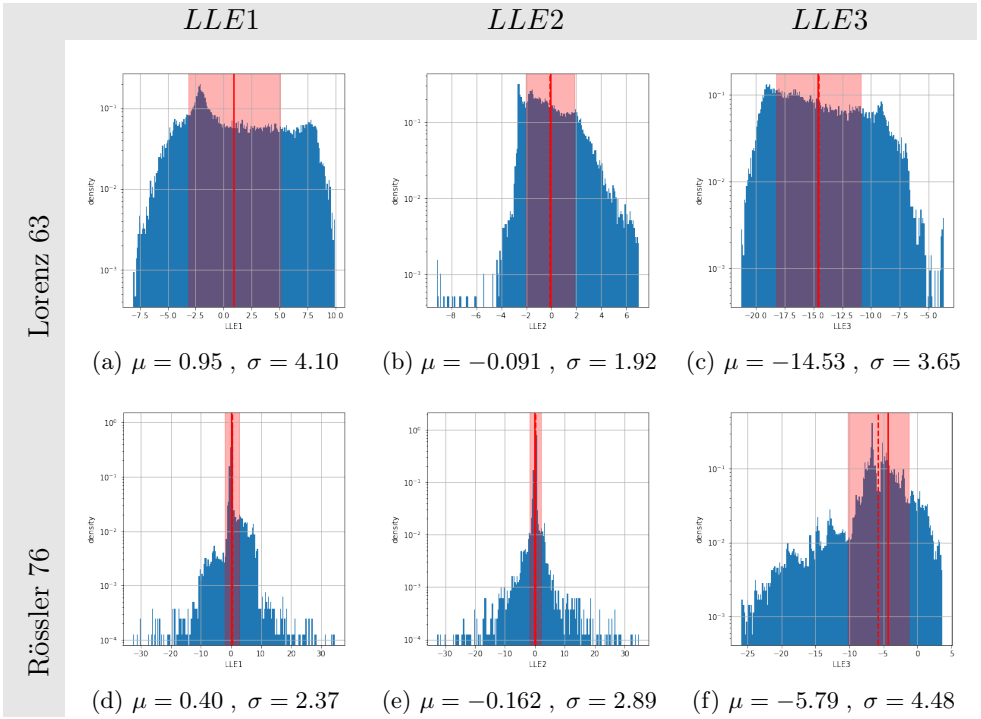


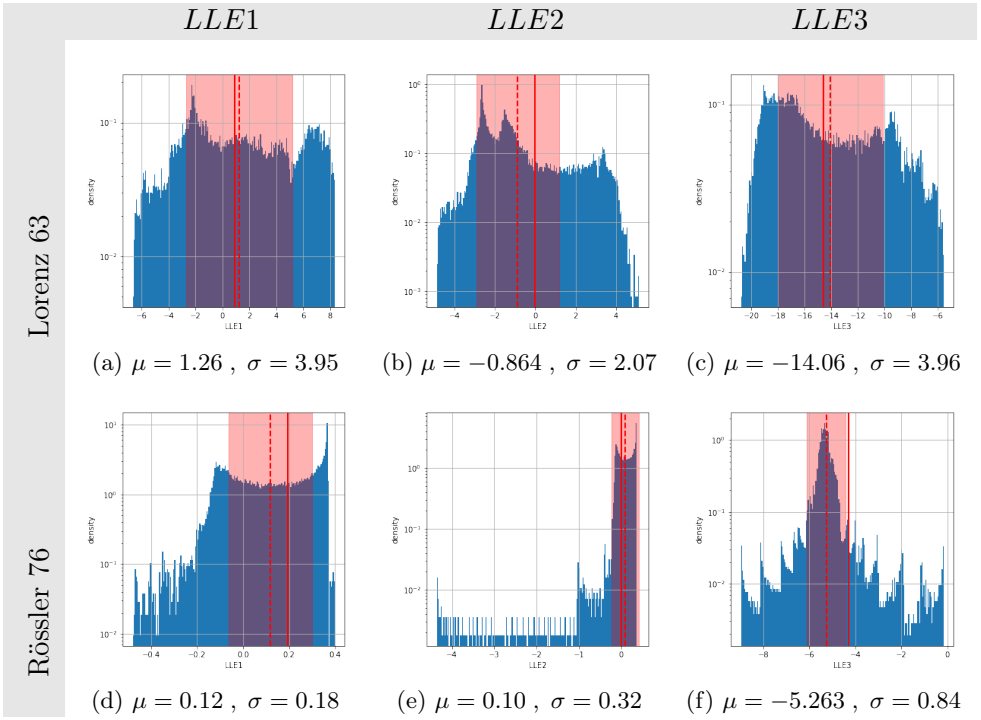
Figure A.7: Short dynamics reconstruction for ESN models. We compare the two systems, Lorenz 63 and Rössler 76, in the short term by comparing the reconstructed trajectories (blue) with the test dataset (orange) starting at the same initial point.



(a) $\mu = 0.95$, $\sigma = 4.10$ (b) $\mu = -0.091$, $\sigma = 1.92$ (c) $\mu = -14.53$, $\sigma = 3.65$

(d) $\mu = 0.40$, $\sigma = 2.37$ (e) $\mu = -0.162$, $\sigma = 2.89$ (f) $\mu = -5.79$, $\sigma = 4.48$

Figure A.8: ESN1 Random matrix. Distributions of Local Lyapunov Exponents (LLE) with mean μ and standard deviation σ. They are computed following the algorithm reported in subsection 2.8.3 with $\tau = 4$. The continuous red line indicates the true value computed from the test dataset, while the dashed red line is the average of the distribution and the colored red area is the displacement from the mean of 1σ.



(a) $\mu = 1.26$, $\sigma = 3.95$ (b) $\mu = -0.864$, $\sigma = 2.07$ (c) $\mu = -14.06$, $\sigma = 3.96$

(d) $\mu = 0.12$, $\sigma = 0.18$ (e) $\mu = 0.10$, $\sigma = 0.32$ (f) $\mu = -5.263$, $\sigma = 0.84$

Figure A.9: ESN2 Erdo-Reny graph. Distributions of Local Lyapunov Exponents (LLE) with mean μ and standard deviation σ. They are computed following the algorithm reported in subsection 2.8.3 with $\tau = 4$. The continuous red line indicates the true value computed from the test dataset, while the dashed red line is the average of the distribution and the colored red area is the displacement from the mean of 1σ.

A.2 Parameter Estimation with Regularised Autoencoders

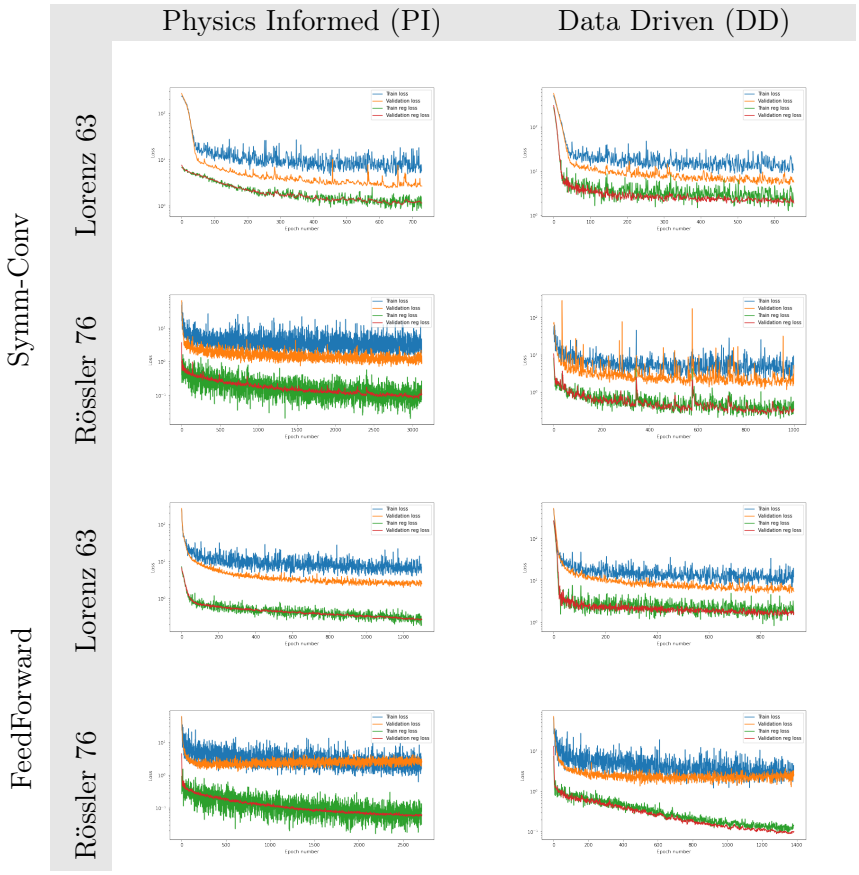


Figure A.10: Training (blue), validation(orange), train regularization (green) and validation regularization (red) losses for Autoencoders models. The regularization strength for Physic Informed loss (left column) is chosen to be $\gamma = 100$ for Lorenz 63 and $\gamma = 1000$ for Rössler 76 in order to make the regularization loss of the same order of magnitude of the reconstruction loss.

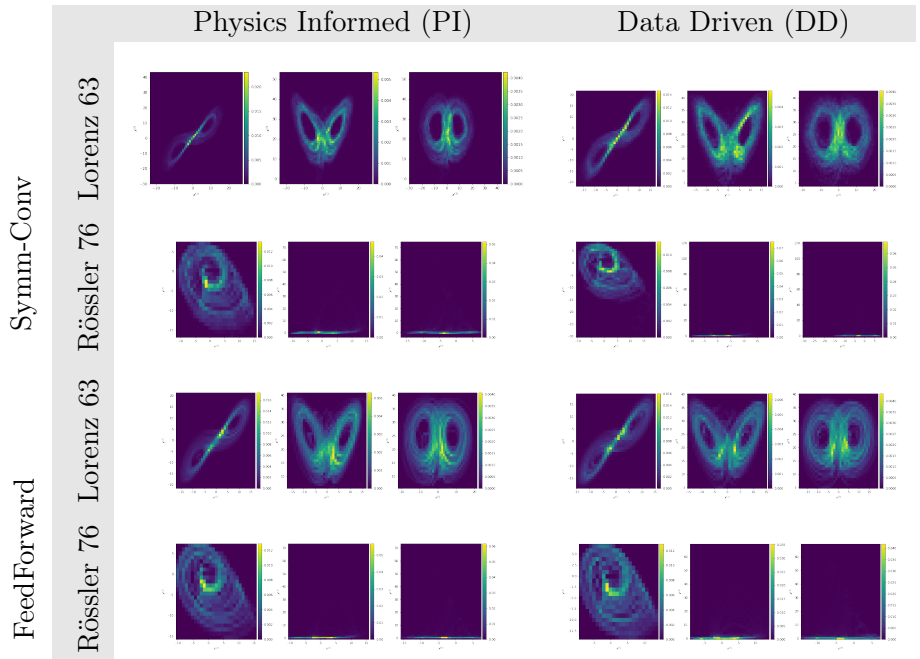


Figure A.11: Projected invariant measures of reconstructed attractors for different Autoencoder models and systems.

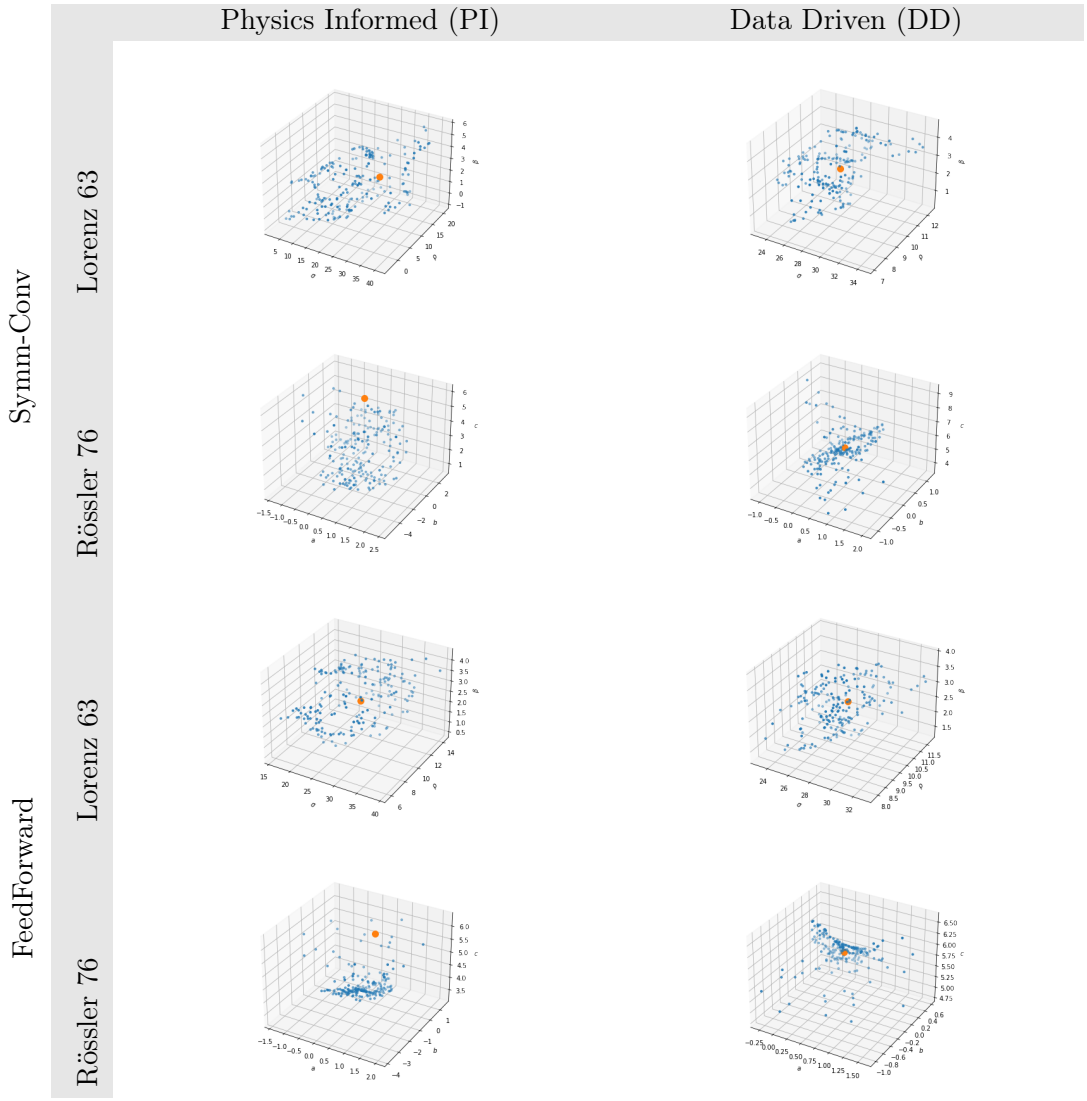


Figure A.12: Distributions of learned parameters in parameters space for different Autoencoder models and systems. The true set of parameters is denoted with the red dot.

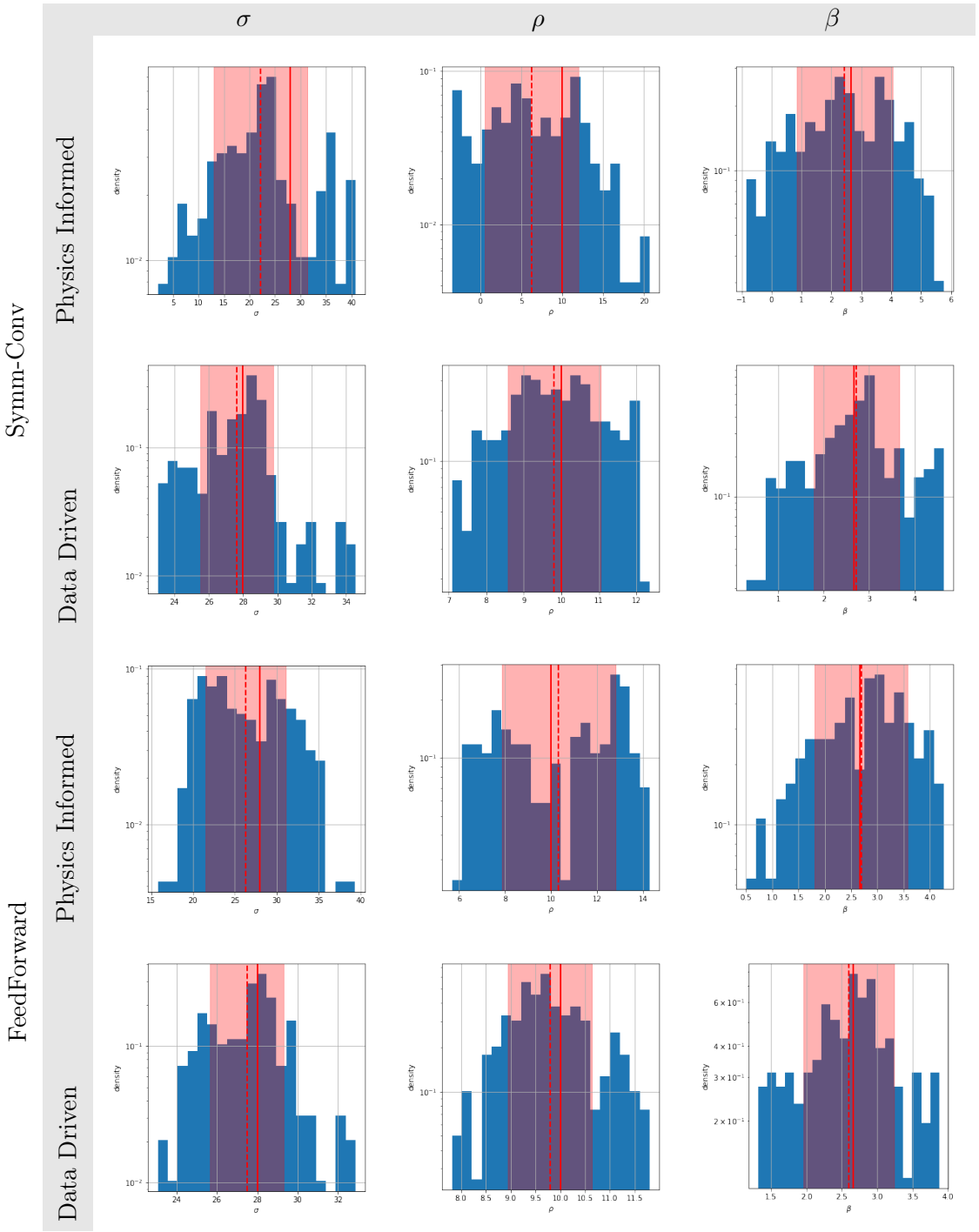


Figure A.13: Distribution of learned parameters for Lorenz 63 system. The mean is denoted by the dotted red line, while the true value is the continuous red line. The shaded red area is included in a σ displacement from the mean.

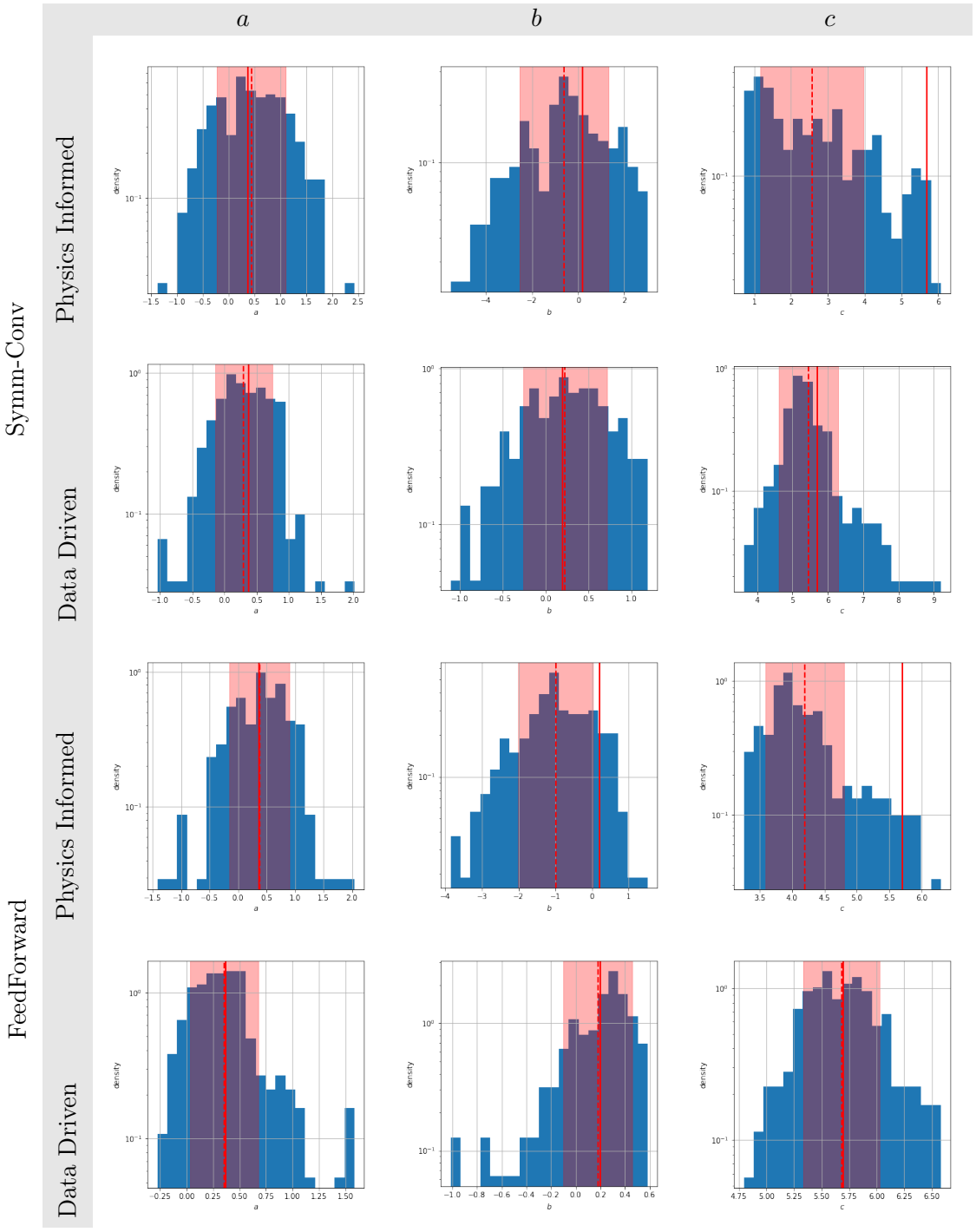


Figure A.14: Distribution of learned parameters for Röessler 76 system. The mean is denoted by the dotted red line, while the true value is the continuous red line. The shaded red area is included in a σ displacement from the mean.

A.3 Unsupervised Learning of Forcing in Lorenz 96

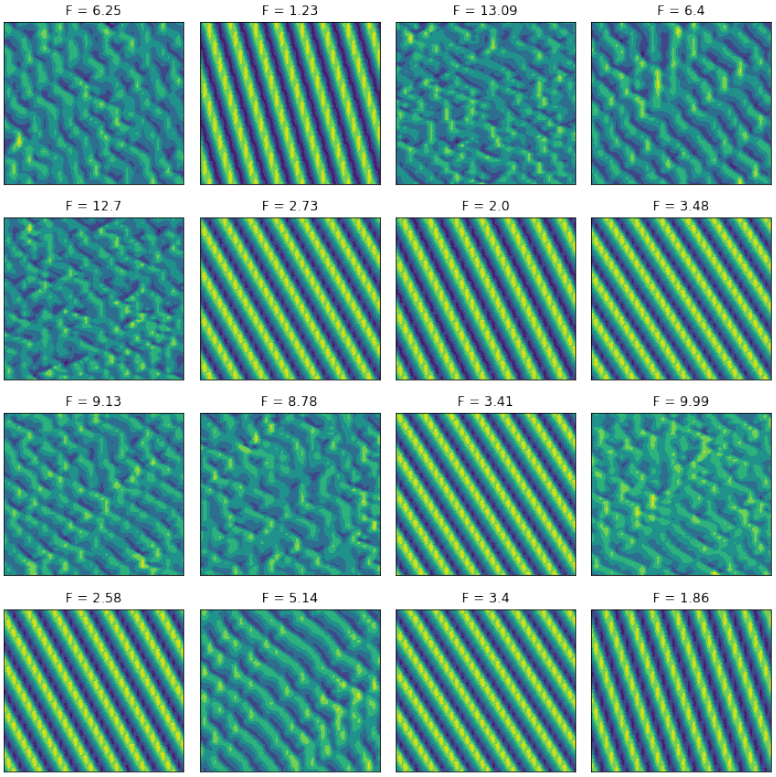
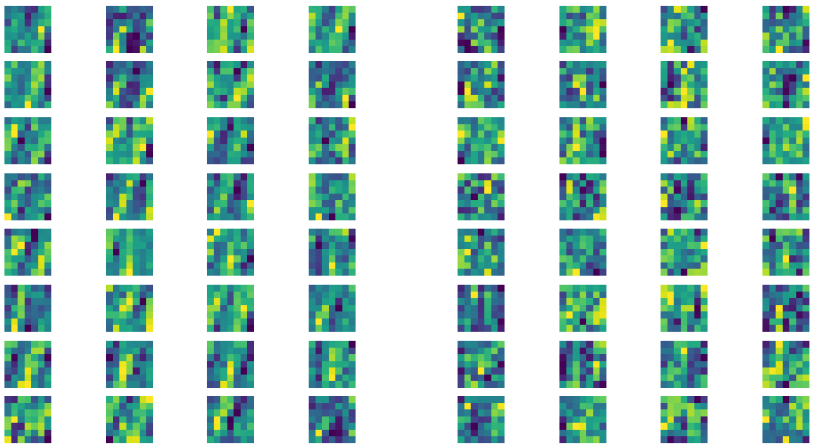


Figure A.15: Examples of the training dataset for the second experiment of this section.



(a) Normalized Inputs. (b) Non-Normalized Inputs.

Figure A.16: Learned convolutional filters of the first layer for a) the Normalized Inputs encoder and b) Non-Normalized. We can appreciate that the filters seem to have learned horizontal and vertical edges as well as little islands, which happen in the chaotic regime.