VRIJE
UNIVERSITEIT
BRUSSEL

Graduation thesis submitted in partial fulfilment of the requirements for the degree of Master in Science: Physics and Astronomy

NEURAL OPERATORS FOR CHAOTIC SYSTEMS

Novel Neural Operator architectures, and predicting chaotic behaviour

Jari Beysen

January 2025

Promotor: prof. dr. Vincent Ginis Co-Promotor: Floriano Jean Tori

sciences and bioengineering sciences

Introduction

Nature is chaotic. From physics and chemistry to biology and even the social sciences; chaos is present in every discipline [1]. Since the early 1960s, when Edward Lorenz first discovered chaotic behaviour in a dynamical system [2] he developed to study atmospheric dynamics, it was obvious that chaotic behaviour is not something to take lightly. The model Lorenz was studying (Lorenz-63 system) is currently not of much interest in the field of atmospheric dynamics, but is now seen as a litmus test for models that try to predict, understand, or even 'defeat' Chaos. The latter is a rather loose term because, mathematically speaking, chaos can not be overcome by improving the accuracy of a model since any deviation, by its definition, will (initially) grow exponentially in time. For now, predicting if or when chaotic behaviour can occur in a system is still a non-trivial problem, certainly when studying a family of differential equations, and not just a singular system. Chaos theory is very much an active field of study due to its incompatibility with reductionism [3]. In recent years, many advances have been made regarding modeling chaotic systems without relying on numerical techniques. Numerical techniques are not useful in every situation; they have to be executed sequentially, and are not usable when the system itself is only partially known, and only data generated from this system is available.

Numerical techniques focus on approximating solutions to equations that are not analytically solvable; meanwhile data-driven techniques approximate solutions in situations where the governing equations are not known, and only some solution samples. Data-driven techniques are often sufficiently accurate when the behaviour of a system is more important than the precision of individual realizations. Such approximations can be carried out by black box methods present in the field of *Machine Learning* (ML). The interplay between ML and the exact sciences is still developing, but undeniably gaining traction. A great example of this interplay in the field of Bio-Informatics is the development of AlphaFold2 [4], which is able to predict the structure of folded proteins with an accuracy comparable to experimental results. The 2024 Nobel Prize in Physics [5], awarded to J. Hopfield and G. Hinton for their development of the Hopfield Network and the Boltzmann Machine respectively, indicates a growing interest in the interdisciplinary affairs between Physics and ML. The nomination reflects the utility that concepts derived from physical theories can have in other sciences. This thesis shall also walk that interdisciplinary bridge between physics and computer science, starting from Reservoir Computing (RC) [6] which is unreasonably reliable at approximating the underlying dynamics in systems that exhibit chaotic behaviour [6].

Reservoir computing is a type of Recurrent Neural Network (RNN) that satisfies the conditions set by the Universal Approximation Theorem for Neural Networks [7]. Neural Networks, just like a Taylor expansion or Fourier decomposition, approximate functions. When a problem requires a model to learn systems of parametrized differential equations, Neural Networks are not sufficient because they cannot generalize across systems. A different approach to learning

a function is learning the operator (time integral in the context of dynamics) that is applied to the differential equation to develop the timeseries. Neural Operators (NO) are Neural Networks which satisfy the conditions set by the universal approximation theorem for operators [8]. Even though this theorem was published in 1995, it took until 2019 for an architecture to be developed in search of approximating operators. The architecture developed by Lu Lu Pengzhan Jin, and George Em Karniadakis, named the DeepONet [9] (Deep Operator Network) uses two different Neural Networks which together create a basis for the input and output function. This development created a boom in research, leading to many other forms of Neural Operators. The DeepONet is not quite suitable for chaotic systems. Chaos presents itself over long time intervals, and is (by nature) complex to an extent that it appears to be stochastic. The previous two observations are obstacles for any feedforward Neural Network because the map from $F : t \to y(t)$ will be too difficult to reasonably learn. DeepONets are inherently flexible regarding the network architectures used internally. This flexibility is not universal; they cannot function with methods such as Reservoir Computing because Linear Regression cannot be integrated into the readout form of the DeepONet.

In this thesis, the notion of perturbation growth over time defined by the Lyapunov exponent will be extended to estimate how errors grow in the context of iterative techniques. In the following chapters, two novel Neural Operator architectures will be developed. The first architecture generalizes the DeepONet into an *ExtremONet* such that it allows Linear Regression to be used. This technique will then be extended to permit Reservoir Computing which was named the *EchONet*. It will be shown that the ExtremONet can outperform the DeepONet on a test parametrized differential equation in both accuracy and training time. The EchONet will then be used to predict whether a system is chaotic with or without timeserie data. Furthermore, this Lyapunov exponent prediction, in combination with the error growth approximation, can be used to create approximate uncertainties on predictions made by the model, increasing the reliability of predictions made by the EchONet.

This thesis consists of 4 chapters. Chapter 1 shall explore the concept of Chaos in non-linear dynamics, and the effects it has on iterative integration schemes. The predictability time of the Runge-Kutta 4 method will be used to show the validity of this novel error growth equation. Chapter 2 is dedicated to Machine Learning. The shortcomings of Linear Regression will be demonstrated, which will lead into a discussion about Neural Networks. Finally, the advantages of Extreme Learning Machines, which combine the strengths of Linear Regression and Neural Networks, will be explored. Chapter 3 is dedicated to Neural Operators and the DeepONet. The adjustments necessary to create the ExtremONet will be explained, together with a thorough exploration of its performance in comparison with the DeepONet. Chapter 4 is the final chapter, and will be dedicated to exploring Reservoir Computing and how to combine it with the ExtremONet to form the EchONet. The EchONet will be thoroughly tested on the entire parameter space of the Lorenz-63 system. It will also be shown that this model does not suffer from many drawbacks inherent to Recurrent Neural Networks, which will result in the possibility of Lyapunov exponent and timeserie predictions without a need for initial timeserie data. The EchONet will also be used to explore the chaotic zones of a generalized Lorenz-96 system, containing many different advection basis terms that can sometimes lead to Chaos.

Contents

1 Non-linear dynamics and chaos **5**
- 1.1 Non-linear maps . . . 6
  - 1.1.1 Aperiodic behaviour . . . 6
  - 1.1.2 The Lyapunov exponent for maps . . . 8
  - 1.1.3 Chaos . . . 10
- 1.2 Non-linear Differential equations . . . 11
  - 1.2.1 Lyapunov spectrum in continuous systems . . . 11
  - 1.2.2 Lorenz-63 and strange attractors . . . 13
  - 1.2.3 Parameter space complication and predictability time . . . 18

2 Machine Learning **22**
- 2.1 Supervised learning . . . 23
  - 2.1.1 Basis expansion and regularization . . . 24
- 2.2 Neural Networks . . . 28
- 2.3 Extreme Learning Machines . . . 29

3 Neural Operators **31**
- 3.1 Universal Approximation Theorem for Operators . . . 31
- 3.2 DeepONet . . . 32
- 3.3 Extreme Operator Network (ExtremONet) . . . 33
  - 3.3.1 Hyperparameter analysis . . . 38
  - 3.3.2 Data dependency, network dimensionality . . . 39
- 3.4 Discussion . . . 41

4 Reservoir and Operators **43**
- 4.1 Recurrent methods . . . 43
  - 4.1.1 Exploding & Vanishing Gradient problem . . . 44
  - 4.1.2 Training time . . . 44
- 4.2 Echo-State property . . . 45
- 4.3 Reservoir Computing and memory for dynamical systems . . . 45
- 4.4 Reservoir topology . . . 50
- 4.5 Conditional Lyapunov exponent and prediction standard deviation . . . 50
- 4.6 Echo Operator Network (EchONet) . . . 52
  - 4.6.1 Parametrized Lyapunov exponent prediction . . . 55
  - 4.6.2 Network design . . . 56
  - 4.6.3 Data dependency . . . 59
  - 4.6.4 Decoupling of training and predicting phases . . . 64

4.7 Generalized Lorenz-96 and the road towards EchONets for chaotic PDEs 67

5 Conclusion **75**
5.1 Future prospects 76

Appendices **78**
A.1 ExtremONet tables 79
B.2 Dissipative basis elements of the Generalized Lorenz -96 system, and examples . 80
C.3 An Alternative EchONet 83

Non-linear dynamics and chaos

Non-linear dynamics is the study of systems where the response to a perturbation is not proportional to perturbation, implying that it can not be described by linear equations. Non-linear systems have the ability to generate very complex behaviour in the time domain, often making them unpredictable. This complex behaviour can lead to the appearance of stochasticity, even though they are fully deterministic. The phenomenon that leads to the appearance of randomness, and which is the field's prime phenomenon of study is Chaos, which differs from the colloquial meaning. A significant reason for this interest is because of the sensitivity to initial conditions, which implies that two neighbouring trajectories will exponentially diverge over some time. Detecting and understanding this behaviour in a non-linear system of interest is paramount in the modeling process.

The study of Meteorology and fluid dynamics is where the sensitivity to initial conditions, and its impact was first discovered by Edward Lorenz in 1963. He was interested in studying the dynamics of forced dissipative hydrodynamic flow [2] which are present in atmospheric dynamics, specifically the convection equations of Saltzmann. His findings demonstrated that thermally forced non-conservative hydrodynamical systems are unstable, implying that small errors in the predictions lead to drastically different behaviour due to their aperiodicity. He noted that this behaviour makes it impossible for long-term predictions of this system to be accurate, unless the initial condition is known perfectly (it is obviously impossible to measure the position and momentum of every molecule in the atmosphere).

It is a common consensus that long term weather forecasts are not very reliable, which is not the fault of the models, but an intrinsic property of the atmosphere. To demonstrate the gravity of accurate weather predictions, there is a notable example from history which can convince anyone that understanding predicting chaotic behaviour should be of top priority. In June of 1944, the Allied forces were planning the liberation of France [10]. They were to do so by storming the coast of Normandy, which took place on the 6th of June but was scheduled for the 5th. D-day was planned such that the weather was clear and would not interfere with the operation. On the 3rd of June, weather forecasts showed that a severe storm was approaching, which would disrupt the plan, and put thousands of men's lives in danger. They were able to avoid the storm, but still found themselves in challenging weather. They had no way to make sure that the prediction that they were relying on was accurate. Even worse, the predictions demonstrated that the next window of opportunity would be two weeks later. If they had chosen to delay their operations, they would have come face to face with the most extreme weather in the English Channel for the next 20 years.

This event, and the apocalyptic risk it carried is a dramatic example, but it does provide motiv-

ation to understand the principles of non-linear dynamics.

This chapter starts by giving an introduction to non-linear dynamics and the emergence of chaos, starting from discrete systems such as the logistic map. Continuous systems will be explored afterwards by examining the Lorenz-63 system. This chapter will conclude by discussing the effects Chaos has on modeling timeseries, and a novel error propagation scheme based on Chaos with a perturbation source term.

1.1 Non-linear maps

Dynamical systems that are discrete in the time domain are referred to as maps. Maps are a construct of interest in many different contexts [11]; a tool for analysing differential equations in the form of difference equations, the study of reproduction in animal populations, simulating chaos. In general, all maps can be described by some iterative equation:

$$y_{n+1} = f(g_n, y_n, y_{n-1}, ..., y_0). \tag{1.1}$$

The vector y_{n+1} is dependent on all its previous iterations and some other map $g(\cdot)$, which will be referred to as a forcing term. If $f(\cdot)$ is isolated and only dependent on values of y, then it will be of the form:

$$y_{n+1} = f(y_n, y_{n-1}, ..., y_0). \tag{1.2}$$

A familiar example for any physicist is the Euler method, which is used for numerical integration of differential equations. A differential equation of the form

$$\dot{y} = f(y) \tag{1.3}$$

can be discretized by letting $dt \to \Delta t$ and $dy \to \Delta y = y_{n+1} - y_n$, returning a (non-linear) map

$$y_{n+1} = y_n + f(y_n)\Delta t. \tag{1.4}$$

Linear maps are well-studied, and are not of much particular research interest since their behaviour is understood; they cannot give rise to chaotic dynamics because they cannot be sensitive to initial conditions (will be shown later). If $f(\cdot)$ is not described by a first-order polynomial, then it generates a non-linear map. Non-linear dynamics are much more challenging to decipher. To demonstrate this, let us explore a simple non-linear 1-dimensional map that can give rise to chaotic behaviour.

1.1.1 Aperiodic behaviour

The logistic map is a discrete version of the logistic differential equation, used to model population growth with a carrying capacity which was first described by the Belgian physicist Pierre François Verhulst [12]. The map, unlike the continuous equation, can produce unpredictable behaviour in certain regions of the parameter space. This map can be defined as:

$$y_{n+1} = r y_n (1 - y_n). \tag{1.5}$$

Where the parameter space of interest is $0 \leq r \leq 4$. To demonstrate some of the behaviour present in the parameter space, the timeseries and phase space of two instances ($r = \{3.4, 3.7\}$) over a period of 200 iterations will be generated.

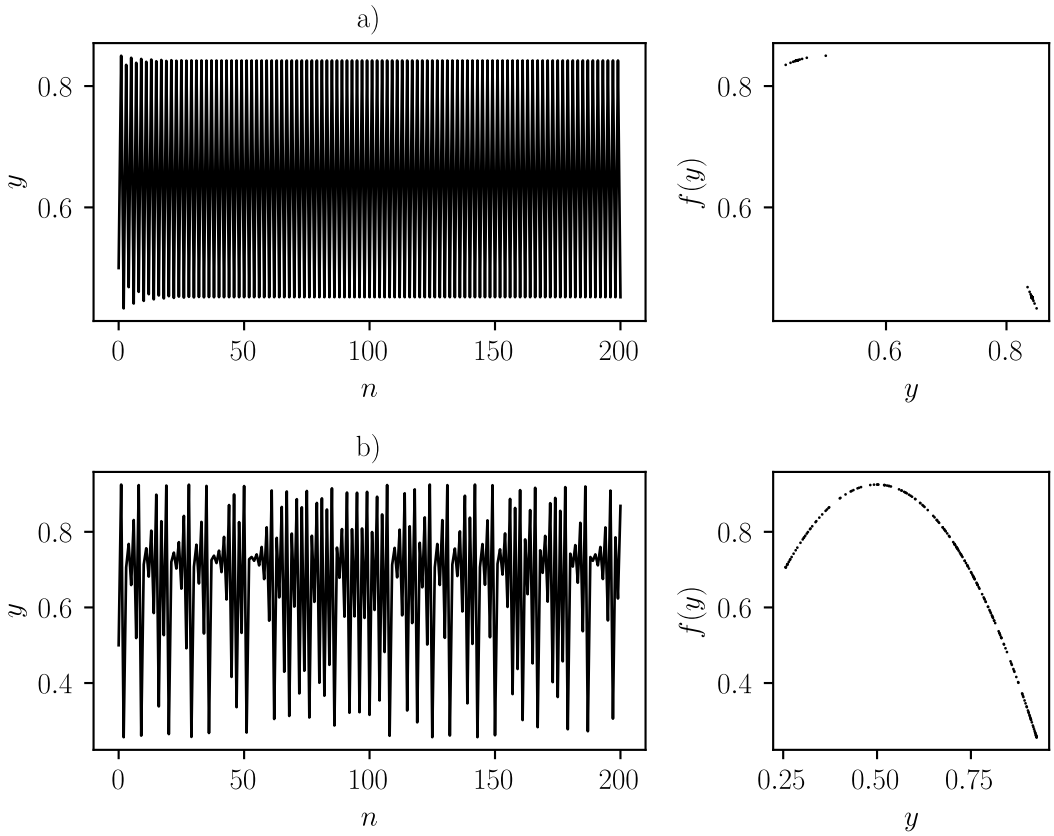


Figure 1.1: a) Timeseries and phase space of the logistic map with $r = 3.4$. b) Timeseries and phase space of the logistic map with $r = 3.7$.

When $r = 3.4$, there is a transient phase[1] which converges into a 2-cycle orbit as demonstrated in figure. When $r = 3.7$, there is no consistent periodic behaviour present, making this system aperiodic. The phase space, after the transient phase and when $n \to \infty$, is dense unlike the previous system which converges into an oscillation between two values.

To effectively study the behaviour of the entire system, a bifurcation diagram[2] was created by sampling 5000 values in $0 \leq r \leq 4$, and letting each system develop over 5000 iterations, where the first 500 were (qualitatively) assumed to be in a transient phase and thus discarded. This length of this transient phase is determined by the initial position, and can be monitored by studying the phase space. The result is shown in figure 1.2.

[1] Non-stable configuration that will relax into some stable configuration.
[2] A diagram where the stable points of a trajectory are plotted against a bifurcation parameter.

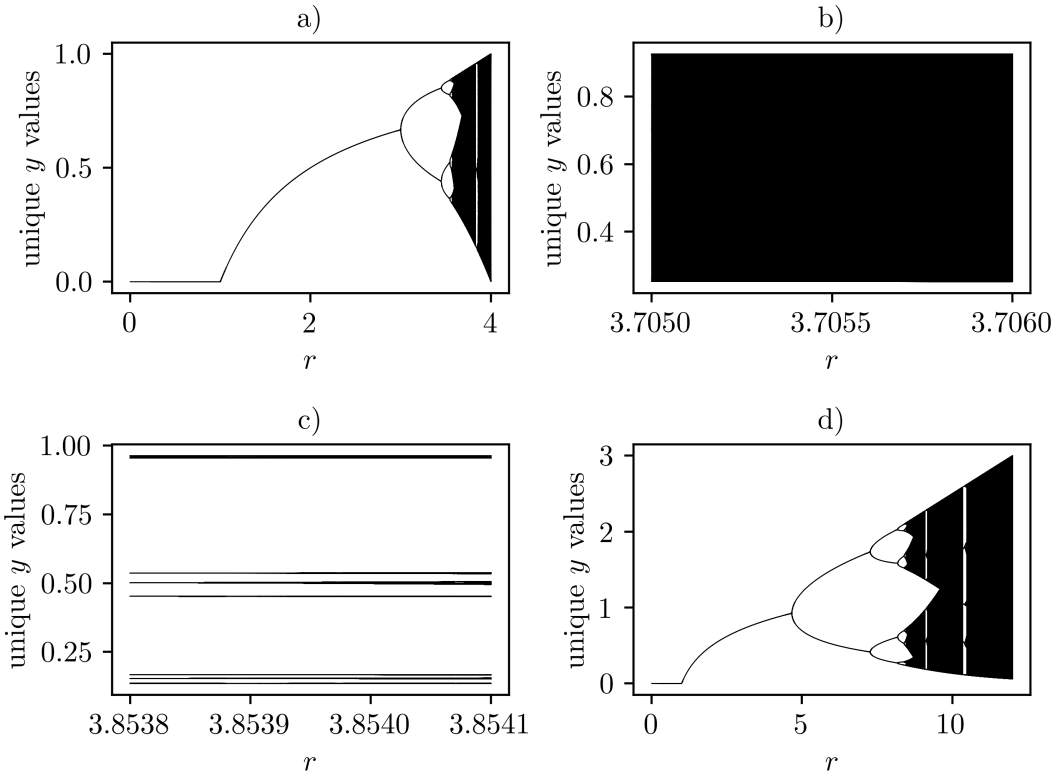


Figure 1.2: a) Bifurcation diagram of the logistic map with $0 \leq r \leq 4$. b) Bifurcation diagram of the logistic map with $3.705 \leq r \leq 3.706$. c) Bifurcation diagram of the logistic map with $3.8538 \leq r \leq 3.8541$. d) Bifurcation diagram of $y_{n+1} = r\tanh(y_n)(1-\tanh(y_n))$ with $0 \leq r \leq 12$.

Between 0 and 3 in Fig 1.2a., the system converges to a stable point. Between 3 and approximately 3.45 the system develops a 2-cycle, which keeps doubling until a critical point of $r_c \approx 3.5699456$ where there is an infinite cycle, or what was previously referred to as aperiodic behaviour. Beyond this threshold, most of the region is aperiodic, with short intervals of periodic behaviour. Two of these regions are shown in Fig. 1.2b and 1.2c. The bifurcation diagram can also be viewed in a different light; when r crosses the critical value r_c, the logistic map becomes space-filling, the dimension of this 'curve' is not actually 1, it has a fractal dimension. This fractal nature can be related to the Hausdorff dimension which is 0.538 at $r = r_c$. An interesting thing to note is that all one-dimensional non-linear maps with parabolic maxima display the same behaviour, known as the Feigenbaum universality. An example of this phenomenon is demonstrated in Fig. 1.2d, where $y_n \to \tanh(y_n)$ in eq. 1.5.

1.1.2 The Lyapunov exponent for maps

As mentioned before, the hallmark of Chaos is the sensitivity to initial conditions. In the literature, a measure for this sensitivity is the Lyapunov exponents, which are the exponential divergence between two neighbouring trajectories. In total, there are as many of these exponents as there are dimensions, where the largest of these will be denoted as Λ, and all the Lyapunov

exponents as a whole will be referred to as the Lyapunov spectrum.

$$||y'(t) - y(t)|| = ||\delta y(t)|| = ||\delta y(0)||e^{\Lambda t}. \tag{1.6}$$

This exponential divergence will only last for a short time if the system is compact. In this thesis, a slightly different notion will be used to mitigate this problem; the exponent will be defined by the average exponential divergence of a neighbouring trajectory at every time point. This definition is valid and a direct consequence of Oseledec's Theorem [13] which is the theorem of multiplicative ergodicity, and reflects the methodology for numerically deriving this exponent.

$$||\delta y(t + \Delta t)|| = ||\delta y(t)||e^{\Lambda \Delta t}. \tag{1.7}$$

In the context of maps, this is changed to:

$$||\delta y_{n+1}|| = ||\delta y_n||e^{\Lambda} \tag{1.8}$$

from which we can get the expectation value of the Lyapunov exponent:

$$\Lambda = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-2} \ln\left(\frac{||\delta y_{n+1})||}{||\delta y_n||}\right) \tag{1.9}$$

The divergence of trajectories can be linked to the local Jacobian by letting $\delta y_n \to dy_n$. Therefore, the exact definition of the largest Lyapunov exponent is:

$$\Lambda = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-2} \ln(\|J(y_n)\|) \tag{1.10}$$

Where $\|\cdot\|$ is the natural matrix norm, and $J(y_n)$ is defined as:

$$J(y_n) = \frac{dy_n}{dy_0} = \Pi_{i=0}^{n-1} \frac{dy_{i+1}}{dy_i}. \tag{1.11}$$

In the next chapter, the Lyapunov exponent will be derived for continuous systems. Revisiting the logistic map, the same setup for the bifurcation diagram was used. The Lyapunov exponent was calculated for 5000 values of r in various ranges. To do so, a timeseries of 5000 points was generated ($y(0) = 0.5$), where the 500 first were discarded to ensure no transient dynamics were captured.

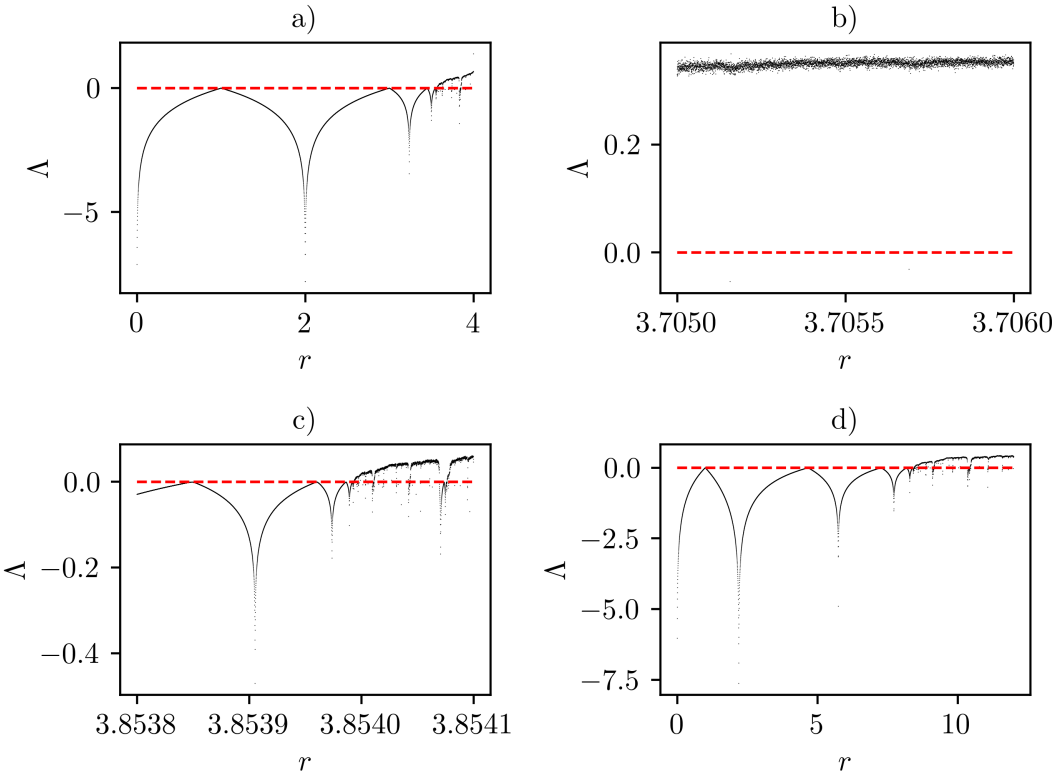


Figure 1.3: a) Lyapunov exponents of the logistic map with $0 \leq r \leq 4$.b) Lyapunov exponents of the logistic map with $3.705 \leq r \leq 3.706$.c) Lyapunov exponents of the logistic map with $3.8538 \leq r \leq 3.8541$. d) Lyapunov exponents of $y_{n+1} = r\tanh(y_n)(1-\tanh(y_n))$ with $0 \leq r \leq 12$.

In this plot, it is displayed that there are many semi-stable trajectories when $0 \leq r \leq r_c$ where $\Lambda = 0$. Above the critical value, there are intervals present of positive and negative Lyapunov exponents. In the chaotic intervals, as seen in Fig. 1.3b, the dynamics are more or less equivalent, with some fluctuations present due to the timeserie being finite. In the periodic intervals in the chaotic region shown by Fig. 1.3c something fascinating takes place. The Lyapunov exponents are fractal and self-similar in nature. This demonstrates the complexity that can arise in non-linear dynamics, even for trivial-appearing systems. Fig. 1.3d demonstrates that Feigenbaum universality shows up in the Lyapunov exponent behaviour.

1.1.3 Chaos

There is still no agreement on a proper and final definition of Chaos in the literature. For our purposes, a qualitative description is enough. According to *Non-linear dynamics and chaos* [11] by Steven H. Strogatz, chaos can be summarized into three points:

1. "Aperiodic long-term behaviour" means that there are trajectories which do not settle down to fixed points, periodic orbits, or quasiperiodic orbits. For practical reasons, we should require that such trajectories are not too rare. For instance, we could insist that

there be an open set of initial conditions leading to aperiodic trajectories, or perhaps that such trajectories should occur with non-zero probability, given a random initial condition.

2. "Deterministic" means that the system has no random or noisy. inputs or parameters. The irregular behaviour arises from the system's nonlinearity, rather than from noisy driving forces.

3. "Sensitive dependence on initial conditions" means that nearby trajectories separate exponentially fast, i.e., the system has a positive Lyapunov exponent.

The Lyapunov exponent is frequently used as a test for Chaos, and will be the main subject of study in this thesis.

1.2 Non-linear Differential equations

The language of nature is differential equations. Non-linear differential equations are fundamental in describing a vast array of physical phenomena, ranging from the dynamics of planetary systems to the behaviour of fluid flow and the complex interactions within biological systems. Unlike their linear counterparts, non-linear differential equations exhibit behaviours that are often far more intricate, similar to the transition between linear and non-linear maps.

In physics, non-linear differential equations often emerge when the system's response is not proportional to the applied forces or when feedback mechanisms create complex interactions between system variables. For instance, the motion of a pendulum becomes non-linear for large oscillations due to the sinusoidal term in its governing equation. Similarly, non-linearities in fluid dynamics, governed by the Navier-Stokes equations, can lead to turbulence, one of the most challenging and poorly understood phenomena in classical physics. Other examples include the non-linear Schrödinger equation in quantum mechanics and non-linear field equations in general relativity.

In this thesis only systems of ordinary differential equations (ODEs) will be discussed. These equations are of the form:

$$\dot{y} = f(y) \tag{1.12}$$

Where $y(t)$ is a vector dependent on time. We require that $f(\cdot) : U \to \mathbb{R}^n$ defined on some open set $U \subset \mathbb{R}^n$ is a C^r function, implying that it is differentiable to some order $1 \leq r$. Partial differential equations can display chaotic behaviour [14]. An example of such a system is the Kuramoto-Sivashinsky equation:

$$\frac{\partial^4 y}{\partial x^4} + \frac{\partial^2 y}{\partial x^2} + y\frac{\partial y}{\partial x} + \frac{\partial y}{\partial t} = 0 \tag{1.13}$$

which is a forth-order partial differential equation that is chaotic for some domain sizes [15]. The amount of storage and raw compute necessary to study the parameter space of partial differential equations, in combination with the absence of literature of some techniques that will be described in the next section pose a roadblock, and is outside of the scope of this master thesis.

1.2.1 Lyapunov spectrum in continuous systems

Tracking the dependence on initial conditions of continuous systems is slightly more tricky. The Jacobian in eq. 1.11 only applies to maps since $t \in \mathbb{R}$ is connected in $\mathbb{R}$ but $n \in \mathbb{N}$ is not connected

in $\mathbb{R}$. To circumvent this issue, the concept of flow [16] is introduced. The flow $F^{\cdot}(\cdot) : U \times \mathbb{R} \to \mathbb{R}^n$ generated by the values of $y(t)$ generated by some function $f(\cdot)$ starting at an initial condition $y_0 \in U$ over some time $t \in \mathbb{R}$ such that:

$$\dot{F^t}(y) = f(F^t(y)). \tag{1.14}$$

The set $\{F^t(y_0)|t \in \mathbb{R}\}$ is the trajectory through y_0. From this form, we can define the *Variational Equation* which defines the tangent flow, which is the tangent space to the trajectory generated at some y_0 over some time t:

$$\dot{T}_t(y_0) = \frac{df(F^t(y_0))}{dy} T_t(y_0) = J_y(f(F^t(y_0)))T_t(y_0). \tag{1.15}$$

The Lyapunov spectrum can be found via the QR Decomposition method [17], which states that the exponents can be found via a QR decomposition of the tangent flow. Let us define $D \equiv J_y(f(F^t(y_0)))$, from which we say that:

$$T_t(y_0) = Q_t R. \tag{1.16}$$

Where Q is an orthogonal matrix and R is an upper triangle matrix with positive diagonal elements R_{ii}. Inserting this into eq. 1.15 we get:

$$\dot{Q}R + Q\dot{R} = DQR, \tag{1.17}$$

from which the equation for the diagonal elements can be found:

$$\dot{R}_{ii} = (Q^T DQ)R_{ii}. \tag{1.18}$$

From these diagonal elements, the Lyapunov exponents are:

$$\Lambda_i = \lim_{t \to 0} \frac{1}{t} \ln(R_{ii}) \tag{1.19}$$

The Lyapunov exponent Λ is now $\max\{\Lambda_i \| 0 \leq i \leq n-1\}$. In many situations, this R_{ii} will diverge rapidly. To counter this, the log of the norm $\|R_{ii}(t)\|$ when solving eq. 1.18 will be tracked. Using the Euler method, one can track the norm and use the following algorithm to find the Lyapunov Spectrum:

Algorithm 1: Euler QR decomposition Lyapunov Spectrum

Data: D,N,dt
Result: Lyapunov Spectrum Λs
$l \leftarrow \mathbf{0}$;
$T \leftarrow \mathbb{I}$;
$n \leftarrow 0$;
for N **do**
  $T \leftarrow (\mathbb{I} + D_n dt)T$;
  $Q, R \leftarrow$ QR-decomp(T);
  $l \leftarrow l + \ln(|diag(R)|)$;
  $T \leftarrow Q$;
  $n \leftarrow n + 1$;
end
$\Lambda s \leftarrow \frac{l}{Ndt}$;

When this algorithm is fed with equidistantly sampled D_n, it will produce an approximation for the Lyapunov spectrum, the largest of which is the Lyapunov exponent. Another option is to use a Runge-Kutta scheme, which will be more precise at the cost of compute time. The following algorithm uses RK4 instead of the Euler method:

Algorithm 2: RK4 QR decomposition Lyapunov Spectrum

Data: D,N,dt
Result: Lyapunov Spectrum Λs
$l \leftarrow \mathbf{0}$;
$T \leftarrow \mathbb{I}$;
$n \leftarrow 0$;
for N **do**
  $k_1 \leftarrow D_n T$;
  $k_2 \leftarrow D_n(T + dtk_1/2)$;
  $k_3 \leftarrow D_n(T + dtk_2/2)$;
  $k_4 \leftarrow D_n(T + dtk_3)$;
  $T \leftarrow T + (dt/6)(k_1 + 2k_2 + 2k_3 + k_4)$;
  $Q, R \leftarrow$ QR-decomp(T);
  $l \leftarrow l + \ln(|diag(R)|)$;
  $T \leftarrow Q$;
  $n \leftarrow n + 1$;
end
$\Lambda s \leftarrow \frac{l}{Ndt}$;

1.2.2 Lorenz-63 and strange attractors

Chaos does not have to occur in the entire space a system is defined on. A system can also be chaotic in subsets of its phase space when it is attracting. An attracting subset implies that any trajectory that enters the subset will stay in the subset. When an attracting subset is chaotic, we refer to it as a *strange attractor*.

The best understood system that has a strange attractor is the Lorenz-63 system, which was the first one discovered by Lorenz in the early 60s.

$$\begin{aligned} \dot{y_0} &= \sigma(y_1 - y_0) \\ \dot{y_1} &= y_0(\rho - y_2) - y_1 \\ \dot{y_2} &= y_0 y_1 - \beta y_2. \end{aligned} \tag{1.20}$$

This 3-dimensional system of ordinary differential equations has become a litmus test for many tools related to modeling non-linear dynamics, and will be the system of focus in this thesis. As mentioned before, this system reflects convection in the atmosphere, where y_0 is related to the convection rate, y_1, y_2 is related to horizontal and vertical temperature variation. The parameters σ, ρ, and β are related to the Prandtl number, Rayleigh number, and the dimension of the layers [2]. To demonstrate the emergence of chaos, three different timeseries with a starting

position of $(1, 0, 0)$ over a period of 50 time units were generated, where ρ was varied between 20 and 28.

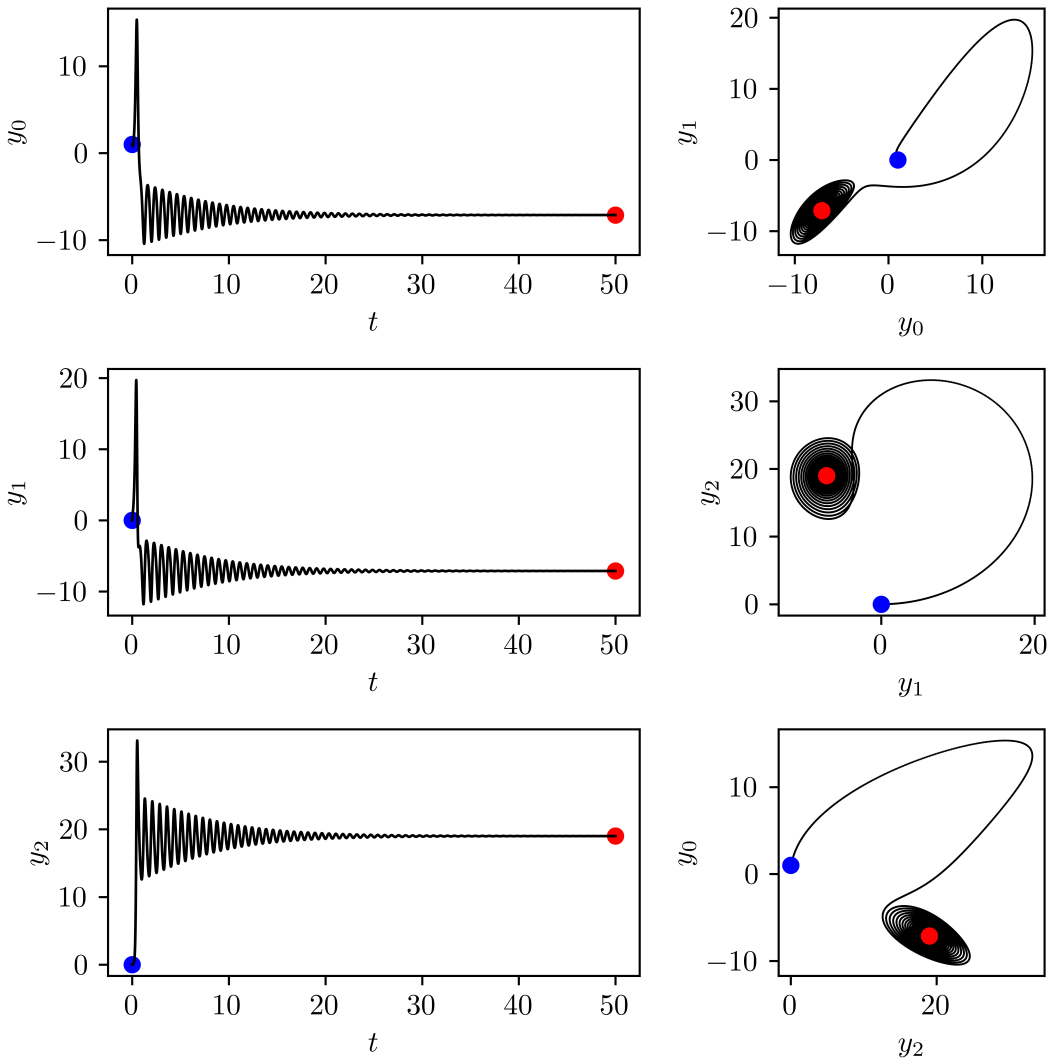


Figure 1.4: Timeseries and phase space of the Lorenz-63 system with $\sigma = 10$, $\rho = 20$, and $\beta = 8/3$.

When ρ is small enough, a dissipative system arises where the transient dynamics fade into a constant value. This constant value is referred to as a limit point.

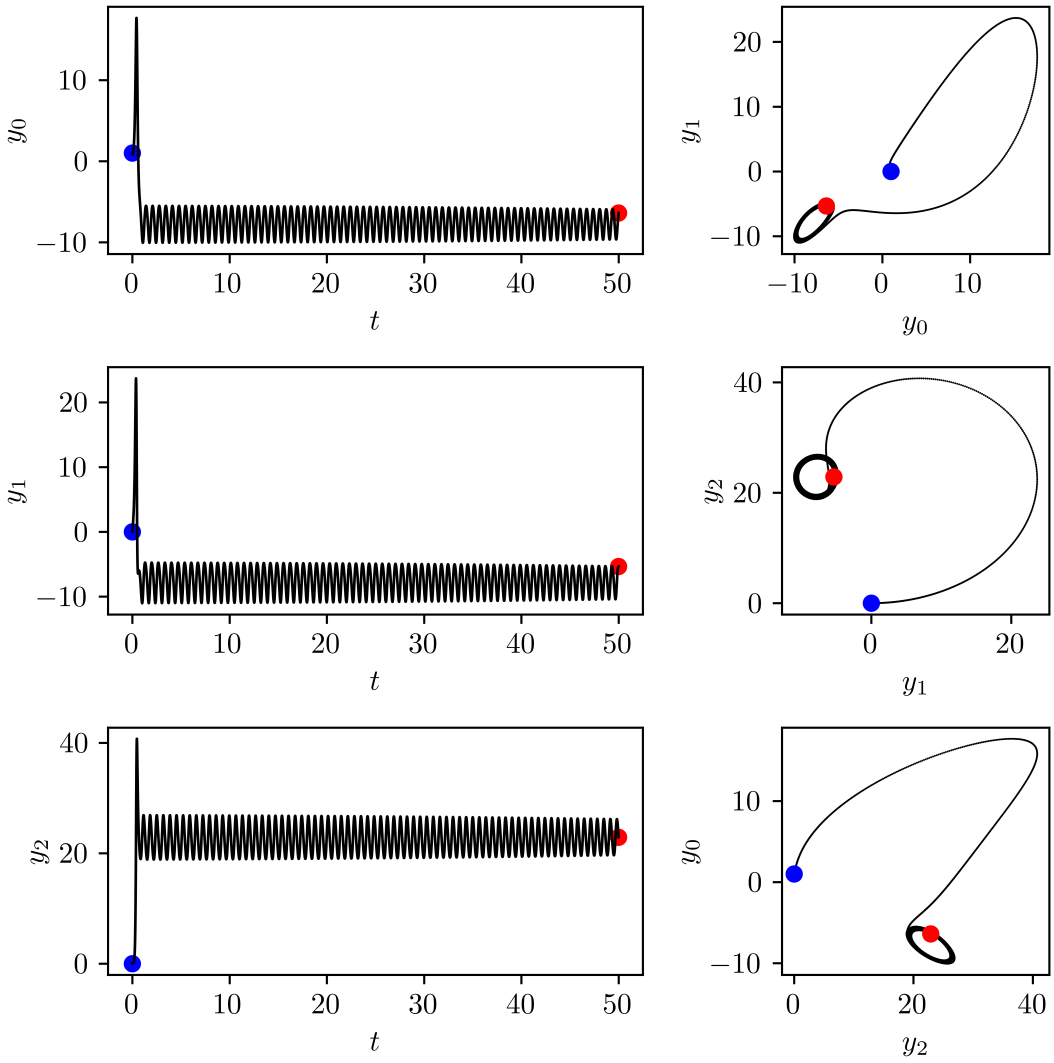


Figure 1.5: Timeseries and phase space of the Lorenz-63 system with $\sigma = 10$, $\rho = 24$, and $\beta = 8/3$.

Increasing the value of ρ to 24 shows that an almost-periodic system arises, which is still slightly dissipative as seen in the phase space plot. This deceivingly suggests that when ρ increases, the system will move to a non-dissipative periodic orbit.

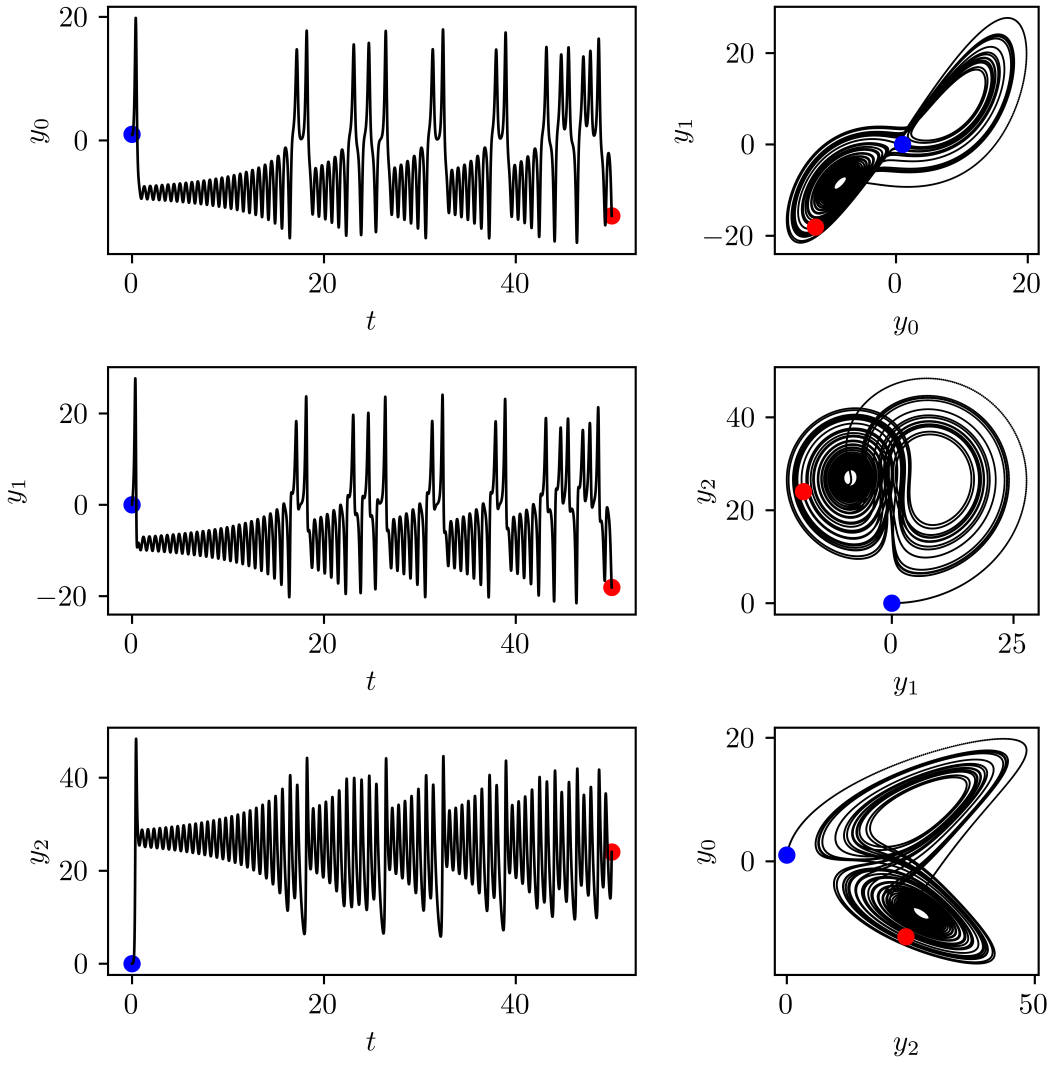


Figure 1.6: Timeseries and phase space of the Lorenz-63 system with $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$.

When ρ crosses a critical value of $\rho_{crit} \approx 24.74$, the limit points turn into repulsors. The trajectory becomes a phase space-filling curve with a dimension of approximately 2.06. This space-filling curve suggests that the system has moved into a chaotic region, which implies that the system contains a strange attractor. To verify this, the values σ, ρ and β were varied to determine which regions in the parameter space are chaotic. The values that were not varied were kept constant at the $\sigma = 10$, $\rho = 28$, or $\beta = 8/3$. 20 Equidistant values were sampled to find the 2 nearest-to-0 points, and a linear interpolation between the two was used to find the approximate zero-point. The timeseries were generated for $t \in [0, 200]$ time units with a timestep of 0.01.

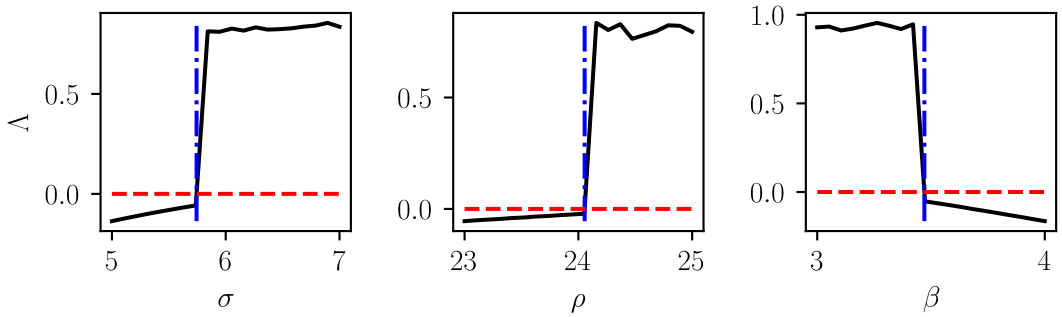


Figure 1.7: a) Lyapunov exponent of the Lorenz-63 system with $5 \leq \sigma \leq 10$, $\sigma_c \approx 5.74$. b) Lyapunov expone nt of the Lorenz-63 system with $20 \leq \rho \leq 28$, $\rho_c \approx 24.05$. c) Lyapunov exponent of the Lorenz-63 system with $8/3 \leq \beta \leq 14/3$, $\beta_c \approx 3.47$.

The zero-points indicate that an effect similar to a phase transition takes place. After crossing certain parameter values, a discontinuous jump in the Lyapunov exponent takes place. This jump takes place because the system either dissipates into the limit points or becomes chaotic; where the system is periodic exactly on the zero-point parameter values.

To investigate further, 3 figures were created where the Lyapunov exponent was calculated for varying $\sigma - \rho$, $\rho - \beta$, and $\beta - \sigma$. These plots map out a more complete picture of the parameter space, and are fundamental to understanding the behaviour of the Lorenz-63 system. Fig. 1.8 was generated from a 10×10 grid from timeseries between $t \in [0, 200]$ with a timestep of 0.01. The first 25% were discarded to avoid transient dynamics.

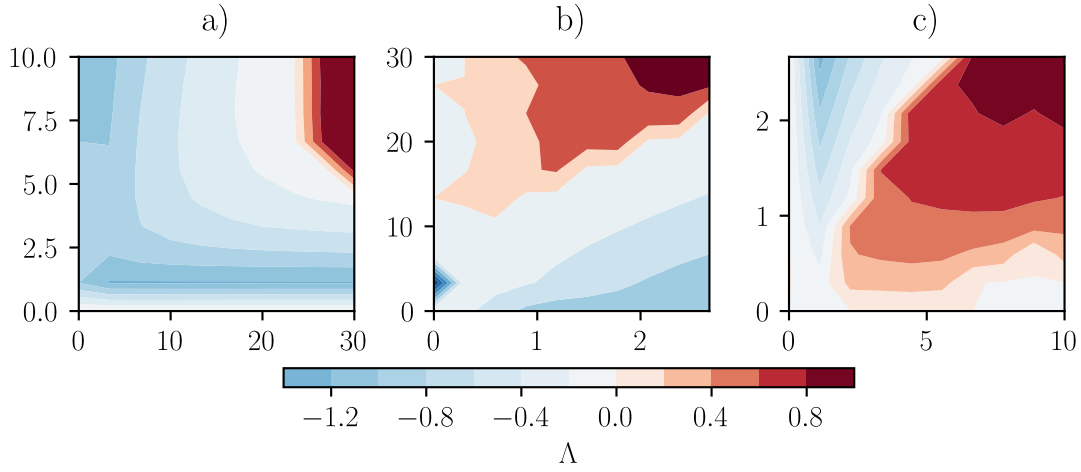


Figure 1.8: a) $\sigma - \rho$ plane Lyapunov exponent of the Lorenz-63 system with $\beta = 8/3$. b) $\rho - \beta$ plane Lyapunov exponent of the Lorenz-63 system with $\sigma = 10$. c) $\beta - \sigma$ plane Lyapunov exponent of the Lorenz-63 system with $\rho = 28$.

These plots show the continuation of Fig. 1.7 where the sharp boundary is again present. This

concludes an exploration of the chaotic dynamics of the Lorenz-63 system.

1.2.3 Parameter space complication and predictability time

Gathering Lyapunov exponents is computationally expensive. They require small timesteps for the QR-decomposition method to work. Not only that, but the contour plots generated above need to be generated sequentially and cannot be parallelized. When dealing with large parameter spaces, this can prove to be a large bottleneck when computation time is of relevance. Usually a sacrifice can be made by letting the timestep size increase. In the case of chaotic systems, a concept named *predictability time* renders this futile.

The Lyapunov exponent of a system gives a quantitative measure of the average exponential divergence. The definition of this exponent can be algebraically manipulated to the concept of predictability time in the following way:

$$\begin{aligned} \epsilon_p &= \epsilon_0 e^{\Lambda t_p} \\ \rightarrow t_p &= \frac{1}{\Lambda} \ln\left(\frac{\epsilon_p}{\epsilon_0}\right) \end{aligned} \tag{1.21}$$

where $\epsilon_p \equiv \epsilon(t_p)$ and $\epsilon_0 \equiv \epsilon(0)$, with $t_p > 0$. This can be interpreted as some error ϵ_0 will on average grow or shrink to an error ϵ_p after some time t_p, where t_p is the predictability time. Therefore if a system has a positive Lyapunov exponent, then if the numerical solver makes an error ϵ_0 then it will grow to some ϵ_p after t_p time units. There is no way to circumvent this since the exponential growth of neighbouring trajectories is fundamental to chaotic systems. Eq. 1.21 is not quite correct in the case of numerical integration or any other iterative method. The error of the previous iteration will propagate according to the Lyapunov exponent, and then will make a new error unrelated to the propagation. **The author suggests a novel approximation:**

$$|\epsilon(t + \Delta t) - \epsilon(t)| \approx \sqrt{(\epsilon(t)\bar{\Lambda}(\Delta t)\Delta t)^2 + (\delta \Delta t)^2}. \tag{1.22}$$

This equation comes forth when one assumes that an iterative process generates a normally distributed error with zero-mean and standard deviation δ_m, and is intuitively derived by starting from the definition of a positive Lyapunov exponent, which defines exponential propagation of a perturbation, and adding this normally distributed error as a source term. The $\bar{\Lambda}$ is used to indicate that the Lyapunov exponent of the approximated system might not be equal to the true Lyapunov exponent. The propagated error and source term error are assumed to be orthogonal because of the symmetry that cancels non-orthogonal directions when examining the expectation value of the sum of two random vectors $\mathbb{E}\{\|a + b\|\} = \sqrt{\|a\|^2 + \|b\|^2}$. This can be turned into a differential equation by letting $\Delta t \rightarrow 0$ and assuming that $\bar{\Lambda} \rightarrow \Lambda$:

$$\dot{\epsilon} = \sqrt{\Lambda^2 \epsilon^2 + \delta_m^2} \tag{1.23}$$

which can be solved for $\Lambda \geq 0$ via hyperbolic substitutions to be ($\epsilon(0) = 0$):

$$\epsilon(t) = \frac{\delta_m}{\Lambda} \sinh(\Lambda t). \tag{1.24}$$

To get a result for $\Lambda \leq 0$, an analytical continuation $\Lambda \rightarrow i\Lambda$ in eq. 1.22 can be used to get a shrinking error propagation term, and results in ($\epsilon(0) = 0$):

$$\epsilon(t) = \frac{\delta_m}{|\Lambda|} \sin(|\Lambda| t), \tag{1.25}$$

which does not align with reality because $\dot{\epsilon} \geq 0 \ \forall t$. The result for $\Lambda < 0$ is therefore unreliable. From intuition it is clear that $0 \leq \epsilon \leq \delta_m/(\Lambda \Delta t)$, where the upper bound is reached when the source term δ_m is in balance with the error propagation $\Lambda \Delta t$. In the limit where $\Lambda \to 0$, the l'Hôpital's rule dictates that $\epsilon(t) = \delta_m t$. Therefore, the error growth rules can be summarized as:

$$\epsilon(t) = \begin{cases} \frac{\delta_m}{\Lambda} \sinh(\Lambda t) & \text{if} \quad \Lambda > 0 \\ \delta_m t & \text{if} \quad \Lambda = 0 \\ \frac{\delta_m}{|\Lambda|} \sin(|\Lambda| t) & \text{if} \quad \Lambda < 0 \quad \text{and} \quad |\Lambda| t < \pi/2 \\ \frac{\delta_m}{|\Lambda|} & \text{if} \quad \Lambda < 0 \quad \text{and} \quad |\Lambda| t \geq \pi/2 \end{cases} \tag{1.26}$$

Generally, it is not possible to find some exact value for δ_m, in the case of RK4, the error has to be of order $\delta \Delta t \sim \Delta t^5 \to \delta \sim \Delta t^6$ [18]. A similar predictability time can now be derived when $\Lambda \geq 0$:

$$t_p = \frac{1}{\Lambda} \text{arcsinh}(\frac{\Lambda \epsilon_p}{\delta_m}) \tag{1.27}$$

when $\Lambda \leq 0$, the equation changes to

$$t_p = \frac{1}{\Lambda} \text{arcsin}(\frac{\Lambda \epsilon_p}{\delta_m}) \tag{1.28}$$

Where $\epsilon_p > \frac{-\delta_m}{\Lambda}$, which is because negative Lyapunov exponents put a limit on how large ϵ_p can become. To demonstrate the severity of this issue, 1000 identical systems were solved with the RK4 solver with different timesteps, and the time until an error of 0.1 with respect to the system with the smallest dt was tracked. The initial position was set such that it was located in the attractor where $\sigma = 10$, $\rho = 28$, $\beta = 10$. The estimation of the Lyapunov exponents for 20 samples in $\Delta t \in [10^{-3}, 10^{-1}]$ was measured for timeseries with $t \in [0, 1000]$. To get a estimation for δ_m, it was modelled to be of the form $\delta_m(dt, a, b) = a dt^b$, and the mean squared error (MSE) between t_p and its true value was minimized via the Levenberg–Marquardt algorithm for for 100 samples of $\Delta t \in [10^{-4}, 1]$ (where the estimation of Λ was more or less accurate).

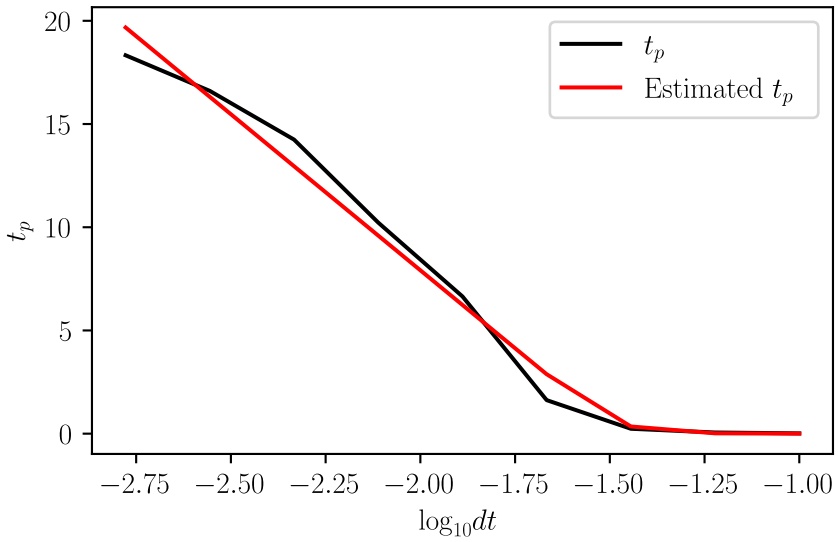


Figure 1.9: Predictability time and estimated predictability time for $\epsilon = 1$ and $\delta - m(dt) = adt^b$ with $a = 1.43 \cdot 10^6 \pm 1.13 \cdot 10^5$ and $b = 5.97 \pm 0.13$.

The values of a and b fall in line with the estimation of δ_m being dependent on Δt^{-6}, giving validity to eq. 1.25. The real and estimated predictability time reflect that for less accurate numerical schemes the driving force for error is the numerical error; meanwhile, more accurate schemes suffer from limitations imposed by the Lyapunov exponent of the system.

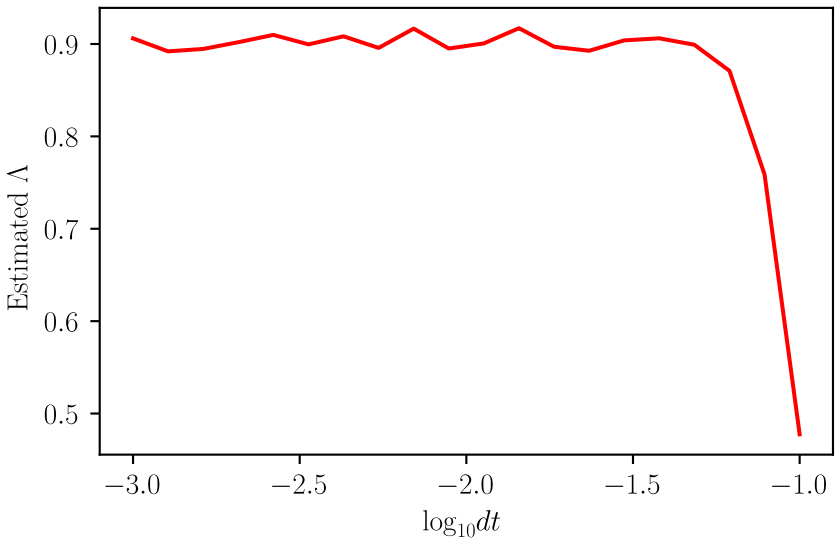


Figure 1.10: Lyapunov exponent estimation for varying timesteps.

The Lyapunov exponent is not very dependent on the timestep size, as seen in Fig. 1.10. When the timestep for this system is smaller than $10^{-1.5}$, the results are almost identical. This shows that capturing the behaviour of a system is much less intensive than capturing correct trajectories when a system is chaotic.

The dependence of predictability time on the timestep size of the integration scheme alerts us that it is very necessary to know the Lyapunov exponent governing a system. Calculating this exponent requires a large amount of timeseries data for every parameter realization, which might not always be feasible. An even larger issue appears when the parameter realization of some experimental timeseries is unknown or hard to reconstruct, making it infeasible to predict if a system is chaotic, and therefore estimate the predictability time.

Chaos also leads us to a more metaphysical re-evaluation of our methods. Reductionism is the underlying philosophy of physics. Reducing problems into their individual components to understand reality has been fruitful in most avenues. Complexity science is composed of many different fields where reductionism fails to provide meaningful answers; emergent phenomena cannot be described by studying components (phase transitions in Ising models). Even though Chaos theory is not situated in Complexity science, it can be viewed in a similar light. The need for a more holistic approach provides incentive to move beyond numerical integration schemes, and explore the viability of artificial intelligence. Numerical integration can not be parallelized efficiently, unlike many machine learning models which inherently forward data in parallel via matrix multiplications. This capability might be very useful when the training time of a model is shorter than the sequential integration time of the system. To use Artificial Intelligence as a method to advance incomplete models, or to discover new phenomena, the foundations have to be constructed first.

Machine Learning

The sciences have slowly but surely accepted the necessity for Machine Learning in the scientific process. An example in the field of particle physics is the the ATLAS project at CERN. ATLAS employs machine learning techniques to investigate atypical decay patterns associated with long-lived particles (LLPs) [19]. The algorithms are specifically designed to detect rare events, such as displaced jets and unexpected energy deposits, which stand out from conventional collision signatures. By leveraging these models, researchers can effectively filter out background noise, thereby enhancing the sensitivity of LLP detection. The focus lies on unique spatial and energetic signatures captured within the detector. This methodological advancement using machine learning significantly boosts the efficiency and precision in pinpointing potential LLP events amidst the extensive data generated by particle collisions.

The 2024 Nobel prize in physics [5] was unsurprisingly controversial in the physics community and seems to indicate a cultural shift in the direction of physics research. Many doubt whether Hopfield and Hinton's work should have been eligible for a Nobel Prize in physics in the first place. It can be argued that they are since their work is founded on the principles of the Ising model, which is an ongoing field of study in statistical physics. It is clear that Machine Learning's presence in the field is growing, and therefore warrants an exploration into its capabilities in modeling non-linear dynamics.

Neural Networks have gained the most traction since most personal computers are now equipped with the computational power necessary to train them. However, some newer architectures and more advanced problems setting still require intensive training cycles. These training cycles come in the form of gradient-based optimizers, which update the parameters of the model by taking the derivative of some loss function with respect to these parameters and updating them accordingly. Such optimizers frequently require on the order of $10^4 - 10^5$ iterations to converge on some (local) minimum.

A clever solution to this issue is the *Extreme Learning Machine* (ELM) which, in essence, is a randomized single-layer Neural Network equipped with a linear readout that is trainable via Linear Regression. This training style can be described as learning from a random non-linear high-dimensional embedding. In the upcoming sections, the theory of basis expansion will be explained, which smoothly transitions into Extreme Learning. Its capabilities will be demonstrated in comparison to standard Neural Networks.

2.1 Supervised learning

There are three branches in Machine Learning; namely supervised learning, unsupervised learning, and reinforcement learning [20]. The branch of relevance in this thesis is supervised learning, where the aim is to learn some map $F : x \in X, y \in Y \rightarrow F(x) = y$. Where X can be a collection of discrete and continuous compact subspaces . When Y is a collection of discrete compact subspaces, then approximating the map $F(\cdot)$ is referred to as a classification problem. On the other hand if Y is a collection of continuous compact subspaces, then approximating the map $F(\cdot)$ is called a regression problem. In the case of regression, X is usually a collection of continuous compact subspaces as well.

'Learning' is quite a vague term. It is easier to first define the properties of a model $G_\theta(\cdot)$ that has learned some map $F(\cdot)$, a model $G_\theta(\cdot)$ defined on the same space as $F(\cdot)$ is ϵ-close to a map $F(\cdot)$ if $\forall x \in X : \mathcal{L}(F(x), G_\theta(x)) < \epsilon$, where $\mathcal{L}(\cdot, \cdot)$ is some metric of interest. This metric is called the loss function. Learning the parameters θ is now the process:

$$\arg\min_\theta \mathop{\mathbb{E}}_{x \sim D} \{\mathcal{L}(F(x), G_\theta(x)\} \tag{2.1}$$

where x is sampled from some distribution D. The goal of all machine learning models is to 'generalize' from some sampled couples $\{x, y\}$ to the entire subspace $\{X, Y\}$ they were sampled from. Generalization implies that the values of θ found in the the sampled couples and values found if one were to draw infinite samples covering the entire subspace are approximately equal. This can be formulated as:

$$\arg\min_\theta \mathop{\mathbb{E}}_{x \sim D} \{\mathcal{L}(F(x), G_\theta(x)\} \approx \arg\min_\theta \frac{1}{|\{X\}|} \int_{x \sim D} \mathcal{L}(F(x), G_\theta(x)) dx \tag{2.2}$$

The obstacle of this minimization process is that eq. 2.1 is descriptive, not prescriptive. Carrying out this minimization is done by letting some algorithm, namely an optimizer, find the best parameters. In certain cases, this optimization process can be done in one step. An example of this is Linear Regression, where the loss function is the mean squared error (MSE):

$$\mathcal{L}_{MSE}(y, G_\theta(x)) = \mathbb{E}\{(y - G_\theta(x))^2\} = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - G_\theta(x_i))^2 \tag{2.3}$$

Linear in this case refers to a model that is described by a first order polynomial:

$$G_\theta(x) = \theta_0 + \theta_1 x \tag{2.4}$$

Since the MSE of a linear function is always convex, and therefore only has 1 minimum, the global minimum can be found by asserting that the gradient equals 0. Let us say that $\theta_0 = 0$ for simplicity's sake, then for θ_1:

$$0 = \frac{\partial \mathcal{L}_{MSE}(y, G_\theta(x))}{\partial \theta_1} \tag{2.5}$$

$$= \frac{\partial}{\partial \theta_1} \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \theta_1 x_i)^2 \tag{2.6}$$

$$= \frac{1}{n} \sum_{i=0}^{n-1} 2(y_i - \theta_1 x_i) x_i \tag{2.7}$$

$$\rightarrow \theta_1 = \mathbb{E}\{\frac{y}{x}\} \tag{2.8}$$

When θ_0 is not set to zero, or x is a vector, the result is [21]:

$$\theta = (x^T x)^{-1} x^T y \tag{2.9}$$

The trick here is that θ_0 can be included by creating the couple $x \to x \otimes 1$, the value of θ associated with the feature that is always 1 is θ_1.

2.1.1 Basis expansion and regularization

A model will not always generalize to the entire input subspace. A trained model can suffer from two effects (not simultaneously), namely underfitting and overfitting. Both these effects will lead to the model's loss being larger over the entire input space than the training set. Underfitting, as the name suggests, occurs when a model is not expressive enough. For example, training a linear model to approximate a polynomial of a higher order. Let us define a polynomial function:

$$y(x) = 1.2x^2 - 2x + 1 \tag{2.10}$$

5 training samples were drawn uniformly from $-10 \leq x \leq 10$. The one-step training result is shown in fig. 2.2

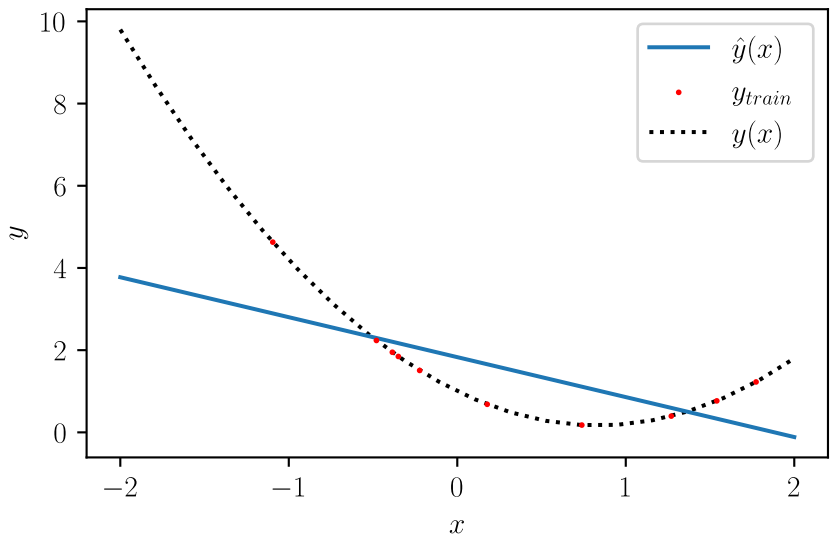


Figure 2.1: Linear Regression where $\hat{y}(x) = 1.83x - 0.97$, $\mathcal{L}_{train} = 0.67$ and $\mathcal{L}_{test} = 4.92$

The learned model did not generalize from the training data to the entire subspace, which in this case is underfitting. This is because a polynomial of order 1 cannot approximate a polynomial of order 2 since it cannot prove a basis. This can easily be solved by a trick called *basis expansion*. As the name suggests, the input space is embedded into a higher-dimensional space which is orthogonal. For a polynomial, this is simply $x \to 1 \otimes x \otimes x^2 \otimes ... \otimes x^r$. In the above example we let $x \to 1 \otimes x \otimes x^2$.

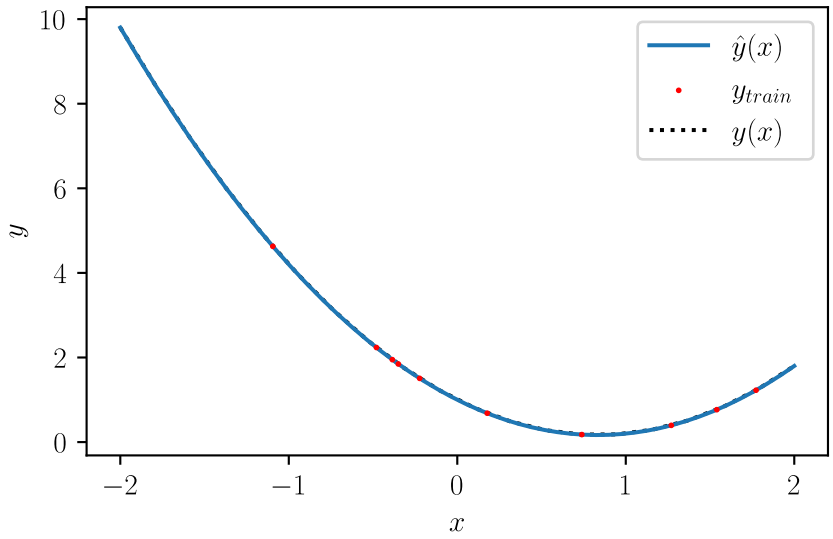


Figure 2.2: Linear Regression where $\hat{y}(x) = 1.2x^2 - 2x + 1$, $\mathcal{L}_{train} = 0.0$ and $\mathcal{L}_{test} = 0.0$

This does not result in an approximation, but in retrieving the actual function $y(\cdot)$. A basis expansion for linear equations is in general of the form:

$$\hat{y}(x) = \sum_{i=0}^{k} \theta_i b_i(x) = \theta \cdot b(x), \tag{2.11}$$

where $\cdot$ is the dot product. Because polynomials can provide a basis for any analytical function due to Taylor's theorem, then eq. 2.11 can learn any dataset exactly if $b_i(x) = x^i$ and there are $k + 1$ training samples. An issue arises when there is noise present. Assume that there is some normally distributed noise present in the observation of the training samples:

$$y_{train}(x) = 1.2x^2 - 2x + 1 + \epsilon \tag{2.12}$$

where $\epsilon \sim \mathcal{N}(0, 1)$. Let us draw 10 equidistantly sampled points as training data from $-2 \leq x \leq 2$, and learn a maximally complex polynomial of order 9.

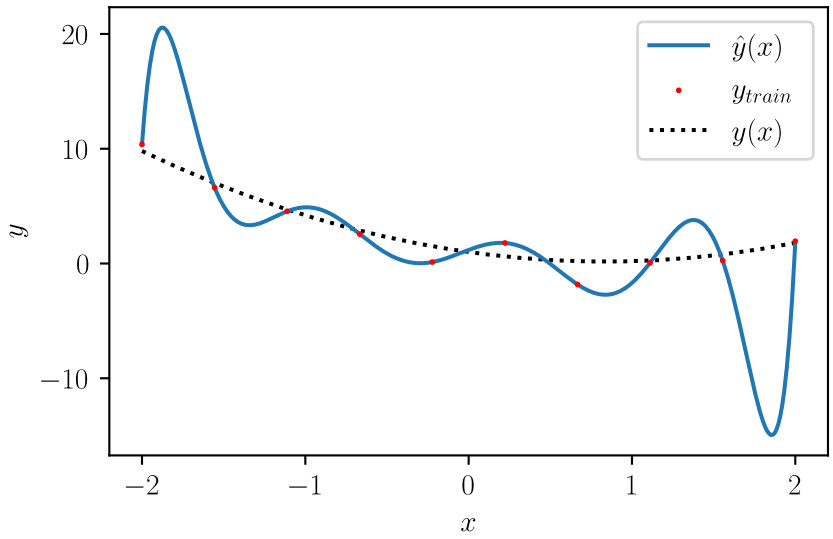


Figure 2.3: Polynomial Linear Regression where $\hat{y}(x) = 1.18 - 5.30x - 5.03x^2 - 33.42x^3 + 8.80x^4 + 38.04x^5 - 3.86x^6 - 15.12^7 + 0.51x^8 + 1.90x^9$, $\mathcal{L}_{train} = 0$ and $\mathcal{L}_{test} = 17.34$

A pattern that was not present in the map was learned. Overfitting occurs when a model that is overly complex is fed with too little data. There are three solutions to overfitting; reduce the complexity of the model before training, increase the amount of training samples, or *regularization*. As we will see later, some models require to be overly complex, and it is not always possible to increase the amount of training data to account for that complexity. Regularization is a strategy to artificially limit the complexity of a model by adjusting the loss function. The adjustments to this loss function are compared to some validation set to find the optimal configuration. There are two primary types of regularization, namely l_1 and l_2 (which refer to the l_1 and l_2 norm). These regularizations place a penalty on the norm of the found parameters. l_1 norm in practice is used to ignore features that do not carry information. l_2 Is used to penalize large values of parameters:

$$\mathcal{L}_{MSE,r}(y, G_\theta(x)) = \mathbb{E}\{(y - G_\theta(x))^2 + r|\theta|^2\} = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - G_\theta(x_i))^2 + r|\theta|^2 \tag{2.13}$$

for which an analytical solution exists:

$$\theta^r = (x^T x + r\mathbb{I})^{-1} x^T y \tag{2.14}$$

Only l_2 regularization will be used in this thesis because there is no analytical solution to l_1 regularized Linear Regression. Finding the best regularization value r and respective θ^r can be translated to the algorithm:

Algorithm 3: l_2-regularized Linear Regression

Data: x_{train},y_{train},x_{val},y_{val},r_s
Result: θ^r,r
$\mathcal{L}_{optimal} \leftarrow \inf$;
$r_{optimal} \leftarrow 0$;
$\theta^r_{optimal} \leftarrow None$;
for $r \in r_s$ **do**
  $\theta^r \leftarrow (x^T_{train}x_{train} + r\mathbb{I})^{-1}x^T_{train}y_{train}$;
  $\mathcal{L}_{val} \leftarrow MSE(y_{val}, \theta^r \cdot (1 \otimes x_{val}))$;
  if $\mathcal{L}_{val} < \mathcal{L}_{optimal}$ **then**
    $\mathcal{L}_{optimal} \leftarrow \mathcal{L}_{val}$;
    $r_{optimal} \leftarrow r_{val}$;
    $\theta^r_{optimal} \leftarrow \theta^r$;
  end
end
An extra validation set, generated equivalently to the training set. This validation set is usually separated internally from the training set. $\theta^r_{optimal}, r_{optimal}$;

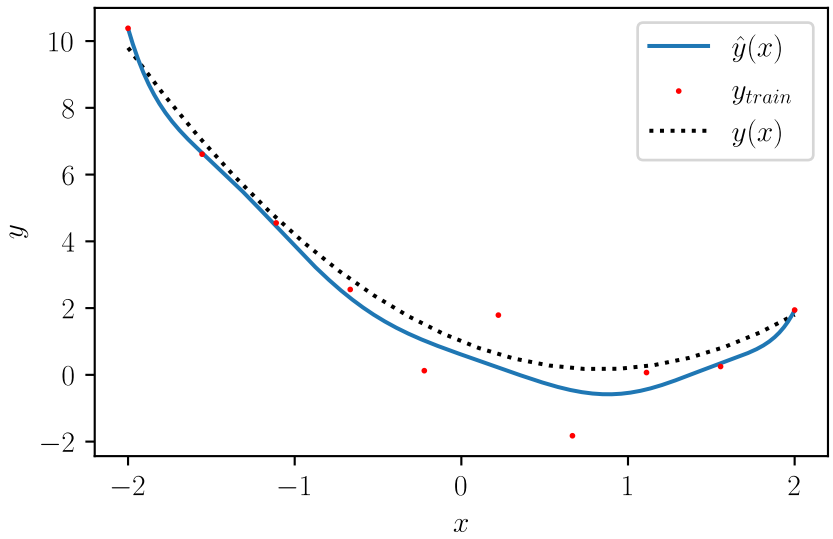


Figure 2.4: Polynomial l_2 Linear Regression where $\hat{y}(x) = 0.61 - 1.77x + 0.21x^2 - 0.97x^3 + 1.33x^4 + 0.70x^5 - 0.57x^6 - 0.17^7 + 0.07^8 + 0.01x^9$ and $r = 0.0933$, $\mathcal{L}_{train} = 0.55$ and $\mathcal{L}_{test} = 0.24$

l_2 regularization was able to reduce the validation error by eliminating large values of θ. Polynomial regression of order $k-1$ when k samples are present is a perfect example of overfitting. It does not actually learn a function, but rather performs an interpolation between points, specifically Lagrange interpolation [22]. This type of interpolation breaks down when the dimension of the input data is not 1-dimensional, however. To encompass any polynomial achievable by an interpolation for x of k-th order and d dimensions, the basis expansion would consist of the set of all multiplied polynomials up until power k as in Taylor's theorem for multivariate functions,

which yields:

$$\dim \theta = \sum_{j=0}^{k} \binom{d+j-1}{j} = \binom{d+k}{k} \tag{2.15}$$

if $y(x)$ is a vector of dimension D, then there would be

$$\dim \theta = D\binom{d+k}{k} \tag{2.16}$$

terms. To demonstrate the sheer magnitude, assume some function $f(\cdot)$ that takes an input of dimension 10 and maps it to 3 dimensions is interpolated by a polynomial of order 10. This would imply that $\dim \theta = 554268$, which would require at least 554269 training points. Another difficulty lies in the instability of polynomials due to their tendency to quickly diverge to infinity. This is computationally irresponsible, and brings us to a discussion of Neural Networks.

2.2 Neural Networks

Function approximation is not a trivial task, when a model is explainable a threshold is placed on the performance. Explainability, like for example basis expanded Linear Regression, serves a purpose when understanding the underlying process is of importance. These type of models are referred to as white box models due to their transparency. When the map that is trying to be approximated is too complex for most white box models and the underlying process is not that relevant, but the results it produces are, then more complex black box models are favourable.

Neural Networks are non-explainable models, constructed in a network-like structure where the connections are trainable matrices, and the nodes are non-linear functions (frequently called activation functions). Neural Networks are universal approximators for functionals [23]:

$$|f(x) - \theta_W^1(\theta_W^0 x + \theta_b^0)| < \epsilon \tag{2.17}$$

As long as they are sufficiently parametrized, they can learn (almost) every function. The standard feedforward Neural Network layer has the following architecture:

$$\mathcal{N}_{\theta^i}^i(x) = f^i(\theta_W^i x + \theta_b^i) \tag{2.18}$$

The entire architecture of an n layer Neural Network that has a layer width of k is the composition of all its layers.

$$\mathcal{N}_\theta(x) = \mathcal{N}_{\theta^n}^n \circ \mathcal{N}_{\theta^{n-1}}^{n-1} \circ ... \circ \mathcal{N}_{\theta_0}^0(x) \tag{2.19}$$

The matrices θ_W^i are shaped such that they lift from the input dimension to a latent dimension if $i = 0$, go down to the output dimension if $i = n-1$, otherwise propagate in a hidden dimension. Respectively they would be of the form $\mathbb{R}^{k \otimes d}$, $\mathbb{R}^{D \otimes k}$, and $\mathbb{R}^{k \otimes k}$. $f^i(\cdot)$ when $i \neq n-1$ is some non-linear (activation) function, and the identity function when $i = n-1$. The activation function used is dependant on the situation, but should not make a large impact as long as it is non-linear.

Finding the optimal θ to approximate a function cannot be solved analytically, nor can a global minimum be found in most situation. Finding these local minimum is done by gradient-based optimizers. Non-gradient-based optimizers, like metaheuristic optimizers, are not used frequently because they require many more function calls than their gradient counterpart.

2.3 Extreme Learning Machines

Neural Networks are slow to train, they require many function calls for the optimizer to converge to a minimum. Extreme Learning [24] optimization can be achieved in one step because it relies on Linear/Ridge Regression. Extreme Learning Machines (ELMs) are very wide 1-layer Neural Networks that are randomly initialized and left untrained. The readout is trained via Ridge Regression. The model architecture is the following:

$$\mathcal{N}(x) = f(Wx + b) \tag{2.20}$$

$$\hat{y}(x) = \theta_0 + \theta_1 \mathcal{N}(x) \tag{2.21}$$

W is a uniformly sampled matrix $\mathbb{R}^{k \otimes d}$, b is a normal sampled vector $\mathbb{R}^{k \otimes 1}$. The trainable parameters θ_0 and θ_1 are a matrix and vector of the shape $\mathbb{R}^{D \otimes k}$ and $\mathbb{R}^{D \otimes 1}$. This strategy is similar to a basis expansion, where the basis is now a combination of random transformations of the input data instead of a polynomial expansion.

To compare the performance of Neural Networks and Extreme Learning Machines four results are of importance, namely training error, test error, training time and prediction time. To make a fair comparison, a Neural Network of 2 layers consisting of 10 nodes each, and an Extreme Learning Machine consisting of 140 nodes were initialized such that they both have 141 trainable parameters. These two networks were trained to approximate a noisey Sinc-like function:

$$y(x) = \text{sinc}(0.5x^2 - 0.2x + 1) \tag{2.22}$$

$$y_{train}(x) = y(x) + \epsilon \tag{2.23}$$

Where the noise ϵ was normally distributed with $\mu = 0$ and $\sigma = 0.01$. The data was sampled such that $-2 < x < 2$, 1000 samples were used for training, and 100000 were used to measure the prediction time and the test error. The data was normalized before entering both the Neural Network and the Extreme Learning Machine, which is done by letting $x_{in} \to (x - \bar{\mu}(x))/\bar{\sigma}(x)$. The Neural Network was trained by the AdamW [25] optimizer for 20000 iterations with a learning rate of 0.001. In the ELM, W was uniformly sampled in $[-12, 12]$ and b was normally sampled with $\mu = 0$ and $\sigma = 12$.

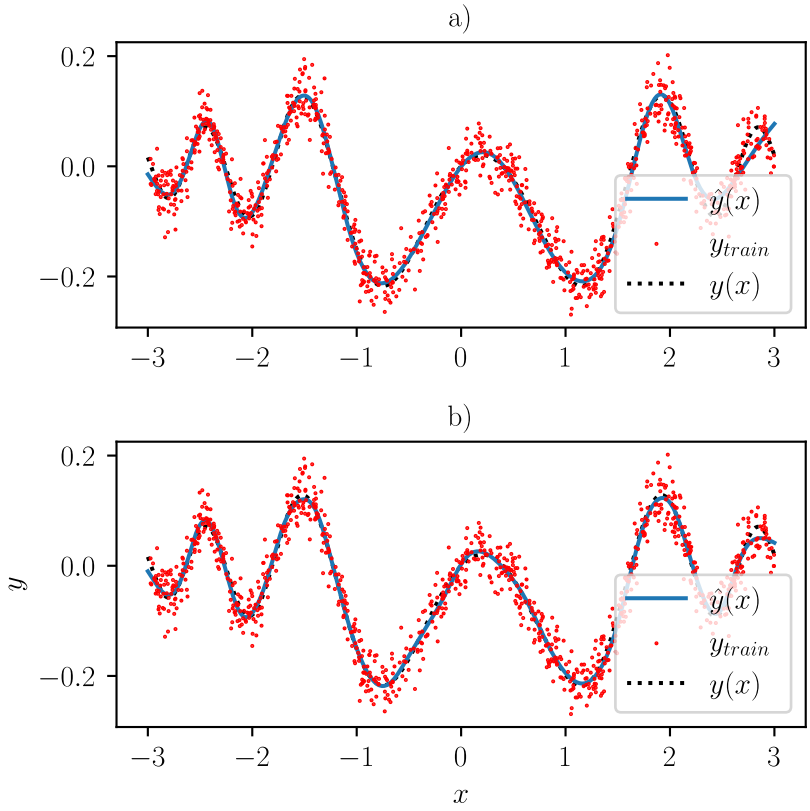


Figure 2.5: a) Neural Network prediction of the noisey Sinc-like function; $\mathcal{L}_{train} = 0.0951$, $\mathcal{L}_{test} = 0.000135$, $T_{train} = 50.2s$, $T_{predict} = 0.031s$. b) Extreme Learning Machine prediction of the noisey Sinc-like function; $\mathcal{L}_{train} = 0.0830$, $\mathcal{L}_{test} = 0.000045$, $T_{train} = 0.3s$, $T_{predict} = 0.047s$.

The ELM was able to outperform the NN slightly regarding the train and test error. The training time was $\sim 17000\%$ faster, while the prediction time was $\sim 30\%$ slower. The main drawback of Extreme Learning is that the performance is dependent on the magnitude of W and b, which are hyperparameters and therefore have to be fine-tuned outside of the training loop. Extreme Learning Machines can sometimes fail to match the performance of Neural Networks when the function is too complex since it can only be one layer deep, unlike Neural Networks which can be as deep as necessary. Therefore, ELMs should only be deployed when the function to be learned is reasonably learnable by a 1-layer NN.

Neural Operators

Operators are the most important mathematical object for Physics. Without linear operators such as the derivative, modern Physics would be limited to a pre-Newtonian era. An operator $G : X \to Y$ takes in a space X and produces a space Y. Usually these spaces are function spaces (or infinite sequences). Moving forward, an operator is a map defined on a compact set $V \subset C(K_1)$, where $C(K_1)$ are all continuous functions defined on some compact set of a Banach space $K_1 \subset X$, where X is the Banach space in question. The operator $G : V \to C(K_2)$ maps the compact space V into another space of all continuous functions $C(K_2)$ of a compact space $K_2 \subset X$.

In the above definition, G does not have to be linear, which is the case in much of Physics. Take for example the gradient $\nabla : V \to C(K_2)$, which is linear since $\nabla(\alpha f(\cdot) + \beta g(\cdot)) = \alpha\nabla(f(\cdot)) + \beta\nabla(g(\cdot))$. It is clear that linear operators are a very small set of all possible operators. For example, if $u(\cdot)$ is some non-linear function defined on V, then defining the operator $G : V \to u(\nabla(V))$ is clearly not linear because $G(\alpha f(\cdot) + \beta g(\cdot))) = u(\alpha\nabla(f(\cdot)) + \beta\nabla(g(\cdot))) \neq \alpha G(f(\cdot)) + \beta G(g(\cdot))$.

A more relevant example is finding the solution to some initial value problem of a differential equation $\frac{dy}{dx} = u(y(x), x)$ where $y(0) = 0$. If $u(y(x), x) = u(x)$, then we say that the operator $G(u(\cdot))(x) \to y(x) = \int_0^x u(x')dx'$. This does not have a simple integral solution if $u(y(x), x)$ is non-linear in $y(x)$.

The goal of a Neural Operator $\mathcal{G}_\theta$ in the context of solving an initial value problem is now to approximate $G(u(\cdot))(x) \to y(x)$, where $u(\cdot) \in V$, which can not be done with Neural Networks because they can only learn a single $u(\cdot)$ per network instance. Moving forward, parametrized systems refer to initial value problems where $\frac{dy}{dx} = u_\alpha(y(x), x)$, where $u_\alpha(\cdot)$ is some function $u(\cdot)$ dependant on some finite parameters α creating a compact set $V \subset C(K_1)$. For instance the polynomials: $u_\alpha(y) = K^k_\alpha[y] = \alpha_0 + \alpha_1 y + \alpha_2 y^2 + ... + \alpha_k y^k$ with finite y. For ease of notation, from here on out we denote sampling this function by $u(y(s), s') \equiv u(s)$.

3.1 Universal Approximation Theorem for Operators

The Universal Approximation Theorem for Operator (UATO) [8] shows that for compact sets $K_1 \in X$, $K_2 \in \mathbb{R}^d$, $V \in C(K_1)$, a non-linear operator $G : V \to C(K_2)$ between function spaces can be approximated arbitrarily close by some Neural Network. For any $\epsilon > 0$, there are some features $s \in K_1$, $\theta^{(R)}_W$, $\theta^{(B)}_W$, $\theta^{(B)}_b$, $\theta^{(T)}_W$, and $\theta^{(T)}_b$, where R, T, B is a naming convention, such

that.

$$\left|G(u(\cdot)(x) - \sum_{i=1}^{p}\sum_{j=1}^{n}\theta_{j(W)}^{i(R)} f_B\big(\sum_{l=1}^{r}\theta_{jl(W)}^{i(B)}u(s_l) + \theta_{j(b)}^{i(B)}\big) f_T\big(\theta_{i(W)}^{(T)}x + \theta_{i(b)}^{(T)}\big)\right| < \epsilon \tag{3.1}$$

The features s will in the future be referred to as sensor locations, and the values $u(s) \in V$ as sensors. Sensors are measurements of the function $u(\cdot)$ that the operator G takes as input. The shapes are now $\theta_W^R \in \mathbb{R}^{p\otimes n}$, $\theta_W^T \in \mathbb{R}^p$, $\theta_W^B \in \mathbb{R}^{p\otimes n\otimes r}$, $\theta_b^T \in \mathbb{R}^p$, and $\theta_b^B \in \mathbb{R}^{p\otimes n}$. Note that some trainable matrices are no longer rank-2 tensors but rank-3 because there is a new dimension p, which represents a functional basis expansion. This basis expansion:

$$G(u(\cdot))(x) \approx \sum_{i=1}^{p}\psi_i(u(\cdot))\phi_i(x) = \sum_{i=1}^{p}(\psi(u(\cdot)) \odot \phi(x))_i \tag{3.2}$$

is similar to a functional basis expansion. In this case the basis is a set of functions instead of vectors.

3.2 DeepONet

The first Neural Operator, developed by G.E. Karniadakis and L.L. Pengzhan Jin in 2020, is called the *Deep Operator Network* (DeepONet) [9]. The architecture is designed to fulfil the requirements set by theorem 3.1.

$$\mathcal{G}_\theta(u(s),(x)) = \sum_{i=1}^{p}\mathcal{B}_\theta^i(u(s))\mathcal{T}_\theta^i(x) + b_\theta^{(R)} \tag{3.3}$$

The operator network consists of two fully connected Neural Networks which output $k \otimes p$-sized matrices[1], one for the sensors $u(s)$ which is called the *branch* network $\mathcal{B}_\theta(u(s))$, and one for the input x which is called the *trunk* network $\mathcal{T}_\theta(x)$. The above configuration is called an 'unstacked' DeepONet, there is also a 'stacked' version where every index in p is now a different Neural Network and produces a rank-2 tensor, instead of a network that produces a rank-3 tensor.

[1] This can be achieved be producing a kp-size output and then rolling into $k \otimes p$.

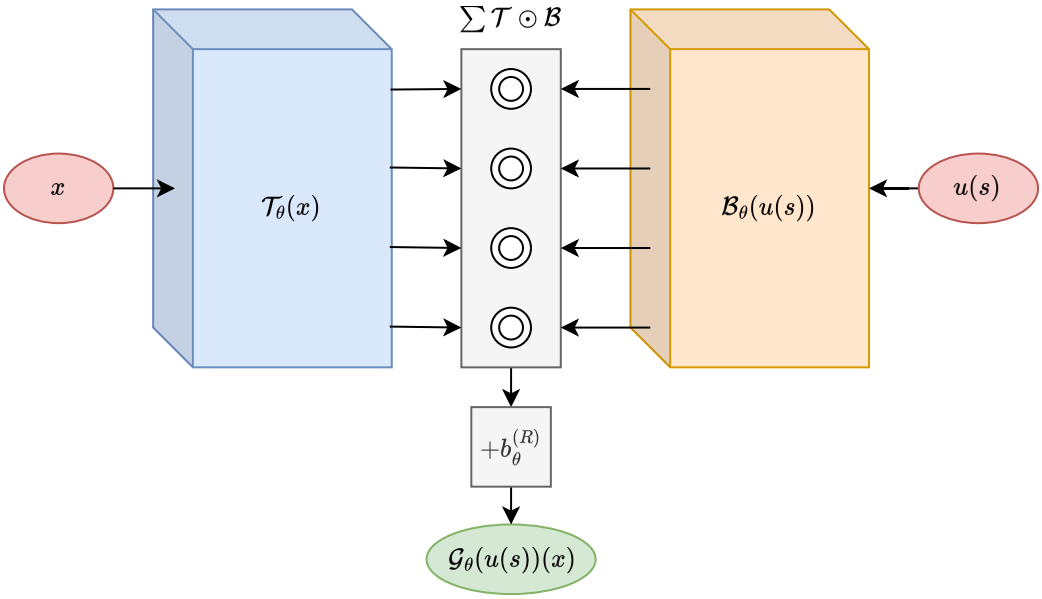


Figure 3.1: Unstacked DeepONet diagram with $p = 4$.

The problem persisting in Neural Network optimization speed is now more problematic since there are two larger networks. Therefore, it might be interesting to design a novel Neural Operator that uses the concepts of Extreme Learning to speed up training, with the trade-off that prediction times are slower.

3.3 Extreme Operator Network (ExtremONet)

Developing new Neural Network architectures is as much theory-driven as it is intuition-based. In a recent presentation given by Dr. Bahri regarding her work in a first-principles approach to understanding Deep Learning by relying on Statistical Physics [26], she echoed this sentiment when she said:

> *"A few years ago, at one of the major Machine Learning conferences, one of the invited speakers very famously made the allusion that Deep Learning is not really a science or engineering; it's alchemy. It is mixing things, designing models in a somewhat unprincipled way without knowing what will and won't work, and having some basic principles"*—Bahri, Yasaman

The DeepONet, as the quote predicted, does not completely follow the UATO structure (there is no θ_W^R present), yet it is still able to approximate. The DeepONet form unfortunately does not permit Extreme Learning. The reason this form does not work for ELM's is because one cannot apply Linear Regression to $\mathcal{B}_\theta$ and $\mathcal{T}_\theta$ separately. Therefore, the author suggests a new Neural Operator that does allow Extreme Learning Machines to be used, and resembles the UATO in a similar but different way, and is equivalent to eq. 3.2. The **novel** *Extreme Operator Network*

(ExtremOnet) architecture is the following:

$$\mathcal{B}(u(s)) = f_B(W^B u(s) + b^B) \tag{3.4}$$
$$\mathcal{T}(x) = f_T(W^T x + b^T) \tag{3.5}$$
$$\hat{y}(u(s))(x) = \theta_0 + \theta_1(\mathcal{B}(u(s)) \odot \mathcal{T}(x)) \tag{3.6}$$

where $\odot$ is the Hadamard product of matrices, W^B is a uniformly sampled matrix of the shape $k \otimes r$, W^T is a uniformly sampled matrix of the shape $k \otimes d$, the bs are normally sampled vectors of the shape $k \otimes 1$. The trainable parameters θ_0 and θ_1 are a matrix and vector of the shape $D \otimes k$ and $D \otimes 1$. The Ws are now rank-2 tensors, created by unrolling the $\sum_i \sum_j$ in the UATO, implying that $pk \to k'$.

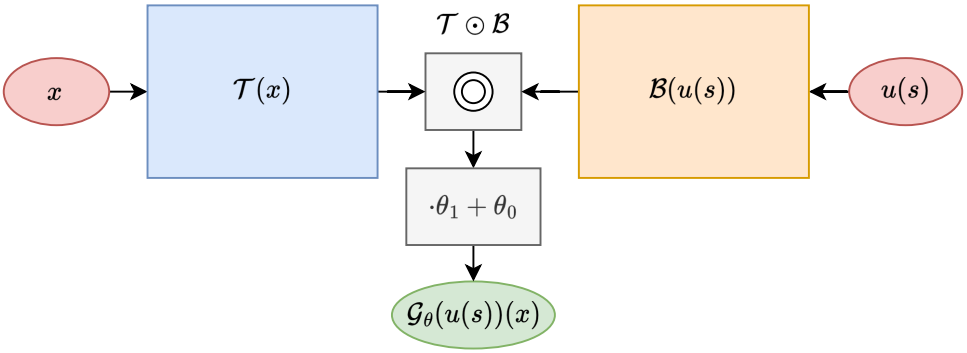


Figure 3.2: ExtremONet diagram.

Currently, there is no approximation theorem for random single layer Neural Operators, from here on named Approximation Theorem for Extreme Learning Neural Operators (ATELNO). From Extreme Learning theory, one can reasonably assume that it extends to operators as well.

To compare the performance of the DeepONet and the ExtremOnet, a parametrized initial value problem

$$\dot{y} = \sin(\alpha y + t) \tag{3.7}$$

was solved for $y(0) = 0$. The parameters were uniformly sampled in $[0, 3]$, t was uniformly sampled in $[0, 10]$, with $r = 100$ sensor locations s were uniformly sampled from $[0, 1]$, and $s' = 0$ in all instances. 5000 couples of $(t, y(t), u_\alpha(s))$ were used for training, where every instance of the couple $y(t)$ and $u_\alpha(s)$ was generated by a unique α. The test set consisted of 1000 points in the same intervals as the training set. The DeepONet consisted of two 2-layer 10 wide Neural Networks with a p-size of 10 resulting in 1481 trainable parameters, which were optimized for 40 000 iterations by the AdamW optimizer with a learning rate of 0.0001. The ExtremONet consisted of two 1000-wide Extreme Learning Machines with $\sigma_{trunk} = 1$ and $\sigma_{branch} = 3$, which consisted of 1001 total trainable parameters. W^B was randomly initialized such that it had a sparsity of $c_B = 0.1$. Both models will use the hyperbolic tangent as a non-linear activation function.

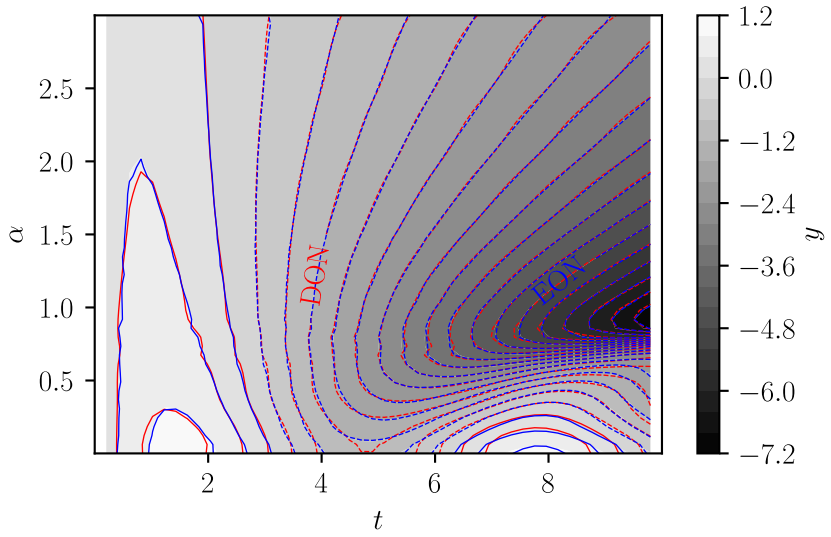


Figure 3.3: Approximation of $y(x)$ by DeepONet and ExtremOnet.

The ExtremONet was able to outperform the DeepONet whilst having fewer trainable parameters and training 2 orders of magnitude faster. A clear issue arises in the prediction time. Because the ELMs are 500 nodes wide, the prediction time is significantly slower. To get a statistically relevant picture, the above test was ran 30 times:

model	$\mathcal{L}_{train}$	$\mathcal{L}_{test}$	t_{train}	t_{pred}
DeepONet	$1.0 \cdot 10^{-3}$ $\pm 5.6 \cdot 10^{-3}$	$1.1 \cdot 10^{-2}$ $\pm 5.7 \cdot 10^{-3}$	475.21 $\pm 9.79s$	0.0062 $\pm 0.0076s$
ExtremONet	$2.6 \cdot 10^{-4}$ $\pm 3.0 \cdot 10^{-4}$	$3.3 \cdot 10^{-4}$ $\pm 6.1 \cdot 10^{-5}$	0.94 $\pm 0.06s$	0.0874 $\pm 0.0122s$

Table 3.1: DeepONet and ExtremONet comparison

To reduce this issue, a special operation has to be defined. One first has to observe that using only one Trunk network index per Branch Network index is inefficient. If an index in the Branch network that carries information about the functional basis is paired with a Trunk network index that carries no information if it is paired with this Branch network index, then the model loses potential performance, and therefore is inefficient. There are k^2 possible combinations that can be achieved by taking all possible Branch and Trunk index combinations, which is too large to realistically handle. To solve this, Branch network indices should be paired with multiple, but not all, Trunk network indices, and vice versa. Random Selection $Rs_q : A \to A'$ projects a rank-2 tensor of size $k \otimes D$ to a rank-2 tensor of size $(q + k) \otimes D$ by randomly sampling $k + q$ unique pairs from the respective k rows. This is equivalent to two sparse matrix multiplications which are $(q + k) \otimes k$ matrices Rs^{qB} and Rs^{qT}.

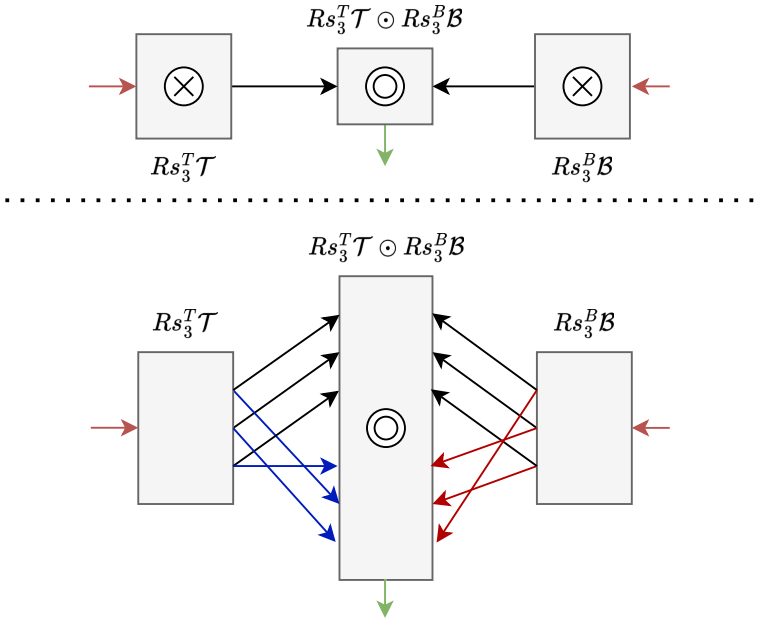


Figure 3.4: Random Selection process diagram where $k = 3$ and $q = 3$.

These matrices can be created such that every pair is unique via the following algorithm:

Algorithm 4: Random selection matrices

Data: K, q
Result: Rs^{qT}, Rs^{qB}
$Rs^T \leftarrow 0^{(q+k)\otimes k}$;
$Rs^B \leftarrow 0^{(q+k)\otimes k}$;
$i \leftarrow 0$;
for k **do**
  $Rs^T_{ii} \leftarrow 1$;
  $Rs^B_{ii} \leftarrow 1$;
  $i \leftarrow i + 1$;
end
$a \leftarrow \text{list}(k, .., q + k)$;
$i \leftarrow 0$;
for q **do**
  $a_i \leftarrow a[i]$;
  $b_i \leftarrow \text{rand}(\text{list}(i, .., k))$;
  $Rs^T_{ia_i} \leftarrow 1$;
  $Rs^B_{ib_i} \leftarrow 1$;
  $i \leftarrow i + 1$;
end

Sparse matrix multiplication can be faster when the matrices in question are sparse enough (qualitatively $\sim$ 1% sparse). This operation in some sense mimics the summation over p in UATFO, reducing the size of the matrix multiplication in the ExtremONet. The randomly

selected ExtremONet is now:

$$\mathcal{B}(u(s)) = Rs_q^B f_B(W^B u(s) + b^B) \tag{3.8}$$
$$\mathcal{T}(x) = Rs_q^T f_T(W^T x + b^T) \tag{3.9}$$
$$\hat{y}(u(s))(x) = \theta_0 + \theta_1(\mathcal{B}(u(s)) \odot \mathcal{T}(x)) \tag{3.10}$$

W^B is a uniformly sampled matrix of the shape $k \otimes r$, W^T is a uniformly sampled matrix of the shape $k \otimes d$, the bs are normally sampled vectors of the shape $k \otimes 1$. The trainable parameters θ_0 and θ_1 are a matrix and vector of the shape $D \otimes (q + k)$ and $D \otimes 1$.

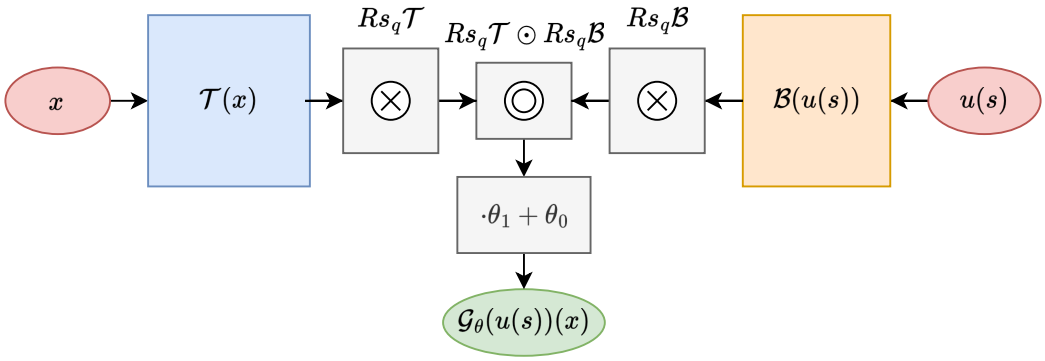


Figure 3.5: Random Selection ExtremONet diagram.

This model was tested against the ExtremONet with the ELMs having 100 nodes, and Rs_{400} and the rest equivalent.

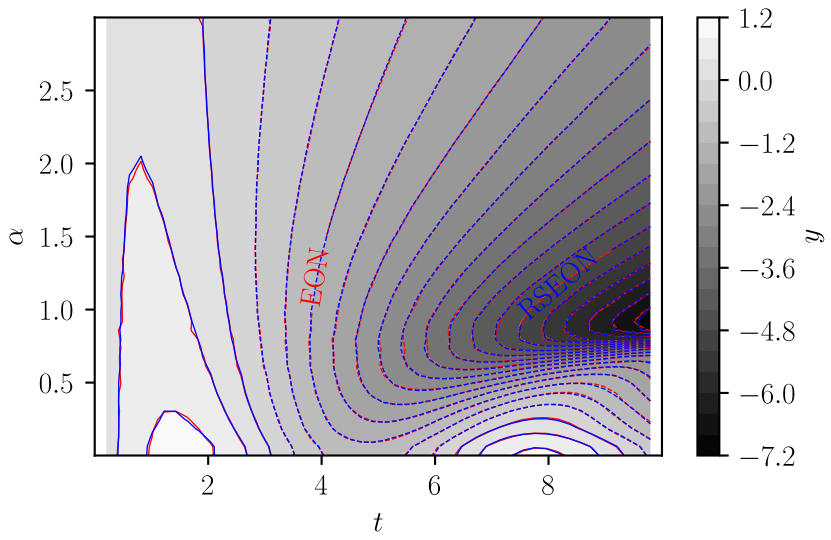


Figure 3.6: Approximation of $y(x)$ by ExtremONet and randomly sampled ExtremeONet.

The RSExtremONet optimized with a final error $\mathcal{L}_{train} = 0.00025$ and $\mathcal{L}_{test} = 0.00044$ after training for $0.81s$, with a prediction time of $0.0322s$. The performance is equivalent to the ExtremONet, whilst the prediction time was vastly reduced and now 3 orders of magnitude faster at training. To get a clearer picture, the above test was run 30 times:

model	$\mathcal{L}_{train}$	$\mathcal{L}_{test}$	t_{train}	t_{pred}
ExtremONet	$2.6 \cdot 10^{-4}$ $\pm 3.0 \cdot 10^{-4}$	$3.3 \cdot 10^{-4}$ $\pm 6.1 \cdot 10^{-5}$	0.94 $\pm 0.06s$	0.0874 $\pm 0.0122s$
RS ExtremONet	$2.9 \cdot 10^{-4}$ $\pm 9.3 \cdot 10^{-5}$	$3.5 \cdot 10^{-4}$ $\pm 1.1 \cdot 10^{-4}$	0.83 ± 0.07	0.0348 $\pm 0.0061s$

Table 3.2: Effect of Random Selection on ExtremeONet

Since training takes much more time in general, one can conclude that the ExtremONet with Random Selection is preferable in many situations. Another non-negligible benefit is the simplicity of using Ridge Regression, because it is easy to interpret and diagnose.

3.3.1 Hyperparameter analysis

The ExtremeONet is dependent on multiple hyperparameters; sparsity of the branch matrix, norm of the matrices, network width, and in the case of Random Selection, q. A range of tests were executed for 30 iterations; the results of which can be found in appendix A.1. Lower sparsities appear to produce the best results, as seen in table 1, 2 with respect to the mean and standard deviation of the train and test loss. In table 3, and 4 it is apparent that higher Branch norms perform better up until $\sigma_B = 5$, which indicates that the operators perform best when the transformation is more non-linear. This is because $\tanh(x) \approx x$ for small x, therefore the Branch network is linear for small norms. The opposite is true for Trunk networks, a network norm $\sigma_T = 1$ is the most performant, as seen in table 5, 6. Finally, a parameter sweep was done over $0.05 < c_B < 0.5$,$0.5 < \sigma_T < 3$, and $1 < c_B < 10$ to find the model with the lowest test error. This sweep was done by brute-force random sampling in the hyperparameter space for 1000 iterations.

	c_B	σ_T	σ_B	$\mathcal{L}_{train}$	$\mathcal{L}_{test}$
argmin $\mathcal{L}_{train}$	0.44	2.48	3.14	$2.0{\cdot}10^{-4}$	$2.5{\cdot}10^{-4}$
argmin $\mathcal{L}_{test}$	0.27	1.01	7.53	$2.0{\cdot}10^{-4}$	$2.4{\cdot}10^{-4}$

Table 3.3: Brute-force hyperparameter sweep of ExtremONet to minimize the train and test loss respectively.

	c_B	σ_T	σ_B	$\mathcal{L}_{train}$	$\mathcal{L}_{test}$
argmin $\mathcal{L}_{train}$	0.15	0.61	3.95	$2.3{\cdot}10^{-4}$	$2.6{\cdot}10^{-4}$
argmin $\mathcal{L}_{test}$	0.15	0.61	3.95	$2.3{\cdot}10^{-4}$	$2.6{\cdot}10^{-4}$

Table 3.4: Brute-force hyperparameter sweep of Random Selection ExtremONet to minimize the train and test loss respectively.

The ExtremONet performs slightly better without Random Selection regarding training error. The test errors are within standard deviation as seen in the tables in the appendix, therefore this performance difference is statistically irrelevant. This indicates that Random Selection ExtremONet is preferable due to the shorter training times.

3.3.2 Data dependency, network dimensionality

There are some important questions that need answering left; how does the ExtremONet performance depend on the amount of training data, how many sensors are necessary to function as an operator, how does the network width impact the performance and how large should/can q be when Random Selection is used.

To answer these questions, a new dataset was generated. Similar to the previous section, eq. 3.7 was solved for $y(0) = 0$. The parameters were uniformly sampled in $[0, 3]$, t was uniformly sampled in $[0, 10]$, with $r = 200$ sensor locations, which s were uniformly sampled from $[0, 1]$. 10000 Couples of $(t, y(t), u_\alpha(s))$ were used for training, where every instance of the couple $y(t)$ and $u_\alpha(s)$ was generated by a unique α. The test set consisted of 1000 points in the same intervals as the training set. The networks used the hyperparameteres found in table 3.3 and 3.4.

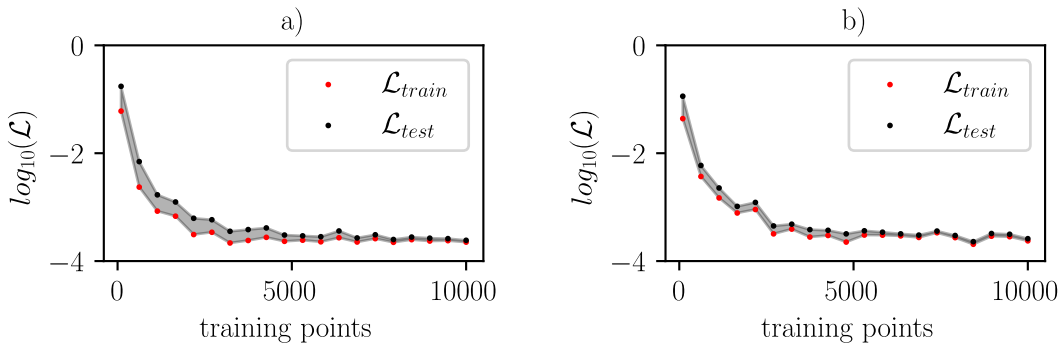


Figure 3.7: a) Data dependency of the ExtremONet. Data dependency of the ExtremONet with Random Selection.

The two models do not overfit because of the Ridge Regression procedure, where internally a validation set is created to check whether the model is overfitting or not. As a result, both models suffer from minimal overfitting. Random Selection performs slightly better, which is most likely because its branch and trunk network have fewer internal parameters, shrinking the hypothesis space. Both models converge to a consistent performance after 5000 training points.

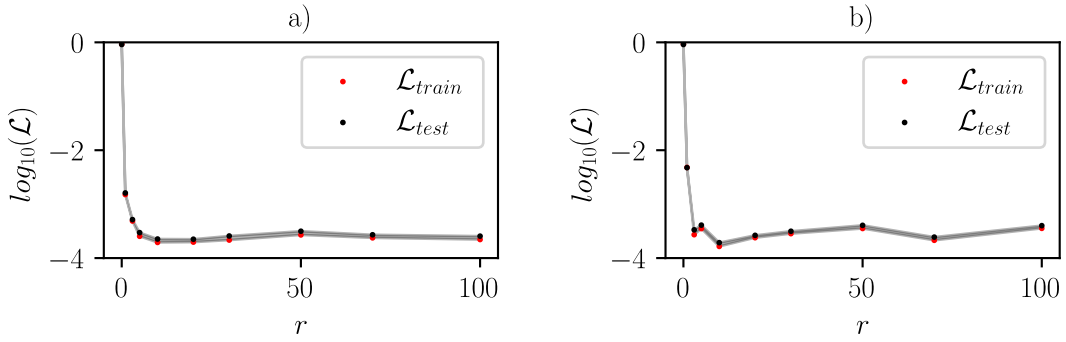


Figure 3.8: a) Sensor dependency of the ExtremONet. Sensor dependency of the ExtremONet with Random Selection.

At 0 sensors, both models reduce to normal Extreme Learning Machines, and cannot learn the problems as expected. Both models need almost no sensors, and 5 seem to be enough to converge. This minimal dependence is atypical since in the original DeepONet paper [9] it was shown that for a fairly similar problem at least 100 sensors were necessary to converge. This indicates that the ExtremONet might be more data-efficient due to the large network width, or makes use of some other unknown effect. The Random Selection ExtremONet lost performance when r was of the same magnitude or greater than the branch width. This is a quite typical example of the curse of dimensionality[2].

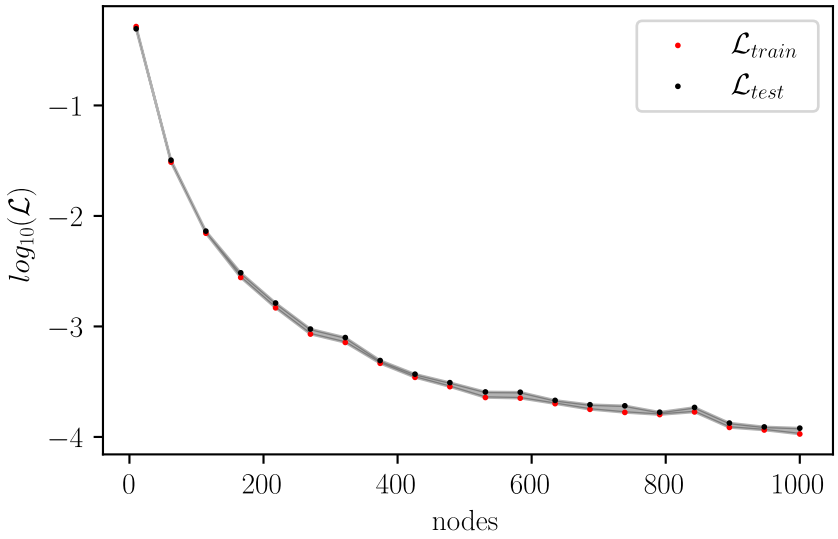


Figure 3.9: Network width k dependency of the ExtremONet.

Extreme Learning Machines suffer from diminishing returns, unlike standard Neural Networks which can have multiple hidden layers. ELMs are restricted to one. Single layer networks do not

[2]The amount of data necessary to generalize grows exponentially with the feature space size.

scale linearly in performance with respect to their network width, the performance increase drops exponentially, which causes an issue for complex problems. Another issue is memory complexity, if a problem requires a very wide Neural Network to perform well, then it is most likely that an ELM needs to be orders of magnitude wider to be as performant, having a large impact on the amount of memory necessary to train and predict. Every system should be studied carefully before deciding whether to use DeepONets and ExtremONets.

To explore the behaviour of the Random Selection ExtremeONet, both q and the number of nodes need to be varied.

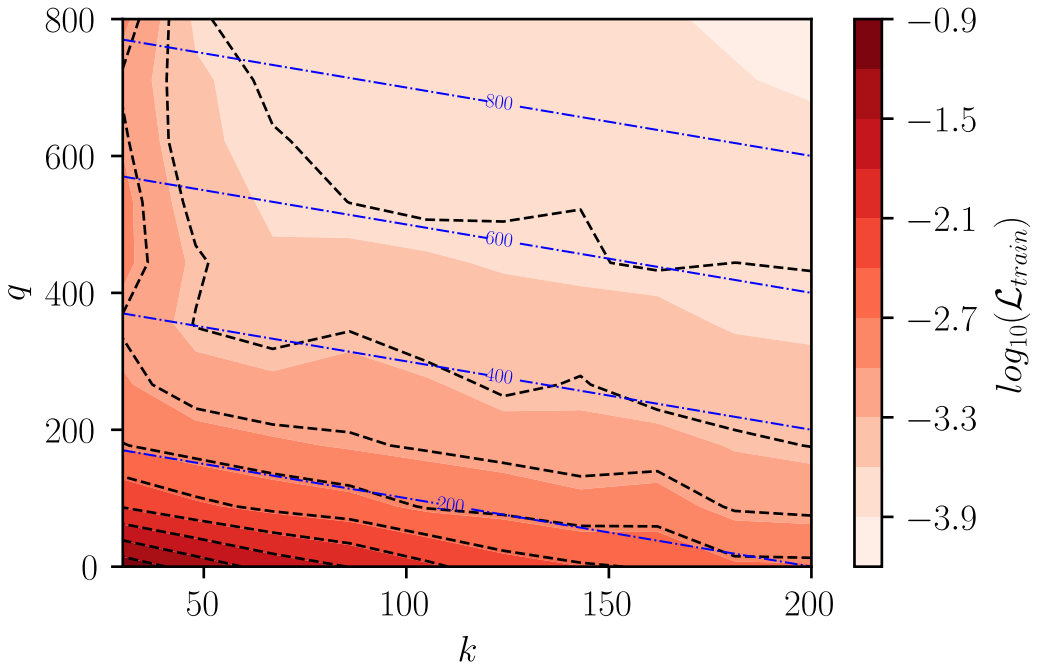


Figure 3.10: Network width $k + q$ dependency of the ExtremONet with Random Selection.

The blue iso-lines run parallel to the iso-error lines, indicating that for low q, the Random Selection version is identical. For larger q and less nodes this behaviour does not appear. This is because the individual ELMs do not have enough nodes to express a basis, and therefore cannot perform well. The lower density in the upper right quadrant demonstrates the diminishing returns from increasing the total model width. From the above plot one can conclude that Random Selection for low-ish q is preferable to the vanilla version. This concludes the investigation of the ExtremONet.

3.4 Discussion

In this chapter, a novel Neural Operator was proposed based on Extreme Learning Machines and eq. 3.2. It was shown that this model is more performant than a small DeepONet with an equal

amount of trainable parameters when approximating the differential equation $\dot{y} = \sin(\alpha y + t)$. Furthermore and adjustment to the final architecture was made (Random Selection) which drastically reduced prediction time. It was also shown that these models do not need many sensors to accurately model an operator.

A reason for this performance increase might be because of the unrolling of rank-3 tensors into rank-2 with higher dimensions. This impacts the sum over p in eq. 3.2. In the DeepONet architecture, according to the [9], p is usually set above 10 for good performance. Increasing this number implies that the parameters of the Branch and Trunk net increase drastically. The ExtremONet combines the output dimensions of the Trunk and Branch net with this variable p creating a sum over pk instead. Therefore the actual basis size is increased by 2 orders of magnitude in the ExtremONets used, increasing the probability of being able to construct a performant basis without increasing the parameter count of the network.

ExtremONet suffers from various issues. There are many hyperparameters which need to be tuned outside of the training loop. This is not a big issue since the training time is short, and these do not impact the performance much as long as they are within a reasonable range. A larger issue is the lack[3] of network depth. Extreme Learning Machines usually are not deeper than one layer, which is not preferable when dealing with complexer problems. the ExtremONet might not be suitable for every situation where the DeepONet is.

In the next two sections, a solution for dynamical systems will be provided in the form of Reservoir Computing. By moving to phase (configuration) space, it is possible to reduce the complexity of any problem by adjusting the timestep size which makes the ExtremONet as performant as Reservoir Computing on single systems. Reservoir Computing comes with its own flaws, but we shall see that many of these will automatically be resolved when combined with Neural Operators.

[3]There are 'deep' Extreme Learning Machines in the literature, but they do not benefit like normal Neural Networks do from this depth increase

Reservoir and Operators

Neural Networks and Neural Operators learn a map between compact spaces. This implies that for dynamical systems, if t is in the input space for a Neural Network, then it can not generalize beyond the subspace where t is sampled from. For example a Neural Network that learned a dynamical system with $t \in [0, 10]$ cannot accurately predict what happens at $t = 100$. To predict longer timeseries, one has to provide training data from those timeseries. This also implies that the complexity of the network will be at least larger than before since the compact input space is now larger. Determining whether a system is chaotic, as mentioned in chapter 1, requires a substantial amount of data over a large period of time. This fact, combined with the large network requirements of models like the DeepONet, hinders most hardware to train on such systems in a timely manner.

Learning long timeseries without enlarging the compact space can be done with recurrent models, as long as the dynamical systems contains an attractor (making it compact in space), which moves the problem to phase space. In this chapter, recurrent approaches to Neural Networks will be discussed. A fundamental issue regarding optimization will be explored and resolved by introducing Reservoir Computing. RC will then be combined with the previously explored ExtremONet with Random Selection to create a Recurrent Neural Operator that soley relies on Ridge Regression. Various advantages of this architecture over standard RC will be explored, together with new zero-shot capabilities. These zero-shot capabilities allow the EchONet, which combines the ExtremONet with Reservoir Computing, to function like a standard numerical integrator without knowledge of the system.

4.1 Recurrent methods

Recurrent Neural Networks (RNN) are a popular Neural Network architecture when dealing with temporally causal data. This can be in the form of mappings, differential equations, or even speech recognition and machine translation. The origin can be traced back to J. Hopfield's work in the early 1980's, namely *Hopfield Networks* [27], better known as the Ising Model. These Hopfield Networks are able to store a (temporary) representation of data through time, also known as memory. The first instance of a general RNN was introduced by J.L. Elman [28] in 1990 in the context of natural language processing. Many applications that previously relied on RNNs have recently been replaced by attention based models such as the Transformer. For our purpose, recurrent methods are more than adequate.

4.1.1 Exploding & Vanishing Gradient problem

A common difficulty that most Recurrent Neural Network architectures suffer from when incautious are the *Exploding Gradient* and *Vanishing Gradient* phenomena [29]. Since RNN's are explicitly iterative in nature, the jacobian linking different iterations tend towards extrema. This behaviour can be demonstrated with the following derivation. For universality's sake, let us study the hidden state of a general RNN:

$$\begin{aligned} h_{i+1} &= G_\theta(h_i, y_i) \\ \hat{y}_{i+1} &= F_\phi(h_i) \end{aligned} \tag{4.1}$$

The gradient of the hidden state with respect to the hidden state weights can be derived, giving the following expression:

$$\frac{\partial h_i}{\partial \theta} = J = \Pi_{j=0}^{k} J_j \tag{4.2}$$

where $J_j \equiv \frac{\partial G}{\partial \theta}\big|_{h_{i-j}}$. Imagine that 'roughly speaking' for every jacobian $\|J_j\| > 1$, then we get that for

$$i \to \infty \implies \|J\| \to \infty$$

on the other hand when $\|J_j\| < 1$ we see that

$$i \to \infty \implies \|J\| \to 0$$

It is clear that he previously mentioned phenomena are two sides of the same coin. This has the effect that the model will not properly train on values for large k since most optimizers (excluding meta-heuristic optimizers) require the jacobian's elements in their update rule. This poses a serious hazard, for example in the widespread *Stochastic Gradient Descent* (SGD):

$$\begin{aligned} \Delta\phi &= -\eta \frac{\partial \mathcal{L}}{\partial \phi} \\ \phi_{i+1} &= \phi_i + \Delta\phi \end{aligned} \tag{4.3}$$

Where the loss $\mathcal{L}$ is an explicit function of y and $\hat{y}$ (usually $\frac{1}{k}\|y - \hat{y}\|^2$) and therefore of J since $\hat{y}$ is connected to h via $F(\cdot)$. There are ways around this issue, for example one can use *Gradient Clipping* where one artificially limits the maximum of the gradient in the update rule. More preferably one would create a model which is not infinitely causal in the gradients. In recent years, deep learning architectures like the *Long-Short-Term-Memory* (LSTM) and *Gated Recurrent Unit* (GRU) have dealt with both phenomena by creating internal hidden states which control the dependency on the previous iterations. Another model type that does not suffer from exploding and vanishing gradients is called *Reservoir Computing* (RC), these models rely on a similar strategy to *Extreme Learning* where the input sequence gets lifted to a random non-linear high-dimensional space (embedding) from which the model tries to learn.

4.1.2 Training time

Recurrent Neural Networks, just like standard feedforward Neural Networks, require a substantial amount of iterations to train. RNNs suffer from an additional computational tax due to their iterative nature. Every forward prediction of n training points require n network forwardings, heavily increasing the amount of computation necessary, unlike feedforward NNs which forward data in parallel due to matrix multiplication. If one were to use a recurrent network as a trunk

network in the previously mentioned DeepONet, then the amount of data that needs forwarding increased further, making training time unnecessary long.

There are alternatives which circumvent this problem, like Transformers which use an attention mechanism that is not iterative, or linear RNNs which can parallelize due to some clever mathematical tricks. A different solution is to decouple the forwarding of the hidden state and the training phase on the hidden state, which is what Reservoir Computing is able to do.

4.2 Echo-State property

The foundation of all recurrent models is built on the idea that models equipped with some form of temporally fading, short-term memory can approximate any timeserie. RC models exploit something called the *Echo-State Propery* (ESP), which can be guaranteed when the architecture in question obeys the following (simplified[1]) rules [30]:

> *The network is uniformly state contracting*, implying that for a hidden-state update rule T there exists a sequence δ_k such that for all hidden state states $h, h^{'}$ and input sequences y_t from some index n the L_2-norm $d(T(h, y_{t'}), T(h^{'}, y_{t'})) < \delta_k$ for all k , where $t^{'} \in \{n, n+k\}$.
>
> *The network is state forgetting* if for all input sequences y_t and a hidden-state update rule T there exists a sequence δ_k such that for all hidden state states $h, h^{'}$ from some index n the L_2-norm $d(T(h, y_{t'}), T(h^{'}, y_{t'})) < \delta_k$ for all k, where $t^{'} \in \{n-k, n\}$.
>
> *The network is input forgetting* if for all input sequences y_t and a hidden-state update rule T there exists a sequence δ_k such that for all sequences of the form $Wy_{t'}$,$Vy_{t'}$, all states $h, h^{'}$ which are end-compatible with $Wy_{t'}$,$Vy_{t'}$ (implying that $T(h(i), \cdot) = h(i+1)$), it holds that $d(h, h^{'}) < \delta_k$ for all k, where $t^{'} \in \{n-k, n\}$

These statements are guaranteed when the hidden-state update rule takes the following form:

$$T(h(i), y(i)) = f(Rh(i) + Wy(i)) \tag{4.4}$$

where the spectral radius of R is smaller or equal to 1, and the activation function $f(\cdot)$ is bounded.

4.3 Reservoir Computing and memory for dynamical systems

The mechanism that allows RC to approximate chaos is called synchronization [31]. Synchronization between systems implies that their dynamics synchronize, for example, the phases of two pendulums attached to the same beam will become equal after some time. For chaotic systems, this cannot fully occur because they are sensitive to initial conditions. However, it can be shown that some of their dynamical properties can still synchronize. Reservoir Computing can be viewed as a random, high-dimensional system driven by a system of interest. As a result, the dynamics of the reservoir can synchronize to the driving system, making it a valuable asset in understanding the properties of a system where only some time-series data is available. The prototypical Reservoir Computing model can be described as an Extreme Learning Machine in

[1] The rules are simplified, the exact definitions can be found in the cited paper

phase space equipped with the Echo State Property. There are many variants of RCs; they can have linear pass-through and can be modeled by complex physical systems such as photonic computing or quantum systems, or Next Generation Reservoir Computing which relies on a random non-linear autoregressor. The standard approach is the following:

$$h_{i+1}(y_i) = f(Rh_i + W_y y_i + b_h) \tag{4.5}$$
$$\hat{y}_{i+1} = \theta_0 + \theta_1 h_{i+1} \tag{4.6}$$

where h is a high dimensional hidden state, $f(\cdot)$ is a bounded non-linear function (to ensure the hidden dynamics are compact), θ are trainable parameters, R is a random matrix named the reservoir matrix, W_y is a random matrix named the input matrix, and b_h is a random vector named the bias term. The hidden-state forwarding phase consists of iteratively executing eq. 4.5 until the end of the training data is reached. Training is done by using Ridge Regression to find the optimal θ in eq. 4.6. Predicting data with an RNN-like structure is different from feedforward models. The result is not grid-invariant, meaning that data is equidistantly spaced. Predictions can only be made on points after the training phase, and has to be done iteratively. Predictions are done by letting $y_i \to \hat{y}_i$ in eq. 4.5.

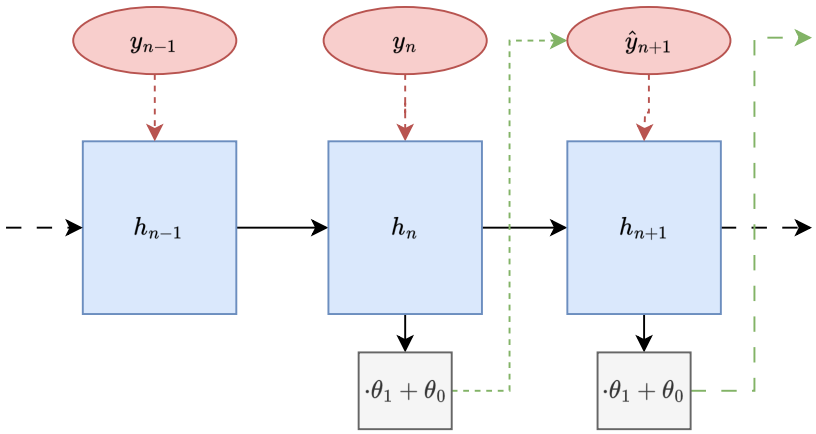


Figure 4.1: Reservoir Computing diagram.

The link between RNNs and dynamical systems is quite evident. Both are causal equations; previous iterations impact the current value and not the other way around. If one merges eq. 4.5 and 4.6, the link to dynamics becomes clear:

$$\begin{aligned} \hat{y}_{i+1} &= \theta_1 + \theta_1 h_{i+1} \sigma(Rh_i + W_y y_i + b_h) \\ &= f(y_i, h_i), \end{aligned} \tag{4.7}$$

when one extends this reasoning to h_i:

$$\hat{y}_{i+1} = f(y_i, y_{i-1}, ..., y_0), \tag{4.8}$$

which is the exact form a general map in eq. 1.2. Recurrent Neural Networks and therefore Reservoir Computing is a discrete dynamical system described by a general map. To be more precise, Reservoir Computing is input and state forgetting, implying that

$$\hat{y}_{i+1} = f(y_i, y_{i-1}, ..., y_{i-l}) \tag{4.9}$$

where l is some soft threshold indicating the amount of iterations information retains decipherable in the model (memory). It is quite evident that RC can also learn continuous systems just by stating its relation to the Euler numerical integration method. If the model does not have an Echo State Property, then eq. 4.9 reduces to:

$$\hat{y}_{i+1} = f(y_i) \tag{4.10}$$

The intuition here is that the above form receives less information about a system, therefore one can hypothesize that the ESP allows more insight into the underlying dynamics of a system, and can outperform a model without memory. Reservoir Computing has been very successful at learning chaotic behaviour [32]. To demonstrate why memory can be effective from a different perspective not shown in the literature, let us assume the model without memory $f(y_i) = \theta_0 \tanh(Ay_i + b)$ makes a small δ_m, (where $\|A\|$, and $\|b\|$ are small) at every iteration after the training data of length n, then the error propagated according to:

$$\begin{aligned} \mathbb{E}\{\epsilon_{n+2}\} &= \mathbb{E}\{\|\theta_0 \tanh(A_0(y_{n+1} + \delta_m) + b_0) - y_{n+2}\|\} \\ &\leq \mathbb{E}\{\|\theta_0 \tanh(A_0 y_{n+1} + b_0) - y_{n+2}\| + \theta_0 A_0 \delta_m\} \\ &= (1 + \theta_0 A_0)\delta_m \end{aligned} \tag{4.11}$$

If Λ in the system is positive and the model learned the system ($\hat{\Lambda} \approx \Lambda$), then $\|\theta\| \geq 1$. It is clear that an error made will propagate and grow. Let us study an autoregressive model that has a memory period of 2. Let us define this model as $g(y_i, y_i{-}1) = \theta_1 \tanh(A_0 y_i + b_0) + \theta_2 \tanh(A_0 y_{i-1} + b_0)$. Because $f(\cdot)$ and $g(\cdot)$ are trained to learn the same system via Ridge Regression, one can assume that if the system is smooth, and the sampling timestep is small that $\|\theta_1\| \approx \|\theta_2\| \leq \|\theta_0\|$

$$\begin{aligned} \mathbb{E}\{\epsilon_{n+1}\} &= \mathbb{E}\{\|\theta_1 \tanh(A_1(y_n + \delta_m) + b_1) + \theta_2 \tanh(A_2 y_{n-1} + b_2) - y_{n+1}\|\} \\ &\leq (1 + \theta_1 A_1)\delta_m \\ &\leq (1 + \theta_0 A_0)\delta_m \end{aligned} \tag{4.12}$$

This example shows qualitatively that error propagation in a model with memory is less pronounced than in models without because it captures information from the past that have smaller errors, making them robust. This example also intuitively extends to RNN-based RC system because it has been shown that autoregressive RC is equivalent to a linear RNN-based RC model with non-linear readout, as mentioned in [33]. This does not imply that errors will not grow according to the Lyapunov exponent. From the above example, one can postulate that memory will have a time-delayed effect on the error propagation:

$$\frac{\delta_m + \mathbb{E}\{\epsilon_{n+1}\}e^{\Lambda dt}}{\delta_m + \mathbb{E}\{\epsilon_{n+l}\}e^{\Lambda dt}} \leq \frac{\mathbb{E}\{\epsilon_{n+l+1}\}}{\delta_m + \mathbb{E}\{\epsilon_{n+l}\}e^{\Lambda dt}} \leq 1 \tag{4.13}$$

which can be proven by showing an upper boundary:

$$\begin{aligned}\mathbb{E}\{\epsilon_{n+l+1}\} &= \mathbb{E}\{\| \sum_{i=1}^{l} \theta_i \tanh(A_i(y_{n+i} + \epsilon_{n+i}) + b_i) - y_{n+l+1}\|\} \\ &\leq \delta_m + \mathbb{E}\{\sum_{i=1}^{l} \theta_i A_i \epsilon_{n+i}\} \\ &\leq \delta_m + \mathbb{E}\{\sum_{i=1}^{l} \theta_i A_i \epsilon_{n+l}\} \\ &= \delta_m + \mathbb{E}\{\epsilon_{n+l}\} e^{\Lambda dt}\end{aligned} \tag{4.14}$$

and a lower boundary:

$$\begin{aligned}\mathbb{E}\{\epsilon_{n+l+1}\} &= \mathbb{E}\{\| \sum_{i=1}^{l} \theta_i \tanh(A_i(y_{n+i} + \epsilon_{n+i}) + b_i) - y_{n+l+1}\|\} \\ &\geq \delta_m + \mathbb{E}\{\sum_{i=1}^{l} \theta_i A_i \epsilon_{n+1}\} \\ &= \delta_m + \mathbb{E}\{\epsilon_{n+1}\} e^{\Lambda dt}\end{aligned} \tag{4.15}$$

any perturbation will still suffer from exponential growth caused by a positive Lyapunov exponent, but not as severely as a model without memory because it allows insight into previous predicted states with less error. This will only hold as long as information about previous states provides information for a prediction, which only holds up until a certain l_c. An obscure observation has to be stated, these boundaries only hold for positive Lyapunov exponent. This is because the trajectory of the system is space filling, such that any error made by the model will most likely still be in the compact space that was trained on. However if $\Lambda < 0$ it cannot be guaranteed that perturbations will still end up in the compact space it was trained on, making it impossible to know how the model will behave.

To provide some evidence for eq. 4.13, it was generated for the Lorenz-63 system with $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$. The inequality was generated for for 100 different training sets situated in the attractor by letting the system evolve for $50s$ with a timestep of 0.01. The Ridge Regression was done for $r \in [10^{-10}, 1]$. The models were 50 nodes wide (each A and b) and generated with $l \in [1, 10]$, $\|A\| = 0.1$, and $\|b\| = 0.1$.

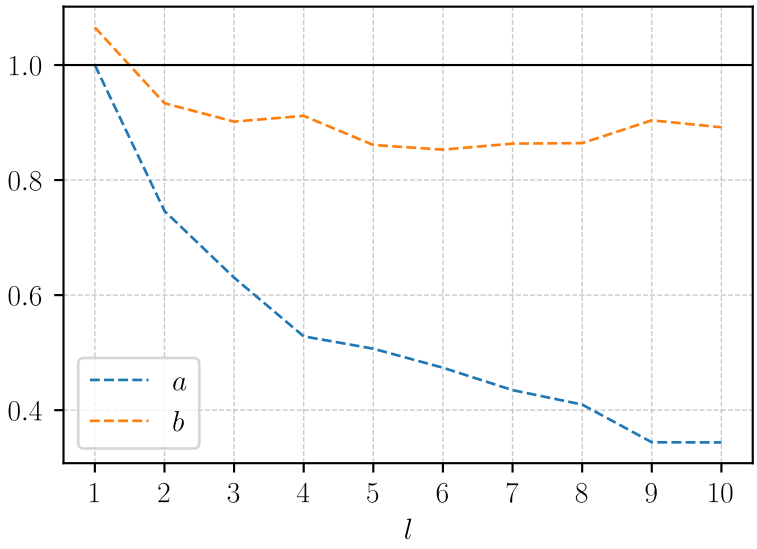


Figure 4.2: Inequality for different l, $a = \frac{\delta_m + \mathbb{E}\{\epsilon_{n+1}\}e^{\Lambda dt}}{\delta_m + \mathbb{E}\{\epsilon_{n+l}\}e^{\Lambda dt}}$, $b = \frac{\mathbb{E}\{\epsilon_{n+l+1}\}}{\delta_m + \mathbb{E}\{\epsilon_{n+l}\}e^{\Lambda dt}}$ with $\Lambda = 0.89$.

The above figure agrees with eq. 4.13 except for $l = 1$ which is most likely because the model was not able to accurately learn the system. This does not inform about the actual predictive performance of models with memory. The inequality is only valid when $\tanh(x) \sim x$, which leads us to ask if a model can actually learn when it is approximately linear.

Is more memory always better?

Deducing l in eq. 4.10 is not well defined nor analytically feasible in standard Reservoir Computing. There is no complete consensus on how to measure this abstract concept of memory. A possible metric is memory capacity, as used in [34], consists of the the sum over time-delayed autocorrelation between a noise input signal and a trained Reservoir Computing system. More precisely:

$$Mc_\tau = \frac{\mathrm{Cov}(s_{i-\tau}, \hat{s}_i^\tau)}{\sigma^2(s_{i-\tau})\sigma^2(s_i^\tau)} \tag{4.16}$$

$$Mc = \sum_{\tau=1}^{\tau_m} Mc_\tau \tag{4.17}$$

Where s is a signal of gaussian noise, $\hat{s}^\tau$ is the prediction of a RC model that takes in s_i as input and tries to approximate $s_{i+\tau}$ via Linear Regression. An important result demonstrated in [34] was that higher sparsity of W_y and R in eq. 4.5 leads to more memory. Another important result was that the spectral radius $\rho(R)$ should be as close as possible to 1, implying that RC performs best near the edge of internal chaos.

More importantly does more memory guarantee better performance? This question can be answered with information theory as demonstrated in [35] where a different substitute for memory was used. This results can be summarized as $I_{y_0}(\epsilon_0, \epsilon_n^L) \geq I_{y_0}(\epsilon_0, \epsilon_n^{NL})$ which states that information about an initial perturbation dissipates faster in a system that is more non-linear. This

is a problem because non-linearity is the cornerstone of the approximation theorems underlying all of the models so far. This is a remarkable result that has unfortunate consequences. In RC, the driving factor of non-linearity is the norm of W_y and b_h which dictates in which regime the regime the reservoir operates. If $\|W_y\|$ and $\|b_h\|$ are small and $\rho(R) < 1$, then $\tanh(x) \sim x$ and therefore approximately linear with better memory capacity. For more 'complex' systems one should not set the norm too small such that the approximative capabilities are not lost in favour of memory.

4.4 Reservoir topology

There have been many proposals for the topology of the reservoir matrix R. The ESP only states that $\rho(R) \leq 1$, and from experimental data it was shown that RC performs better when the sparsity of R (c_R) is low. This does not provide guidelines for the internal structure of this matrix. It has been shown that the performance is mostly independent of the topology of R, therefore reducing the amount of hyperparameters and statistical effects is a good course of action. There is a topology named Simple Cycle which has 1 hyperparameter and is not stochastically initialized. This topology consists of a matrix with only elements on the off-diagonal of the matrix and one extra at a corner to create a cycle of length D in a reservoir matrix of dimension $D \otimes D$. This matrix is initialized via:

$$\begin{cases} R_{ij} = \rho, & \text{if } i = (j+1) \bmod D \\ R_{ij} = 0, & \text{if } i \neq (j+1) \bmod D \end{cases} \tag{4.18}$$

It has been shown that this topology has equivalent expressive power [36], which has been experimentally confirmed [37].

4.5 Conditional Lyapunov exponent and prediction standard deviation

Determining the Lyapunov exponent of a system requires a lot of data as discussed in chapter 1. A solution to situations where there is an absence of data, Reservoir Computing can be deployed to learn the underlying dynamics. From a trained RC model, it is plausible to approximate the Lyapunov exponent since the jacobians of eq. 4.5 can be analytically determined. The exponent of this model is frequently named the conditional Lyapunov exponent since the hidden state of the RC model is driven by the input. To determine this exponent, one needs to determine $J_n \equiv \frac{d\hat{y}_n}{dy_0}$. After some mathematical manipulation, this jacobian for a training set of length m and a total length n is found to be:

$$J_n = A\big(\Pi_{i=0}^{n-1}(R + tAW_y)d_{i+1}\big)W_y \tag{4.19}$$

$$d_i = \text{diag}\left(\frac{\partial}{\partial y_i} f(Rh_i + W_y y_i + b_h)\right) \tag{4.20}$$

$$\begin{cases} t = 1, & \text{if } i \geq m \\ t = 0, & \text{if } i < m \end{cases} \tag{4.21}$$

t Switches from 0 to 1 when the feedback mechanism of eq. 4.6 is active. Diag implies that the elements are placed on a diagonal matrix. In the case where $f(\cdot) = \tanh(\cdot)$:

$$\frac{\partial}{\partial y_i} \tanh(Rh_i + W_y y_i + b_h) = \frac{1}{\cosh^2(Rh_i + W_y y_i + b_h)} \tag{4.22}$$

This result is crucial moving forward, and allows Reservoir Computing to quickly and effectively find the largest Lyapunov exponent internally. This approximation to the Lyapunov exponent of the system can be used in eq. 1.25 to estimate the error the model makes with respect to time. To demonstrate, a 100-wide RC model with $\rho = 0.5$, $\sigma_{W_y} = 1$, $\sigma_{b_h} = 1$, and $s_{W_y} = 0.5$ was trained on the first 50% of a Lorenz-63 system with $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$ integrated over $t \in [0, 200]$ with $\Delta t = 0.01$. The model was forwarded until a random time point between $t \in [100, 200]$ was reached, after which the model was used to predict the time series, which was done 1000 times to get the expectation value of $\epsilon(t)$. Estimating the model error can be done via: $\delta_m \approx \sqrt{\mathcal{L}_{val}}$. The Lyapunov exponent was predicted using eq. 4.19 and eq. 1.10.

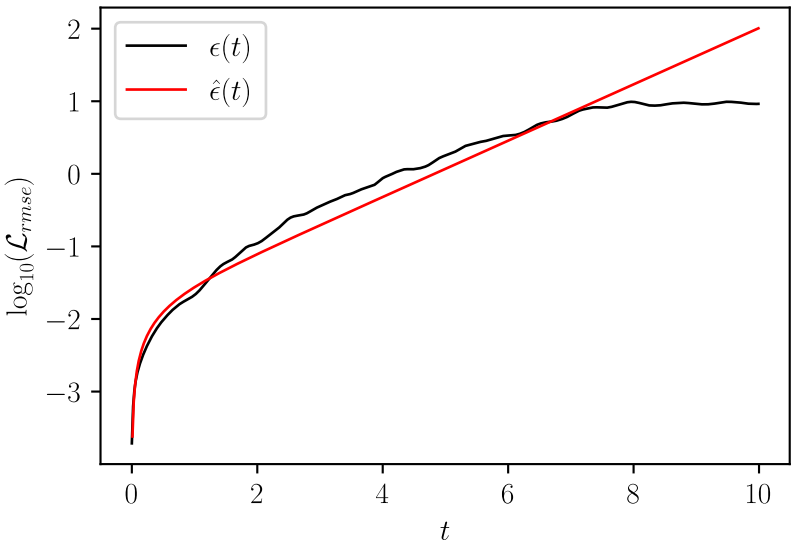


Figure 4.3: Average error prediction $\hat{\epsilon}(t)$ plotted against the real $\epsilon(t)$ of an RC model trained on the Lorenz-63 system.

The slight downward trend visible after $t = 7$ is due to the Lyapunov exponent only being valid for smaller perturbations. This plot demonstrates that eq. 1.25 is remarkably useful to estimate the error that a recurrent method accumulates. This error estimation can also be used to create uncertainty bounds on the prediction the model makes if one views δ_m as the standard deviation of the error made by the model.

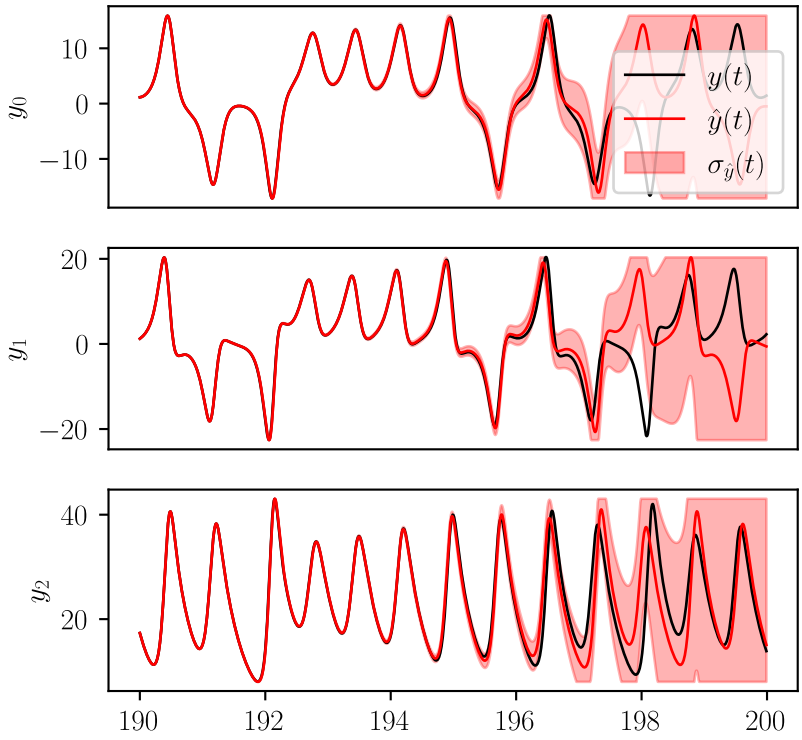


Figure 4.4: Average error prediction $\hat{\epsilon}(t)$ plotted against the real $\epsilon(t)$ of an RC model trained on the Lorenz-63 system.

The above plot highlights the effectiveness of eq. 1.25 when it is used to propagate the standard deviation. The bounds were cut off at the minimum and maximum value of the timeserie predicted by the model to limit the possible predictions to the attractor. Creating such bounds is far from trivial and usually requires specialized models such as Bayesian Neural Networks to create.

4.6 Echo Operator Network (EchONet)

As demonstrated in the previous chapter, feedforward Neural Networks are not ideal when one wants to predict far into the future because of the restraint compact spaces induce on the minimum complexity of the model. Chaotic dynamics are captured over long time periods, therefore making models such as Reservoir Computing preferable. After exploring the ExtremONet, it was shown that it was more performant than a small DeepONet on the test differential equation. The next step in this thesis is to merge the two fields, and create a novel Recurrent Neural Operator,

namely the **novel** *Echo State Neural Operator (EchONet).*

$$\begin{aligned} \mathcal{B}(u(s)) &= f_B(W^B u(s) + b^B) \\ \mathcal{T}_{i+1}(y_i) &= f_T(R\mathcal{T}_i + W^T y_i + b^T) \\ \hat{y}_{i+1}(u(s))(y_i) &= \theta_0 + \theta_1(Rs_q^B \mathcal{B}(u(s)) \odot Rs_q^T \mathcal{T}_{i+1}(y_i)) \end{aligned} \tag{4.23}$$

where W^B is a uniformly sampled matrix of the shape $k \otimes r$, W^T is a uniformly sampled matrix of the shape $k \otimes D$, the bs are normally sampled vectors of the shape $k \otimes 1$. The trainable parameters θ_0 and θ_1 are of the shape $D \otimes 1$ and $D \otimes (k+q)$. Random Selection was used by default because the dimension of R impacts the forwarding time of $\mathcal{T}$ because of the k^2 complexity dependence of square matrix multiplication.

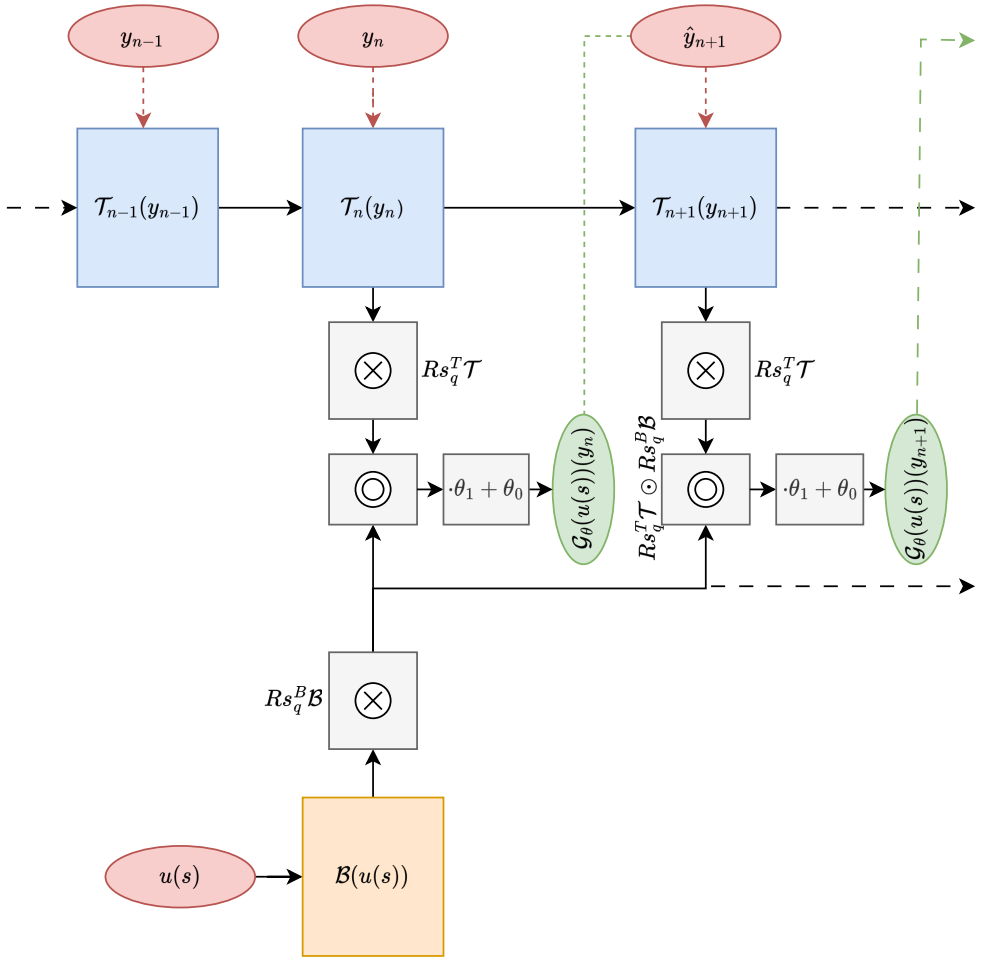


Figure 4.5: Prediction diagram of the EchONet.

This novel model is identical to the ExtremONet, with the addition of an echo-state property, which forces the network to function fundamentally different from other feedforward Neural

Operators. Instead of approximating the operator

$$G(u(y(s), s'))(x) \to y(x) = \int_0^x u(y(x'), x')dx', \tag{4.24}$$

it now approximates

$$G(u(y(s), s'))(y(x)) \to y(x + \Delta x) = \int_x^{x+\Delta x} u(y(x'), x')dx', \tag{4.25}$$

which is a less complex task if Δx is small. Because of the different mechanisms of the ExtremONet and Reservoir Computing, there are many questions to ponder upon.

To determine the ideal parameters, two hyperparameter searches for a 200-wide EchONet model ($k = 100$, $q = 100$) were performed on the Lorenz-63 system with $\sigma = 10$, $\rho \in [0, 30]$, and $\beta = 8/3$, and $\sigma \in [0, 10]$, $\rho \in [0, 30]$, and $\beta \in [0, 8/3]$. 300 uniformly distributed realizations of the parameter space were generated for $t \in [0, 50]$ with $\Delta t = 0.01$. The ranges brute-force searched were $\rho(R) \in [0, 1]$, $c_T \in [0, 1]$, $c_B \in [0, 1]$, $\sigma_T \in [0, 3]$, and $\sigma_B \in [0, 3]$.

	$\rho(R)$	c_T	c_B	σ_T	σ_B	$\mathcal{L}_{train}$	$\mathcal{L}_{test}$
argmin $\mathcal{L}_{train}$	0.46	0.48	0.33	0.40	2.33	$2.5{\cdot}10^{-9}$	$2.7{\cdot}10^{-9}$
argmin $\mathcal{L}_{test}$	0.46	0.48	0.33	0.40	2.33	$2.5{\cdot}10^{-9}$	$2.7{\cdot}10^{-9}$

Table 4.1: Brute-force hyperparameter sweep of the EchoNet on the Lorenz-63 system with $\sigma = 10$, $\rho \in [0, 30]$, and $\beta = 8/3$ to minimize the train and test loss respectively.

	$\rho(R)$	c_T	c_B	σ_T	σ_B	$\mathcal{L}_{train}$	$\mathcal{L}_{test}$
argmin $\mathcal{L}_{train}$	0.52	0.56	0.35	0.30	0.56	$1.6{\cdot}10^{-8}$	$1.8{\cdot}10^{-8}$
argmin $\mathcal{L}_{test}$	0.52	0.56	0.35	0.30	0.56	$1.6{\cdot}10^{-8}$	$1.8{\cdot}10^{-8}$

Table 4.2: Brute-force hyperparameter sweep of the EchoNet on the Lorenz-63 system with $\sigma \in [0, 10]$, $\rho \in [0, 30]$, and $\beta \in [0, 8/3]$ to minimize the train and test loss respectively.

Which is 5 orders of magnitude smaller than the loss of the ExtremONet on a much less complex differential equation and smaller time period. This remarkable performance demonstrates the effectiveness of moving to phase space.

4.6.1 Parametrized Lyapunov exponent prediction

Estimating the Lyapunov exponent is mostly identical to standard Reservoir Computing since only the readout strategy has been altered, therefore:

$$J_n = S\big(\Pi_{i=0}^{n-1}(R + tSW_y)d_{i+1}\big)W_y \tag{4.26}$$

$$S = (\text{diag}((Rs_q^B f_B(W^B u(s) + b^B)) \odot A)Rs_q^T \tag{4.27}$$

$$d_i = \text{diag}\left(\frac{\partial}{\partial y_i} f_T(Rh_i + W_y y_i + b_h)\right) \tag{4.28}$$

$$\begin{cases} t = 1, & \text{if } i \geq m \\ t = 0, & \text{if } i < m \end{cases} \tag{4.29}$$

t Switches from 0 to 1 when the feedback mechanism of eq. 4.6 is active. Diag implies that the elements are placed on a diagonal matrix.

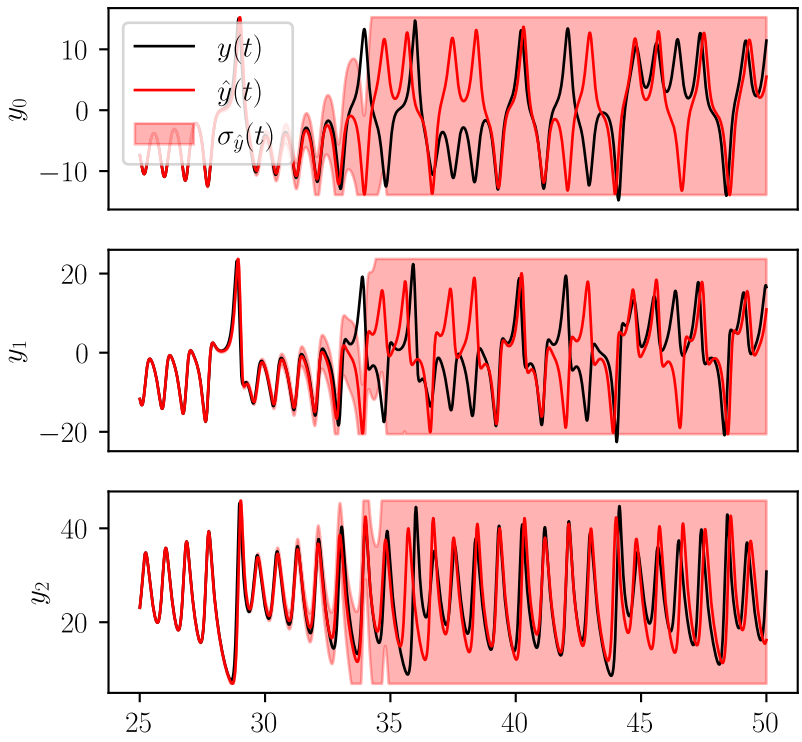


Figure 4.6: Standard deviation prediction for a chaotic Lorenz-63 system by an EchONet

The error prediction capabilities were also tested. Two 200-wide ($k = 100$, $q = 100$) EchONets with $\rho = 0.5$, $\sigma_B = 0.1$, $\sigma_T = 0.11$, $c_B = 0.5$, and $c_T = 0.5$ were trained on the first 50% of two Lorenz-63 datasets, each containing 300 time series with $\sigma = 10$, $\rho \in [0, 30]$, and $\beta = 8/3$

and $\sigma \in [0, 10]$, $\rho \in [0, 30]$, and $\beta \in [0, 8/3]$ integrated over $t \in [0, 200]$ with $\Delta t = 0.01$. The model was forwarded until a random time point between $t \in [100, 200]$ was reached, after which the model was used to predict the time series, which was done 1000 times to get the expectation value of $\epsilon(t)$.

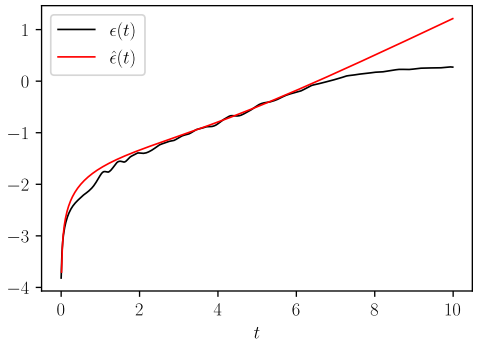


Figure 4.7: Average error prediction $\hat{\epsilon}(t)$ plotted against the real $\epsilon(t)$ of an EchONet model trained on the Lorenz-63 system with varying ρ.

The EchONet was able to almost perfectly predict the error growth of its predictions, indicating that the Lyapunov exponent predictions, in combination with eq. 4.26, are functioning as intended.

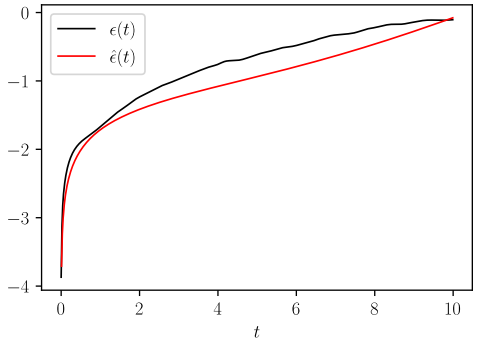


Figure 4.8: Average error prediction $\hat{\epsilon}(t)$ plotted against the real $\epsilon(t)$ of an EchONet model trained on the Lorenz-63 system with varying ρ, σ, and β.

The error growth for the entire parameter space of the Lorenz-63 system was not exact, which indicates that more data is necessary to properly generalize to the entire space.

4.6.2 Network design

The introduction of an Echo State Property into an ExtremONet design should result in better performance. To test this, the hyperparameter search datasets (varying ρ; varying σ, ρ, and β) were learned by EchONets with varying $\rho(R)$, $\sigma_B = 0.5$, $\sigma_T = 0.5$, $c_B = 0.5$, and $s_{W_T} = 0.5$. This experiment was repeated 30 times to get a statistically relevant picture.

	$\rho(R) = 0$	$\rho(R) = 0.25$	$\rho(R) = 0.5$	$\rho(R) = 0.75$	$\rho(R) = 1$
$\mathcal{L}_{train}$	$3.7 \cdot 10^{-9}$ $\pm 6.6 \cdot 10^{-10}$	$2.4 \cdot 10^{-9}$ $\pm 5.4 \cdot 10^{-10}$	$1.5 \cdot 10^{-9}$ $\pm 3.8 \cdot 10^{-10}$	$2.5 \cdot 10^{-9}$ $\pm 7.6 \cdot 10^{-10}$	$4.9 \cdot 10^{-9}$ $\pm 1.9 \cdot 10^{-9}$
$\mathcal{L}_{test}$	$5.4 \cdot 10^{-9}$ $\pm 1.7 \cdot 10^{-9}$	$3.3 \cdot 10^{-9}$ $\pm 1.0 \cdot 10^{-9}$	$2.6 \cdot 10^{-9}$ $\pm 8.0 \cdot 10^{-10}$	$4.6 \cdot 10^{-9}$ $\pm 4.5 \cdot 10^{-9}$	$1.9 \cdot 10^{-8}$ $\pm 7.7 \cdot 10^{-9}$

Table 4.3: Spectral radius $\rho(R)$ dependence of the EchONet for the L-63 system with varying ρ.

	$\rho(R) = 0$	$\rho(R) = 0.25$	$\rho(R) = 0.5$	$\rho(R) = 0.75$	$\rho(R) = 1$
$\mathcal{L}_{train}$	$2.6 \cdot 10^{-8}$ $\pm 5.9 \cdot 10^{-9}$	$8.6 \cdot 10^{-9}$ $\pm 2.4 \cdot 10^{-9}$	$8.5 \cdot 10^{-9}$ $\pm 3.2 \cdot 10^{-9}$	$1.3 \cdot 10^{-8}$ $\pm 6.7 \cdot 10^{-9}$	$5.3 \cdot 10^{-8}$ $\pm 2.8 \cdot 10^{-8}$
$\mathcal{L}_{test}$	$2.7 \cdot 10^{-8}$ $\pm 6.0 \cdot 10^{-9}$	$9.6 \cdot 10^{-9}$ $\pm 2.9 \cdot 10^{-9}$	$9.3 \cdot 10^{-9}$ $\pm 3.7 \cdot 10^{-9}$	$1.6 \cdot 10^{-8}$ $\pm 7.4 \cdot 10^{-9}$	$6.4 \cdot 10^{-8}$ $\pm 3.2 \cdot 10^{-8}$

Table 4.4: Spectral radius $\rho(R)$ dependence of the EchONet for the L-63 system with varying σ, ρ, and β.

In both situations a spectral radius of $\rho(R) = 0.5$ resulted in the best mean train-and test error; and for the varying ρ dataset, it resulted in the lowest standard deviation of the train-and test error.

The Random Selection process was effective in reducing the dimensionality of the ExtremONet branch and trunk networks. This benefit is not guaranteed to work for phase space methods since the problem is drastically reduced in complexity. To test the dependence of performance on the network dimensionality, the previous two datasets were trained on by various combinations of k and q.

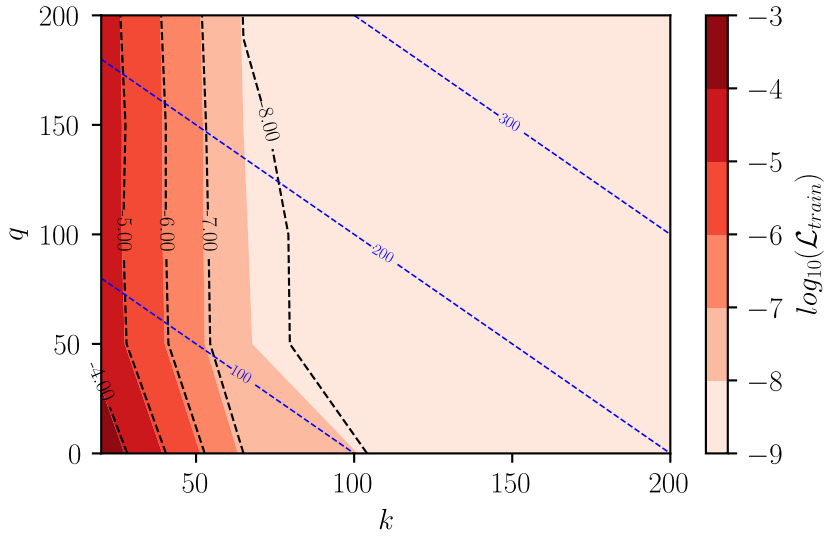
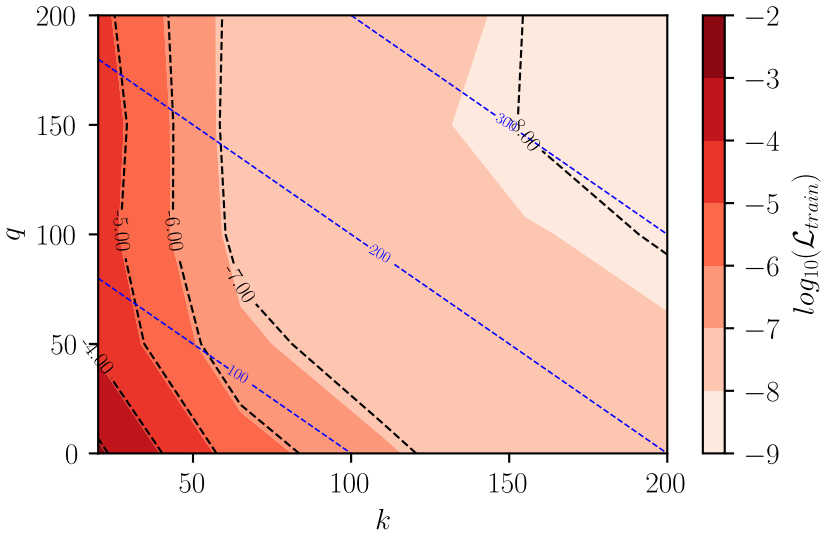


Figure 4.9: Average test and train error of the EchONet with $\rho(R) = 0.5$, $c_T = 0.50$, $c_B = 0.35$, $\sigma_T = 0.40$, $\sigma_B = 2.30$ and varying k and q, trained on the Lorenz-63 with varying ρ. The black dotted lines represent test error, the red gradient contour represents train error, and the blue dotted line represents constant $q + k$.

When only ρ varies, the parameter space is small and simple in nature, which is reflected in the above figure; increasing q has almost no impact because a low-dimensional function basis is enough to properly approximate the entire parameter space. Increasing k does improve the performance, but is redundant after $k = 100$. This hard limit could be due to the nature of numerically integrating which produces an error of the order dt^{-5} for RK-4 and therefore limiting the loss to around 10^{-10}. The errors made by the numerical integration scheme is not per se unlearnable, but Occam's Razor dictates that it is the most probable cause. If true, then the EchONet is equally accurate as the numerical scheme used to generate the data, which is encouraging.



0.46

Figure 4.10: Average test and train error of the EchONet with $\rho(R) = 0.5$, $c_T = 0.55$, $c_B = 0.35$, $\sigma_T = 0.3$, $\sigma_B = 0.55$ and varying k and q, trained on the Lorenz-63 with varying σ, ρ, and β. The black dotted lines represent test error, the red gradient contour represents train error, and the blue dotted line represents constant $q + k$.

When the entire parameter space is varied, the picture slightly changes. Because the parameter space is now more complex, the model does benefit from a larger basis obtained by increasing q. The EchONet was also able to learn the problem to the same accuracy as the data was generated by, showing that by sacrificing grid-invariance, it is possible to learn any possible realization (compact) of the Lorenz-63 system without issue. Being able to learn the dynamics to a certain accuracy does not mean that the model can retrieve the Lyapunov exponent when it is in prediction mode, relying on self-feedback. If the model is unstable, accurately predicting timeseries and Lyapunov exponents is impossible.

A final remark is that there appears to be no relation between model test error and the predicted Lyapunov exponent error. This was confirmed by generating 100 random 200-wide ($k = 100$, $q = 100$) models with $\rho(R) \in [0, 1]$, $c_T \in [0, 1]$, $c_B \in [0, 1]$, $\sigma_T \in [0, 1]$, $\sigma_B \in [0, 1]$ and training them on the hyperparameter search dataset and predicting the Lyapunov exponents of some test set consisting of 30 timeseries, generated in a similar manner to the training set. The Pearson correlation between the test error and Lyapunov error was -0.08, indicating that there is no correlation.

4.6.3 Data dependency

Reservoir Computing is built upon equidistantly sampled data (not grid invariant), meanwhile the DeepONet architecture trains on a single parameter space realization and therefore only one grid-invariant point per time series. There is no reason except for efficiency to only sample one point, which does not per se impact the performance. The EchONet is thus not grid invariant in its sensor locations nor its time point sampling. One has to carefully analyse the impact of dataset construction on the performance of the model.

To test the impact of dataset construction on performance, a 200-wide EchONet model ($k = 100$, $q = 100$) was trained on the Lorenz-63 system with $\sigma = 10$, $\rho \in [0, 30]$, and $\beta = 8/3$, and $\sigma \in [0, 10]$, $\rho \in [0, 30]$, and $\beta \in [0, 8/3]$. 500 uniformly distributed realizations of the parameter space were generated for $t \in [0, 50]$ with $\Delta t = 0.01$. The parameter space realizations were varied from $500 n^p_{train}/n^p_{total}$ with $n^p_{train} \in [0, 1]$, the amount of randomly selected time points was varied from $5000 n^t_{train}/n^t_{total}$ with $n^t_{train} \in [0, 1]$.

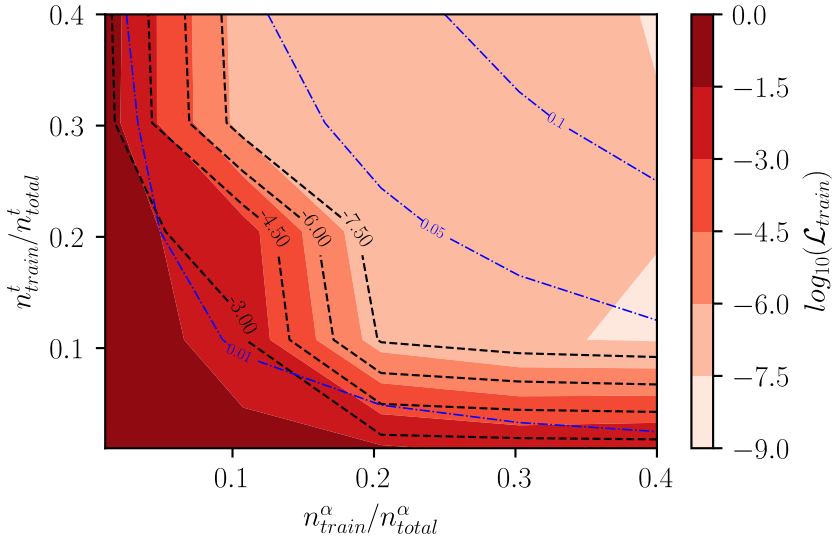


Figure 4.11: Average test and train error of the EchONet, trained on the Lorenz-63 with varying σ, and with varying amounts of training samples and parametrizations. The black dotted lines represent test error, the red gradient contour represents train error, and the blue dotted line represents constant $q + k$.

Unexpectedly, there are not many parameter realizations necessary. 75 Different realizations with at least 7500 time points each were enough to converge. This result is very surprising since the DeepONet used 1 realization per timepoint to train (as prescribed in [9]). On the other hand, one can compensate for a lack of time points by training on more parameter realizations. These two extremes are further enforced by the path of the blue lines, which indicate a constant amount of data, and the error contour lines.

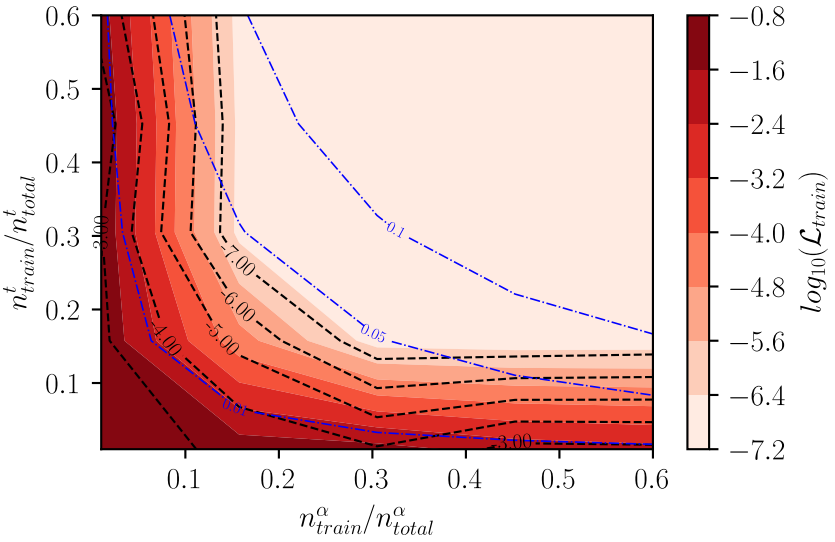


Figure 4.12: Average test and train error of the EchONet, trained on the Lorenz-63 with varying σ, ρ, β, and with varying amounts of training samples and parametrizations. The black dotted lines represent test error, the red gradient contour represents train error, and the blue dotted line represents constant $q + k$.

For the entire parameter space, the situation is identical. There is less data required here to converge to a minimum possible error because the network is too small to learn the entire space.

The amount of sensor locations, as was tested in the ExtremONet section, needs to be tested again. To do so, the above datasets were tested on their respective EchONet. The amount of available sensor features was varied for 10 different values with $r \geq 1$ for 30 iteration to get a statistically relevant picture.

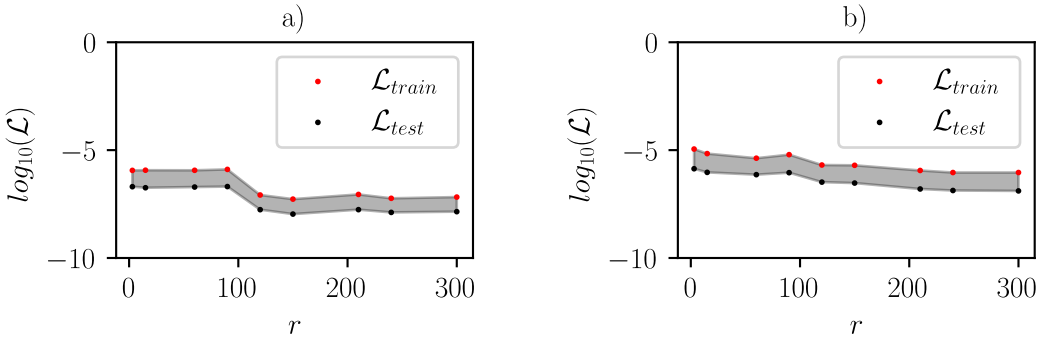


Figure 4.13: a) Mean train and test errors of the EchONet trained on the Lorenz-63 system with varying ρ and r. b) Mean train and test errors of the EchONet trained on the Lorenz-63 system with varying σ, ρ, β, and r.

The EchONet did not benefit much from an increase in sensor data, except for the case where only ρ was varied, which displays a sudden drop in train and test error around the $k = 100$ mark. There might be such a drop for the Lorenz-63 dataset where σ, ρ, and β were varied; this would then occur in a region where $r \geq q + k$, which will hinder the performance. It might be the case that the dataset where the entire parameter space is varied benefits from a very large Branch network size, which could be studied in a later work.

Another question worth investigating is what the smallest final time point is which is necessary to generalize to the whole space. Because there is a warm-up period present in Reservoir Computing due to the transient dynamics caused by the initial hidden state, it might not be possible to generalize. This should be remediable by lowering $\rho(R)$ until the initial state does not impact the result much, and the system develops in a similar manner initially for all parameter realizations (similar transient dynamics before being trapped by an attractor). Another problem is that the initial period of a timeserie is usually transient, and therefore does not directly inform about the dynamics in the attractor.

Showing whether the EchONet can generalize just by learning from the transient phase can be done in a similar way to the previous section; instead of taking random time points, only the first $100nf\%$ were used to train on, and tested against the entire test dataset. The same datasets were used, and the parameter space realizations were varied from $500n^p_{train}/n^p_{total}$ with $n^p_{train} \in [0, 1]$, the final time point was varied from $5000n^f_{train}/n^f_{total}$ with $n^f_{train} \in [0, 1]$.

Determining whether a system is in a transient phase or a stable configuration for an entire parameter space is not necessary. If the system can learn from the transient dynamics only, then there should only be a small threshold present because of the transient phase of the hidden state, which can be reduced by reducing $\rho(R)$. This will qualitatively be shorter than the transient phase of the system, as seen in Fig. 1.6.

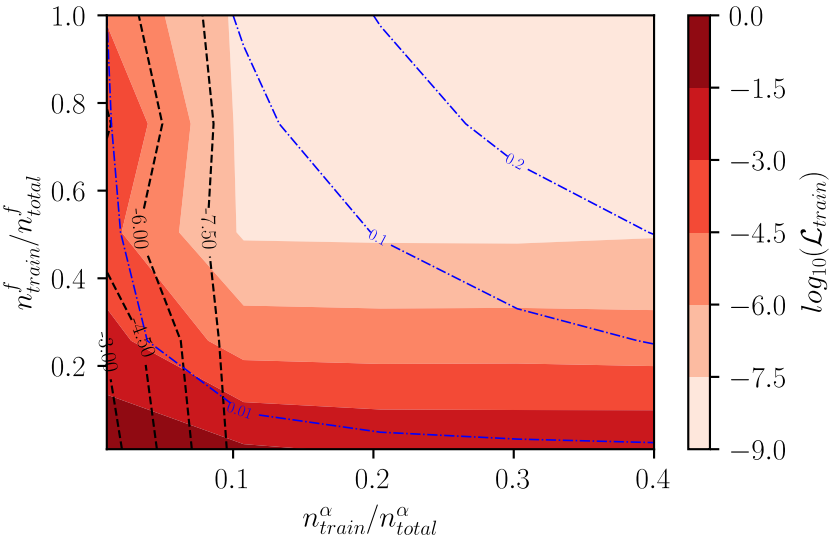


Figure 4.14: Average test and train error of the EchONet, trained on the Lorenz-63 with varying σ, and with varying amounts of training samples and final time point. The black dotted lines represent test error, the red gradient contour represents train error, and the blue dotted line represents constant $q + k$.

From the above plot one can deduce that the final timepoint should be at least $t = 25$, which indicates that it cannot generalize from transient dynamics to stable dynamics for $\rho \in [0, 30]$. This final point is most certainly not due to the transient dynamics of the hidden state of the model, because with $\rho(R) = 0.5$, an initial perturbation reduces to approximately 10^{-10} after $\log_{0.5}(10^{-10}) = 33$ iterations (t=0.33).

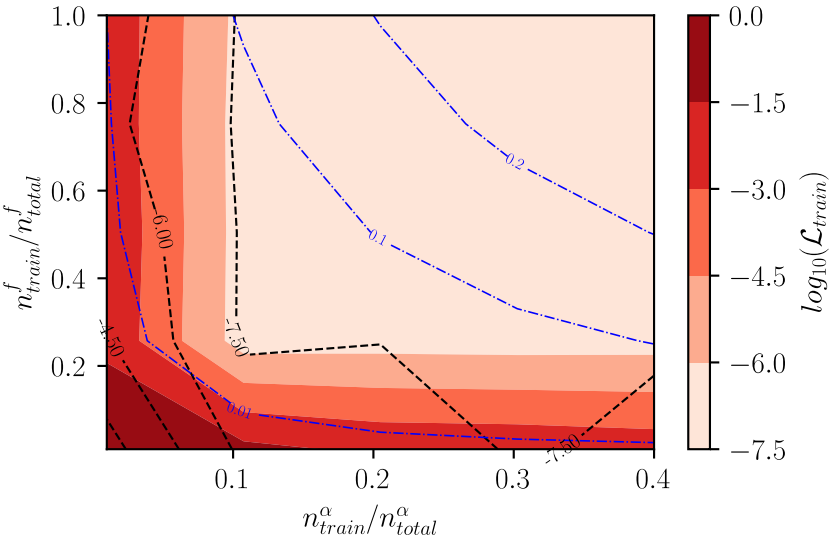


Figure 4.15: Average test and train error of the EchONet, trained on the Lorenz-63 with varying σ, ρ, β, and with varying amounts of training samples and final time point. The black dotted lines represent test error, the red gradient contour represents train error, and the blue dotted line represents constant $q + k$.

4.6.4 Decoupling of training and predicting phases

A problem with Recurrent Neural Networks is that the prediction phase is always sequential to the training phase. If one trains for $t \in [0, t_1]$ then the model can predict what happens at $t \in [t_1, t_2]$. The EchONet does not suffer from this limitation because the model generalized over the entire parameter subspace. No information except for an initial position (which is identical for all training timeseries) is required. This is a big leap forward for RNNs. Because,in theory, EchONets are able to zero-shot predictions which can be very beneficial in many situations, for example in predicting the Lyapunov exponent of a system without actual temporal data, just the sensor measurements. Of course this requires that many realizations of $u(\cdot)$ are available to train on. Zero-shot timeserie predictions are equivalent to model-free numerical integration. As far as the author is aware, there is no other recurrent model in the literature that is able to achieve this.

Zero-shot predictions

To test whether the model is able to predict the Lyapunov exponent of a system without timeserie data, two 200-wide ($k = 100$, $q = 100$) EchONet with $\rho = 0.5$, $\sigma_B = 0.1$, $\sigma_T = 0.11$, $c_B = 0.5$, and $s_{W_T} = 0.5$ were trained on the two Lorenz-63 datasets, each containing 300 timeseries with $\sigma = 10$, $\rho \in [0, 30]$, and $\beta = 8/3$ and $\sigma \in [0, 10]$, $\rho \in [0, 30]$, and $\beta \in [0, 8/3]$ integrated over $t \in [0, 200]$ with $\Delta t = 0.01$. The test sets, which contained 100 timeseries each, of which the

Lyapunov exponents were predicted, were generated in the same manner as the training set. n_f represents the amount of sequential timeserie data presented to the predicting phase.

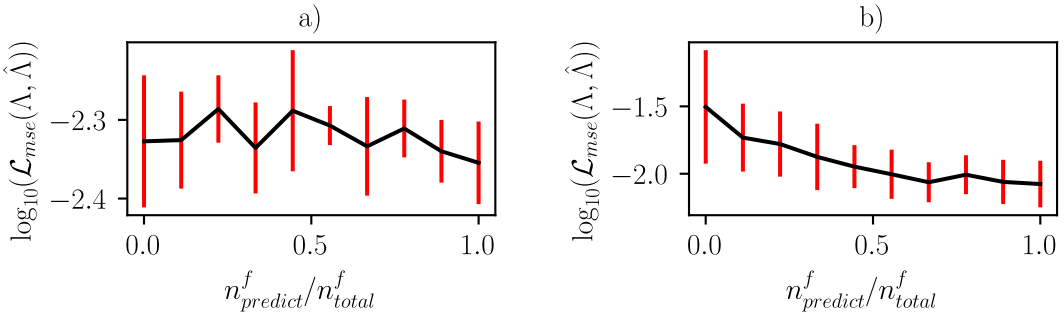


Figure 4.16: a) Mean and standard deviation of Lyapunov exponent prediction errors with respect to the final timepoint of the initial input data for an EchONet trained on the Lorenz-63 system with varying ρ. b) Mean and standard deviation of Lyapunov exponent prediction errors with respect to the final timepoint of the initial input data for an EchONet trained on the Lorenz-63 system with varying σ, ρ, and β.

The Lyapunov predictions of a dataset where ρ was varied is lightly dependent on the amount of initial data present. This dependence appears to be almost negligible, indicating that the model was able to learn the underlying dynamics without relying on initial data. The fact that the dependence is quasi-linear, it can be deduced that the transient dynamics of the initial hidden state bare no impact on the behaviour of the trajectory.

The situation is quite similar for the entire phase-space of the Lorenz-63 system, from which it is clear that one does not need timeserie data to properly estimate the Lyapunov exponent of a system. The model was able to circumvent the issue of initial data that prevents Reservoir Computing to be used in many situations where experimental data is not widely available. An important subtilty is that the datasets contain a transient phase, where the Lyapunov exponent is not valid. Therefore, any prediction of $\epsilon(t)$ is not per se valuable because it will most likely overestimate the growth in chaotic systems, and underestimate the growth in non-chaotic systems.

To demonstrate the usefulness of the EchONet zero-shot capabilities, fig. 1.8 was recreated in more detail. 3 figures were generated, where the Lyapunov exponent was calculated for varying $\sigma - \rho$, $\rho - \beta$, and $\beta - \sigma$. Each plot consists of a 20×20 grid from timeseries between $t \in 0 - 200$ with a timestep of 0.01. The first 25% were discarded to avoid transient dynamics. A 150-wide ($k = 75$, $q = 75$) EchONet was trained on 500 random timeseries with $t \in [0, 50]$, in the generated parameter intervals, and then zero-shot predicted the Lyapunov exponent. The first 25% of the predicted timeseries were discarded to avoid transient dynamics. In this situation, the EchONet behaves like a numerical integrator that has no direct insight into the system except for the sensor measurements.

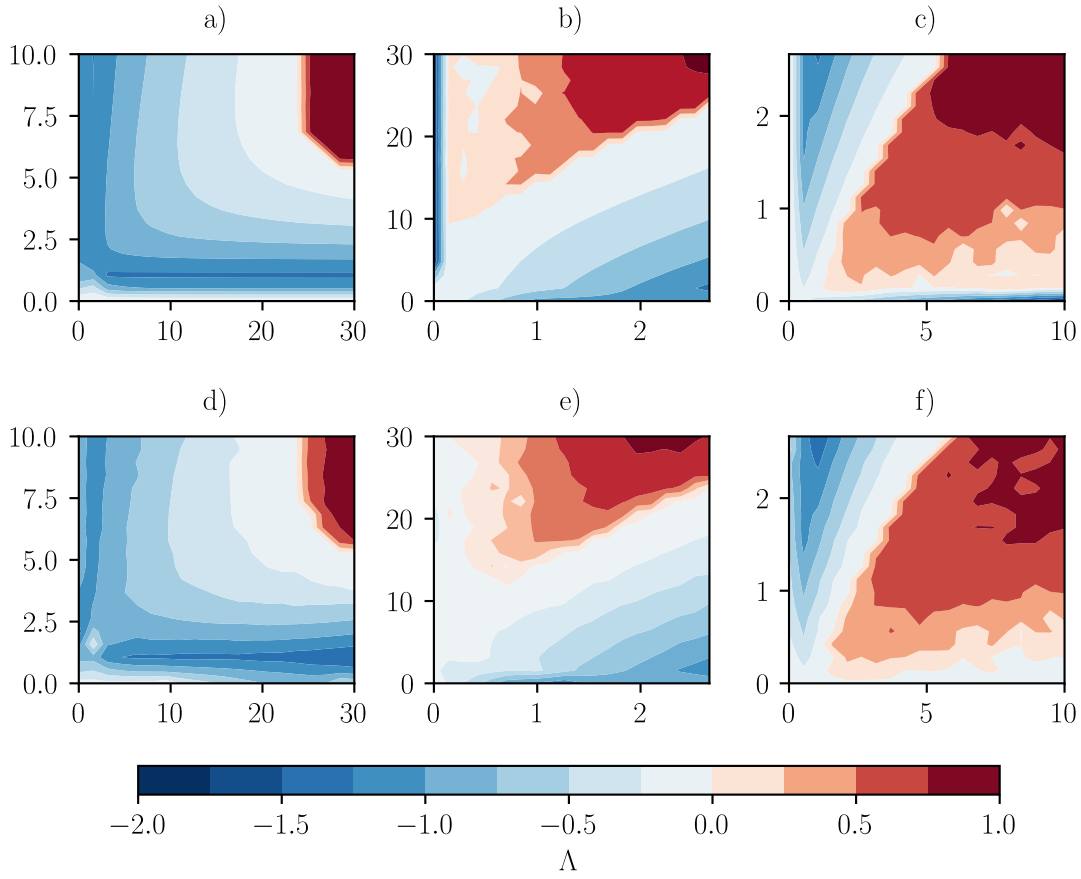


Figure 4.17: a) $\sigma - \rho$ plane Lyapunov exponent of the Lorenz-63 system with $\beta = 8/3$. b) $\rho - \beta$ plane Lyapunov exponent of the Lorenz-63 system with $\sigma = 10$. c) $\beta - \sigma$ plane Lyapunov exponent of the Lorenz-63 system with $\rho = 28$. d) EchONet zero-shot prediction of $\sigma - \rho$ plane Lyapunov exponent of the Lorenz-63 system with $\beta = 8/3$. e) EchONet zero-shot prediction of $\rho - \beta$ plane Lyapunov exponent of the Lorenz-63 system with $\sigma = 10$. f) EchONet zero-shot prediction of $\beta - \sigma$ plane Lyapunov exponent of the Lorenz-63 system with $\rho = 28$.

The small EchONet was able to approximate the figures without outliers, demonstrating that it did generalize to the entire parameter space. The fact that it generalized is interesting since it was only trained on short timeseries, which themselves did not carry enough information to determine the Lyapunov exponents. This is a hallmark indicator that the model does function like a Reservoir Computing model. It took the numerical method $919s$ to generate each individual plot, meanwhile the EchONet only took $576s$ per plot. This demonstrates the usefulness of the parallelization strategy embedded into the architecture.

Interpretation

These zero-shot predictions still require some data, in the form of sensor data. Sensor data can be seen as the time-derivative of the system at different locations $u(s) = \frac{d}{dt}y(\cdot)|_{y(0)=s}$, which does not rely on a developed timeserie. Such sensor measurements could be approximated by letting a system, with the sensor locations as initial conditions, develop just like a timeserie for 1 step, and using finite-differences to approximate this time derivative $\frac{d}{dt}y(\cdot)|_{y(0)=s} \approx \frac{y(\Delta t)-y(0)}{\Delta t}|_{y(0)=s} = \frac{s(\Delta t)-s(0)}{\Delta t}$. Such a trade-in could be very helpful when a system takes a long time to develop learnable dynamics, such as weather systems or astronomical trajectories.

4.7 Generalized Lorenz-96 and the road towards EchONets for chaotic PDEs

The Lorenz-63 system is a 'simple' chaotic system. The behaviour in three dimensions can be visually explored via phase-space diagrams. The Lorenz-96 model is an N-dimensional locally coupled differential equation that, similar to the Lorenz-63 system, can exhibit Chaos for certain parametrizations [38].

$$\dot{y}_i = y_{i-1}(y_{i+1} - y_{i-2}) - y_i + \mu \tag{4.30}$$

where μ is some forcing term. The equation is similar to a discretized 1-dimensional partial differential equation in the sense that it only exhibits neighbour interactions. These neighbour interactions mimic advection terms in partial differential equations, which are of the form $\frac{\partial y}{\partial t} = -u\nabla y$, where u is some velocity field. Advection is the main driving factor of Chaos in fluids and weather systems [39]. Due to this resemblance, the equation can be modified such that it adheres to toroidal boundary conditions ($y_{N+1} = y_1$); J. Kerin and H. Engler analysed a generalization of this system [38], which allows for complexer parametrization:

$$\dot{y}_i = CG(y) - By + \mu(t) \tag{4.31}$$

where C, and G are diagonal matrices, and the map G creates an advection term, which is

- **quadratic**: $G(y) = \frac{1}{2}B(y, y)$ where $B : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ is symmetric and bilinear.
- **energy preserving**: $y^t G(y) = 0$, where $E \equiv y^t y$ is a constant of motion, and referred to as the energy of the system.
- **equivariant**: the permutation $\rho(y) \equiv \{y_2, y_3, .., y_d, y_1\}$ on the group G is equivalent to the permutation on the input elements $\rho(G(x)) = G(\rho(x))$, and therefore G is $\langle\rho\rangle$-invariant.
- **k-localized**: The only non-zero elements of the matrix G are located within $G_{i,i\pm k}\ \forall i$. In other words, the largest index distance from the diagonal of any non-zero elements is k, such that the system it generates can only interact with $\pm 1, \pm 2, ..., \pm k$-th neighbours.

They proved that for small forcing terms $\mu(t)$, there are unique and globally stable solutions. In their paper, they studied a simple parametrization of the standard Lorenz-96, which is equivalent to an advection term $G_3(y)$, which is a basis element of the class of all advection terms with $k = 2$, denoted as $\mathscr{G}_2$. The basis elements of this family of maps can be denoted as (for $k \in \{1, 2, 3\}$):

- $k = 1$:

$$G_1(y)_i = y_{i+1}^2 - y_i y_{i-1}, \quad \bar{G}_1(y)_i = y_{i-1}^2 - y_i y_{i+1} \tag{4.32}$$

- $k = 2$:

$$\begin{aligned} &G_1(y)_i = ..., \quad \bar{G}_1(y)_i = ... \\ &G_2(y)_i = y_{i+2}^2 - y_i y_{i-2}, \quad \bar{G}_2(y)_i = y_{i-2}^2 - y_i y_{i+2} \\ &G_3(y)_i = y_{i-1} y_{i+1} - y_{i-2} y_{i-1}, \quad \bar{G}_3(y)_i = y_{i+1} y_{i-1} - y_{i+2} y_{i+1} \end{aligned} \tag{4.33}$$

- $k = 3$:

$$\begin{aligned} &G_1(y)_i = ..., \quad \bar{G}_1(y)_i = ... \\ &G_2(y)_i = ..., \quad \bar{G}_2(y)_i = ... \\ &G_3(y)_i = ..., \quad \bar{G}_3(y)_i = ... \\ &G_4(y)_i = y_{i+3}^2 - y_i y_{i-3}, \quad \bar{G}_4(y)_i = y_{i-3}^2 - y_i y_{i+3} \\ &G_5(y)_i = y_{i+2} y_{i+3} - y_{i-2} y_{i+1}, \quad \bar{G}_5(y)_i = y_{i-2} y_{i-3} - y_{i+2} y_{i-1} \\ &G_6(y)_i = y_{i+1} y_{i+3} - y_{i-1} y_{i+2}, \quad \bar{G}_6(y)_i = y_{i-1} y_{i-3} - y_{i+1} y_{i-2} \end{aligned} \tag{4.34}$$

Random combinations of these basis elements will still be advection terms. $\bar{G}(y)_i$ is a reflection of $G(y)_i$, via $x_{i+k} \to x_{i-k}$. Linear combinations of these advection basis terms have not been studied yet, only $G_3(y)$ (and $\bar{G}_3(y)$ by symmetry) have received attention because it was the original formulation of the L-96 system. Studying these linear combinations is not trivial; a system of d dimensions and with all advection terms $k = 3$, a source term, and a drag term would result in a differential equation with $26d$ separate terms, where $24d$ terms are quadratic. A generalized L-96 differential equation now takes the form:

$$\dot{y}_i = \sum_{G_j \in \mathscr{G}_k} \alpha_j G_j(y)_i + \sum_{\bar{G}_n \in \mathscr{G}_k} \alpha_n \bar{G}_n(y)_i - y_i + \mu(t). \tag{4.35}$$

A reduced example of this differential equation is the Orszag-McLaughlin family, which is a linear combination of G_3 and $\bar{G}_3$. To reduce the parametric complexity of the general case, the $\bar{G}$ terms will be removed due to their symmetry with G, and the parameters α will be normalized such that it creates a hyperplane in the $\mathscr{G}$-space, which is done by letting $\alpha_j \to \beta \alpha_j / \sum_i \alpha_i$, and setting the total value with a constant β. Only considering the plane cases will produce a map of the same total energy E regardless of the advection parametrization, and reduces the parameter space dimension by 1. We shall also assume a constant forcing term μ.

$$\dot{y}_i = \sum_{G_j \in \mathscr{G}_k} \alpha_j G_j(y)_i - y_i + \mu. \tag{4.36}$$

A generalized Lorenz-96 system with a right-handed advection basis $\mathscr{G}_k$, and dimension d will be referred to as a $\{\mathscr{G}_k, d\}$-system; the left-handed form will be denoted as $\{\bar{\mathscr{G}}_k, d\}$. Studying this differential equation with numerical techniques can be time-consuming. If one wants to determine the regions in the parameter space, the amount of sampled timeseries necessary can become too large to generate sequentially. For example, if $k = 3$, then $\{\mu, \alpha\} \in \mathbb{R} \times [0, 1]^5$. If one wants to equidistantly sample every dimension, then 10^6 timeseries will have to be generated sequentially. If an EchONet is able to generalize for the entire space $\mathscr{G}_3$, then it could be

used to generate all these timeseries in parallel, which can be orders of magnitude faster as long as the model size does not grow exponentially with respect to the parameter space dimension and the system dimension. This should not be the case according to the Manifold Hypothesis [40].

To test whether the EchONet is able to learn larger parameter spaces, which appeared to be the case for the Lorenz-63 system, $\{G_0, ..., G_6\}$ were first learned individually to discern which elements generate chaotic behaviour for positive forcing terms. The non-chaotic basis terms will be displayed in appendix B.2. Afterwards, the terms that showed potential chaotic behaviour were put on the surface of a hyperplane of side length $\beta = 1$ such that it can be compared with the individual basis elements.

The systems produced by individual basis elements were learned by a 200-wide ($k = 100$, $q = 100$) EchONet, with $\rho(R) = 0$, $\sigma_B = 0.5$, $\sigma_T = 0.5$, $c_B = 0.5$, and $c_T = 0.5$ and 100 sensors. The ESP property was disabled such that there was no transient phase that could hinder zero-shot predictions. The sensor location were uniformly sampled between $s \in [-10, 10]^d$. This impacted the general performance of the model with regard to training and test error, but was necessary because the timeseries showed aggressive and short transient behaviour. The only basis elements that produced chaotic behaviour for positive forcing constants were G_3, G_5, and G_6.

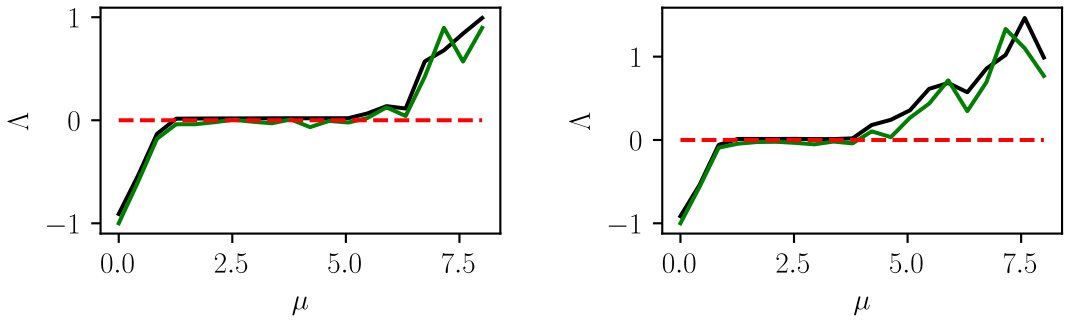


Figure 4.18: Lyapunov exponent with respect to the forcing constant μ and the zero-shot approximated values by an EchONet for G_3 (classic L-96), $d = 6$, and $d = 10$. Green is the true value, black is the predicted value.

G_3, also known as the classic Lorenz-96 system, displayed similar behaviour in its lowest possible dimension configuration and a higher-dimensional configuration. The transition into Chaos happened earlier in the 10-dimensional configuration, around the $\mu = 5$ mark.

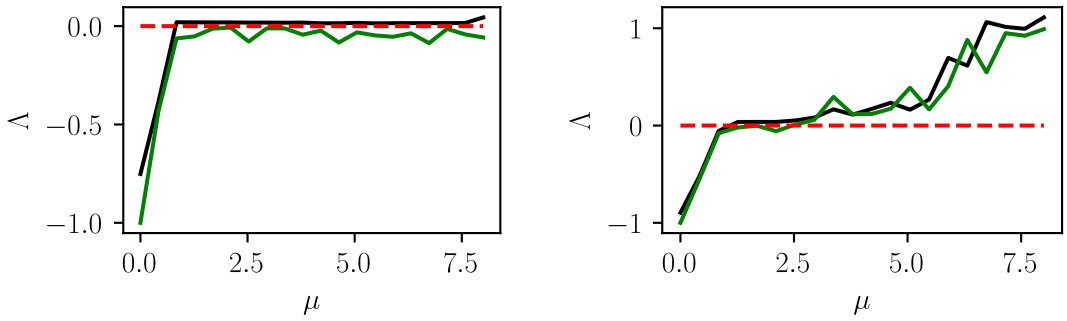


Figure 4.19: Lyapunov exponent with respect to the forcing constant μ and the zero-shot approximated values by an EchONet for G_5, $d = 8$, and $d = 10$. Green is the true value, black is the predicted value.

G_5's behaviour is dimension dependent. The 8-dimensional systems quickly moved to stable periodic behaviour, which did not happen in the 10-dimensional case, where around the $\mu = 2.5$ mark the system slowly moved towards more chaotic trajectories.

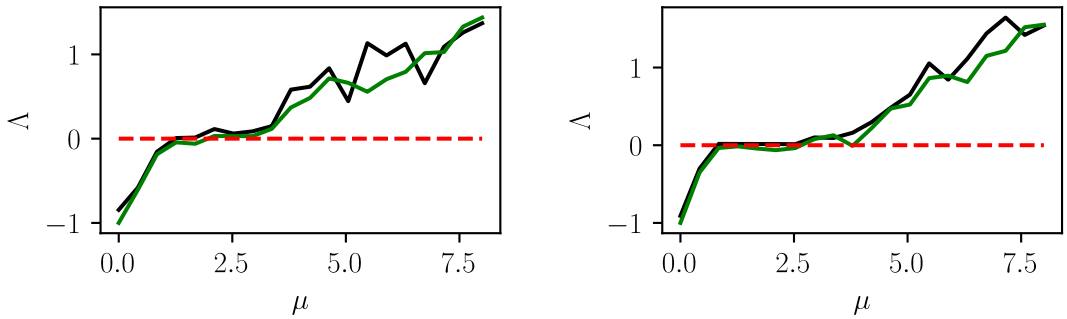


Figure 4.20: Lyapunov exponent with respect to the forcing constant μ and the zero-shot approximated values by an EchONet for G_6, $d = 8$, and $d = 10$. Green is the true value, black is the predicted value.

The G_6 timeseries have opposite dimension-relations to G_3. The systems turned chaotic later with respect to μ for the higher-dimensional configuration.

All basis element configurations were learnable by a relatively small EchONet. The mean squared train-and test error were approximately 10^{-10}, and the mean squared zero-shot Lyapunov exponent error was approximately 0.012, which is reasonably accurate. To test whether the EchONet can learn a high-dimensional system, with a high dimensional parameter space, a 600-and 800-wide ($k = 300$, $q = 300$, and $k = 400$, $q = 400$) EchONet, with $\rho(R) = 0$, $\sigma_B = 0.5$, $\sigma_T = 0.5$, $c_B = 0.5$, and $c_T = 0.5$ was trained on two instances of eq. 4.36, namely $\{\mathscr{G}_3/\{G_1, G_2, G_4\}, 8\}$ and $\{\mathscr{G}_3/\{G_1, G_2, G_4\}, 10\}$ to explore the impact of differently sized systems on the Lyapunov exponent. The $\alpha_1 - \alpha_2$ planes were equidistantly sampled over a 10×10 grid.

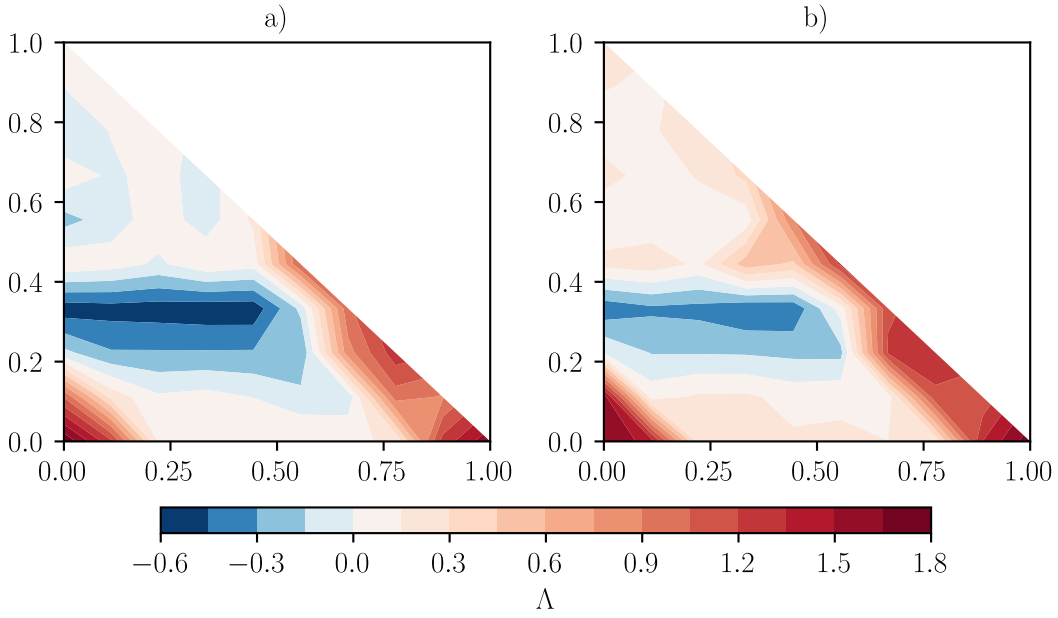


Figure 4.21: a) Lyapunov exponent with respect to α_1-α_2 plane ($\alpha_3 = 1 - \alpha_1 - \alpha_2$), for $\{\mathscr{G}_3/\{G_1, G_2, G_4\}, 8\}$ and $\mu = 8$. b) Zero-shot approximation by an EchONet.

The model trained to a train-and test error of $3.8 \cdot 10^{-8}$. The zero-shot Lyapunov exponent predictions had a mean squared error of 0.052. The model was able to accurately learn the entire parameter space, and was able to predict that certain combinations of the advection basis terms in their respective chaotic regions did not result in chaos.

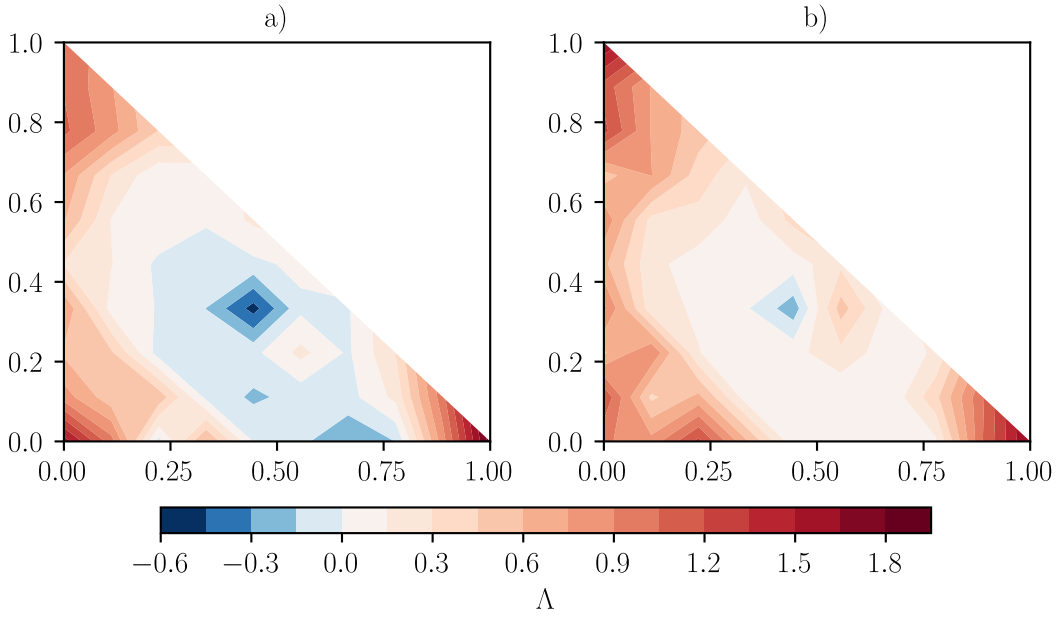


Figure 4.22: a) Lyapunov exponent with respect to α_1-α_2 plane ($\alpha_3 = 1 - \alpha_1 - \alpha_2$), for $\{\mathscr{G}_3/\{G_1, G_2, G_4\}, 10\}$ and $\mu = 8$. b) Zero-shot approximation by an EchONet.

In the case of $d = 10$, the model trained to a train-and test error of $2.8 \cdot 10^{-7}$ and $3.3 \cdot 10^{-7}$. The zero-shot Lyapunov exponent predictions had a mean squared error of 0.071. The model was able to learn the behaviour similarly to the $d = 8$ case, but was unable to accurately predict the central 'diffuse zone'.

The EchONets were able to accurately predict the μ and G dependence of the Lyapunov exponent without knowledge of the timeseries, only relying on sensor data. A problem appears when the prediction times are taken into consideration. The data for plot a) in Fig. 4.21 took $49s$ to generate, meanwhile the approximation in plot b) took $355s$ to generate. The data for plot a) in Fig. 4.22 took $72s$ to generate, and the approximation in plot b) took $978s$ to generate. These jacobians are generated by chains of high-dimensional matrix multiplications, which cannot be reduced in size.

Mixed approach for accelerated in-silico experiments

When exploration of large parameter spaces is the goal, and the parametrization of the systems are fully known, a compromise can be made. To fully benefit from the parallel nature of the EchONet, the Lyapunov exponent predictions can be done with the real jacobian of the system, evaluated at the timeserie points approximated by the EchONet. This mixed approach allows for an exponential speed up with respect to the numerical integration. Sequential numerical integration will be of time complexity $O(nN)$, where N is the amount of timeseries, and n is the length of the timeseries; The EchONet predictions have a time complexity of $O(nN^{\kappa})$, where $0 < \kappa < 1$. This less-than-linear relation guarantees that after some amount of timeseries generations is reached, the EchONet will be faster. Fig. 4.21, and 4.22 with the mixed approach:

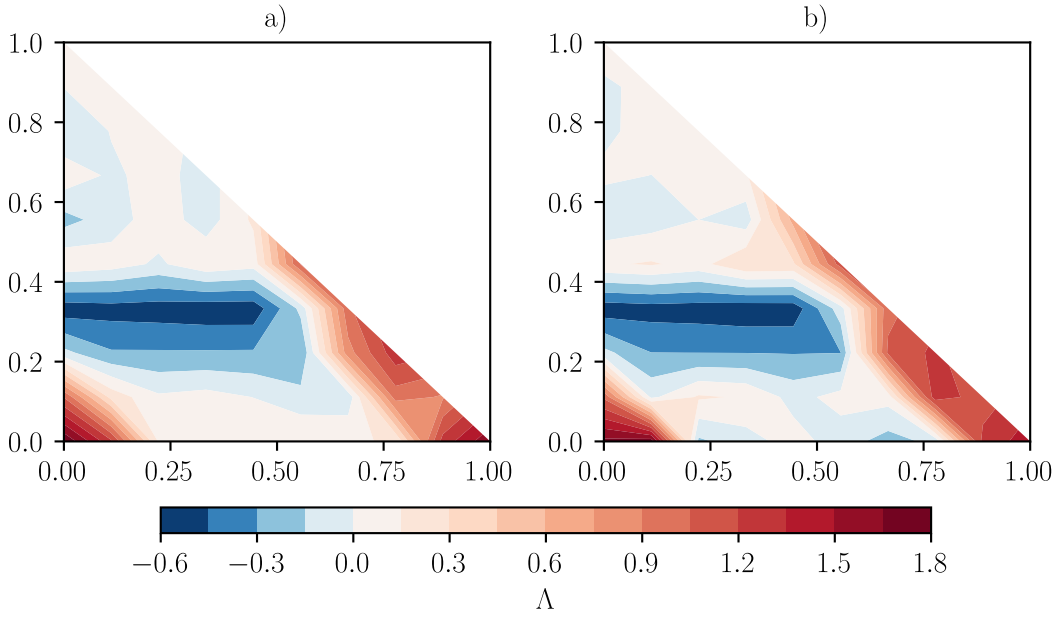


Figure 4.23: a) Taken from Fig. 4.21. b) Lyapunov exponent approximation, mixed approach

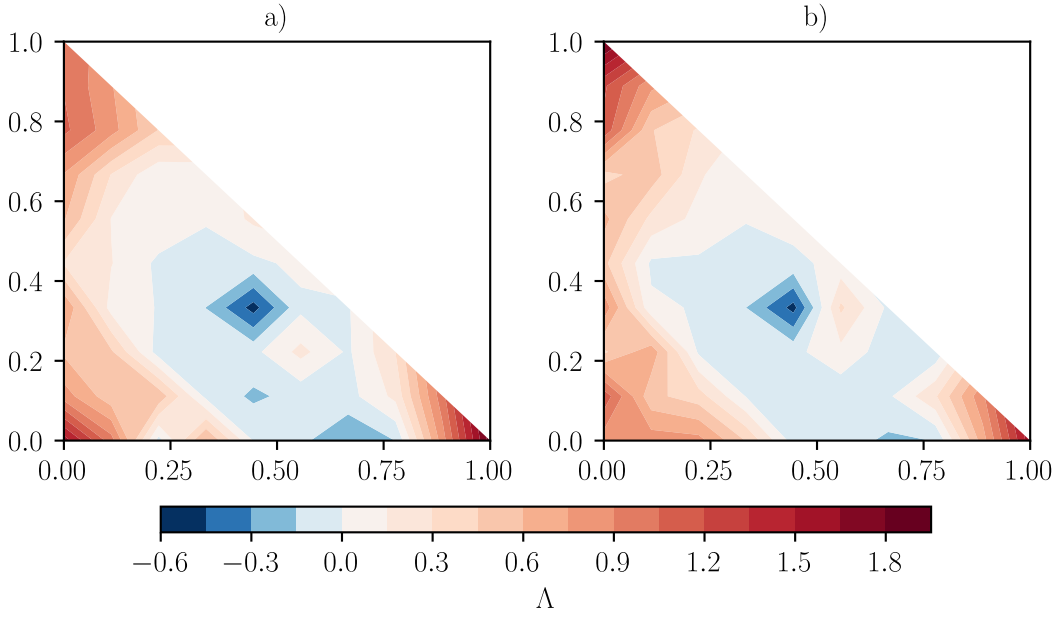


Figure 4.24: a) Taken from Fig. 4.22. b) Lyapunov exponent approximation, mixed approach.

The mixed approach zero-shot Lyapunov exponent predictions had a mean squared error of 0.051 and 0.063, and were therefore more precise than the fully data-driven approach. The Lyapunov exponent prediction times were $10s$ and $47s$. This concludes an exploration of the Echo State Operator Network.

Conclusion

This thesis began by identifying the fundamental limitations of existing approaches to chaotic systems, particularly numerical integration methods and classical machine learning techniques. Traditional numerical methods, such as the Runge-Kutta method, face limitations when applied to chaotic systems because errors propagate exponentially over time, as dictated by the system's Lyapunov exponent. This makes long-term predictions unreliable, even with small time steps and high computational resources. To estimate the reliability of iterative methods, a novel error propagation scheme, based on the definition of the Lyapunov exponent with a perturbation source term was proposed. It was shown that this method was able to correctly identify the time step error dependence of RK4, indicating that there is some validity to this equation. Numerical integration is only applicable when the system is known exactly. When there is missing information, Machine Learning techniques are necessary. Recurrent Neural Networks (RNNs) suffer from inherent drawbacks such as the exploding and vanishing gradient problem, long training times, and an inability to generalize across systems. This inability to generalize to multiple parametrizations of a system is inherent to traditional function approximators. To overcome these challenges, the thesis proposed two novel Neural Operator architectures:

- **ExtremONet**: This architecture generalized the existing DeepONet, a Neural Operator designed to approximate operators in function spaces. By introducing the capability to integrate Linear Regression into its design, ExtremONet improved both the training efficiency and accuracy of predictions for parametrized systems of ODE's. Linear Regression's inclusion reduced the time complexity associated with gradient-based optimization, a process that often requires thousands of iterations to converge. The ExtremONet, therefore, provided a faster alternative without compromising on accuracy, especially for systems where the operator could be well-approximated by smaller DeepONets, as it cannot benefit to the same extent from deeper layers as traditional Neural Networks.

- **EchONet**: Building on the architecture of the ExtremONet, the EchONet introduced the integration of Reservoir Computing (RC) into the Neural Operator form. Reservoir Computing is well-suited for chaotic systems due to its synchronization properties; the reservoir dynamics align with those of the chaotic system, even in the absence of complete knowledge about its governing equations. The EchONet used this synchronization to predict both timeseries data and Lyapunov exponents, the key measure of chaos, without requiring initial timeserie information.

The EchONet was tested rigorously across the parameter space of the Lorenz-63 system, a canonical chaotic system used as a litmus test in non-linear dynamics, and finally a generalized Lorenz-96 system with a large parameter space. The EchONet's ability to decouple the training

and prediction phases further enhanced its efficiency; once trained, the model could generate predictions rapidly and in parallel, unlike traditional numerical methods that are inherently sequential. The zero-shot capabilities are the first of its kind in the field of Reservoir Computing, and therefore warrants further investigation.

5.1 Future prospects

The findings of this thesis open up many new avenues worthwhile to explore.

Other chaotic systems

The EchONet was only tested on the Lorenz-63 and generalized Lorenz-96 systems, which are the most studied chaotic systems of differential equations. There are many more systems which are frequently studied in the same context, namely the the Rössler attractor, and the Hénon map.

Generalized DeepONet

The ExtremONet relied on a generalized form of the DeepONet. It was not investigated whether this reworked operator form is superior without relying on Extreme Learning Machines and only using traditional Deep Neural Networks.

Grid-invariance

Reservoir Computing and the DeepONet are both not grid-invariant methods. Timeserie data input into an RC model has to be equidistantly sampled in time, and sensor data input into the Branch network of a DeepONet has to be collected in the same locations. These restriction might make the EchONet obsolete in most situations. There are grid-invariant version of the DeepONet, most remarkably the RI-DeepONet [41]. This reworked architecture was published when this thesis was in its final stages, and therefore was not tested in the context of the EchONet.

Initial position invariance

The current EchONet formulation was only tested on systems where the initial conditions were identical. Systems with varying initial conditions should still be learnable, but much harder than constant initial conditions. It is hard to hypothesize the impact it will have on the reservoir dimension dependence of the train-and test error.

Partial differential equations

Partial differential equations were not considered in this thesis because detecting Chaos in partial differential equations is a much more laborious and complex task. It has been shown that Reservoir Computing can be used to approximate (chaotic) PDE's such as the Schrödinger equation [42]. Adjusting the EchONet in a similar manner might be fruitful for studying parametrized PDEs.

Multiple compact spaces

The parametrizations of the Lorenz-63 that were studied in this thesis were all singular connected compact spaces. It might be interesting to investigate whether it is possible to extend the input

space to multiple types of disconnected compact spaces, each representing a parametrized chaotic system.

Alternative Echo State Neural Operator

In the appendix, an alternative formulation of the EchONet was defined, which could potentially outperform the standard EchONet because it allowed for a variable spectral radius of the reservoir matrix. The author was unable to find a strategy to use this effect as a benefit, and therefore abandoned this effort. More information can be found in appendix C.3.

Appendices

A.1 ExtremONet tables

	$c_B = 1$	$c_B = 0.5$	$c_B = 0.2$	$c_B = 0.1$	$c_B = 0.05$
$\mathcal{L}_{train}$	$9.2 \cdot 10^{-4}$ $\pm 1.7 \cdot 10^{-4}$	$5.1 \cdot 10^{-3}$ $\pm 7.6 \cdot 10^{-5}$	$2.9 \cdot 10^{-4}$ $\pm 5.9 \cdot 10^{-5}$	$2.3 \cdot 10^{-4}$ $\pm 3.4 \cdot 10^{-5}$	$1.9 \cdot 10^{-4}$ $\pm 3.6 \cdot 10^{-5}$
$\mathcal{L}_{test}$	$1.1 \cdot 10^{-3}$ $\pm 2.2 \cdot 10^{-3}$	$6.3 \cdot 10^{-4}$ $\pm 1.4 \cdot 10^{-3}$	$3.4 \cdot 10^{-4}$ $\pm 8.0 \cdot 10^{-4}$	$3.1 \cdot 10^{-4}$ $\pm 9.0 \cdot 10^{-4}$	$2.3 \cdot 10^{-4}$ $\pm 4.1 \cdot 10^{-4}$

Table 1: ExtremONet test and train loss for different branch sparsities, $\sigma_T = 1$, $\sigma_B = 3$, 1000 nodes.

	$c_B = 1$	$c_B = 0.5$	$c_B = 0.2$	$c_B = 0.1$	$c_B = 0.05$
$\mathcal{L}_{train}$	$1.1 \cdot 10^{-2}$ $\pm 2.0 \cdot 10^{-3}$	$5.7 \cdot 10^{-3}$ $\pm 3.5 \cdot 10^{-3}$	$2.3 \cdot 10^{-3}$ $\pm 7.0 \cdot 10^{-4}$	$2.1 \cdot 10^{-3}$ $\pm 8.3 \cdot 10^{-4}$	$1.8 \cdot 10^{-3}$ $\pm 6.9 \cdot 10^{-3}$
$\mathcal{L}_{test}$	$1.4 \cdot 10^{-2}$ $\pm 2.7 \cdot 10^{-3}$	$7.1 \cdot 10^{-3}$ $\pm 4.2 \cdot 10^{-3}$	$2.9 \cdot 10^{-3}$ $\pm 1.0 \cdot 10^{-3}$	$2.3 \cdot 10^{-3}$ $\pm 8.3 \cdot 10^{-4}$	$2.1 \cdot 10^{-3}$ $\pm 6.9 \cdot 10^{-4}$

Table 2: Random Sampling ExtremONet test and train loss for different branch sparsities, $\sigma_T = 1$, $\sigma_B = 3$, 100 nodes, $q = 10$.

	$\sigma_B = 0.5$	$\sigma_B = 1$	$\sigma_B = 3$	$\sigma_B = 5$	$\sigma_B = 10$
$\mathcal{L}_{train}$	$2.3 \cdot 10^{-3}$ $\pm 2.9 \cdot 10^{-4}$	$7.4 \cdot 10^{-4}$ $\pm 2.7 \cdot 10^{-4}$	$2.0 \cdot 10^{-4}$ $\pm 5.0 \cdot 10^{-5}$	$2.9 \cdot 10^{-4}$ $\pm 2.3 \cdot 10^{-4}$	$9.8 \cdot 10^{-4}$ $\pm 9.4 \cdot 10^{-4}$
$\mathcal{L}_{test}$	$2.6 \cdot 10^{-3}$ $\pm 3.1 \cdot 10^{-4}$	$8.5 \cdot 10^{-4}$ $\pm 2.4 \cdot 10^{-4}$	$2.7 \cdot 10^{-4}$ $\pm 6.0 \cdot 10^{-5}$	$3.5 \cdot 10^{-4}$ $\pm 2.5 \cdot 10^{-4}$	$1.1 \cdot 10^{-3}$ $\pm 1.0 \cdot 10^{-3}$

Table 3: ExtremONet test and train loss for different branch norms, $c_B = 3$, $\sigma_T = 1$, 1000 nodes.

	$\sigma_B = 0.5$	$\sigma_B = 1$	$\sigma_B = 3$	$\sigma_B = 5$	$\sigma_B = 10$
$\mathcal{L}_{train}$	$2.0 \cdot 10^{-3}$ $\pm 1.4 \cdot 10^{-4}$	$6.2 \cdot 10^{-4}$ $\pm 6.0 \cdot 10^{-5}$	$2.3 \cdot 10^{-4}$ $\pm 5.0 \cdot 10^{-5}$	$2.5 \cdot 10^{-4}$ $\pm 8.0 \cdot 10^{-5}$	$7.5 \cdot 10^{-4}$ $\pm 1.7 \cdot 10^{-4}$
$\mathcal{L}_{test}$	$2.3 \cdot 10^{-3}$ $\pm 2.2 \cdot 10^{-4}$	$8.6 \cdot 10^{-4}$ $\pm 1.5 \cdot 10^{-4}$	$3.1 \cdot 10^{-4}$ $\pm 1.1 \cdot 10^{-4}$	$3.0 \cdot 10^{-4}$ $\pm 1.4 \cdot 10^{-4}$	$9.5 \cdot 10^{-4}$ $\pm 2.0 \cdot 10^{-4}$

Table 4: Random Selection ExtremONet test and train loss for different branch norms, $c_B = 0.1$, $\sigma_T = 1$, 100 nodes, $q = 10$.

	$\sigma_T = 0.5$	$\sigma_T = 1$	$\sigma_T = 3$	$\sigma_T = 5$	$\sigma_T = 10$
$\mathcal{L}_{train}$	$3.9 \cdot 10^{-4}$ $\pm 6.0 \cdot 10^{-5}$	$1.9 \cdot 10^{-4}$ $\pm 3.0 \cdot 10^{-5}$	$2.3 \cdot 10^{-4}$ $\pm 9.0 \cdot 10^{-5}$	$3.5 \cdot 10^{-4}$ $\pm 1.0 \cdot 10^{-4}$	$1.2 \cdot 10^{-3}$ $\pm 4.5 \cdot 10^{-4}$
$\mathcal{L}_{test}$	$4.9 \cdot 10^{-4}$ $\pm 1.4 \cdot 10^{-4}$	$2.4 \cdot 10^{-4}$ $\pm 5.0 \cdot 10^{-5}$	$3.0 \cdot 10^{-4}$ $\pm 1.0 \cdot 10^{-4}$	$4.5 \cdot 10^{-4}$ $\pm 1.3 \cdot 10^{-4}$	$1.6 \cdot 10^{-3}$ $\pm 4.5 \cdot 10^{-4}$

Table 5: ExtremONet test and train loss for different branch norms, $c_B = 0.1$, $\sigma_B = 3$, 1000 nodes.

	$\sigma_T = 0.5$	$\sigma_T = 1$	$\sigma_T = 3$	$\sigma_T = 5$	$\sigma_T = 10$
$\mathcal{L}_{train}$	$4.2 \cdot 10^{-4}$ $\pm 1.4 \cdot 10^{-4}$	$2.9 \cdot 10^{-4}$ $\pm 1.7 \cdot 10^{-4}$	$4.0 \cdot 10^{-4}$ $\pm 2.2 \cdot 10^{-4}$	$6.5 \cdot 10^{-4}$ $\pm 4.5 \cdot 10^{-4}$	$1.8 \cdot 10^{-3}$ $\pm 1.1 \cdot 10^{-3}$
$\mathcal{L}_{test}$	$5.3 \cdot 10^{-4}$ $\pm 2.5 \cdot 10^{-4}$	$3.5 \cdot 10^{-4}$ $\pm 2.2 \cdot 10^{-4}$	$5.1 \cdot 10^{-4}$ $\pm 2.6 \cdot 10^{-4}$	$8.2 \cdot 10^{-4}$ $\pm 5.1 \cdot 10^{-4}$	$2.4 \cdot 10^{-3}$ $\pm 1.2 \cdot 10^{-3}$

Table 6: Random Selection ExtremONet test and train loss for different branch norms, $c_B = 0.1$, $\sigma_B = 3$, 100 nodes, $q = 10$.

B.2 Dissipative basis elements of the Generalized Lorenz -96 system, and examples

Lyapunov exponent predictions

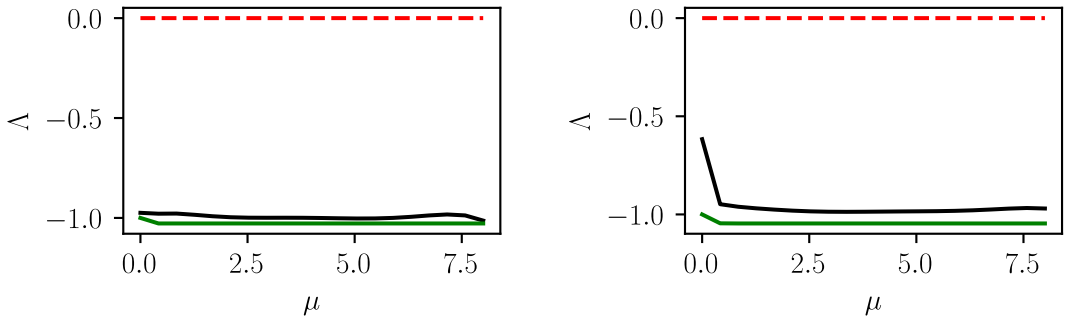


Figure 1: Lyapunov exponent with respect to the forcing constant μ and the zero-shot approximated values by an EchONet for G_1, $d = 4$, and $d = 10$. Green is the true value, black is the predicted value.

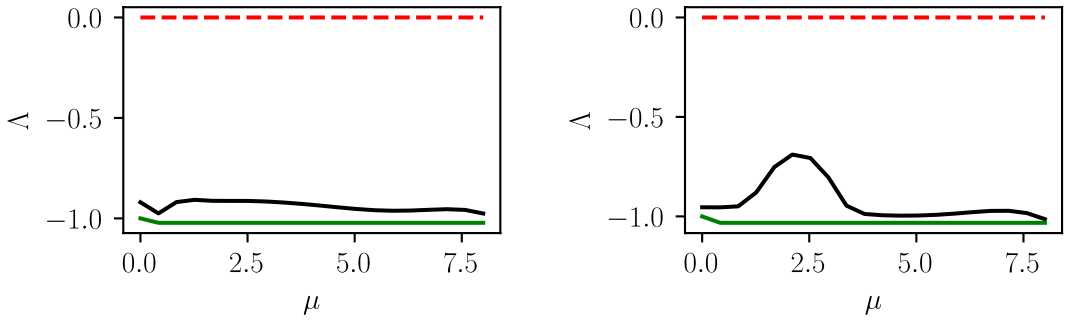


Figure 2: Lyapunov exponent with respect to the forcing constant μ and the zero-shot approximated values by an EchONet for G_2, $d = 6$, and $d = 10$. Green is the true value, black is the predicted value.

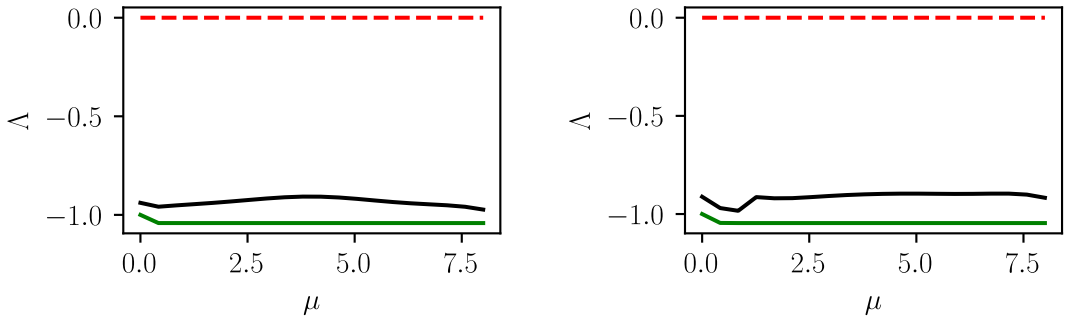


Figure 3: Lyapunov exponent with respect to the forcing constant μ and the zero-shot approximated values by an EchONet for G_4, $d = 8$, and $d = 10$. Green is the true value, black is the predicted value.

Examples

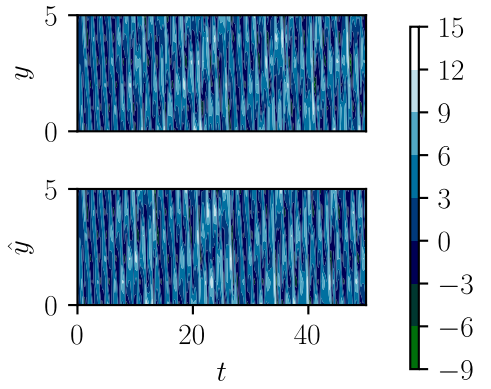


Figure 4: Timeserie example with G_3, $d = 6$, and $\mu = 8$, and its zero-shot approximation by an EchONet.

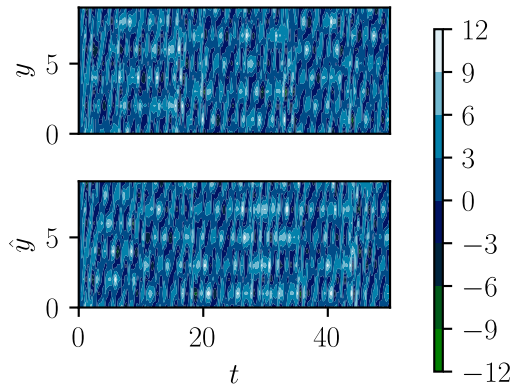


Figure 5: Timeserie example with G_5, $d = 10$, and $\mu = 8$, and its zero-shot approximation by an EchONet.

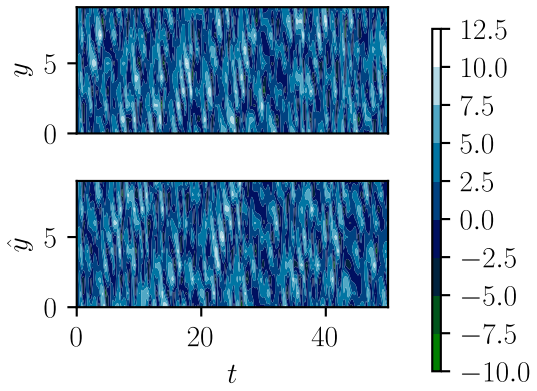


Figure 6: Timeserie example with G_6, $d = 10$, and $\mu = 8$, and its zero-shot approximation by an EchONet.

C.3 An Alternative EchONet

There is a different formulation of the EchONet that was purposefully left out due to some yet unsolved problems. The hidden state of a Reservoir Computing system can also be turned into an operator:

$$\begin{aligned} \mathcal{T}_{i+1}(u(s))(y_i) &= f_T(R\mathcal{T}_i + W^T y_i + b^T) \odot f_B(W^B u(s) + b^B) \\ \hat{y}_{i+1}(u(s))(y_i) &= \theta_0 + \theta_1 \mathcal{T}_{i+1}(u(s))(y_i) \end{aligned} \tag{1}$$

effectively creating an identical model to the EchONet in the prediction phase (without Random Selection), but very different in the training phase. Because the Operator aspect is now integrated into the hidden state, Random Selection cannot be implemented because the dimensions in the hidden state forwarding process have to be identical.

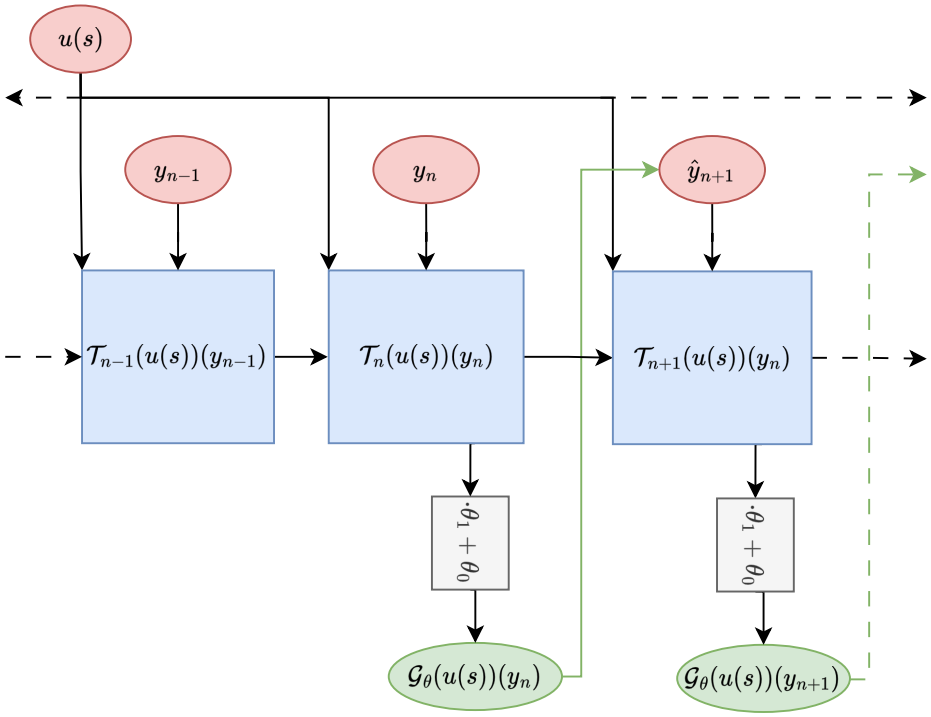


Figure 7: Prediction diagram of the Alternative EchONet

This reformulation of the EchONet now affects the Echo State Property because the Hadamard product with Branch network effectively projects each $\mathcal{T}_n^i \to \mathcal{B}_i \mathcal{T}_n^i$ with $|\mathcal{B}_i| \leq 1$, therefore shrinking the norm of the hidden state

$$\begin{aligned} \|(\mathcal{T}_n^0, \mathcal{T}_n^1, ..., \mathcal{T}_n^k) \odot (\mathcal{B}_0, \mathcal{B}_1, ..., \mathcal{B}_k)\| &= \sqrt{(\mathcal{T}_n^0 \mathcal{B}_0)^2 + (\mathcal{T}_n^1 \mathcal{B}_1)^2 + ... + (\mathcal{T}_n^k \mathcal{B}_k)^2} \\ &\leq \|(\mathcal{T}_n^0, \mathcal{T}_n^1, ..., \mathcal{T}_n^k\|. \end{aligned} \tag{2}$$

This norm reduction is equivalent to altering the spectral radius $\rho(R)$ based on the sensor data. If one writes out a reduced form of the update rule for $\mathcal{T}$ for two iterations without input or bias terms, it can be analysed further:

$$\begin{aligned} \mathcal{T}_{i+1}(u(s))(y_i) &= f_T(R\mathcal{T}_i) \odot \mathcal{B} \\ &= f_T(R(f_T(R\mathcal{T}_{i-1}) \odot \mathcal{B})) \odot \mathcal{B} \\ &= f_T((R \odot \mathcal{B}^t) f_T(R\mathcal{T}_{i-1})) \odot \mathcal{B} \end{aligned} \tag{3}$$

The Reservoir matrix is now altered via $R_i \to R_i' = R_i \odot \mathcal{B}^t$ (R' will be referred to as the effective Reservoir matrix, and $\rho(R')$ the effective spectral radius) which reduces the norm of R, but not always $\rho(R)$. There is an abuse of notation present since $\mathcal{B}$ is a vector. To be more accurate, let us define $\mathcal{B}^k \equiv (\mathcal{B} \otimes^{k-1} \mathcal{B})^t$. An example of this is

$$R = \begin{pmatrix} 0.5 & 0.5 \\ -0.5 & 1 \end{pmatrix} \tag{4}$$

$$\mathcal{B}^k = \begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix} \tag{5}$$

Which results in $\rho(R) \approx 0.866$ and $\rho(R \odot \mathcal{B}^k) \approx 1.15$, and to be sure $\|R\| \approx 1.33$. This example extends to any dimension $k \geq 2$ by padding R with zero-valued rows and columns, and padding $\mathcal{B}^k$ such that $\mathcal{B}^k_i = 0 \ \forall i > 2$. In general, for any matrix A, where $AA^t \neq A^tA$ (A and A^t do not commute). The inequality $\rho(A) < \|A\|$ always holds [43]. The matrix A is referred to as a non-normal matrix. This shows that the Echo State Property might not be guaranteed if R is scaled via $R \to \rho R/\rho(R)$ and should instead be scaled via $R \to \rho R/\|R\|$.

There is a subtle possible benefit over the standard EchONet. From Reservoir Computing theory, we know that more memory capacity can be beneficial for less complex systems, which in turn would benefit from a larger spectral radius $\rho(R)$. The opposite is also true, complexer systems benefit from a smaller spectral radius. If one is able to create an inverse correlation between the complexity of the system and the effective spectral radius for each timeserie instance input into the alternative EchONet, then this alternative EchONet could outperform the standard formulation. One can assume that the complexity of the timeseries generated by some parametrized differential equation is directly correlated to the complexity of the governing equation, which can be measured with the sensor data $u(s)$.

The author was not able to create this inverse correlation. Many problems arose in the development process. Defining some arbitrary complexity of a function is not trivial, nor is creating such an inverse correlation because any optimization process involving the spectral radius in turn requires the repeated calculation of the eigenvalues of a very large matrix (which is too slow to reasonably attempt). A more analytical approach will be necessary. A first step could be exploring non-stochastic upper-triangular matrices. These matrices, by Schur's unitary triangularization theorem [44], are always non-normal. The requirements to create a worthwhile alternative EchONet are confirming that upper-triangular reservoir matrices are performant, and find some functional structure for $\mathcal{B}$ such that it shrinks or grows the spectral radius of R' depending on the complexity of the sensor data $u(s)$ whilst functioning as a Branch network and generating a upper-triangular matrix R' when the Hadamard product is taken with R.

Another option is to avoid the effective spectral radius issue by using the simple-cycle reservoir matrix, for which all $\rho(R) = \rho(R \odot \mathcal{B}^k)$. This allows the model to be completely equivalent to the standard EchONet, which could be tested in a future project.

Bibliography

[1] C. Oestreicher, 'A history of chaos theory,' en, *Dialogues in Clinical Neuroscience*, vol. 9, no. 3, pp. 279–289, Sep. 2007, ISSN: 1958-5969. DOI: 10.31887/DCNS.2007.9.3/coestreicher. [Online]. Available: https://www.tandfonline.com/doi/full/10.31887/DCNS.2007.9.3/coestreicher (visited on 22/12/2024).

[2] E. N. Lorenz, 'Deterministic Nonperiodic Flow,' en, in *The Theory of Chaotic Attractors*, B. R. Hunt, T.-Y. Li, J. A. Kennedy and H. E. Nusse, Eds., New York, NY: Springer New York, 2004, pp. 25–36, ISBN: 978-1-4419-2330-1 978-0-387-21830-4. DOI: 10.1007/978-0-387-21830-4_2. [Online]. Available: http://link.springer.com/10.1007/978-0-387-21830-4_2 (visited on 22/12/2024).

[3] R. Sapolsky, *21. Chaos and Reductionism*, Stanford University, 2011.

[4] Z. Yang, X. Zeng, Y. Zhao and R. Chen, 'AlphaFold2 and its applications in the fields of biology and medicine,' en, *Signal Transduction and Targeted Therapy*, vol. 8, no. 1, p. 115, Mar. 2023, ISSN: 2059-3635. DOI: 10.1038/s41392-023-01381-z. [Online]. Available: https://www.nature.com/articles/s41392-023-01381-z (visited on 22/12/2024).

[5] *The Nobel Prize in Physics 2024*, Oct. 2024. [Online]. Available: https://www.nobelprize.org/prizes/physics/2024/press-release/.

[6] Schrauwen, Benjamin and Verstraeten, David and Van Campenhout, Jan, 'An overview of reservoir computing: Theory, applications and implementations,' eng, in *Proceedings of the 15th European Symposium on Artificial Neural Networks. p. 471-482 2007*, 2007, pp. 471–482. [Online]. Available: %7Bhttp://doi.org/1854/11063%7D.

[7] M. T. Augustine, 'A SURVEY ON UNIVERSAL APPROXIMATION THEOREMS,' Indian Institute of Technology Bombay, India, Tech. Rep., 2024. [Online]. Available: https://arxiv.org/pdf/2407.12895.

[8] Tianping Chen and Hong Chen, 'Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems,' *IEEE Transactions on Neural Networks*, vol. 6, no. 4, pp. 911–917, Jul. 1995, ISSN: 10459227. DOI: 10.1109/72.392253. [Online]. Available: http://ieeexplore.ieee.org/document/392253/ (visited on 22/12/2024).

[9] L. Lu, P. Jin, G. Pang, Z. Zhang and G. E. Karniadakis, 'Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators,' en, *Nature Machine Intelligence*, vol. 3, no. 3, pp. 218–229, Mar. 2021, ISSN: 2522-5839. DOI: 10.1038/s42256-021-00302-5. [Online]. Available: https://www.nature.com/articles/s42256-021-00302-5 (visited on 22/12/2024).

[10] K. McCabe, *D-Day landings and the most crucial weather forecast in history*, Jun. 2024. [Online]. Available: https://www.rmets.org/metmatters/d-day-landings-and-most-crucial-weather-forecast-history.

[11] S. H. Strogatz, *Nonlinear dynamics and chaos*. Westview Press, 2015.

[12] B. Delmas, 'Pierre-François Verhulst et la loi logistique de la population,' *Mathématiques et sciences humaines*, no. 167, Sep. 2004, ISSN: 0987-6936, 1950-6821. DOI: 10.4000/msh.2893. [Online]. Available: http://journals.openedition.org/msh/2893 (visited on 22/12/2024).

[13] S. Filip, 'Notes on the multiplicative ergodic theorem,' en, *Ergodic Theory and Dynamical Systems*, vol. 39, no. 5, pp. 1153–1189, May 2019, ISSN: 0143-3857, 1469-4417. DOI: 10.1017/etds.2017.68. [Online]. Available: https://www.cambridge.org/core/product/identifier/S0143385717000682/type/journal_article (visited on 22/12/2024).

[14] Y. C. Li, 'Chaos in Partial Differential Equations,' 2009, Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.0909.0910. [Online]. Available: https://arxiv.org/abs/0909.0910 (visited on 22/12/2024).

[15] R. A. EDSON, J. E. BUNDER, T. W. MATTNER and A. J. ROBERTS, 'LYAPUNOV EXPONENTS OF THE KURAMOTO–SIVASHINSKY PDE,' *The ANZIAM Journal*, vol. 61, no. 3, pp. 270–285, 2019. DOI: 10.1017/S1446181119000105.

[16] M. Sandri, 'Numerical Calculations of Lyapunov Exponents,' University of Verona, Italy, Tech. Rep. [Online]. Available: https://venturi.soe.ucsc.edu/sites/default/files/Numerical_Calculation_of_Lyapunov_Exponents.pdf.

[17] T. M. Janaki and R. Govindan, 'Lyapunov Exponents for Continuous-Time Dynamical Systems,' Department of Mathematics, Indian Institute of Science, Bangalore , India, Tech. Rep., 2003.

[18] S. V., 'Estimation of Error in Runge-Kutta Fourth Order Method,' *International Journal of Emerging Technologies and Innovative Research (www.jetir.org)*, vol. 5, no. 2, pp. 1590–1596, Feb. 2018, ISSN: 2349-5162. [Online]. Available: https://www.jetir.org/papers/JETIR1802301.pdf.

[19] A. collaboration, *ATLAS seeks out unusual signatures of long-lived particles*, CERN, 2022. [Online]. Available: https://home.cern/news/news/physics/atlas-seeks-out-unusual-signatures-long-lived-particles.

[20] Z.-H. Zhou, *Machine Learning*, en. Singapore: Springer Singapore, 2021, ISBN: 9789811519666 9789811519673. DOI: 10.1007/978-981-15-1967-3. [Online]. Available: https://link.springer.com/10.1007/978-981-15-1967-3 (visited on 22/12/2024).

[21] W. N. van Wieringen, *Lecture notes on ridge regression*, Version Number: 8, 2015. DOI: 10.48550/ARXIV.1509.09169. [Online]. Available: https://arxiv.org/abs/1509.09169 (visited on 22/12/2024).

[22] J. Szabados and P. Vértesi, *Interpolation of Functions*, en. WORLD SCIENTIFIC, Oct. 1990, ISBN: 978-9971-5-0915-6 978-981-4335-84-3. DOI: 10.1142/0861. [Online]. Available: https://www.worldscientific.com/worldscibooks/10.1142/0861 (visited on 22/12/2024).

[23] K. Hornik, M. Stinchcombe and H. White, 'Multilayer feedforward networks are universal approximators,' en, *Neural Networks*, vol. 2, no. 5, pp. 359–366, Jan. 1989, ISSN: 08936080. DOI: 10.1016/0893-6080(89)90020-8. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0893608089900208 (visited on 22/12/2024).

[24] G.-B. Huang, Q.-Y. Zhu and C.-K. Siew, 'Extreme learning machine: Theory and applications,' en, *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, Dec. 2006, ISSN: 09252312. DOI: 10.1016/j.neucom.2005.12.126. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0925231206000385 (visited on 22/12/2024).

[25] I. Loshchilov and F. Hutter, *Decoupled Weight Decay Regularization*, Version Number: 3, 2017. DOI: 10.48550/ARXIV.1711.05101. [Online]. Available: https://arxiv.org/abs/1711.05101 (visited on 22/12/2024).

[26] Y. Bahri, *DDPS — "A first-principles approach to understanding deep learning"*. [Online]. Available: https://www.youtube.com/watch?v=7UQNHPAAaNk&t=3140s.

[27] H. Ramsauer, B. Schäfl, J. Lehner *et al.*, *Hopfield Networks is All You Need*, Version Number: 3, 2020. DOI: 10.48550/ARXIV.2008.02217. [Online]. Available: https://arxiv.org/abs/2008.02217 (visited on 22/12/2024).

[28] J. L. Elman, 'Finding Structure in Time,' en, *Cognitive Science*, vol. 14, no. 2, pp. 179–211, Mar. 1990, ISSN: 0364-0213, 1551-6709. DOI: 10.1207/s15516709cog1402_1. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1207/s15516709cog1402_1 (visited on 22/12/2024).

[29] R. Pascanu, T. Mikolov and Y. Bengio, 'On the difficulty of training recurrent neural networks,' in *Proceedings of the 30th International Conference on Machine Learning*, S. Dasgupta and D. McAllester, Eds., ser. Proceedings of Machine Learning Research, Issue: 3, vol. 28, Atlanta, Georgia, USA: PMLR, Jun. 2013, pp. 1310–1318. [Online]. Available: https://proceedings.mlr.press/v28/pascanu13.html.

[30] H. Jaeger, 'The "echo state" approach to analysing and training recurrent neural networks – with an Erratum note1,' *Fraunhofer Institute for Autonomous Intelligent Systems*, Jan. 2010. [Online]. Available: https://www.ai.rug.nl/minds/uploads/EchoStatesTechRep.pdf.

[31] S. Boccaletti, J. Kurths, G. Osipov, D. Valladares and C. Zhou, 'The synchronization of chaotic systems,' en, *Physics Reports*, vol. 366, no. 1-2, pp. 1–101, Aug. 2002, ISSN: 03701573. DOI: 10.1016/S0370-1573(02)00137-0. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0370157302001370 (visited on 22/12/2024).

[32] S. Shahi, F. H. Fenton and E. M. Cherry, 'Prediction of chaotic time series using recurrent neural networks and reservoir computing techniques: A comparative study,' en, *Machine Learning with Applications*, vol. 8, p. 100 300, Jun. 2022, ISSN: 26668270. DOI: 10.1016/j.mlwa.2022.100300. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2666827022000275 (visited on 22/12/2024).

[33] D. J. Gauthier, E. Bollt, A. Griffith and W. A. S. Barbosa, 'Next generation reservoir computing,' en, *Nature Communications*, vol. 12, no. 1, p. 5564, Sep. 2021, ISSN: 2041-1723. DOI: 10.1038/s41467-021-25801-2. [Online]. Available: https://www.nature.com/articles/s41467-021-25801-2 (visited on 22/12/2024).

[34] T. L. Carroll, 'Optimizing memory in reservoir computers,' en, *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 32, no. 2, p. 023 123, Feb. 2022, ISSN: 1054-1500, 1089-7682. DOI: 10.1063/5.0078151. [Online]. Available: https://pubs.aip.org/cha/article/32/2/023123/2835760/Optimizing-memory-in-reservoir-computers (visited on 22/12/2024).

[35] M. Inubushi and K. Yoshimura, 'Reservoir Computing Beyond Memory-Nonlinearity Trade-off,' en, *Scientific Reports*, vol. 7, no. 1, p. 10 199, Aug. 2017, ISSN: 2045-2322. DOI: 10.1038/s41598-017-10257-6. [Online]. Available: https://www.nature.com/articles/s41598-017-10257-6 (visited on 22/12/2024).

[36] B. Li, R. S. Fong and P. Tiňo, 'Simple Cycle Reservoirs are Universal,' 2023, Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.2308.10793. [Online]. Available: https://arxiv.org/abs/2308.10793 (visited on 22/12/2024).

[37] A. Rodan and P. Tino, 'Minimum Complexity Echo State Network,' *IEEE Transactions on Neural Networks*, vol. 22, no. 1, pp. 131–144, Jan. 2011, ISSN: 1045-9227, 1941-0093. DOI: 10.1109/TNN.2010.2089641. [Online]. Available: http://ieeexplore.ieee.org/document/5629375/ (visited on 22/12/2024).

[38] J. Kerin and H. Engler, 'On the Lorenz '96 model and some generalizations,' *Discrete & Continuous Dynamical Systems - B*, vol. 27, no. 2, p. 769, 2022, ISSN: 1531-3492, 1553-524X. DOI: 10.3934/dcdsb.2021064. [Online]. Available: https://www.aimsciences.org/article/doi/10.3934/dcdsb.2021064 (visited on 26/12/2024).

[39] J. H. E. Cartwright, M. Feingold and O. Piro, 'An Introduction to Chaotic Advection,' en, in *Mixing*, H. Chaté, E. Villermaux and J.-M. Chomaz, Eds., vol. 373, Series Title: NATO ASI Series, Boston, MA: Springer US, 1999, pp. 307–342, ISBN: 978-1-4613-7127-4 978-1-4615-4697-9. DOI: 10.1007/978-1-4615-4697-9_13. [Online]. Available: http://link.springer.com/10.1007/978-1-4615-4697-9_13 (visited on 27/12/2024).

[40] C. Fefferman, S. Mitter and H. Narayanan, 'Testing the manifold hypothesis,' en, *Journal of the American Mathematical Society*, vol. 29, no. 4, pp. 983–1049, Feb. 2016, ISSN: 0894-0347, 1088-6834. DOI: 10.1090/jams/852. [Online]. Available: https://www.ams.org/jams/2016-29-04/S0894-0347-2016-00852-4/ (visited on 28/12/2024).

[41] B. Bahmani, S. Goswami, I. G. Kevrekidis and M. D. Shields, *A Resolution Independent Neural Operator*, Version Number: 3, 2024. DOI: 10.48550/ARXIV.2407.13010. [Online]. Available: https://arxiv.org/abs/2407.13010 (visited on 23/12/2024).

[42] L. Domingo, J. Borondo and F. Borondo, 'Adapting reservoir computing to solve the Schrödinger equation,' en, *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 32, no. 6, p. 063 111, Jun. 2022, ISSN: 1054-1500, 1089-7682. DOI: 10.1063/5.0087785. [Online]. Available: https://pubs.aip.org/cha/article/32/6/063111/2835823/Adapting-reservoir-computing-to-solve-the (visited on 22/12/2024).

[43] L. N. Trefethen, 'Spectra and Pseudospectra: The Behaviour of Non-Normal Matrices and Operators,' en, in *The Graduate Student's Guide to Numerical Analysis '98*, M. Ainsworth, J. Levesley and M. Marletta, Eds., vol. 26, Series Title: Springer Series in Computational Mathematics, Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 217–250, ISBN: 978-3-642-08503-1 978-3-662-03972-4. DOI: 10.1007/978-3-662-03972-4_6. [Online]. Available: http://link.springer.com/10.1007/978-3-662-03972-4_6 (visited on 22/12/2024).

[44] K. D. Ikramov, 'The canonical Schur form of a matrix with simple eigenvalues,' en, *Doklady Mathematics*, vol. 77, no. 3, pp. 359–360, Jun. 2008, ISSN: 1064-5624, 1531-8362. DOI: 10.1134/S1064562408030101. [Online]. Available: http://link.springer.com/10.1134/S1064562408030101 (visited on 22/12/2024).