# The Scalable Spatial Echo State Network

## *For Detection of Anomalous Ocean Surface Topography*

**Master's Thesis in Computational Physics**

by Jacob Ungar Felding

August 13, 2021

**Supervisor**

Associate Professor James E. Avery

**University of Copenhagen**

Niels Bohr Institute

## Acknowledgements

During the trying times of a pandemic, lockdowns and working from home, the people that have helped and encouraged me during the creation of this thesis have my most sincere appreciation.

My supervisor, Associate Professor James Avery, in particular has my gratitude for being within reach, digitally, and ready to discuss all aspects of my work – literally day and night – undeterred by any weekends or holidays. I have greatly enjoyed learning from you, and sharing the occasional beer.

For listening to my complaints at trying times and for many joyful breaks I want to thank the people of *eScience* at the Niels Bohr Institute, and my office mates throughout the last year. In addition, Carl Johnsen has provided me with much guidance, and the computational resources to perform my experiments; oftentimes at the cost of rendering his machine *unusable* to himself. At eScience, Rasmus Munk has been forever helpful in sorting out a multitude of technical issues for me, and discussing approaches to high performance computing. David Marchant has provided a lot of pedagogical feedback and stylistic tips for the writing of this thesis.
To Markus Jochum and the people of *Team Ocean* at NBI, I am grateful for the facilitation of CESM simulation data used in this thesis (as has Avery).

The thing that I most look forward to after the final sprint of writing this thesis is spending some time with my friends and family who have supported my personal journey in more ways than can meaningfully be expressed on paper; I am *deeply* grateful to you all.

## Abstract

Global climate model simulations like the CESM can inform scientists, decision makers and the public of the likely future climate of planet Earth. The Sixth Assessment Report of the IPCC, for example, is the latest landmark evaluation of the scientific consensus wrt. climate change, and many of its conclusions are drawn from analysis of climate model simulations.

While *climate* refers to long-term weather trends that can often be examined by statistical measures, the simulations are necessarily much more fine grained, with time scales of days rather than decades. The implication is that simulations are, to a large extent, *unexplored*; ripe with information on the evolution of e.g. ocean currents that is deeply impactful to populations in the near future at regional levels. While such analysis by oceanographers is possible, the immensity of the simulation data prohibit manual evaluation at scale. It is clear that automatic or semi-automatic methods of detecting anomalous ocean transitions are necessary.

In a step towards such anomaly detection, this thesis presents a machine learning method for *high-resolution spatio-temporal* forecasting by feeding simulation data with no exogenous information of the underlying physics – except for the tuning of a few hyperparameters.
The proposed method is a type of Recurrent Neural Network called an Echo State Network (ESN) that is *efficient* to train, and *effective* at forecasting. The work builds on and extends *'Adaptive Anomaly Detection in Chaotic Time Series with a Spatially Aware Echo State Network'* (Heim and Avery 2019). Their *spatial ESN* (SESN) prototype showed great promise at spatio-temporal prediction, but technical limitations inhibited exploration of oceans at grand scales that is necessary for automatic analysis of ocean simulations.

The main contributions that make the SESN spatially scalable are (1) the construction of high-dimensional reservoirs by avoiding diagonalization, limited only by memory, (2) the implementation of an efficient *spatially sensitive* loss function with the introduction of robust forward-backward transforms for regression, and (3) the implementation of dimensionality reduction that stabilizes optimization and combats over-fitting.

The SSESN implementation and the spatial loss functions are made available as Python 3 packages at GitHub[1,2].

As shown in the report, the SSESN provides forecasting on both climate model simulations and simpler calibration systems. The thesis also exemplifies the SSESN's application for anomaly detection in ocean systems, whereas a robust and fully automated approach is left as further work.

[1] SSESN Implementation: *https://github.com/jfelding/esn/*
[2] Image Euclidean Distance Functions: *https://github.com/jfelding/IMED*

# Notation

## *Scalars, Vectors and Matrices*

$a$ A scalar

$\mathbf{a}$ A vector, or row-major vectorized matrix $\text{vec}(\mathbf{A})$

$\mathbf{A}$ A matrix

## *Indices*

I chose to *preserve* the font when indexing multi-element structures regardless of whether the result is a scalar or array:

$a_i$ $i^{\text{th}}$ scalar sample

$\mathbf{a}_i$ Single (scalar) element of a vector $\mathbf{a}$, or the $i^{\text{th}}$ vector sample from a vector space $\mathbf{a}_i \in \mathcal{A}$

$\mathbf{A}_{(i,:)}$ Vector from $i$th row of matrix $\mathbf{A}$

$\mathbf{A}_{(i,j)}$ Scalar element of matrix $\mathbf{A}$ found in its $i$th row and $j$th column

## *Variables*

| | |
|---|---|
| $\mathcal{X}$ | Input space with $\mathbf{x}_t \in \mathcal{X}$ |
| $\mathcal{Y}$ | Output space with $\mathbf{d}_t \in \mathcal{Y}$ and $\mathbf{y}_t \in \mathcal{Y}$ |
| $\mathcal{D}$ | Training set containing inputs and targets such that $(\mathbf{x}_t, \mathbf{d}_t) \in \mathcal{D} \subset (\mathcal{X} \times \mathcal{Y})$ |
| $\mathcal{L}_2$ | Function space of the $\mathcal{L}_2$ norm, and all square-integrable functions |
| $\mathbf{x}_t$ | Input at time $t$ – Dimension $N_{\text{input}}$ |
| $\mathbf{y}_t$ | Output at time $t$ – Dimension $N_{\text{output}}$ |
| $\mathbf{d}_t$ | Target at time $t$ |
| $\mathbf{h}_t$ | Hidden state vector at time $t$ with dimension $N_{\text{hidden}}$ |
| $N_{\text{input}}$ | Dimensionality of input sequence at time $t$, flatted if larger than 1D. |
| $N_{\text{output}}$ | Output dimensionality. In this report typically equal to $N_{\text{input}}$ |
| $N_{\text{hidden}}$ | Dimensionality of high-dimensional hidden state vector |
| $N_{\text{train}}$ | Number of harvested hidden states that are regressed upon |
| $N_{\text{trans}}$ | Number of initial training set observations reserved for initialization of $\mathbf{h}_{t=0}$ |
| $N_{\text{pred}}$ | Number of free-running prediction steps, post-training |
| $\mathbf{W}_{\text{ih}}$ | Matrix representation of input maps, initial transform of inputs to the hidden state dimension (ih: input-to-hidden) |
| $\mathbf{W}_{\text{hh}}$ | The reservoir, a matrix representing a linear map $w : N_{\text{hidden}} \to N_{\text{hidden}}$ (hh: hidden-to-hidden) |
| $\mathbf{W}_{\text{ho}}$ | Readout matrix, optimized using multiple least squares (ho: hidden-to-output) |
| $\mathbf{H}$ | Hidden state matrix consisting of 'harvested' states $\mathbf{h}_t$, with resulting dimensions $(N_{\text{train}} \times N_{\text{hidden}})$ |
| $\ell(\mathbf{y}_t, \mathbf{d}_t)$ | Loss Function applied to output and target |
| $\hat{\mathscr{L}}(h, \mathcal{D})$ | Empirical loss of the selected hypothesis $h$ on the entire training set $\mathcal{D}$. |
| $\mathscr{L}(h)$ | Expected loss, generalization error, or *out-of-sample* error using the loss metric $\mathscr{L}(h) = \mathbb{E}[\ell(h(x), d)]$ |
| $\mathrm{d}(\cdot, \cdot)$ | Euclidean distance following from discretization of the $\mathcal{L}_2$ norm |

## Abbreviations

| | |
|---|---|
| BPTT | Back-Propagation Through Time |
| CESM | Community Earth System Model |
| CNN | Convolutional Neural Network |
| CPU | Central Processing Unit |
| CSR | Compressed Sparse Row matrix format |
| DCT | Discrete Cosine Transform |
| DFT | Discrete Fourier Transform |
| ESN | Echo State Network |
| ESP | Echo State Property |
| FFT | Fast Fourier Transform |
| GPU | Graphics Processing Unit |
| IMED | Image Euclidean Distance |
| iST | Inverse Standardizing Transforms of the IMED. This thesis introduces robust frequency representation approaches to the iST. |
| LSTM | Long Short-Term Memory network |
| MA | Moving Average |
| ML | Machine Learning |
| MSE | Mean Squared Error |
| PCA | Principal Component Analysis |
| RNN | Recurrent Neural Networks |
| SESN | Spatial Echo State Network (Heim and Avery 2019) |
| SSESN | Scalable SESN (this project) |
| SSH | Sea Surface Height (CESM variable) |
| ST | Standardizing Transform of the IMED |
| SVD | Singular Value Decomposition |

# Contents

# Chapter 1
# *Introduction*

---

This thesis develops a machine learning (ML) method to predict chaotic spatio-temporal series governed by differential equations of physics. The project builds on work by Niklas Heim and James Avery, and implements solutions to improve and stabilize predictions. Most importantly, my work allows exploration of datasets with much larger spatial dimensions than before. This was previously unfeasible due to both time and memory constraints.

With this accomplishment, realistic spatio-temporal predictions of chaotic systems, like ocean currents, are achievable and very fast even without exploiting accelerators like GPUs.

As a use case in computational physics, I show that simple methods for *anomaly detection* can be built on top of the ML-predictions to discover areas in the oceans where *large-scale topographical changes occur* over different time scales, and in a seemingly unpredictable manner.

## 1.1 Spatio-Temporal Learning

The main subject of this thesis is the development of a machine that produces spatio-temporal predictions of high quality and with high spatial resolution. The task is to take an image sequence as input, and make *good* estimates of what will happen *after* the sequence has ended, for a number of frames. In this thesis the image sequences considered are governed by continuos differential equations describing e.g. the laws of physics. When the sequence obeys such mathematical or physical constraints that difficult task becomes feasible in principle. This thesis is limited to approximating dynamics of such spatio-temporal series.

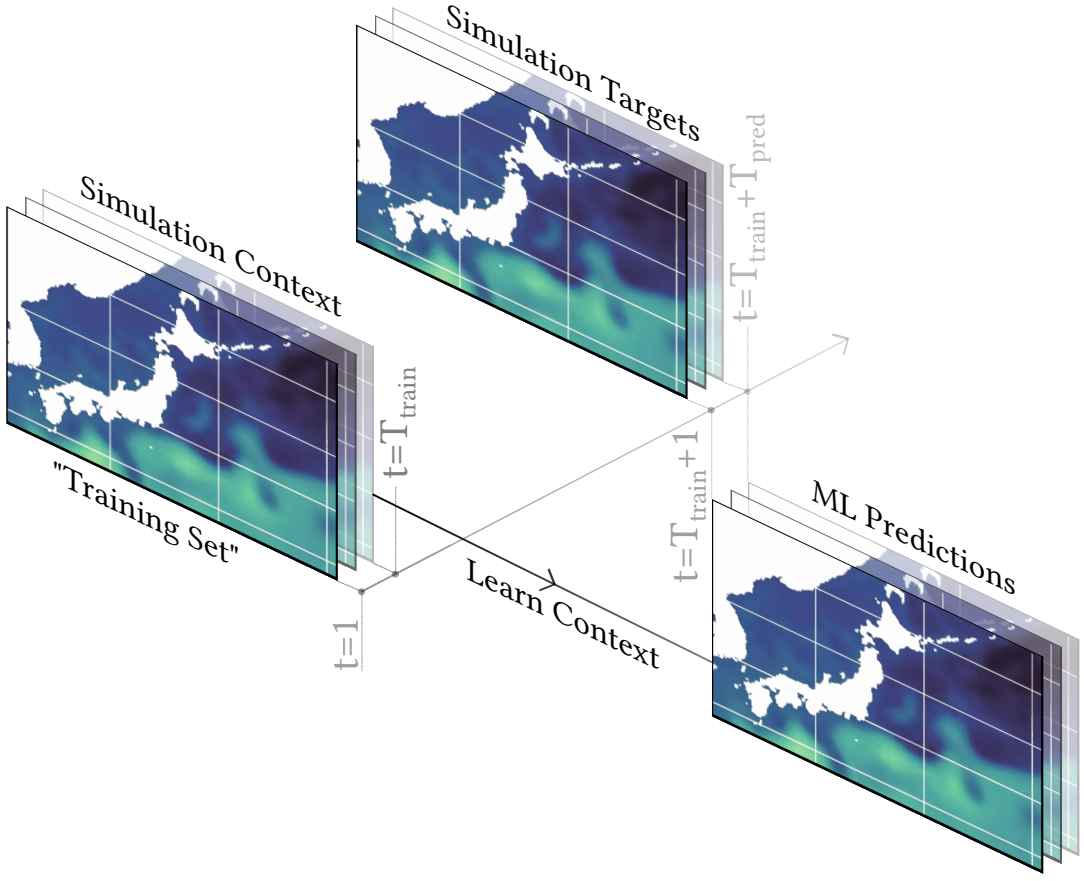



Figure 1.1: The image sequence prediction task. The available sequence is split into *training* and *target* subsets. The *training set* is available to accomplish the task using *supervised* learning approaches. The target set is unseen by the learner, and may only be used to evaluate the quality of the learner's predictions (in a simple learning setting). Note: For prediction of image sequences, the 'target' for input of time $t$ is the frame at time $t + 1$.

**Gradient-Based and Reservoir Computing Approaches**

It has recently become popular to apply so-called *deep learning* for video-frame prediction (Oprea et al. 2020), which typically involves heavy *gradient-based optimization methods* that necessitate GPU acceleration for high-dimensional time series like spatio-temporal ones. I take another path laid out by the work of then-master's student Niklas Heim and our common master's project supervisor, James Avery. Heim's excellent thesis is available on GitHub, see (Heim 2018). More recently and concisely their work was presented in an article preprint, see (Heim and Avery 2019) with a major code revision available at (Heim 2021).

Their work, contrary to the current zeitgeist, is based on the realization that recurrent neural networks (RNNs) can be extremely hard to optimize due to time constraints, but also wrt. prediction quality *because* they often fail. RNNs are the class of machine learning predictors most commonly applied for sequential prediction tasks; further described in Chaper 2, with a more solid foundation available in (Hammer 2000).

The work of Heim and that of this thesis therefore utilize a specific type of RNN called *echo state network* (ESN), which takes a *reservoir computing* approach to sequential modelling for which optimization is very fast, but requires understanding of the RNN dynamics to function properly.

The spatial ESN prototype implemented by Heim and Avery allows *spatio-temporal* prediction within reasonable time for spatial dimension around 30x30 and is limited by memory consumption around $120 \times 120$ (See benchmarks in Sec. 7.8), but the resource and time consumption scales poorly with the spatial resolution and makes predictions on high-resolution data infeasible.

Conventional ESNs, which are not generally used for spatially correlated data, are also described in Chapter 2, and the main contribution of my thesis (in most chapters) to the prediction of image sequences is the development of such ESNs for very high-dimensional data, and especially space and time correlated data for which I further develop a distance metric function called the Image Euclidean Distance (IMED) that is more sensitive to correlation along dimensions, and less sensitive to small pertubations, and white noise. This work is described in Chapter 6.

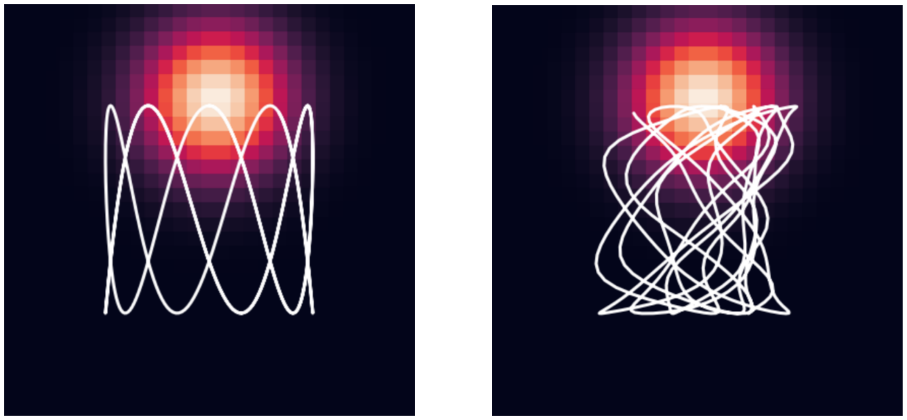
As the aim of the thesis project is mainly the creation of an ESN for high-resolution input,

I refer to my particular ESN as a **Scalable Spatial Echo State Network (SSESN)**.

## 1.2 Testing of the SSESN

While developing my SSESN method of image sequence prediction for high-resolution sequences, it is important to ensure that the approach is capable of approximating spatially coherent structures (often more so than minimizing an error measure, which can be done in many ways).

The project therefore started by testing ideas for the SSESN on artificially created sequences of spatial 'orbs' that move in either a deterministic or chaotic manner, as illustrated in Figure 1.2. These *synthetic data* are generated by (Heim and Avery 2019). The individual pixels of a periodically moving orb (left) is relatively easy to predict, while an orb whose path is governed by the Mackey-Glass equations is much harder to get right and spatially coherent (see Sec. 7.4.1). Even if one allows the predicted trajectory to deviate from the true path, and merely values *qualitatively* plausible predictions. At the same time, the speed of the chaotic orb is faster, for an increased prediction challenge.



(a) Orb with periodic trajectory

(b) Orb with Mackey-Glass trajectory

Figure 1.2: Illustration of two different sequences of moving orbs with Gaussian spread. During the sequences, orbs remain spatially cohesive. in (a), the orb takes a simpler, periodic path. In (b), the orb takes a path governed by chaotic behaviour of Mackey-Glass equations. Illustrations are from (Heim and Avery 2019) who also created the synthetic data and figures that are shown with permission.

## 1.3 Predicting Dynamics of the Kuroshio, and Ocean Applications

Of course, the artificial *orbs* are only used for testing and calibration purposes, and to benchmark the SSESN on data of different dimensions, see Chapter 7.

Later, as the SSESN method of performing spatio-temporal prediction began showing promise and feasible spatial scalability the application moved to data that are more interesting physically.

With competent guidance from oceanographers and computational physicists at Team Ocean led by Prof. Markus Jochum of the Niels Bohr Institute [1] the project applies simulation data of the Community Earth System Model (CESM) to the SSESN; continuing the work of (Heim and Avery 2019). The simulations were performed by Team Ocean. The *University Corporation for Atmospheric Research* provide their own introduction to the CESM:

> CESM is a fully-coupled, community, global climate model that provides state-of-the-art computer simulations of the Earth's past, present, and future climate states (UCAR 2021)

The CESM ocean simulations are thus a system for the numerical integration of differentials equations that govern the oceans (and its interplay with the atmosphere). Unlike the ML model that I develop it is deeply 'aware' of the physics involved in ocean dynamics. To such an extent that the model can *reproduce* some ocean phenomena that are known to occur, but are not yet well-understood or predicted. An infamous example that Team Ocean has directed our attention to is the **state transition of the Kuroshio Current** at the coast of Japan.

#### Kuroshio: The Black Tide

The Kuroshio (kuro (黒): *black* + shio (潮): *tide*) is notorious for its bimodal and sudden, lately more uncommon, transitions that have significant consequences to the residents of Japan. In Figure 1.3 (from Sugimoto et al. 2021) the typical paths of the flow are shown in each of its two states: the large meander (low sea level) and non-large meander (high sea level) states. It has been in the low state since 2017, which can be assigned blame for causing a *local greenhouse effect* and increasing temperatures that

[1] Team Ocean can be visited at: `www.gfy.ku.dk/~nuterman/teamocean/index.html`

doubling the number of discomfort days experienced in the Kanto Region (Sugimoto et al. 2021). The transition also gives rise to relocations of marine life habitats and increased snowfall with consequences for e.g. fishing and the economy.
Further description of the bimodality and tabulations of Kuroshio transitions in recent history are available in Kawabe 1985.

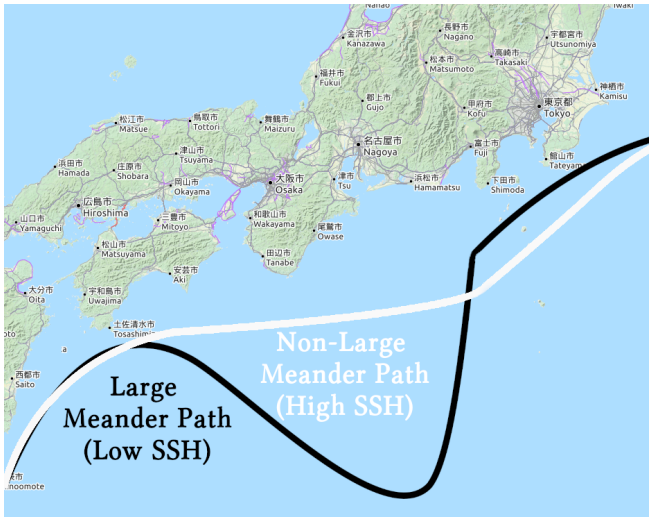



Figure 1.3: The path typically taken by the Kuroshio current of Japan, in each state. Since 2017, Kuroshio has taken the LM path, the first transition since a shift in 2004-2005. Figure heavily inspired by (Sugimoto et al. 2021). Map: ©OpenStreetMap contributors.

Team Ocean have allocated significant resources towards performing the CESM simulations:

> The computational costs of the 0.1° model made it difficult to integrate the high resolution simulations closer to their equilibrium solution given our resources. The total sum of 42 high resolution model years required more than a year to complete on 4096 cores of the BlueGene supercomputer located in Jülich, Germany, and exhausted our resources (Poulsen et al. 2018)

The simulation data include many variables such as sea surface height, temperature, salinity and velocity in a number of sea depths, and is defined on a (0.1°) 3600x2400 spatial grid (a single grid cell, on the surface, is about $10^4\,\mathrm{m} \times 10^4\,\mathrm{m}$ depending on the latitude and longitude). In this thesis we only explore what can be achieved by applying the SSESN to a *single variable*, the sea surface height (SSH), and prediction the future of this variable, as illustrated in Fig. 1.1. There is definitely much further work to do!

The Kuroshio transition is *known to occur* (though it is not easily predictable) as the shift has implications for especially residents of Japan. Because it is known and reproducible in simulations, it is a good starting point for achieving something much greater: **Detecting unknown transitions in the ocean, and predicting them with advance notice**.
Team Ocean tells us that many such transitions likely occur that remain *unknown* to mankind. The transitions may be similar to Kuroshio while others can take place at different scales – of space and time.

## 1.4 Anomaly Detection

The greater purpose of detection anomalous transitions in ocean currents (first Kuroshio, then elsewhere) is a major motivation for obtaining a scalable spatial ESN. Time consumed on developing the latter has sadly limited the time for implementation of the former to only *a week*. Nonetheless, *anomaly detection* by means of the SSESN requires a slightly deeper description, which is mainly provided in Chapter 8. For now, I present the approach in this thesis and its motivation.

#### What is an anomaly?

To detect an anomaly, one often defines *what is normal* to start out. That task requires expertise in the domain in question (expertise that I admittedly do not have, as I have no prior experience with geophysics).
In the context of *ocean anomalies*, or perhaps specified further as *topographical transitions of currents*, one has to consider the *complexity* of oceans.

#### Ocean Turbulence

In accordance with the definition put out by Strogatz (Strogatz 2015, p. 331), oceans current are chaotic (and therefore non-linear) systems:

- Small perturbations of the system can have significant influence in the short and long term,
- The currents are *aperiodic*. While they have strong components of *trend* and *seasonality*, many phenomena such as the behaviour of Kuroshio cannot be described by the large scale components,

- Deviations arising from small changes in initial conditions arise despite the determinism of the currents. Oceans are well-described in the realm of classical and continuum mechanics.
- Turbulence exists at all scales of the ocean, and especially large-scale turbulence like *mesoscale eddies* (10km-100km), the "ocean equivalents of storms" (Ferrari 2021), can significantly influence the dynamics of currents.

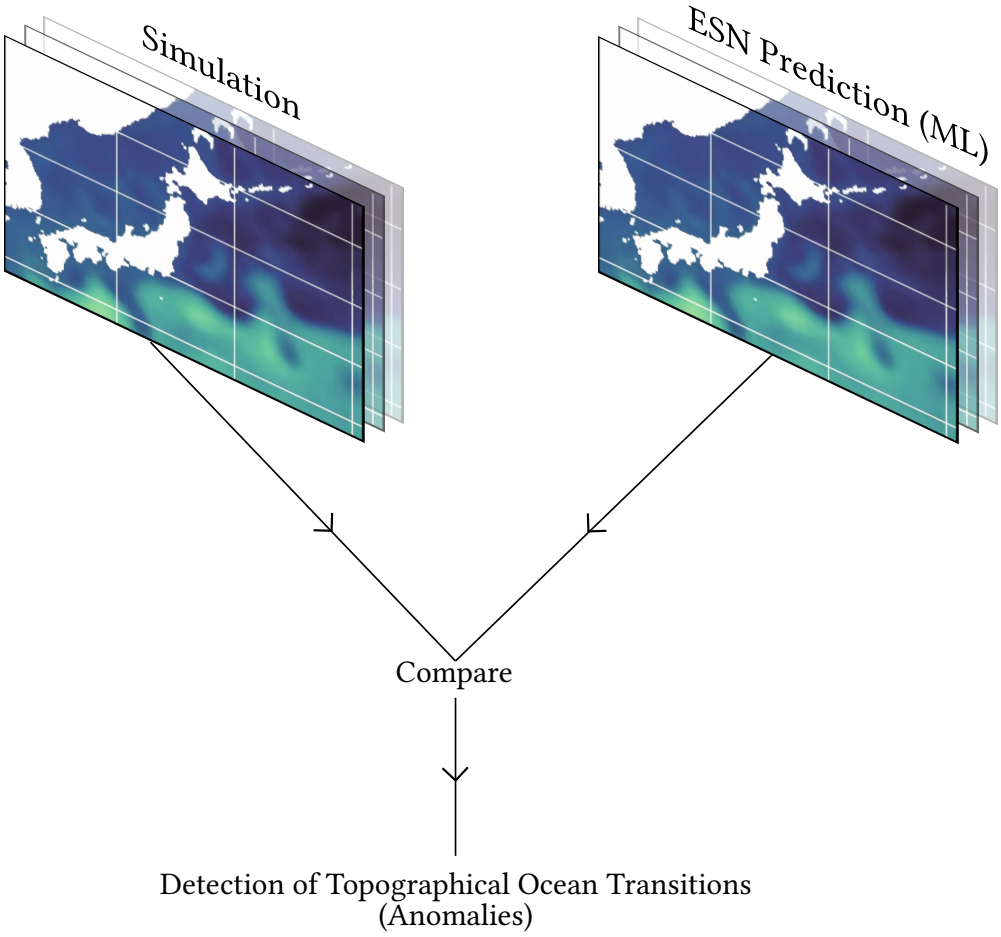



Figure 1.4: Outline of approach to detection of topographical transition in oceans using ML predictions based on simulation output of the CESM model, and a comparison of the simulation data itself. Details of the approach are visualized in Figure 8.1.

### A Workaround to Definition Issues

The question arises therefore: **How do we define normality** for a vast, diverse dynamic non-linear system? Is it possible to define normality in a way that applies to *all* large currents, and not just to specific currents or similar currents (Kuroshio, like the Gulf Stream, is a *western boundary current*)?

The approach of this project is to *avoid* defining normality from direct *physical traits* of each system altogether. Instead, we try to *approximate* the dynamics of the system with a good, but *imperfect* predictor, and **define anomalies as those dynamics that significantly deviate from the short-term estimates of the predictor**. Anomalies, per definition, are phenomena that are *hard to predict* from the previous dynamics (of the training set).

The task is, therefore, to construct the *predictor* and the *detector* in a manner that allows us to *distinguish* small-scale turbulence, numerical noise and the exponentially increasing uncertainty inherent to chaotic systems from large scale transition of currents like Kuroshio. The approach is illustrated in Figure 1.4.

## 1.5 *Why Bother* with High-Resolution Predictions?

The resolution limitations of the spatial ESN prior to this work limits what the predictor can learn about a region from the contextual information *around that region*. Scaling up the resolution at which predictions can take place, then, allows better predictions to be made.

Further, the spatial ESN is to be applied for anomaly detection across the immense data sets with many variables, large regions, and for many time steps. To search the vast *simulated* oceans (or real SSH data), it needs to be *fast* to be practically useful.

In future works, the spatial ESN should apply not only *one variable*, but several to optimize predictions further. For that to be possible, the spatial ESN must *also* be optimized. These are the drawbacks of the current stat of the SESN. In (Heim and Avery 2019) it was necessary to downscale the system, or examine only small areas, to allow high-quality prediction. The method showed promise, and was capable at detecting a known Kuroshio simulation anomaly, but was not ready for any automatic exploration of ocean simulation data.

The proposed methods of this thesis allow the short-term prediction of the CESM SSH variable at grand scales, and possibly for the entire CESM map *at once*, with increased memory.

Note, however, that the work in this thesis *keeps* a major contribution of Heim and Avery, which is the replacement of a linear mapping of the spatial input to the *hidden state space* – conventionally represented as a matrix – with efficient functions that provide spatially

sensitive representations of the input. These *input maps* remain in the SSESN.

## 1.6 Contributions and Reading Instructions

The contributions of this thesis to the field of anomaly detection, echo-state networks and video-frame or image prediction are the following:

1. Allowing extremely **high-dimensional ESN reservoirs** to be constructed with adjustable dynamics by replacing *diagonalization* methods with an empirical eigenvalue *circular law* for certain sparse random matrices, see Chapter 4,

2. The implementation and further development of a simple, but **spatially sensitive loss function** of general utility for volumes with any number of dimensions, while reducing the influence of numerical noise in predictions, see Chapter 6, and GitHub[2],

3. Introduction of robust methods for the **inverse transform** inherent to the loss function for the application in *regression* as opposed to classification only, in Ch. 6.

4. The implementation of dimensionality reduction in the SSESN readout to allow optimization at scale with a lower number of covariates, and to stabilize predictions, see Ch. 5. This leads to *linear scalability* as shown in Sec. 7.8.4,

5. The implementation of the SSESN itself in a simpler, optimized and extendible version available at GitHub[3],

Additionally, I include background material to the understanding of machine learning theory, recurrent neural networks and conventional echo state networks in Chapters 2-3, and demonstrate the capabilities of the SSESN in Chapter 7-8 while comparing spatial scalabilities of the SSESN and SESN.

Supplementary material like animations and hyperparameter configurations are available in a separate branch of the SSESN repository[4].

I hope you enjoy the following chapters, and I welcome you to the world of *spatial Echo State Networks*!

[2]IMED Repository: `https://github.com/jfelding/IMED`
[3]SSESN Repository: `https://github.com/jfelding/esn`
[4]Assets for the thesis: `https://github.com/jfelding/esn/tree/thesis_assets`

# Part I

# Background

# Chapter 2

# *Supervised Machine Learning and RNNs*

---

This section serves only as background information to the development of the echo state network, and the information theory that underlies machine learning methods. Machine learning is a wide field with a plethora of classifiers and regressors or predictors. Machine Learning models obtain their functionality by approximating underlying functions that are learnt inductively from examples – the more, the better as long as sampling is i.i.d. This chapter will introduce both the basic objectives and pitfalls of machine learning methods as well as fundamentals of Echo-State Networks that this thesis will further develop for spatio-temporal problems.

## 2.1 Many Applications, Yet No Free Lunch

Machine Learning (ML) – often more or less interchangeably referred to as *statistical learning*, and *deep* learning – is now used in all realms of society where 'big data' are available. Financial institutions apply ML for risk analysis, advertisers target customers with material they deem 'relevant' to us, a digital assistant can recognize your voice and respond appropriately, particle physicists at CERN can identify subatomic particles in jets of quarks and gluons, and self-driving cars continue to confront our legislators with ethical issues of the image recognition and robotics technology.

Indeed, as digitalization prevails over its analogue counterparts, machine learning is becoming increasingly widespread whether we like it or not.

While machine learning has triumphed at many tasks, no one algorithm trained on general data will perform better than every other. That is, there is no one 'universal' artificial intelligence that can defeat all others at all tasks at once. This is known as the *no free lunch* theorem (Goodfellow et al. 2016), and directs our attention to creating algorithm that work well on problems with specific statistical characteristics.

## 2.2 Objectives of Supervised Machine Learning

### *2.2.1 Introduction*

Supervised machine learning is the subset of learning methods that rely on a training *set* of samples $\mathcal{D} = \{(x_1, d_1), (x_2, d_2), \ldots, (x_{Ntrain}, d_{Ntrain})\}$ in which an input example $x_i \in \mathcal{X}$ is paired with a *true* output $d_i \in \mathcal{Y}$ with $\mathcal{X}$ and $\mathcal{Y}$ being the sets of possible inputs and outputs, respectively (Abu-Mostafa et al. 2012). The notation above does not imply that $x_i, d_i$ are scalar, but that their properties are unspecified.

In classification problems, $d_i$ is referred to as a *label* and is taken from a particular output set inherent to the problem. In digit recognition, labels may be $d_i \in \{0, 1, 2, \ldots\}$ whereas another problem may have possible labels {'cat', 'dog', 'car',...}.

In *regression problems* – **the subject of this thesis** – $d_i$ is instead referred to as the *target* of the input example $x_i$, and the output set $\mathcal{Y}$ may be the space of real numbers $\mathbb{R}$, or indeed a higher-dimensional space like $\mathbb{R}^n$.

The supervised machine learning algorithm seeks an approximation $h$ of the *unknown*

target function $f : \mathcal{X} \rightarrow \mathcal{Y}$ where $h$ is picked from a hypothesis set $\mathcal{H}$ that contains different possible mappings.

### 2.2.2 Induction and the Task of Learning

The training set $\mathcal{D}$ contains a finite number of samples of $f$, the *unknown* function mapping the input space to its output. This provides the learning algorithm with information on $f$ for the observations in the training set $\mathcal{D}$, but no information as to its behaviour *outside* $\mathcal{D}$. Further, many possible examples of hypotheses $h \in \mathcal{H}$ may equally well approximate the samples of $f$ in $\mathcal{D}$, and in some cases classification may be *flawless* on $\mathcal{D}$ ($k$-nearest neighbours with $k = 1$).

*Learning*, is *not* determining $h$ such that $\mathcal{D}$ is approximated, even if perfectly. Instead, it is the approximation of $f$ *outside* of the sample $\mathcal{D}$ that is relevant to learning. Whether *learning* is feasible, however, is determined by whether it is possible to determine $h$ that approximates $f$ *outside* the training set, $\mathcal{D}$.

Learning or *inference*, therefore, is closely related to the *problem of induction*. Famously (*Black Swan Theory* 2021), Roman poet Juvenal in his works used the term 'black swan' to describe a presumed non-existence. Centuries later, Europeans indeed *did* discover them in Australia, and the inductive reasoning that such creatures did not exist was proven wrong, as documented in Fig. 2.1.

Statistical inference is faced with the same issue of whether outliers of data outside the training set may exist that should drastically alter the hypothesis. It turns out, however, that in a probabilistic sense, inductive inference *can* be made under certain conditions.

### 2.2.3 Probabilistic Inference

The reason public polling, for instance, works (within uncertainties and sources of error) although the approach is *inductive* is that it is *random sample* of the population that on average tends to agree with the characteristics of the population (an unbiased estimate, under ideal conditions).

Mathematical theorems have been put forward in this spirit to *bound* the probability of a random variables of a particular hypothesis $h$ having a value in a specified interval. These classic bounding methods include the *Markov, Chebyshev and Hoeffding Inequali-*

*ties*. Under the assumption that $\mathcal{D}$ contains samples that are **independent and identically distributed** (i.i.d.) these inequalities prove that it becomes increasingly unlikely that the random variable deviates a selected amount from the expected as the number of random samples are increased. The key here is that the samples are i.i.d. for the bounds to provably work, unlike samples of swans in medieval Europe, of course, with respect to the global probability distribution of swans.

Figure 2.1: Black swans do exist! Machine learning uses inductive reasoning, but its pitfalls are mitigated by probabilistic inference. Image credit: Kuiphuis 2018.

### 2.2.4 Hypothesis Selection and Assessment

Probabilistic inference mitigates the glaring issues of inductive reasoning with respect to a specific hypothesis. Learning also involves selection of a certain hypothesis $h \in \mathcal{H}$ over others. For this purpose, I introduce the notion of a *loss function*, a measure of how wrong the prediction $h(x_i)$ is wrt. the target $d_i$. The empirical loss or in-sample error is:

$$\hat{\mathscr{L}}(h, \mathcal{D}) = \frac{1}{|\mathscr{D}|} \sum_{x_i, d_i \in \mathscr{D}} \ell(h(x_i), d_i) \tag{2.1}$$

where the output of the hypothesis $h$ is $y_i = h(x_i)$, and $\ell(h(x_i), d_i)$ is a loss function e.g. $\ell(y_i, d_i) = (y_i - d_i)^2$ to evaluate the deviations of the prediction and target.

The objective of machine learning is to minimize *another* measure known as the out-of-sample error, generalization error or *expected error* on *unseen* data sampled from the same distribution as the training set:

$$\mathscr{L}(h) = \mathbb{E}[\ell(h(x), d)] \tag{2.2}$$

The quantity above can be estimated and bounded, but not observed. Low generalization error implies successful inference of the learning algorithm on unseen data from the same

unknown distribution. This is what we want: **Being able to predict $d$ from $x$ reliably when new data is sampled**.

The *training error* $\hat{\mathscr{L}}(h, \mathcal{D})$ is *not* an unbiased estimate of generalization error $\mathscr{L}(h)$. $\mathcal{D}$ was used to teach the hypothesis, and $h$ is therefore specifically adapted to minimize $\hat{\mathscr{L}}(h, \mathcal{D})$, but if care is not taken, this may come at the expense of larger $\mathscr{L}(h)$. This is known as *over-fitting*, and is a significant obstacle to machine learning. Increasing the *model complexity* of $h$ will often decrease the training error, but increase the generalization error, as I will show in Section 2.2.5.

Say we have chosen two hypotheses $h_1$ and $h_2$ as candidates for the best predictor $h^*$. The machine learning algorithm and optimization methods may provide relatively low training errors. This does not tell us which hypothesis to pick as over-fitting may have incurred, and certainly does not indicate the generalization performance. So, we decide to bring in new data sampled from the same distribution. We call this set $\mathcal{D}_{\text{val}}$ the *validation set*, as evaluating the predictors' errors on it can help us determine *which is likely the best option* in terms of minimizing $\mathscr{L}(h)$. Crucially, $\mathcal{D}_{\text{val}}$ is not used for any training, only validation of a number of hypotheses found using training.

So, we evaluate the validation error $\hat{\mathscr{L}}(h, \mathcal{D}_{\text{val}})$ of $h_1$ and $h_2$, and proceed with the one that minimizes this error, and on average results in the lowest $\mathcal{L}(h)$ of the two.

Still, we have not answered how to estimate $\mathscr{L}(h)$. We once again may ask if $\hat{\mathscr{L}}(h, \mathcal{D}_{\text{val}})$ of the best hypothesis is an unbiased estimate of $\mathscr{L}(h)$. With the approach described above, the answer is *it is not*: By using $\mathcal{D}_{\text{val}}$ to *select* the best hypothesis out of more than one such, we have *biased* $\hat{\mathscr{L}}(h^*, \mathcal{D}_{\text{val}})$ because the selected hypothesis $h^*$ depends on $\mathcal{D}_{\text{val}}$. The validation loss cannot be expected to lead to the same value of $\mathcal{L}(h)$.

If we *do* need to *honestly* tell our supervisor or employer what error they should expect when they try out the selected hypothesis on their own data for prediction purposes, we need to bring in even more new data $\mathcal{D}_{\text{test}}$, which we refer to as the *test set*. Finally, we may evaluate the *test error* $\hat{\mathscr{L}}(h, \mathcal{D}_{\text{test}})$, which is otherwise unused, and this test error *is* an unbiased estimator of the generalization loss $\mathscr{L}(h)$.

It is a counter-intuitive point (Seldin 2021) that $\hat{\mathscr{L}}(h_1, \mathcal{D}_{\text{val}})$ is an unbiased estimate of $\mathscr{L}(h_1)$ if we do not also evaluate $\hat{\mathscr{L}}(h_2, \mathcal{D}_{\text{val}})$. And conversely. But as soon as we have used $\mathcal{D}_{\text{val}}$ to pick the *best* hypothesis $h^*$, the estimate has become tainted!

What we *call* the data sets is irrelevant. If we use $\mathcal{D}_{\text{test}}$ to pick $h^*$ using $\hat{\mathscr{L}}(h^*, \mathcal{D}_{\text{test}})$ (which seems to be the case too often), this is no longer an unbiased estimate of $\mathscr{L}(h^*)$ (and we should refer to it as a *validation set* to avoid confusion).

### 2.2.5 Bias-Variance Tradeoff

I have stated without proof that increasing the model complexity may reduce the in-sample error, but tends to increase the generalization error. This is deeply relevant to the subject of this thesis, since a major change in the approach of this thesis is to *restrict the complexity* using dimension reduction methods, see Section 5.1.

If we consider linear regression hypothesis $h_{LS}(x)$, which is applied in this thesis, one can show that the **expected generalization error** can be decomposed (Hastie et al. 2017, p. 223-224) into three terms. We assume that the target is $d = f(x) + \epsilon$ where $\epsilon$ is a white noise term such that $\mathbb{E}[\epsilon] = 0$, $\text{Var}(\epsilon) = \sigma_\epsilon^2$, and $f$ is the true unknown function generating samples.
A model with $p$ parameter will have the following decomposition of the generalization error, where $N$ is the number of samples.

$$\frac{1}{N}\sum_{i=1}^{N}\mathscr{L}(h_{\text{LS}}(x_i)) = \sigma_\epsilon^2 + \frac{1}{N}\sum_{i=1}^{N}(f(x_i) - \mathbb{E}[h_{\text{LS}}(x_i)])^2 + \frac{p}{N}\sigma_\epsilon^2 \tag{2.3}$$

And we see, directly, that the *variance increases with the number of parameters* $p$. We refer to **over-fitting** as the case where *variance is too high*, and **under-fitting** as the case where *bias* is (the second term). It is *central* to avoid either for a least squares fit to generalize well to unseen data. The expression above, still, cannot be evaluated as the true function $f$ is unknown. We also see, that one can never expect to do better than the variance of the targets, $\sigma^2_{epsilon}$, the first term.

### 2.2.6 Probabilistic Inference for Sequential Models

The subject of this thesis is supervised learning for sequential, time-dependent models (RNNs, ESNs). Every sample is therefore *not* independent and identically distributed, and many results from information theory of supervised learning do not hold in this case.
Instead, the i.i.d. condition must be considered for the *time series themselves* rather than

the individual observations that they consist of. Probabilistic induction, thus, holds in these cases (Hammer 2000, p. 53). Besides this reference, I refer the interested reader to (Cesa-Bianchi et al. 2009 (Print: 2006)). This thesis, however, is not on informational theoretical research into such models, but their developments, and I will refrain from delving further into these aspects.

## 2.3 Recurrent Neural Networks

*Recurrent Neural Network* (RNN) can refer to a number of predictors in machine learning that have been widely used to model data with *temporal dependencies* (time series). The class of predictors include *Long Short-Term Memory* (LSTM) and *Echo State Networks*, the latter of which this thesis will extend for use on very high-dimensional time series, like image sequence input. It is pertinent to introduce the broader class of RNNs such that the benefits and drawbacks of ESNs may come to light.
Recurrent Neural Networks have as their basis an **input-driven dynamical system** that *recurrently* refers to its earlier state. A very general description of such system is (Goodfellow et al. 2016, p. 370):

$$\mathbf{h}_t = f\left(\mathbf{h}_{t-1}, \mathbf{x}_t; \theta_{\text{hh}}\right), \tag{2.4}$$

such that $\mathbf{h}_t$ stores the *state* of the system at time $t$; a function $f$ of its previous state $\mathbf{h}_{t-1}$, driving input $\mathbf{x}_t$ at time $t$ and some parameters $\theta_{\text{hh}}$ (hh: hidden-to-hidden). The state, denoted with $h$ due it being called the *hidden state* (hidden, in-between input and output).

To **produce predictions**, a *readout function* $g$ can be defined that extracts information contained in the state $\mathbf{h}_t$ and optionally the input $\mathbf{x}_t$ directly to produce some output $\mathbf{y}_t$:

$$\mathbf{y}_t = g\left(\mathbf{h}_t, \mathbf{x}_t; \theta_{\text{ho}}\right), \tag{2.5}$$

using another set of parameters $\theta_{\text{ho}}$ (ho: hidden-to-output).
In most RNNs, the approach is to **optimize the parameters** included in $\theta_{\text{hh}}, \theta_{\text{ho}}$ in a manner that provides *good prediction output*.
Crucially, **this is not the case for ESNs** in which $\theta_{\text{hh}}$ is more or less *randomly initialized*, and only weights in $\theta_{ho}$ are trained. We do, however, tame the randomness with certain

*properties* that allow ESNs to learn better *without* the explicit optimization of *hidden parameters*.

### 2.3.1 Unfolding the RNN

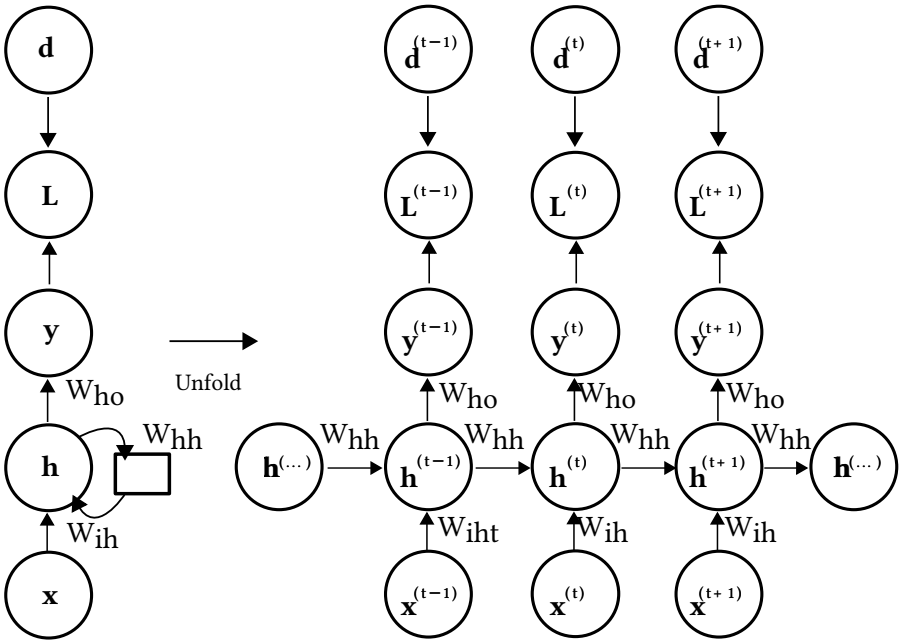



Figure 2.2: Unfolding the RNN graph. Completely inspired by Goodfellow et al. 2016, p. 369, adapted to match notation used in this document. $\mathbf{x}$ is the input sequence, $\mathbf{h}$ the hidden state, $\mathbf{y}$ the output after readout, $\mathbf{d}$ the target output, and $L$ the empirical loss of the output and target given some loss function.

A recurrent neural network applied to a *finite-length data sequence* defined by a recurrent dynamical system like Eq. (2.4) can be regarded as a very **deep feed-forward network**. Due to its recurrence, i.e. the dependency of $\mathbf{h}_t$ on $\mathbf{h}_{t-1}$, the network can be *unfolded* by replacing $\mathbf{h}_{t-1}$ with its recurrent definition $\mathbf{h}_{t-1} = f(\mathbf{h}_{t-2}, \mathbf{x}_{t-1}; \theta)$ repeatedly. This way, we see that the current hidden state $\mathbf{h}_t$ of the system is a function of all previous input (Goodfellow et al. 2016, pp. 370-371):

$$\begin{aligned}\mathbf{h}_t &= h^t(\mathbf{x}_t, \mathbf{x}_{t-1}, ..., \mathbf{x}_1) \\ &= f(\mathbf{h}_{t-1}, \mathbf{x}_t; \theta),\end{aligned}$$

where $h^t$ is a function of all previous input function at time $t$. Unfolding is illustrated in Figure 2.2. This explains why recurrent neural networks are commonly used for time series analysis: they have **memory** since the state contains selected information about (potentially) all past input. The memory property of RNNs is not unlimited, of course, as

the state $\mathbf{h}_t$ retains its dimension between time steps as opposed to expanding whenever new information is made available by processing input $\mathbf{x}_t$.

Further, $\theta_{\text{hh}}$ is *fixed* (after possible optimization) for all time steps, which is a choice that relies on the assumption of *stationarity* of the input time series, i.e. that it is independent of the time index, $t$, while correlation between $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ is of course to be expected. A drawback is that $\theta_{\text{hh}}$ is difficult to optimize exactly because it contains parameters that are shared across all time step. This step is **avoided** by ESNs by simply choosing *only* to optimize output parameters $\theta_{ho}$.

### 2.3.2 Forward Propagation

A standard RNN, including the dynamical system and separate output or *readout* layer is defined by the following equations that spell out explicitly a choice of the functions $f$ and $g$:

$$\mathbf{h}_t = \sigma\left(\mathbf{W}_{\text{ih}}\mathbf{x}_t + \mathbf{W}_{\text{hh}}\mathbf{h}_{t-1} + \mathbf{b}_{\text{h}}\right) \tag{2.6}$$

$$\mathbf{y}_t = \mathbf{W}_{\text{ho}}\mathbf{h}_t + \mathbf{b}_{\text{ho}} \tag{2.7}$$

Here, parameters described as $\theta_{\text{hh}}, \theta_{\text{ho}}$ are more concretely laid out in vector and matrix form such that:

- $\sigma(\cdot)$ is a non-linear activation function, most often the hyperbolic tangent, $\tanh(\cdot)$
- $\mathbf{W}_{\text{ih}}$ is a $\mathbb{R}^{N_{\text{hidden}} \times N_{\text{input}}}$ matrix that transforms (expands, most often) the input vector at time $t$, $\mathbf{x}_t$ to a vector with dimension $N_{\text{hidden}}$
- $\mathbf{W}_{\text{hh}} \in \mathbb{R}^{N_{\text{hidden}} \times N_{\text{hidden}}}$ is the hidden-to-hidden matrix transform that of the last computed hidden state $\mathbf{h}_{t-1}$ from which *recurrent* nets derive their name. I refer to it as the *reservoir matrix* the context of echo state networks.
- $\mathbf{W}_{\text{ho}}$ is a $\mathbb{R}^{N_{\text{output}} \times N_{\text{hidden}}}$ matrix that takes the current hidden state $\mathbf{h}_t$ and produces the desired output.
- $\mathbf{b}_{\text{h}}$ and $\mathbf{b}_{\text{ho}}$ are vectors containing bias parameters.

Since Eqs. (2.6-2.7) are inherently *separated*, the dynamical system can run independently of output generation. After training, however, some tasks require *free-running prediction* where generated output is then used as input, i.e. $\mathbf{y}_t \to \mathbf{x}_{t-1}$. This will be the

approach for the applications (like anomaly detection) using the Spatial ESN presented in this report.

### 2.3.3 General RNN Expressiveness Comes at a Cost

Recurrent neural networks - unlike their ESN variants - are what is called *Turing complete* meaning that they can simulate all *Turing machines* (Siegelmann et al. 1995). Stated popularly, any computable process can be computed by some finite RNN (and there exist infinitely many RNNs to do so). But there is a catch!
The above is only true when the RNN is built using *exact rational numbers*, and not on numbers represented by *today's computers* that use floating point precision. The expressiveness is *limited* by the floating point precision in that such RNNs are *no longer Turing complete*.

*However*, in another twist of events that limitation of expressiveness is also *the only reason RNNs are useful* in practice. This is because *Rice's theorem* will limit what can be computed *in practice* for a Turing complete system.
I.e. there is a *trade-off* between *expressiveness* and *practical approximation capability*. Due to the expressiveness of RNNs they are *hard to train* on real-world computers (impossible to train in many cases given access to rational numbers). *Echo state networks* on the other hand are *less expressive* (expressive enough for most practical problems), but that makes it *easier to find good solutions* in practice, giving them an edge to RNNs in some cases, at the very least if *time* is a factor. As such, the limited expressiveness of ESNs is therefore often to its *benefit*.

### 2.3.4 Training an RNN Can Be Tricky

This section seeks to describe why general RNNs are not always the best choice, over ESNs, despite their theoretically larger expressiveness. As such, it is also a motivation of the choice of ESNs for hard spatio-temporal problems.

#### Gradient-Based Learning

The general RNN training paradigm is to optimize all parameters in $\mathbf{W}_{\text{ih}}$, $\mathbf{W}_{\text{hh}}$, $\mathbf{W}_{\text{ho}}$, $\mathbf{b}_{\text{h}}$, $\mathbf{b}_{\text{ho}}$. This is no simple task as the parameters should be optimal at all time. The most

common approach is *gradient-based* learning applied to the entire unfolded network (see Fig. 2.2). Put simply, gradient-based optimization updates parameters by taking a step in the direction (in a parameter *space*) that decreases the mistakes of the model *the most*. This is measured by the negated *loss gradient* with respect to trained parameters of the model.
The 'loss landscape' in which a set of parameters are a *high-dimensional points* has no guarantee of convexity, so one can easily end up with parameters that are make up a poor local minimum (high loss). A number of advanced gradient-based methods try to combat this issue.

Gradient-based learning is further challenged in RNNs to optimize *throughout all time* steps. The predominant approach for RNNs is *Back-Propagation Through Time*. It is helpful to have Figure 2.2 in mind when contemplating the gradient that must be computed over and over.
Do note that *echo state networks* do not optimize any parameters of the *dynamical system* (Eq. 2.6). That makes it possible to apply *least squares* with a guaranteed **unique solution** for overdetermined problems.

#### Back-Propagation Through Time (BPTT)

*Back-Propagation* is a technique to calculate any gradient, and is almost always used where gradient-based optimization is applied. Back-Propagation Through Time is simply back-propagation applied to the unfolded graph, i.e. through all time steps of the RNN as seen in Fig. 2.2. Back-Propagation relies on the chain rule of differentiation for functions of functions (composite functions). Let $x$ be a real number, $y = g(x)$, $z = f(y)$ then:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{2.8}$$

The BPTT approach allows the computation of gradients, but to evaluate the loss gradient wrt. parameters to update parameters:

- the dynamical system must evolve for the duration of the training set, referred to as an 'epoch',
- all hidden states must be stored for BPTT evaluation,
- upon weight update the process is restarted from scratch

Therefore, large training sets may take a very long time to train, and without guarantee of landing in the global minimum loss.

#### Issues of Training RNNs with Gradient-Based Methods

Perhaps more significantly, the gradient-based learning itself is not trouble-free, and may not lead to great predictive power even when trained for many epochs. Major issues include that can be especially bad for RNNs due to their depth:

- **Vanishing gradients**: When training RNNs, gradients often become *vanishingly small*, which prohibits gradient-based learning (extremely small parameter updates, or ineffective ones). LSTMs are more well-behaved in this aspect, but still hard to train well.
- **Exploding gradients**: In a related case, gradients can become very large, which also makes it hard to find good minima.
- **Bifurcations**: Hidden state space *bifurcations* can emerge. In a dynamical systems perspective on Eq. (2.6) as a fixed point equation, different parameters set during training can produce wildly different outcomes. Even small numerical rounding errors can mean completely different results, and that makes it very hard for gradient-based methods to optimize. When the system encounters a parameter set where a bifurcation occurs, the system can behave very differently from the parameter set that was *before* the last update. As this happens often when training RNNs they can be very unsuccessful even allowed a long time for training. For further description and examples, I refer the interested reader to the appendix of (Heim and Avery 2019).

### *2.3.5 ESN Training: Least Squares*

Until now, I have not been clear enough in distinguishing between ESNs and more general RNNs. This is because echo state networks are given by the same equations Eqs. (2.6-2.7), but with a **vastly different approach to training and optimization**, that separate it from general RNNs.

ESNs will be described further in chapter 3, but since we are at the topic of *optimization*,

it seems pertinent to get into the approach of ESNs.

### An Untrained Dynamical System, and Regularization

As I have stated, ESNs *do not train* the dynamical system (Eq. 2.6), but only the *readout* of Eq. (2.7). This *avoids* the issues that I have briefly discussed with regards to training RNNs, but also imply that the dynamical system of ESNs must be initialized *more carefully* (see Chapter 3).

Since only the readout layer is trained, the readout matrix $\mathbf{W}_{\text{ho}}$ that transforms hidden states $\mathbf{h}_t$ to desirable outputs $\mathbf{y}_t$ can be *optimized using least squares*. The optimization is remarkably different to gradient-based methods!

To be specific, echo state networks take the states $\mathbf{h}_t$ of the training set, and gathers them in a matrix, $\mathbf{H}$ that I refer to as *the hidden state matrix*. This matrix can be used for multiple least squares:

$$\mathbf{W}_{\text{ho}} = (\mathbf{H}^\top \mathbf{H})^{-1} \mathbf{H}^\top \mathbf{D} \tag{2.9}$$

Here, $\mathbf{D}$ is the *training targets*, i.e. also vectors stacked in a matrix format.
Almost always, it is recommendable to *not* apply Eq. 2.9 when training RNNs – especially in the underdetermined case that often occurs when the hidden dimension is large.
Instead, regularized methods such as ridge regression should be used (Lukoševičius 2012).
On that note, the 'spatial echo state network' of (Heim and Avery 2019) used a least squares method (spectral filtering) based on singular value decomposition (SVD) to achieve a less *ill-conditioned* regression problem.

In addition, this thesis will also apply *another* type of regularization using PCA *dimension reduction* on $\mathbf{H}$. This has a number of benefits, and one of them is to make the regression problem *overdetermined*. Another is that it impedes model complexity, which, as we have seen in Section 2.2.5, leads to larger *out-of-sample error*.

### Benefits of Least Squares

The implications of the different approach to RNNs are many.
First of all, a unique solution can be found to overdetermined problems. For underdetermined problems, reasonable solutions can often still be found using regularization. The solutions are often *high quality*, as we do not end up in a local minimum or encounter vanishing gradient problems, or state space bifurcations.
Further, optimization is *fast*. For the high-dimensional **spatio-temporal problems**, general RNNs are really **no alternative**, as I will look at time series that have hundreds of thousands of variables. While gradient-based machine learning is often carried out using heavy GPUs, that has not been necessary when training ESNs in this thesis.
A combination of a CNN and LSTM know as CNN-LSTM *is* an interesting alternative that has less parameters (like CNNs) and mitigates the vanishing gradient problem (like LSTMs). However, optimization is *still* gradient-based. Due to time constraints, this venue has not been explored further in this thesis.

### ESNs Require Larger Hidden State Dimension

It is interesting to consider *why an affine linear fit is effective* for ESNs (we *do* fit the intercept!). Of course, if the activation function was not there, the entire ESN would be *linear*, and therefore collapsible to a network that is not deep (the same is true for general artificial neural networks). The activation function can also be considered a *feature transform* where the hidden state variables are *features*. This is often used (with the 'kernel trick') to allow least squares to fit a linear function in another space, such that the result is a non-linear fit, when transforming back to the original space.
However, there is also another reason why ESNs can be effectively trained using least squares: **we increase the dimension of the hidden state** to gain in some expressiveness that we *lost* when deciding not to optimize the dynamical system (in comparison to general RNNs). Therefore, the hidden state is often *larger* than the *input sequence*. This provides the necessary memory capability of the untrained dynamical system. It also provides a *rich* set of regressors in $\mathbf{H}$ that is used for the least squares fit.
Since larger hidden states are used for the already high-dimensional spatio-temporal series, the spatial ESN requires more memory to train and utilize than general RNNs. That is a small price to pay for effective and efficient application, as we shall see...

# Chapter 3

# *Echo State Network Dynamics*

---

After the description of general RNNs and machine learning of the previous chapter, we turn to familiarizing with the echo state network, specifically. The state of the *spatial ESN* prior to the optimizations from this thesis is also described in Sec. 3.5, as a warm-up to Part II in which the SESN is made *scalable*.

## 3.1 A Review of Echo State Networks

To summarize what has been described in Chapter 2 about RNNs and ESNs, I reproduce below the two RNN/ESN Equations (2.6-2.7), where I choose specifically the activation $\sigma(\cdot) = \tanh(\cdot)$ (applied to all vector elements), which is most well-understood in an ESN context.

$$\mathbf{h}_t = \tanh(\mathbf{W}_{\text{ih}}\mathbf{x}_t + \mathbf{W}_{\text{hh}}\mathbf{h}_{t-1} + \mathbf{b}_{\text{h}}) \tag{3.1}$$

$$\mathbf{y}_t = \mathbf{W}_{\text{ho}}\mathbf{h}_t + \mathbf{b}_{\text{o}} \tag{3.2}$$

Recall that in an RNN context, the two are deeply intertwined, as all weights of $\mathbf{W}_{\text{ih}}$, $\mathbf{W}_{\text{hh}}$, $\mathbf{W}_{\text{ho}}$, $\mathbf{b}_{\text{h}}$, $\mathbf{b}_{\text{out}}$ are *optimized*, whereas the same is only true in an ESN context for $\mathbf{W}_{\text{ho}}$ and $\mathbf{b}_{\text{out}}$.

The first step to familiarizing oneself with ESNs, therefore, is to think of the two equations as *completely separate*, with terminology as in Table 3.1

| **ESN Dynamical System** Eq. (3.1) | **ESN Affine Linear Readout** Eq. (3.2) |
|---|---|
| • Develops hidden state $\mathbf{h}_t$ *recurrently* | • Linear in the non-linear hidden state covariates |
| • Driven by input $\mathbf{x}_t$ | • Transforms hidden state $\mathbf{h}_t$ to output $\mathbf{y}_t$ |
| • Dynamics according to properties of $\mathbf{W}_{\text{ih}}$, $\mathbf{W}_{\text{hh}}$, $\mathbf{x}_t \in \mathcal{X}$ | • $\mathbf{W}_{\text{ho}}$ optimized on $\mathbf{H} = [\mathbf{h}_{t_0}; \mathbf{h}_{t_0+1}; ...]$ only *once* |
| • Driven by input $\mathbf{x}_t$ | • Supervised regression |
| • Non-Linear activation $\sigma$ | |

**Table 3.1:** *Unlike RNNs, it is better to think of ESN as two completely separate parts: A dynamical system, and a linear readout. The dynamical system does not participate in the readout (optimization) task, and is configured a priori. It produces the regressors* $\mathbf{h}_t$ *that the readout system takes as 'input'.*

Note that the role of $\mathbf{b}_{\text{h}}$ is to control the centring of the hidden state. In an ESN context, it is not in general relevant to define a random initialization of $\mathbf{b}_{\text{h}}$, so it is often *dropped*.

In the table, I refer to the first of the two equations as a *dynamical system*, an input driven one, in fact. Depending on the reader's background, it may be helpful to realize that Eq.

(3.1) can be interpreted as the *Euler discretization* of the ordinary differential equation (Lukoševičius 2012, p. 668):

$$\frac{\partial \mathbf{h}(t)}{\partial t} = \tanh\left(\mathbf{W}_{\text{ih}}\mathbf{x}(t) + \mathbf{W}_{\text{hh}}\mathbf{h}(t)\right) - \mathbf{h}(t) \tag{3.3}$$

## 3.2 Echo State Network Dynamics

With *all* weights of the ESN dynamical system chosen *a priori*, i.e. left unoptimized the application of an ESN requires slightly more understanding of the dynamics that may arise from a given pre-configuration of the dynamical system.
Of course, the (training) time series $\mathbf{X}_{N_{\text{train}}} = \left[\mathbf{x}_0; \mathbf{x}_1; ...; \mathbf{x}_{N_{\text{train}}}\right]$ has large impact (as it should) on the dynamics of $\mathbf{h}_t$, but I proceed with no specific assumptions, except that it is reasonably well-behaved numerically, e.g. *does not* give rise to the *constant saturation* of the $\tanh(\cdot)$ activation, which has range $(-1; 1)$, and is not constantly the *zero vector*.

Sometimes, in the literature, the input-to-hidden matrix $\mathbf{W}_{\text{ih}}$ and the hidden-to-hidden matrix $\mathbf{W}_{\text{hh}}$ are referred to, in combination, as *the reservoir*. I will make a distinction between $\mathbf{W}_{\text{ih}}$ as the *input map/matrix* and $\mathbf{W}_{\text{hh}}$ as the *reservoir*. Both are *fixed* during the evolution of the dynamical system.

## 3.3 Input Mapping

I refer to an *input mapping* as the operation of performing a *linear transform* $Wih : \mathbb{R}^{N_{\text{input}}} \rightarrow \mathbb{R}^{N_{\text{hidden}}}$ on the (flattened) input $\mathbf{x}_t$ at time $t$. This occurs at every time step of the ESN dynamical system when $\mathbf{W}_{\text{ih}}\mathbf{x}_t$ is computed. In that case, the linear function $Wih$ is represented as a dense random matrix $\mathbf{W}_{\text{ih}}$.

Conventionally, $N_{\text{hidden}} > N_{\text{input}}$ is chosen, and this is referred to as an **expansion** of the input into a higher-dimensional space. Another step of the dynamical system is the application of $\tanh(\cdot)$, providing a somewhat indirect *non-linear* expansion of the input (scrambled together with previous hidden state $\mathbf{W}_{\text{hh}}\mathbf{h}_{t-1}$).

Input expansion has several as listed in Table 3.2

**Table 3.2:** *A summary of the properties of mapping input to hidden state space.*

| **Input Mapping** |
|---|
| • Expand input from dim $\left(N_{\text{input}}\right)$ to dim $(N_{\text{hidden}})$ |
| • Linear transform in RNNs and ESNs |
| (activation $\sigma(\cdot) = \tanh(\cdot)$ applied later) |
| • High-dimensional hidden state $\mathbf{h}_t$ |
| contains more information, allows memory retention |
| • Like 'kernel trick' linearly inseparable |
| observations may be separable after non-linear expansion |
| • ESN mapping traditionally represented as dense matrix $\mathbf{W}_{\text{ih}}$ |
| • $\mathbf{W}_{\text{ih}}$ may be represented as function $Win(\mathbf{x}_t)$ instead. |
| Approach of (Heim and Avery 2019). |

The ingenuity of (Heim and Avery 2019) is to replace $\mathbf{W}_{\text{ih}}$ by computationally more efficient, and *non-random* linear maps $Wih(\mathbf{x}_t)$. These functions provide several types of spatial information to the ESN dynamical system, like the image gradient, 2D convolutions or low-frequency DCT coefficients. The simplest input map is the input itself — perhaps rescaled using bilinear interpolation.
The different spatial representations of the input image are then flattened, and *concatenated*, defining the hidden state dimension, $N_{\text{hidden}}$.
The input maps are not the primary focus of this thesis, but they are applied in it. I further describe the spatial input maps in Table 7.5 of Chapter 7.

As listed in Table 3.2, a major benefit of expanding input into a higher-dimensional space in an ESN context is that a larger state $\mathbf{h}_t$ simply allows more information about past inputs to be stored than a smaller dimension $N_{\text{hidden}}$ would.
The non-linear activation function allows *richer* states $\mathbf{h}_t$ to be harvested, so that it is possible to successfully construct a linear readout from $\mathbf{h}_t$ to desired output $\mathbf{y}_t$.

## 3.4 Reservoir Dynamics

The reservoir $\mathbf{W}_{\text{hh}}$ is a real, square matrix with dimension $N_{\text{hidden}} \times N_{\text{hidden}}$, and connects the last hidden state $\mathbf{h}_{t-1}$ with the next, i.e. $\mathbf{h}_t = \sigma(\cdots + \mathbf{W}_{\text{hh}}\mathbf{h}_{t-1})$. In Table 3.3, I provide a brief summary of the purposes and properties of the reservoir.

**Table 3.3:** *The role of the reservoir, $\mathbf{W}_{hh}$, in summary.*

| **The Reservoir** |
|---|
| • Create a memory of previous input |
| • Most important intrinsic property: $\rho(\mathbf{W}_{\text{hh}})$, the spectral radius |
| • Connect hidden state elements to a number of other elements |
| • A sparse matrix |

Most importantly, in the table, the reservoir induces memory of previous inputs that is necessary in a temporally correlated system to determine the next state. In principle the larger the hidden state, the more information of previous input can be stored in the hidden state. For complex problems, the larger the better, is a good rule of thumb (as long as the readout is configured properly).

It is not uncommon for 'traditional ESNs', i.e. non-spatial models, to have $N_{\text{hidden}} \sim 10^4$ (Lukoševičius 2012), whereas this thesis will push the hidden state dimension to order $\mathbf{10^6}$ on reasonable hardware that was not configured for the sole purpose of running ESNs, see Section 7.8.2.

If the reservoir is *poorly preconfigured*, however, the memory ability will *not emerge* in the dynamical system. In ESN literature, the **echo state property** is the term describing whether the dynamical system is applicable as a model for learning (should be thought of as a necessary condition, not a sufficient one). Next, I build up the necessary terminology to more clearly define the echo state property. I then discuss practical aspects. Most importantly, the *spectral radius*.

### 3.4.1 Eigenspectrum and Spectral Radius of the Reservoir

In ESNs, the properties of the square reservoir matrix $\mathbf{W}_{\text{hh}}$ are of crucial importance to the expressiveness and stability of the network, and the subject of much study. In particular, the eigenvalues $\lambda_i$ of the reservoirs are valuable for interpreting (or designing) the effects of the linear transformation that the reservoir *is*. Its eigenvalues are solutions of the eigenvalue equation:

$$\mathbf{W}_{\text{hh}}\mathbf{v} = \lambda\mathbf{v} \tag{3.4}$$

Where $\mathbf{v}$ is the eigenvector that corresponds to the eigenvalue $\lambda$ (a scalar). That is, the eigenvectors inform of us certain *directions* in which the intricate linear transform can be understood *simply* as multiplication by a scalar.

Real eigenvalues therefore indicate how much *stretching* will occur in the direction of the corresponding eigenvector. A complex eigenvalue $\lambda = re^{i\phi}$, on the other hand, can be interpreted as rotation in the plane spanned by $\text{Re}(\mathbf{v})$, $\text{Im}(\mathbf{v})$ with an angle $\phi$ and a radius $r = |\lambda|$ (stretch). As the reservoir is real, the presence of a complex eigenvalue $\lambda$ guarantees the existence of another *eigenpair* with complex conjugate values $\{\overline{\lambda}, \overline{\mathbf{v}}\}$ resulting in the *same* rotation.

The geometric interpretation of eigenvalues provides some intuition that they should not be *too extreme* for a reservoir, to avoid unstable dynamics from e.g. *signal amplification*. A lot of discussion around echo state network design therefore focuses on the **spectral radius** of the reservoir. It poses and *upper bound* to the stretching effect of the linear transform in any direction:

$$\rho(\mathbf{W}_{\text{hh}}) = \max_{1 \le i \le N_{\text{hidden}}} |\lambda_i| \tag{3.5}$$

### 3.4.2 The Echo State Property

A colloquial statement of the echo state property (ESP) is that:

> *The driving input $\mathbf{x}_t$ at time $t$ defines the dynamical system for some duration, but cease to do so at a later (finite) time.*

Loosely, this ensures that close sequential driving inputs are incorporated into the hidden state until their replacement by other driving input at a later time, while *echoing* in the system until then.

For a more formal treatment, see (Yildiz et al. 2012).

It is a common misunderstanding in the ESN world that the ESP is guaranteed whenever $\rho(\mathbf{W}_{\text{hh}}) < 1$, allowing a network with fading memory capabilities. On the contrary, the ESP depends on the fixed parameters of $\mathbf{W}_{\text{hh}}, \mathbf{W}_{\text{ih}}$ *and driving input*.
In fact, the criterion is only sufficient for the zero vector input, while in many other cases $\rho > 1$ also satisfies the ESP.

Sufficient conditions for $\mathbf{W}_{\text{hh}}$ to satisfy the ESP for all input include (Yildiz et al. 2012):

1. The maximum singular value of $\mathbf{W}_{\text{hh}}$ is less than unity $\sigma_{\max}(\mathbf{W}_{\text{hh}}) < 1$, where singular values $\sigma$ are the square roots of non-negative eigenvalues of $\mathbf{W}_{\text{hh}}^{\top}\mathbf{W}_{\text{hh}}$.

2. $\mathbf{W}_{\text{hh}}$ is *diagonally Schur stable*, implying the existence of a positive definite matrix $\mathbf{P}$ such that $\mathbf{W}_{\text{hh}}^{\top}\mathbf{P}\mathbf{W}_{\text{hh}} - \mathbf{P}$ is negative definite.

While both conditions guarantee the satisfaction of the ESP, the conditions are not *necessary*, and the first condition is especially **restrictive** and often leads to poor memory capabilities.
In practice, $\rho$ **is tuned** according to the problem at hand more or less manually, and can have a significant influence on the prediction dynamics. Tuning it is done by creating a random initialization of a reservoir $\tilde{\mathbf{W}}_{\text{hh}}$, determining its spectral radius $\tilde{\rho}$ by diagonalization (using e.g. ARPACK); then rescaling the reservoir to $\mathbf{W}_{\text{hh}} = [\rho/\tilde{\rho}]\,\tilde{\mathbf{W}}_{\text{hh}}$ to achieve a spectral radius of $\rho$. This simple approach is allowed by the eigenvalue equation (3.4). Diagonalization is a computationally intensive operation, and the *Arnoldi algorithm*, a version of which is applied by ARPACK, is $\mathcal{O}\left(kN^2 + k^2N\right)$ in time complexity (Lee et al. 2009). Here, N is the matrix dimension, and $k$ is the number of large eigenvalues to find (one, for the spectral radius).
**This thesis will show a way to get rid of the diagonalization** in order to create much larger hidden states, and reservoirs in conjunction.

The intuition behind the ESP as dependent on $\rho$ is that the activation $\tanh(\cdot)$ may be increasingly saturated as time increases if the *reservoir amplifies the previous state* produced using non-zero input over and over. When the largest eigenvalue, in magnitude,

is below 1, the input to the activation function is *often* **contractive**. On the other hand, a *small* spectral radius will push the hidden state closer to the origin, where the behaviour of $\tanh(\cdot)$ is less **non-linear**. This will decrease the memory capacity, and is often a good choice if the predictions only depend on near-history (Lukoševičius 2012).

It should be stressed that different reservoirs can have the same spectral radii, but different eigenspectra and performance. It is common to report ESN approximation performance as an average over several trials.

### *3.4.3 Beyond the Echo State Property*

Selecting the spectral radius and other hyperparameters of ESNs is in practice very *hands on*, but the knowledge that a spectral radius around unity is a good starting point, is useful information that is not as relevant for general RNN implementation.

Beyond the memory and spectral radius that I have mentioned, in Table 3.3, there is the subject of *sparsity*, and the *distribution* of non-zero values in the reservoir.
A common approach (Lukoševičius 2012) is to pick non-zero values from the *symmetric uniform distribution*, *standard Gaussian*, or (less commonly) from binary options i.e. $\{-a, a\}$. The scaling, or endpoints, of the distribution is not significant when $\rho$ is later chosen by *rescaling* the matrix, as explained in Section 3.4.2. The first two options yield often indistinguishable results for the same $\rho$, whereas the latter is less popular due to lack of 'richness' arising from the binary numbers.

*Sparsity* is not always a significant parameter, and often chosen to be 1% or 10%.

I generally encourage the reader to visualize the impact of the reservoir as that of its eigenspectrum rather than in terms of the non-zero values themselves. The value at a row and column $(i, j)$, can, however, be thought of as a *neuron* connecting the $j$'th element of $\mathbf{h}_{t-1}$ to the $i$'th element of $\mathbf{h}_t$. The number of non-zero values in each row of $\mathbf{W}_{\text{hh}}$, then, is the number of connections, or neurons.

In this thesis, I explore the Gaussian and uniformly distributed values. The non-zero

values will be placed *uniformly* in each row. Further, unconventionally, I explore a *fixed* number of non-zero values per row. My particular approach is explained in Section 4.1.

## 3.5 Challenges and Bottlenecks to Spatio-Temporal Learning

The work of (Heim and Avery 2019) has shown that *spatio-temporal prediction* of chaotic series is achievable with a special design of the ESN.
However, their work has also highlighted the challenges that the ESN design poses when the prediction task is *extremely high-dimensional*. Each pixel in an input image at time $t$ makes up a *dimension* of the flattened input vector, $\mathbf{x}_t$.
When their spatial input maps are utilized, the input sized $N_{\text{input}}$ is expanded into an $N_{\text{hidden}}$ sized hidden state. This makes the prediction problem even more high-dimensional. Most of the spatial input maps have the same number of 'pixels' as the original image, and when several maps are chosen, $N_{\text{hidden}}$ may be a multiple of $N_{\text{input}}$ (see Table 7.5).

Next, I present the challenges that the high-dimensional covariates pose, and outline the solutions that I will later present to achieve predictions for spatially high-dimensional image frames.

### Challenges to Construction of the Reservoir

The main challenge, as I have noted, wrt. the reservoir and spatial scalability of the ESN is that the common approach of construction is:

1. Generate a real, random, sparse matrix $\tilde{\mathbf{W}}_{hh}$ with some density, and non-zero values picked from a zero-symmetric probability distribution.
2. Determine the eigenvalue with largest magnitude, $\tilde{\rho}$.
3. Scale $\tilde{\mathbf{W}}_{hh}$ by $\rho/\tilde{\rho}$ to achieve final reservoir, $\mathbf{W}_{\text{hh}}$ with desired spectral radius $\rho$

The second step requires the most computation for large reservoirs ($N_{\text{hidden}} \times N_{\text{hidden}}$). Since the random sparse matrix generally has complex eigenvalues, a simple method like *power iteration* is not applicable for fast determination of $\tilde{\rho}$.
The approach of the scalable SESN, described in Chapter 4 will get rid of the second and third steps entirely using a stochastic law for the eigenspectrum of a certain type of

sparse matrix. That way, the variance of the probability distribution that I sample non-zero values from will approximately determine the spectral radius with high probability, especially for large reservoirs.

For 'small' reservoirs with $N_{\text{hidden}} < 5000$ I advice caution, and best results are most probably achieved by computing $\tilde{\rho}$ with the Arnoldi method of `ARPACK`, for example.

Like (Lukoševičius 2012) I find that the sparsity of the reservoir is not a very significant hyperparameter to the ESN approximation performance. I therefore continue with an approach where the *density of* $\mathbf{W}_{hh}$ *is not fixed* when $N_{\text{hidden}}$ is increased. Instead, the number of *non-zero values* in each row of $\mathbf{W}_{\text{hh}}$ *is*.
This has benefits to both speed and memory consumption.

**Challenges to the Spatial IMED Loss Function**

Another originality of (Heim and Avery 2019) is the implementation of the *Image Euclidean Distance* (IMED) as a loss function that is compatible with least squares optimization. To my knowledge, it is the first time that the IMED has been applied for *regression problems*, and not just classification. This requires both the implementation of a forward transform and a backward transform related to the IMED, and denoted the (inverse) *standardizing transform* (iST/ST).

The original implementation of the SESN, however, should be viewed as a demonstration of IMED utility rather than a *final* version. Their implementation requires the storage of an $\left(N_{\text{x}}N_{\text{y}} \times N_{\text{x}}N_{\text{y}}\right)$ matrix, *and its full diagonalization*. This is prohibitive to spatial scaling with respect to both time and memory consumption. I therefore implement more recent, but similar Fourier transform-based IMED metric that does not require diagonalization. Further, I introduce its inverse transform for regression problems, and also implement a version based on the discrete cosine transform with better boundary conditions for most images.
For better approximation performance, the methods I implement also allow *temporal* correlations to be considered in the loss metric, instead of just the spatial correlations in single images. To my knowledge, that is also a new utilization.

The implementation and improvements to the IMED metric are described in Chapter 6.

### Challenges To Optimization (Readout)

The main challenges of performing multiple least squares in the SESN are listed in Table 3.4.

The fact that the number of trained parameters per input pixel is equal to the hidden state dimension $N_{\text{hidden}}$ entails several issues. When input is high-dimensional, and the hidden state is a multiple of that (3-8× is common for the SESN) the output matrix $\mathbf{W}_{\text{ho}}$ can be prohibitively large in memory, and take an increasing amount of time to optimize using least squares. But that is not all: The sheer number of parameters can lead to over-fitting, i.e. perfect approximation of the training set, but poor results when prediction unseen data (the aim of supervised learning).

The spatial nature of the input maps is also a challenge to least squares, since it makes the elements of $\mathbf{h}_t$ highly correlated. This implies that weights of $\mathbf{W}_{\text{ho}}$ must depend on several mutually dependent covariates, and leads to instability and sensitivity to the behaviour of the dependent covariates.

Lastly, the number of *features* per regressor $\mathbf{h}_t$ can vastly exceed the number of training observations $N_{\text{train}}$, which makes the Gram matrix $\mathbf{H}^\top\mathbf{H}$ un-invertible.

**Table 3.4:** *Challenges posed by the high-dimensional $N_{hidden}$ in the SESN by (Heim and Avery 2019) wrt. optimization of $\mathbf{W}_{ho}$*

| Linear Readout Matrix $\mathbf{W}_{\text{ho}}$ | |
|---|---|
| Memory Footprint | $N_{\text{input}} \times N_{\text{hidden}}$ elements,<br>i.e. $N_{\text{hidden}}$ weights trained per input/output pixel |
| Time Consumption | SVD of $N \times N$: $\mathcal{O}\left(N^3\right)$ (See "LAPACK Benchmark" 1999).<br>Least Squares using the SVD: $\mathcal{O}(N)$ |
| Impact on Learning | *Many parameters*: over-fitting likely.<br>*Collinear Spatial Covariates*: Unstable solution<br>*Ill-Conditioned*: Regressor $\mathbf{H}$ ($N_{\text{train}} \times N_{\text{hidden}}$) has $N_{\text{hidden}} \gg N_{\text{train}}$ |

# Part II

# Development of a Highly Scalable Spatial Echo State Network

# Chapter 4

# *Scaling Up the Reservoir Matrix*

---

The properties of an ESN are deeply dependent on the reservoir matrix $\mathbf{W}_{\text{hh}}$. In this chapter contributions are made that allow the hidden state dimension to scale from traditional values of up to $10^4$, and well into order $10^6$ by avoiding any diagonalization to tune the spectral radius. This is a prerequiste for spatial scalability of the SESN due to its input maps.

## 4.1 Spectral Radius by Design

I have stated in Section 3.4.1 that the spectral radius $\rho$ of the reservoir matrix $\mathbf{W}_{\text{hh}} \in \mathbb{R}^{N_{\text{hidden}} \times N_{\text{hidden}}}$ can be and is conventionally picked by rescaling the matrix. This approach, however, requires that the spectral radius of the initially created reservoir first be determined. The reservoirs that are applied in this thesis are sparse, and up to orders $\left(10^6 \times 10^6\right)$. Therefore, determining the magnitude of the largest eigenvalue can be an extremely time-consuming operation (see benchmarks in 7.8). ARPACK (Lehoucq et al. 1997) is a fast FORTRAN eigenvalue solver for very large and sparse matrices that is conveniently wrapped in the SciPy Python package. The time consumption can vary depending on factors like the size of the matrix, the number of non-zero elements, the clustering of the eigenvalues, the desired precision and whether it is necessary to find corresponding eigenvectors. Convergence is *not guaranteed*, and that scenario becomes increasingly likely when $N_{\text{hidden}}$ is substantially increased. Clearly, a workaround to diagonalization is required for spatial scalability of the spatial ESN.

### *4.1.1 Exploring Spectral Radii*

During the initial exploration of the scalability of the spatial ESN (Heim and Avery 2019), the effect of the sparsity, or density of non-zero values was investigated. The idea is that it may not be necessary to *fix* the relative density (share of non-zero elements in the sparse matrix) when increasing the problem size and $N_{\text{hidden}}$, but perhaps not the complexity of the problem. The motivation for this line of thought is that each element in a row of the reservoir, $\mathbf{W}_{\text{hh}}$, can be thought of as a connection between hidden states $\mathbf{h}_t$ and $\mathbf{h}_{t+1}$. As its dimension, $N_{\text{hidden}}$ decreases the rows (and columns...) more connections are thus made to the existing hidden state, perhaps reducing the need to also increase the average number of non-zero elements per row (given by a fixed density).

Initially, this investigation was meant to see if computations could be spared by **fixing** the *number of non-zero elements per row*, $N_{\text{nzpr}}$, when increasing $N_{\text{hidden}}$, thus increasing the relative sparsity, and reducing (if incrementally) the workload from diagonalizing very large matrices.

### A Pattern in the Largest Eigenvalues

At the time, results indicated that the predictions were not greatly affected by this change (a good thing!), but also revealed something more interesting: While the largest eigenvalue was almost always complex and its real and imaginary parts had vastly different values, its **magnitude was approximately constant** when varying $N_{\text{hidden}}$ with fixed $N_{\text{nzpr}}$. This seems like a *remarkable* stochastic relationship as the positions of the $N_{\text{nzpr}}$ values in each row were uniformly randomly distributed at each initialization; with the values of the elements themselves uniformly distributed in the interval $(-1; 1)$ prior to tuning of $\rho$. The prospect of entirely dropping diagonalization to determine, and rescale the spectral radius of the reservoir was of course enticing, and required further examination. A first result can be seen in Fig. 4.1.

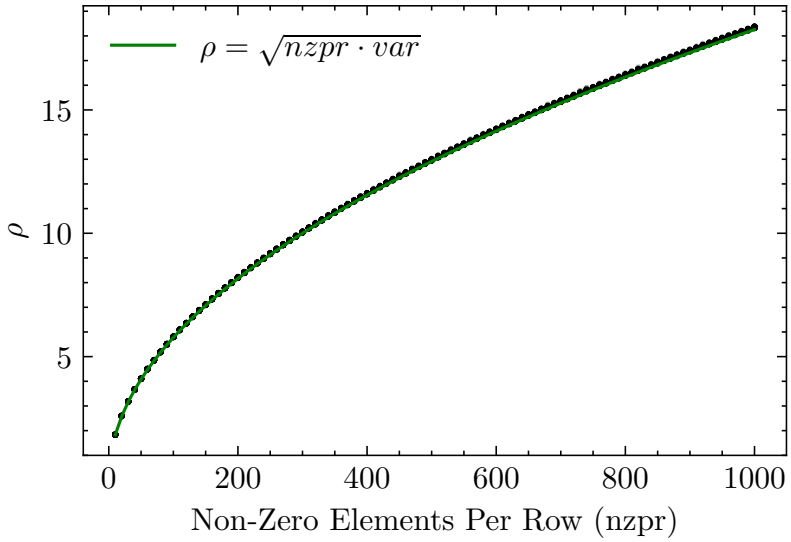



Figure 4.1: Spectral radii of sparse 50000 × 50000 matrices with a certain number of non-zero elements per row (nzpr); In black: 100 samples at each number of non-zero elements are shown. In green: An estimate of the spectral radius, where $var$ represents the variance of the non-zero elements. As these were picked from a uniform distribution $(a; b) = (-1; 1)$, they have variance $var = \frac{1}{12}(a - b)^2 = 1/3$.

From the raw observations (in black in the figure), it was clear that the spectral radius approximately follows a power law on average. Using the `probfit` and `iminuit` (Dembinski et al. 2020) python packages often employed for non-linear fitting in the field of High Energy Physics, It was determined that the relationship was similar to $\rho = c \cdot \sqrt{nzpr}$ with $c \approx 0.57$. When later returning to the problem using a Gaussian distribution of non-zero values, I discovered that $c$ indeed represented the square root of the variance of the distribution in both cases, as plotted in Figure 4.1. I have confirmed this fact for the

uniformly distributed case in other symmetric intervals than $(-1;1)$ (as must be the case given the eigenvalue equation).

The dependence of $\rho$ on $N_{\text{hidden}}$ – the dimension of the reservoir $\mathbf{W}_{\text{hh}}$ – is of course crucial to examine such that *extrapolation* may be utilized for determining the spectral radius of much larger matrices for which it is utterly *unfeasible* to naively compute the magnitude of the largest eigenvalue.

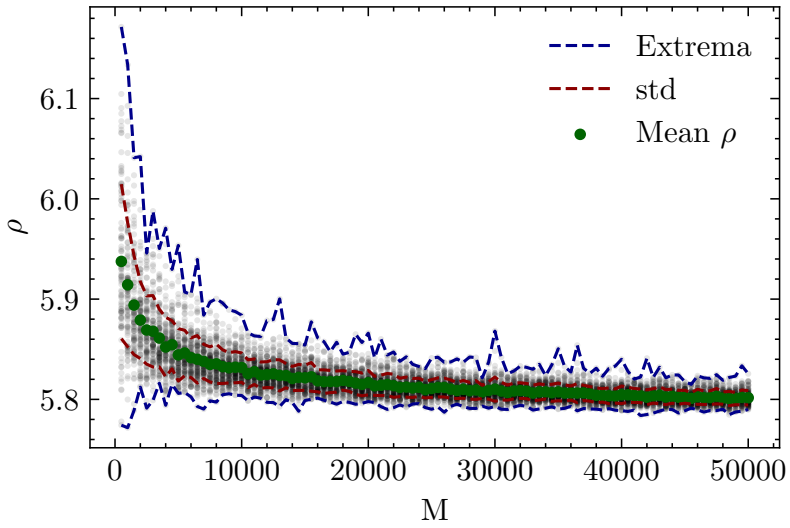
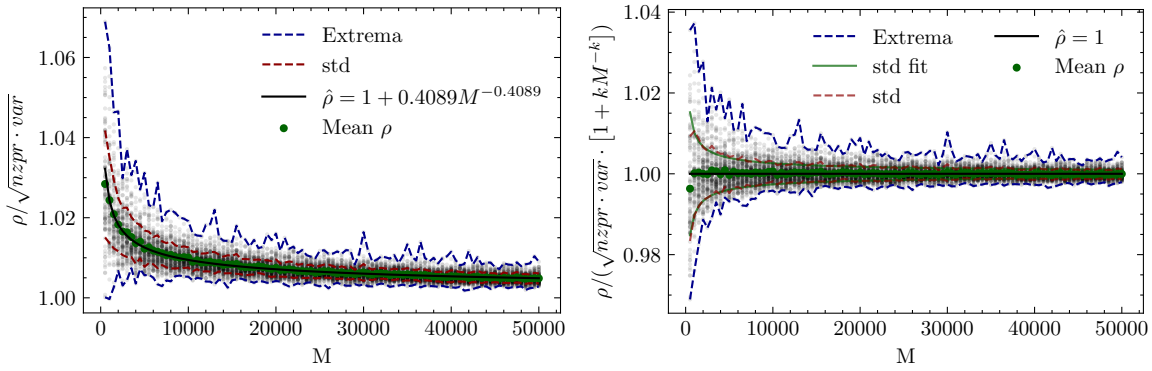



Figure 4.2: Spectral radii of sparse $M \times M$ matrices with a 100 non-zero elements per row ($N_{\text{nzpr}} = 100$); In black: 100 samples at each dimension are shown. In green: Sample mean of the computed spectral radius. All non-zero elements drawn from $(a;b) = (-1;1)$, i.e. variance $var = \frac{1}{12}(a-b)^2 = 1/3$. From the estimated relationship from Figure 4.1 the expected value was $\rho = \sqrt{\frac{100}{3}} \approx 5.77$, but this plot shows that the sample mean converges towards that value as the sparse matrix dimension is increased. The `scipy` ARPACK wrapper was utilized, set to determine the eigenvalue with the largest magnitude to a relative tolerance of $10^{-5}$.

In Figure 4.2 the dependence of the computed spectral radii on the dimension of the sparse matrices is seen. From here, we see that there is indeed more structure than simply $\rho = \sqrt{nzpr \cdot var}$. In fact, ARPACK finds that the spectral radii are slightly above the aforementioned estimate, converging towards it as the reservoir dimensionality increases.

To better visualize deviations from the crude estimate of $\rho = \sqrt{nzpr \cdot var}$, I rescale the observations by $\hat{\rho} = \rho\sqrt{nzpr \cdot var}$ in Figure 4.3(a) and fit a scaled power law $\hat{\rho} = 1 + kM^{-l}$ in that space with parameters $k, l$. It immediately appeared that $k = l$, so in the interest of simplicity, I have reverted to fitting a single parameter, i.e. $\hat{\rho} = 1 + kM^{-k}$. For non-zero values drawn from the uniform distribution – as seen in the figure – the

obtained fit for $\hat{\rho} = 1 + kM^{-k}$ is $k = 0.4098 \pm 0.0003$, while for the Gaussian distribution, the same approach yields (observations not shown) $k_G = 0.4075 \pm 0.0003$. I both cases, the sample standard deviation of the 100 observations was approximately $\sigma_U = 0.588M^{-0.588M}$ and $\sigma_G = 0.583M^{-0.583M}$ for the uniform and Gaussian distributions, respectively. This error estimate is important in deciding a *cut-off* below which the largest eigenvalue of reservoirs should be computed *explicitly* and above which $\rho$ can be estimated with high probability. The parameter fitted to the standard deviation is remarkably close to $1-k$, but it is currently not known, whether this curiosity is spurious or not. Further, the standard deviation may not be a perfect descriptor for the slightly asymmetric distributions of spectra radii.




(a) Initial rescaling

(b) Second rescaling

Figure 4.3: In subfigure (a), observations seen in Fig. 4.2 are rescaled by the reciprocal of the crude model $\rho = \sqrt{nzpr \cdot var}$ to better visualize deviation from the model. In that space, I fit a power law $\hat{\rho} = 1 + kM^{-k}$ to weighted observations that obtains $k = 0.4098 \pm 0.0003$ with $\chi^2/dof = 145.17/99$ (dof=degrees of freedom), or an estimated fitting probability of $p = 1.74 \times 10^{-3}$. Uncertainties are estimated as the sample standard deviation (std) rescaled by $1/\sqrt{N_{\text{obs}}}$ by assumption of Gaussian errors. Strictly, this assumption may be violated the distributions for each dimension value $M$ seem skewed. Uncertainty estimates affect the probability value.
In (b), a second rescaling by applying the obtained fit better displaying deviations from the updated model. Fitting the standard deviation to a power law so as to estimate the error on the spectral radii estimates (unweighted fit) provides $\sigma_G = 0.588M^{-0.588}$. On this scale, an outlier is seen at $M = 1000$. In this plot, $\hat{\rho} = 1$ is not a direct fit.

With the obtained fit for the uniformly distributed non-zero values, another rescaling is shown using the new estimates $\rho = \sqrt{nzpr \cdot var}\left(1 + kM^{-k}\right)$ in Fig. 4.3(b). At this scale, an outlier is seen at the lowest dimension of $M = 1000$. From the plot, we see that the maximum relative deviation from the modelled spectral radius of all 10000 samples equally distributed among 100 values of $M$ is below 4% at $M = 1000$. It converges as the dimensionality is increased. The fitted standard deviation – or average deviation from

the mean – is also displayed in Subfigure (b).

It is pertinent, after the term-by-term modelling, to revisit the $N_{\text{nzpr}}$-dependency originally displayed in Figure 4.1 by again rescaling according to the updated model $\rho = \sqrt{nzpr \cdot var}\left(1 + kM^{-k}\right)$. The result is seen in Figure 4.4 in is seen to be describe well the spectral radii as the number of non-zero values per row, $nzpr$, is varied.

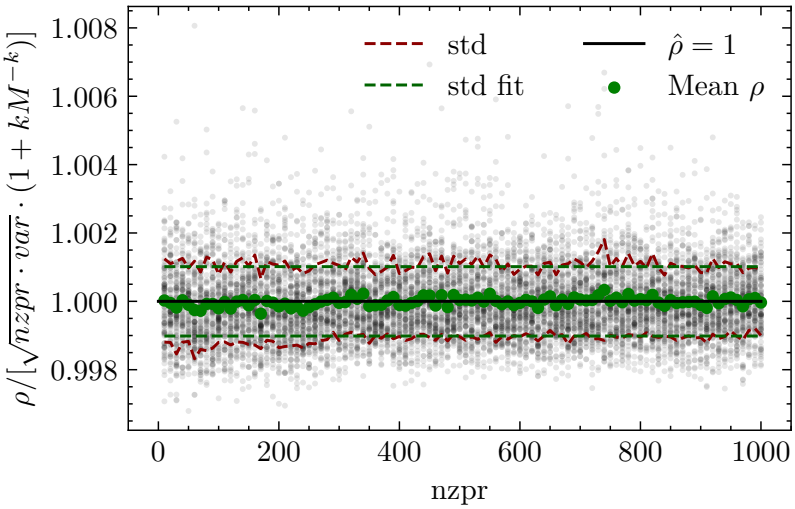



Figure 4.4: Spectral radii of sparse 50000 × 50000 matrices from Fig. 4.1, rescaled according to the model $\rho = \sqrt{nzpr \cdot var}\left(1 + kM^{-k}\right)$ with $k = 0.4098$ for non-zero values sampled from the uniform $[-1; 1]$ distribution (pictured). Also pictured, the fitted standard deviation from Fig. 4.3(b) at the particular instance dimension $M = 50000$; with sample means superimposed. Of the 10000 samples pictured at $M = 50000$, the maximum deviation from the model is 0.8%, and the skewed nature of the spectral radii distribution is more easily observed in this figure.

The exploration allows the **empirical conclusion** that the spectral radius for the random sparse matrices that have a fixed number of non-zero elements per row, spread randomly and uniformly in each row, with values of non-zero elements drawn from probability distributions including – at a minimum – the uniform and Gaussian distributions, can be predicted with high certainty as:

$$\rho = \sqrt{nzpr \cdot var}\left(1 + kM^{-k}\right) \pm sM^{-s} \tag{4.1}$$

Further, after implementing the approach in the spatial ESN, the expected dynamical difference between $\rho < 1$ and $\rho > 1$ is found to *remain* – an indication that the method *works*.

### 4.1.2 A Circular Law: The Eigenspectra of Sparse Fixed-$N_{nzpr}$ Random Reservoirs

From the initial observation that the magnitude of the largest eigenvalue of the random matrices was approximately constant at fixed $N_{\text{nzpr}}$, while the real and imaginary parts varied, a straightforward conclusion is that the maximum eigenvalue tends to lie in the perimeter of a circle with radius

$$\rho = \sqrt{\text{Re}\,(\lambda_{\text{max}})^2 + \text{Im}\,(\lambda_{\text{max}})^2} \tag{4.2}$$

that contains all eigenvalues of the matrix. This can indeed be confirmed by plotting the distribution of the real and imaginary parts of the largest eigenvalues of the random matrices. However, it is even more interesting to discover that the eigenspectrum of these random matrices have eigenvalues that are distributed *homogeneously*. As an example of the complete eigenspectrum of a matrix of the type discussed, see Figure 4.5 in which a dense $10000 \times 10000$ matrix is investigated using the rescaling of Eq. (4.1) (determined using a LAPACK wrapper).

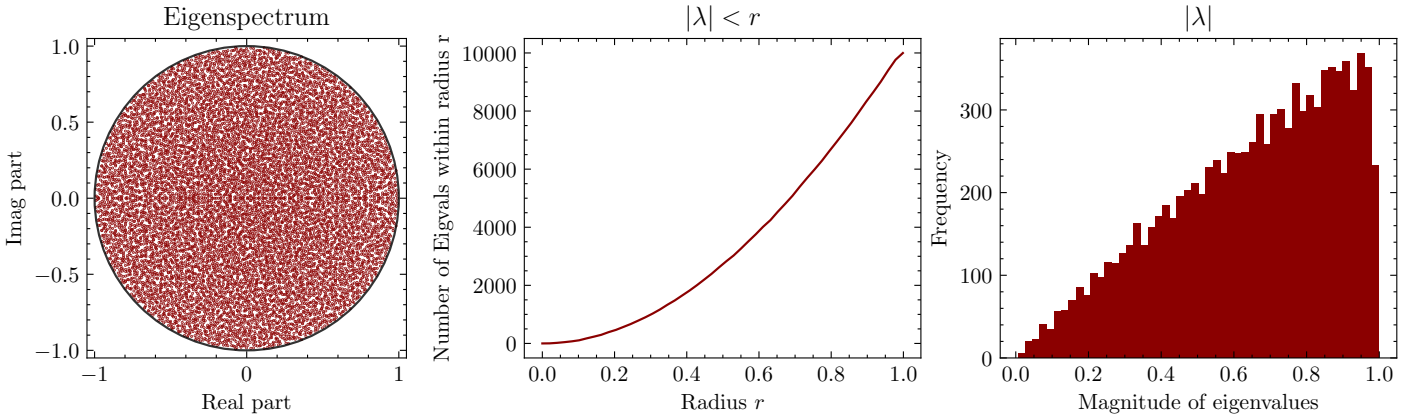



Figure 4.5: Eigenvalues of a dense 10000x10000 matrix with $N_{\text{nzpr}} = 25$ non-zero elements per row picked from the unit Gaussian distribution, rescaled to the unit circle using the estimate of $\rho$ from Eq. (4.1). Left: Eigenvalues are completely contained within a complex disk of radius $r = \rho$. Centre/Right: The number of eigenvalues within the disk grows like $r^2$ – like the area of the disk, and (right) with linearly growing magnitude like the perimeter of a disk of radius $r$ indicating that eigenvalues are distributed homogeneously within the disk. Note: For any complex eigenvalue $\lambda_+ = a + ib$ of a real matrix there is another eigenvalue that is its complex conjugate $\lambda_- = a - ib$, providing symmetry about the imaginary axis. Eigenvalues of the dense matrix are computed using `numpy.linalg.eig` with a LAPACK back-end (Anderson et al. 1999).

### Theoretical Support

It is a remarkable fact that the stochastic orderliness and simplicity arises from a remarkably random process. As I have since discovered, the behaviour is not *completely uncharted* territory to mankind.

There is some theoretical support to the behaviour that is observed empirically in this chapter. The **circular law** states that the eigenvalues of a (dense) random $M \times M$ matrix converge to uniform distribution within the complex disk of radius $\sqrt{M}$ as $M \to \infty$ provided that its elements have zero mean and unit variance. The result has been extended to general variance, yielding a radius of $\sqrt{\text{var} \cdot M}$ (Tao et al. 2008).
This is *similar* to empirical findings of this chapter, *but* we found instead that a sparse matrix, regardless of its dimensionality $N_{\text{hidden}} \times N_{\text{hidden}}$, behaves similar to a dense random $M \times M$ matrix provided that it has exactly $N_{\text{nzpr}} = M$ non-zero elements placed uniformly randomly in each row.
Observations are *not* collected in the dense limit, however, as it is of no interest to the spatial ESN application of this thesis.
To my knowledge, these results are yet to be proven. It has been found in (Wood 2012), however, that the standard circular law holds for random sparse matrices, too, when each element is non-zero with probability $p = 1/M^{1-\alpha}$ with $0 < \alpha \leq 1$. This is of course different from requiring a fixed number of non-zero elements in each row, which seems to place the eigenvalues in a smaller disk as $N_{\text{nzpr}} \leq M$.

### Benefits of the Homogenous Disk of Eigenvalues

The uniform distribution of eigenvalues within the complex disk may be especially beneficial when the matrix is used as a reservoir matrix $\mathbf{W}_{\text{hh}}$ as in this thesis. In (Ozturk et al. 2007) authors argued that the entropy of the *hidden states* $\mathbf{h}_t$ is a good indicator of the network's approximation performance.
They showed that an entropy measure was correlated with ESN predictive performance, and that the entropy was maximized when the eigenspectrum was *a complex disk*, as in Fig. 4.5, although a circular law was not *applied* for the study.

### *4.1.3 Practical Considerations*

Equation (4.1) can vastly benefit the spatial, high-dimensional spatial ESN model, and allow reservoirs that have dimensions *orders of magnitude* higher than in conventional ESNs. For non-zero values drawn from (at least) the uniform or Gaussian distributions it allows predicting the magnitude of the largest eigenvalue.
When ARPACK is used for to determine the eigenvalue that is largest in magnitude, the computational complexity is at least $\mathcal{O}\left(M^2\right)$, and in general $\mathcal{O}\left(kM^2 + k^2M\right)$ when $k$ eigenvalues and corresponding eigenvectors are computed (Lee et al. 2009).
The simple method of *power iteration* to determine the dominating eigenvalue is only $\mathcal{O}(M)$, and was indeed attempted to be implemented for this thesis prior to realizing that asymmetric real matrices (that provide better reservoirs) almost always have complex eigenvalues that never allow the power method to converge.

As an alternative to the highly time-consuming calculation of $\rho$, the spectral radius may now be rescaled using Eq. (4.1) by element-wise multiplication with the generated reservoir. The SSESN at GitHub[1], however, utilizes that the variance of the sampled non-zero values can be adjusted at the time of initialization of the random matrix according to the desired spectral radius, avoiding any *rescaling* for higher efficiency.

[1]Implementation of `reservoir()`: *https://github.com/jfelding/esn/blob/master/esn_dev/hidden.py*

# Chapter 5

# *Dimensionality Reduction*

---

It is important to the memory capabilities of the ESN dynamical system to have *high-dimensional* hidden states. However, when used directly as covariates, they are both *too correlated* in the spatial ESN, and *too many*- This results in overfitting and a more unstable model.

At scale, it also becomes unfeasible to optimize hundreds of thousands of parameters for each pixel in the desired image output (see Sec. 7.8 for a Benchmark). This chapter implements a dimensionality reduction layer using Principal Component Analysis to solve these issues.

## 5.1 Dimensionality Reduction of H

As we have seen in Section 2.3.5, the standard ESN approach to training is to apply a type of *least squares*. In the readout (Eq. 3.2) $\mathbf{W}_{\text{out}}$ is an optimized linear transformation taking hidden state vectors $\mathbf{h}_t \in \mathbb{R}^{N_{\text{hidden}}}$ to the output vector $\mathbf{y}_t \in \mathbb{R}^{N_{\text{output}}}$. In the spatial ESN that can be reshaped to 2D, i.e. $N_{\text{output}} = N_{\text{d1}} \times N_{\text{d2}}$ for visualization. Therefore, $\mathbf{W}_{\text{out}}$ is an $N_{\text{output}} \times N_{\text{hidden}}$ matrix, implying that $N_{\text{hidden}}$ parameters are trained for each pixel in the output.

The approach presents multiple issues of for the high-dimensional spatial ESN both in terms of *prediction performance* and *computational performance*, especially when higher spatial resolution is applied.

It is also common (including in the SESN) sue not only $\mathbf{h}_t$ as regressor, but to append to the hidden state matrix $\mathbf{H}$ the original input at that time step $\mathbf{x}_t$. This of course leads to *even more parameters* being trained.

#### Issue wrt. SESN Scalability

In Section 7.8 I show that this strategy become increasingly unfeasible wrt. time consumption when the hidden state dimension is increased. It simply takes a longer and longer time to optimize the amount of parameters (perhaps not a long time in comparison to gradient-based methods). This is the **main motivation** for this chapter in making the SESN more *scalable* in spatial dimensions.

#### Issues wrt. Predictive Performance

Before presenting the proposed means of dimensionality reduction that restricts the number of trained parameters significantly, this section describes what *other issues* the high-dimensionality may accompany.

- **Collinearity:** The spatial ESN has input that is *highly spatially correlated*. When covariates are correlated, least squares can become unstable since the trained parameters also become mutually dependent, and not uniquely determinable. This is an issue if the input is directly appended to $\mathbf{H}$ as described above. However, collinearities can *also* emerge from the multiplication of the large random sparse matrix $\mathbf{W}_{\text{hh}}$. In any case, it is best that covariates are *decorrelated* before applying least squares.

- **Underdetermination:** When $N_{\text{hidden}}$ is very large, and in the SESN setting often $N_{\text{hidden}} \gg N_{\text{train}}$ the optimization problem is *underdetermined* with no unique optimal solution, but with an infinite number of solutions. Recall from Chapter 2 that the aim of ML is *not* to over-fit the training set, but to fit *unseen observation*. Generalization performance is not stable when the number of features is extremely high (see also Sec. 2.2.5).

**Alternative Approaches in Machine Learning**

The issues of high dimensionality are common in image recognition problems. Convolutional neural networks (CNNs) address the issue by performing convolution(s) on the (transformed) images using kernels with fewer parameters than pixels of the input. Besides providing translational invariance (objects may be recognized at any position in the image), this can greatly reduce the number of trained parameters when compared to fully-connected (dense) neural networks. The CNN strategy is not easily incorporated into ESNs, however. Note that one type of *input map is* a convolution, however.
Another common strategy to avoid over-fitting is to use *early stopping*, which works by monitoring the validation (out-of-sample) error as the model is iteratively trained, and stop the training when the validation error reaches a minimum, instead of, say, the training (in-sample) error. Unfortunately, this strategy is not available with one-shot least squares optimization – the major benefit of ESNs. The same can be said of *dropout layers* in which neurons are deactivated at random during the training process to decrease the reliance of the network on single parameters.

The approach in this work, instead, will be to apply *Principal Component Analysis*. This strategy is also applied in ESNs by (Bianchi, Scardapane, et al. 2018). Their ESN is not created for predicting image sequences, however.

### *5.1.1 Principal Component Analysis*

Principal Component Analysis (Goodfellow et al. 2016, p. 143) is a linear method of dimensionality reduction that projects observations onto *principal components*, or orthogonal directions of greatest variance in the dataset, and removing correlation of variables. If the observations are stored in the $m$ rows of the $m \times n$ data matrix $\mathbf{X}$ these directions

are equivalent to the eigenvectors of the corresponding covariance matrix:

$$\mathbf{K}_{XX} = \frac{1}{m-1}\mathbf{X}^\top\mathbf{X} \tag{5.1}$$

And as such, the transform matrix $\mathbf{V}^\top$ can be found using the eigendecomposition (in practice, SVD is a more efficient approach):

$$\mathbf{X}^\top\mathbf{X} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\top \tag{5.2}$$

Where the columns of $\mathbf{V}$ contain the eigenvectors corresponding to the eigenvalues in diagonal matrix $\mathbf{\Lambda}$, sorted by magnitude. The PCA projection of data then is $\mathbf{Z} = \mathbf{V}^\top\mathbf{X}$. To achieve dimensionality reduction to only $N_{\text{PC}}$ principal components, $\mathbf{V}$ is truncated to the $m \times N_{\text{PC}}$ matrix $\tilde{\mathbf{V}}$, by throwing away directions corresponding to the smallest eigenvalues, and therefore the *least variance* according to Eq. (5.1). The projected data, then is simply $\tilde{\mathbf{Z}} = \tilde{\mathbf{V}}^\top\mathbf{X}$. Orthogonality of the principal components follows from the symmetry of the covariance matrix $\mathbf{K}_{XX}$.

### *5.1.2 Application of PCA in ESNs*

A simple indicator of whether the PCA dimensionality reduction or — in SVD-terms — the *low-rank matrix approximation* can be expected to be well-behaved is the spectrum of eigenvalues $\lambda$ of the covariance matrix $\mathbf{K}_{XX}$ that can be translated to the share of 'explained variability' that $N_{\text{PC}}$ principal components account for by normalizing and summing like so:

$$EV(N_{\text{PC}}) = \left(\sum_{i=1}^{m}\lambda_i\right)^{-1}\sum_{i=1}^{N_{\text{PC}}}\lambda_i \tag{5.3}$$

In that spirit, a real-world example is shown in Figure 5.1 produced from 10 random ESN initializations fed with the same 2D Mackey-Glass sequence computed from the singular values of hidden state matrices $\mathbf{H}$.

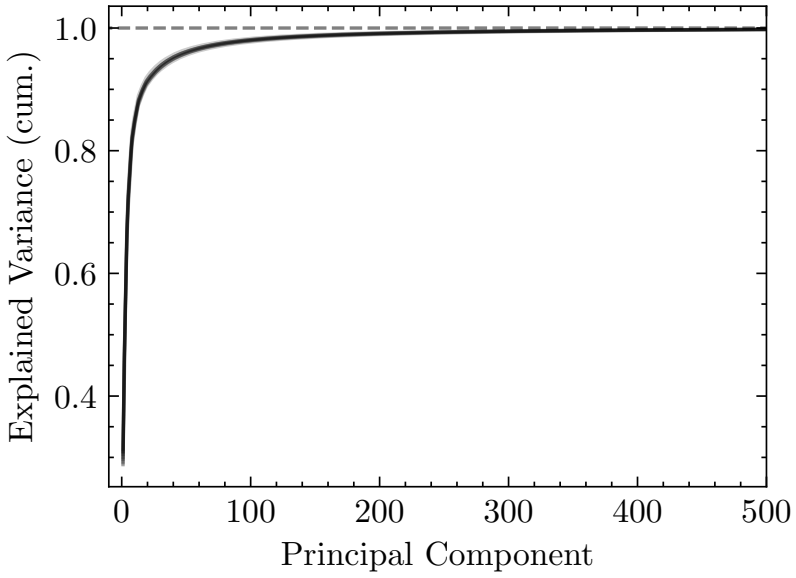



Figure 5.1: Cumulative explained variance of the first 500 principal components of hidden state matrices **H** for 30 × 30 2D Mackey-Glass orb training series with $N_{\text{train}} = 2000$, $N_{\text{hidden}} = 13625$. The plot shows 10 random initializations superimposed. On average, 500 of the 2000 principal components accounted for 99.768% of variance.

At the same time, including only eigenvectors corresponding to the largest eigenvalues of the covariance matrix reduced the condition number of the dimensionality reduced $\tilde{\mathbf{H}}$ over the original **H** by more than a factor 200 in every instance from around $\text{cond}(\mathbf{H}) \sim 4 \times 10^4$ to $\text{cond}\left(\tilde{\mathbf{H}}\right) \sim 2 \times 200$.

Selecting $N_{\text{PC}}$, the number of principal components to use, is not simply a matter of examining explained variance, however. In order to have the benefits of combating overfitting and the curse of dimensionality, I find that comparing the training *MSE* (from a small subset of states) to the MSE for unseen data is useful in determining whether to include more or less principal components. When **training error** is extremely small (effectively null) and out-of-sample MSE is not, it often helps to reduce the number of components to optimize the out-of-sample error - the true aim of a machine learning model. $N_{\text{PC}}$ becomes the *number of optimized parameters per pixel*. It is sensible to pick $N_{\text{PC}} \in (0; \min[N_{\text{hidden}}, N_{\text{train}}])$ at which the eigenvalues/variance can be non-zero. One should also remember to fit an *intercept*, i.e. append a value of e.g. 1 to the dimension reduced hidden states, $\tilde{\mathbf{h}}_t$.

### 5.1.3 Impact of PCA on the ESN Dynamical System

It should be stressed that $\mathbf{H}$ is the matrix consisting of harvested hidden states, each $N_{\text{hidden}}$-dimensional. The PCA layer comes *after* the dynamical system has been running, driven by the entirety of the dataset. The dynamical system (Eq. 3.1) is therefore *indifferent* to the use of PCA in a readout layer (Eq. 3.2) and prediction-time setting. Therefore the memory capabilities of the dynamical system are not affected.

After training, however, predictions are made from the linear readout layer on the reduced state $\tilde{\mathbf{h}}_t$

$$\tilde{\mathbf{h}}_t = \mathbf{V}^\top \left[ \mathbf{h}_t - \frac{1}{N_{\text{hidden}}} \sum_{i=1}^{N_{\text{hidden}}} \mathbf{H}_{(:,i)} \right] \tag{5.4}$$

$$\mathbf{y}_t = \mathbf{W}_{\text{out}} \tilde{\mathbf{h}}_t \tag{5.5}$$

Where $\frac{1}{N_{\text{hidden}}} \sum_{i=1}^{N_{\text{hidden}}} \mathbf{H}_{(:,i)}$ is the mean vector of hidden variables in the training states, a pre-processing step of PCA.

At this point, for free-running predictions, Eq. (3.1) dictates that the prediction, $\mathbf{y}_t$ is fed back to the network as input $\mathbf{x}_{t+1}$. Only in that case, PCA will directly affects the dynamical system. However, only the quality of the prediction is of significance to the stability of the ESN in free-running prediction mode. Which, apparently, PCA can improve dramatically by resolving issues related to high-dimensional ESNs, as discussed.

A benchmark is also available of the newly implemented SSESN with and without a PCA layer in Sec. 7.8.

# Chapter 6
# *A Spatially Sensitive Metric*

---

Supervised machine learning methods apply *optimization* to minimize their mistakes or *loss*. The loss metric is therefore one of the most crucial choices in ML applications. In this thesis, I explore *image sequences*, and that calls for a metric that has *spatial* or *spatio-temporal* sensitivity while maintaining computational *efficiency*. Such a method exists, the *IMED*.

Here, I describe and implement the IMED, and develop an efficient extension with more natural boundary effects for the use case, while extending the method to $n$-dimensional volumes.

Most importantly, I implement the method as *forward and backward* transforms that are *robust*, which allow the use of the method for *p problems*. Robustness was generally ensured with prior implementations.

The method also reduces the SSESN's dependency on image noise, which is crucial for long-term stability of free-running predictions (see E.g. Section 7.5).

## 6.1 You Get What You Ask For

In *business management* they speak of *Key Performance Indicators* (KPI). These are measures – qualitative or (often) quantitative – that managers tend to choose as indicators, and therefore goals, of the team or business that is managed.
For example, a company might want their marketing department to make as many calls to potential new clients as possible in order to acquire streams of revenue. The true aim, of course, is profit. But an indirect KPI might be *the number of calls made* by the marketing team. Such a KPI might have *unintended consequences* as good employees who want to keep their jobs will try to *maximize the KPI*. He or she will make as many calls a day as possible, and might rush through them at the risk of alienating potential customers, or might make calls to clients for whom the service or product is entirely irrelevant. The KPI, therefore, may be *accomplished*, but the ultimate objective of increasing profit is perhaps harmed. In short, *you get what you ask for*.

By analogy, KPIs of business management is not unlike the *loss functions of machine learning*. Minimizing loss is *all* that concerns our optimization tools, and that makes the loss function perhaps the most important design choice of a machine learning model.
Especially for spatial prediction, some type of *spatial sensitivity* of the metric may be needed to learn spatial features. It should also retain efficiency, and, in the context of Echo State Networks, remain a discretization of an $\mathcal{L}_2$ inner product such that least squares regression (as well as Principal Component Analysis, Support Vector Machines and Linear Discriminant Analysis) may be used for one-step optimization of the readout.
At a deeper level, the question, really, is *what is a good prediction?* The answer: *One that resembles the target, of course!* Trying to define resemblance, however is not easy, and my own 'Socratic monologue' seems to come up short.
In lack of loss functions with perfect perceptiveness of spatial features, we must resort to qualitative assessment of the predictions that have minimized some error measure. Which is all we *can* ask of machine learning methods: For the machine, a 'good' prediction is defined *only* as one where trained parameters minimize the loss function.
In this chapter, I present the current state of the metric called *IMED* and extend it to temporally sensitivity that guides machines to produce predictions that are more to our qualitative liking. It can *also* be used for comparison when determining *anomalies*, so it has multiple separate applications in this thesis.

## 6.2 Reading Guidance

In this chapter I describe several methods of implementing the IMED:

1. A naive (original) method using a very large matrix, in Sec. 6.4.
2. A version with tensor decomposition, in Sec. 6.5
3. Faster versions using convolution performed using Fourier transforms or the DCT (new!) in 6.6.1

The first two sections serve more as background, and can be *skipped*, as the SSESN of this thesis utilizes the frequency methods.

The most important contribution of this chapter, besides the implementation in a python package at GitHub[1], is a method for a **robust inverse transforms** that allow the use of IMED for regression problems even as the Gaussian parameter $\sigma$ is increased. The IMED is traditionally only used for classification problems; see Section 6.7.

Besides robust inverse transforms, the motivation for this chapter is to implement *efficient* versions of the IMED. I benchmark the different versions for one, two and three dimensional volumes in Section 6.8.

## 6.3 The $\mathcal{L}_2$ Norm and Euclidean Distance

The space of square-integrable functions, $\mathcal{L}_2$ is the *natural* space in which to discuss *least squares optimization* that ESNs apply (see Section 2.3.5). In the continuos case, least squares minimizes the (squared) $\mathcal{L}_2$ norm of the difference between training *predictions* and *targets*:

$$\langle f-g, f-g\rangle = \int_\Omega (f(x)-g(x))^* (f(x)-g(x))\,dx \tag{6.1}$$

where $f$ and $g$ can in this context be considered training predictions and *targets*, i.e. continuos functions that generate the images. $\Omega$ are rectangular regions that make up the rectangular continuos images. While the continuos description is useful in explaining the IMED, do note that in the discrete case the norm is the square *Euclidean*. For $n$-dimensional vectors it is:

$$\mathrm{d}^2(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_2^2 = (\mathbf{x}_1-\mathbf{y_1})^2 + (\mathbf{x}_2-\mathbf{y_2})^2 + \ldots + (\mathbf{x}_n-\mathbf{y_n})^2 \tag{6.2}$$

[1] IMED package *https://github.com/jfelding/IMED*

In optimization the square, above, is often used instead due to its smooth and differentiable parabolic as opposed to conic shape, see Figure 6.1.

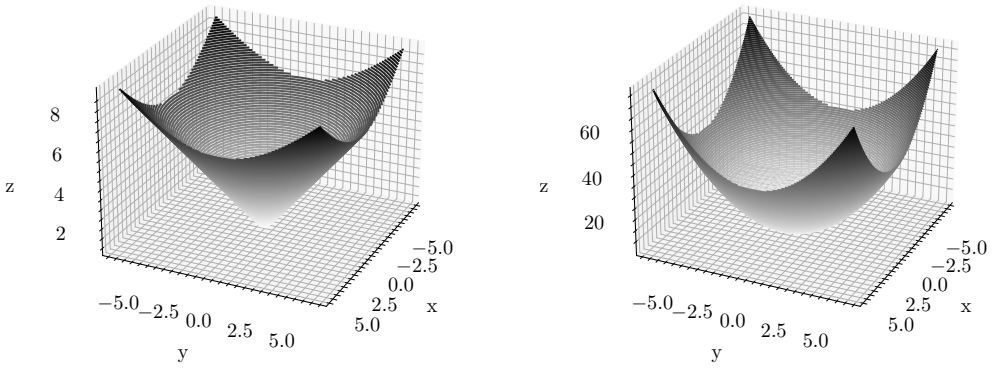
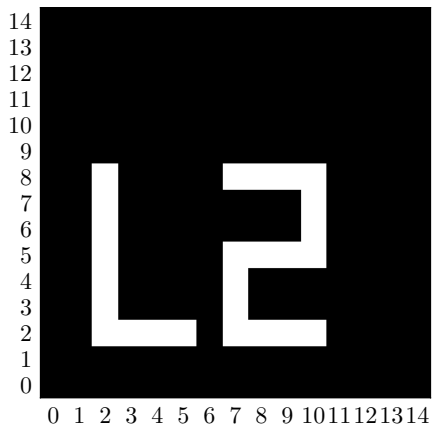


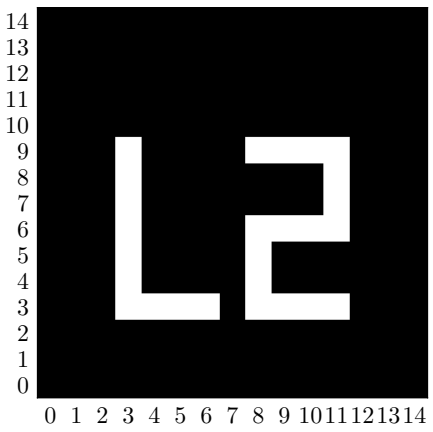

Figure 6.1: The squared Euclidean distance of Eq. (6.2) makes up a smooth paraboloid (right), whereas taking the square root (left) yields a cone that is not differentiable in the origin. The square retains its minimum (right)

In this text, I will implicitly refer to 2D images as $\mathbf{X}$ and its row-major flattened vector version as $\mathbf{x}$. Without further definitions, I will also allow myself to use the straightforward notation $\mathrm{d}(\mathbf{X}, \mathbf{Y}) = \mathrm{d}(\mathbf{x}, \mathbf{y})$.

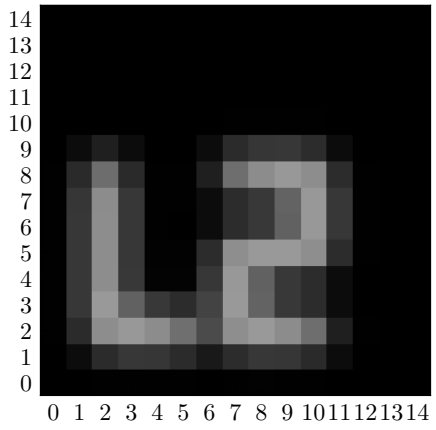
The discrete square norm computed using $\mathrm{d}^2$ is a loss functions that clearly does not take spatial and temporal dependencies into account in images or sequences of images. Conceptually, its pixel-wise nature makes optimization approaches focus on the largest pixel-to-pixel deviations instead of taking into account any spatial or spatio-temporal pattern that may be of much greater importance to the learning task. If one considers the image $\mathbf{X}$ to be a version of $\mathbf{Y}$ in which every pixel is displaced by just one to the right, a good spatial metric should recognize that the two versions are *similar*, but the Euclidean *does not*, as it does not consider any pixel relations as demonstrated in Figure 6.2 where the right column images are displaced versions of the right column images. We now move on to the Image Euclidean Distance; one that is more spatially sensitive.



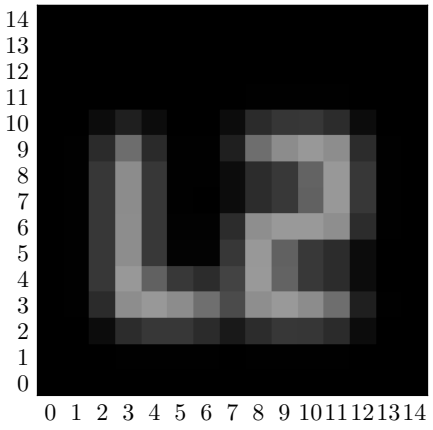
(a) Original image



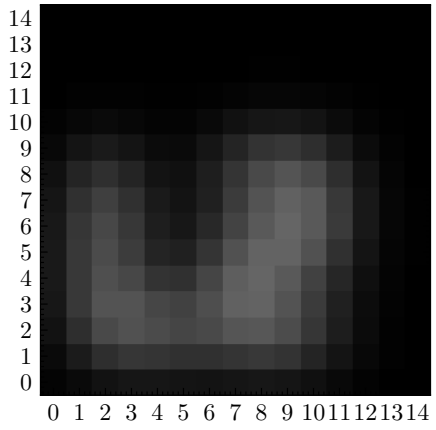
(b) Image displaced one up, one right



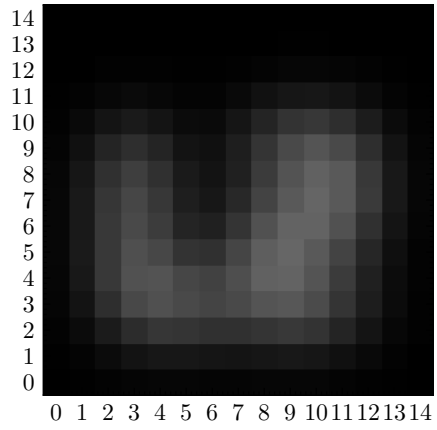
(c) Transformed image, $\sigma = 1$



(d) Transformed displaced image, $\sigma = 1$



(e) Transformed Image, $\sigma = 2$



(f) Transformed displaced image, $\sigma = 2$

Figure 6.2: $15 \times 15$ 2D images. Top row: originals created with binary pixel values of either 0 (in black) or 1 (in white). Right column shows the same image as to the left, but with white pixels displaced by one to the right, and one up. In the middle and bottom rows the top row images have been transformed using the linear convolution standardizing transform (Eq. 6.7) inherent to the IMED; with different values of the Gaussian parameter $\sigma$. The IMED is equivalent to performing the transform of both images, and taking the $\mathcal{L}_2$ pixel-wise distance. It is less sensitive to small displacements. All images shown in greyscale with values in $[0; 1]$. Distances between left and right images, from top to bottom rows:
48 (original $d^2$), 7.0 ($\sigma = 1$), 1.0 ($\sigma = 2$). It is, however, general that higher $\sigma$ lowers the values of the metric.

## 6.4 The Image Euclidean Distance Metric (IMED)

In (Liwei Wang et al. 2005) L. Wang, Y. Zhang and J. Feng introduced the *Image Euclidean Distance* (IMED). They proposed applying a continuous, monotonically decreasing and positive definite function $f$ to weigh the pixel-distance

$$f(|P_i - P_j|) \tag{6.3}$$

to take into account dependencies of neighbouring pixels. The article considered only spatial pixel-distances, but extends to $n$-dimensional images, though this was not implemented. A natural choice for $f$, they found, was the Gaussian
$f(i,j) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{(x_i-x_j)^2+(y_i-y_j)^2}{2\sigma^2}\right)$ where $(x_i, y_i)$ are coordinates of the pixel indexed by $i$ such that $x_i \in \{0,1,...,M-1\}$ and $y_i \in \{0,1...,N-1\}$. When $\mathbf{G}$ is a matrix whose $(i,j)^{\text{th}}$ element is $f(i,j)$, i.e.:

$$\mathbf{G}_{(i,j)} = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{(x_i-x_j)^2+(y_i-y_j)^2}{2\sigma^2}\right) \tag{6.4}$$

then the squared IMED — which is unscrupulously referred to it as the IMED for brevity — is defined as:

$$\mathrm{d}^2_{IMED}(\mathbf{X},\mathbf{Y}) = (\mathbf{y}-\mathbf{x})^\top \mathbf{G}(\mathbf{y}-\mathbf{x}) \tag{6.5}$$

The authors list three main advantages over the Euclidean distance.

- Sensitivity to small perturbations (as demonstrated in Fig. 6.2)
- Simplicity over other spatial similarity measures
- $\mathcal{L}_2$ compatibility allowing direct utility in existing image recognition methods

The authors focus only on application in classification methods, whereas this project will use the IMED as loss function for a **regression problem**, which has required work on robust deconvolution methods, as we shall see in Section 6.7.

#### IMED in the SESN and an Inverse Transform

The naive version of the IMED was implemented in the SESN of Niklas Heim and James Avery (Heim and Avery 2019; Heim 2021), and it is one of the computational limitations

for high-dimensional spatial applications that was left as future work to replace by more efficient versions.

The inverse transform that they used requires determining determining $\mathbf{G}^{-1}$ (also an $NM \times NM$ matrix), which can be done using the eigendecomposition of $\mathbf{G}$. This operation is $\mathcal{O}\left((NM)^3\right)$ for the symmetric matrix.

### 6.4.1 The Standardizing Transform

IMED authors noted that, equivalently to (6.5), each image (prediction and target in the regression case) can be transformed by the unique square root $\mathbf{G}^{1/2}$ for which:

$$\mathbf{G} = \left(\mathbf{G}^{1/2}\right)^{\top} \mathbf{G}^{1/2} = \mathbf{G}^{1/2}\mathbf{G}^{1/2} \tag{6.6}$$

Using this factorization, the discrete IMED can be rewritten

$$\begin{aligned} \mathrm{d}^2_{IMED}(\mathbf{X}, \mathbf{Y}) &= \left[(\mathbf{y}-\mathbf{x})^{\top} \mathbf{G}^{1/2}\right]\left[\mathbf{G}^{1/2}(\mathbf{y}-\mathbf{x})\right] \\ &= \left[\left(\mathbf{G}^{1/2}\mathbf{y} - \mathbf{G}^{1/2}\mathbf{x}\right)^{\top}\right]\left[\left(\mathbf{G}^{1/2}\mathbf{y} - \mathbf{G}^{1/2}\mathbf{x}\right)\right] \\ &= (\mathbf{y}_{\mathrm{ST}} - \mathbf{x}_{\mathrm{ST}})^{\top} (\mathbf{y}_{\mathrm{ST}} - \mathbf{x}_{\mathrm{ST}}) \end{aligned} \tag{6.7}$$

such that the IMED is simply the pixel-wise Euclidean distance of transformed images, with ST subscripts referring to:

$$\mathbf{x}_{\mathrm{ST}} \equiv \mathbf{G}^{1/2}\mathbf{x} \tag{6.8}$$

which authors named the **standardizing transform** (ST).

The standardizing transform, then, can be incorporated in supervised or unsupervised algorithms as a *preprocessing step* to input and/or targets to obtain the IMED without changing any loss functions directly. There is a but, however, as determining $\mathbf{G}^{1/2}$ (also) requires eigendecomposition of $\mathbf{G}$. The unique positive square-root of a positive-definite matrix like $\mathbf{G}$ can be found by its eigendecomposition (the decomposition of $\mathbf{G}$ has to be computed only once):

$$\begin{aligned} \mathbf{G} &= \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\top} \\ \mathbf{G}^{1/2} &= \mathbf{U}\mathbf{\Lambda}^{1/2}\mathbf{U}^{\top} \end{aligned} \tag{6.9}$$

Where the columns of $\mathbf{U}$ are eigenvectors of $\mathbf{G}$ with corresponding eigenvalues along the diagonal of $\mathbf{\Lambda}$. $\mathbf{\Lambda}^{1/2}$ is its (unique) positive root, i.e. $\mathbf{\Lambda} = \left(\mathbf{\Lambda}^{1/2}\right)^{\top} \mathbf{\Lambda}^{1/2} = \mathbf{\Lambda}^{1/2}\mathbf{\Lambda}^{1/2}$ that is simply found by taking the element-wise square root of $\mathbf{\Lambda}$, since it is a diagonal matrix.

### 6.4.2 A Naive Inverse Standardizing Transform

The inverse transform, needed for predictions in real-space using IMED as a loss function, requires the inverse operation, which I shall denote the *inverse standardizing transform* (iST):

$$\mathbf{x} = \mathbf{G}^{-1/2}\mathbf{x}_{\mathrm{ST}} \tag{6.10}$$

Where $\mathbf{G}^{-1/2}$ can be determined by the eigendecomposition of $\mathbf{G}$ in the same fashion.

#### Implementation

In the `IMED` python 3 package, the naive IMED described above is available as a forward and backwards standardizing transform. Please see the GitHub code[2] for the function `ST_fullMat()`. It is the work by Niklas Heim and James Avery and Jacob Felding (a few fixes).

## 6.5 IMED by Kronecker Product Decomposition

The $\mathbf{G}$ matrix with $(NM)^2$ elements can be decomposed as a Kronecker product. This was noted by IMED authors (Liwei Wang et al. 2005), and developed in (B. Sun et al. 2008). Kronecker product decomposition implies:

$$\mathbf{G} = \mathbf{G}_{\hat{x}} \otimes \mathbf{G}_{\hat{y}} \tag{6.11}$$

using $N^2$ and $M^2$ matrices $\mathbf{G}_{\hat{x}}, \mathbf{G}_{\hat{y}}$ respectively . Each is real and symmetric, depending only on the respective subscript coordinate of pixels, i.e.

$$\mathbf{G}_{\hat{x}} = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i - x_j)^2}{2\sigma^2}\right) \tag{6.12}$$

$$\mathbf{G}_{\hat{y}} = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - y_j)^2}{2\sigma^2}\right) \tag{6.13}$$

The separability of $\mathbf{G}$ comes about from the product rule for exponents:

$$\mathbf{G}_{(i,j)} = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i - x_j)^2}{2\sigma^2}\right) \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - y_j)^2}{2\sigma^2}\right) \tag{6.14}$$

$$= g(x_i - x_j)g(y_i - y_j) = (\mathbf{G}_{\hat{x}} \otimes \mathbf{G}_{\hat{y}})_{(i,j)} \tag{6.15}$$

[2] Implementation of the naive IMED *https://github.com/jfelding/IMED/blob/master/IMED/spatial_ST.py*

Two diagonalizations are now required to obtain the ST, but on matrices sized only $M \times M$ and $N \times N$ instead of a single one, sized $(MN \times MN)$.

$$\mathbf{G}^{1/2} = \mathbf{G}_{\hat{x}}^{1/2} \otimes \mathbf{G}_{\hat{y}}^{1/2} = \left(\mathbf{U}_x \mathbf{\Lambda}_x^{1/2} \mathbf{U}_x^\top\right) \otimes \left(\mathbf{U}_y \mathbf{\Lambda}_y^{1/2} \mathbf{U}_y^\top\right) \tag{6.16}$$

To avoid constructing the memory-consuming $\mathbf{G}^{1/2}$ the so-called *vec trick* is applied such that the standardizing transform is:

$$\mathbf{X}_{\text{ST}} = \mathbf{G}_{\hat{x}}^{1/2} \mathbf{X} \mathbf{G}_{\hat{y}}^{1/2} \tag{6.17}$$

To be clear, the vectorization trick is the following observation. A matrix equation:

$$\mathbf{AXB} = \mathbf{C} \tag{6.18}$$

can be rewritten as:

$$\left(\mathbf{A} \otimes \mathbf{B}^\top\right) \mathbf{x} = \mathbf{AXB} \tag{6.19}$$

As before, the inverse standardizing transform can be obtained by computing $\mathbf{G}_{\hat{x}}^{-1/2}$, $\mathbf{G}_{\hat{y}}^{-1/2}$.

#### Extensibility to $n$ Dimensions, and Edge Effects

Kronecker separation largely solves the computational issues of the original IMED method that was applied in (Heim 2021). While it is not implemented, Kronecker decomposition shows that the method is extendible to $n$ dimensions. The standardizing transforms also both yield certain **edge effects** (it is identical to convolution with zero-padding at borders). Other types of convolution than this 'linear convolution' (i.e. periodic/symmetric conv.) provides other edge effects.

#### Time Complexity

For the implementation as part of a spatial echo state network, the IMED should be very *efficient*, and a robust method for *inverse standardizing transform* must also be created. Currently, the matrix methods are not robust, since they rely on discrete diagonalization that does not give precise results (like slightly negative eigenvalues that cannot exist). The time complexities are:

- Naive (initial) method: $\mathcal{O}\left((MN)^3\right)$ in 2D, $\mathcal{O}\left((MNT)^3\right)$ if 3D, and so on,

- The Kronecker decomposition method is: $\mathcal{O}\left(M^3\right) + \mathcal{O}\left(N^3\right) + \mathcal{O}\left(T^3\right) + \ldots$
- Convolution using Fourier Transform is *another way* to implement the IMED (Bing Sun et al. 2015) is $\mathcal{O}(M \log M) + \mathcal{O}(N \log N) + \mathcal{O}(T \log T) + \ldots$

Therefore, I now turn to implementing and extending frequency representation methods for the IMED.

#### Implementation

For the implementation of Kronecker Decomposition IMED, see GitHub code[3] with the function `ST_sepMat()`.

## 6.6 Standardizing Transforms Using Frequency Representations

In (Bing Sun et al. 2015) authors showed that the standardizing transform is a convolution operation that can be performed using frequency representations (Fourier transform). This provides better scalability and – for the purpose of this thesis – avoids diagonalization that is not robust enough in practice for the *inverse transform* necessary for regression problems (see Sec. 6.7).

### *6.6.1 IMED by Fourier Transform*

The standardizing transform is in fact a convolution operation. By using the convolution theorem for Fourier transformations, one can see that, in the continuous case, it allows us to obtain the standardizing transform by taking the square root of the Fourier transform of a Gaussian, which is itself *another* Gaussian. First, I introduce the Fourier transform and that of the Gaussian.

#### The Fourier Transform

There are many slightly different conventions related to the forward and backward (inverse) Fourier transforms, including the use of different signs, normalization constant and frequency $k$ or angular frequency $\omega = 2\pi k$ Here, I simply chose:

$$H(k) = \mathcal{F}(h(x))(k) = \int_{-\infty}^{\infty} h(x) e^{2\pi i k x} \, \mathrm{d}x \tag{6.20}$$

[3]Implementation of Kronecker IMED: *https://github.com/jfelding/IMED/blob/master/IMED/spatial_ST.py*

such that the inverse transform is:

$$h(x) = \mathcal{F}^{-1}(H(k)) = \int_{-\infty}^{\infty} H(k) e^{-2\pi i k x} \, \mathrm{d}k \tag{6.21}$$

**The Fourier Transform of a Gaussian is Another Gaussian**

The Gaussian has the special property that its Fourier transform is another Gaussian function with a spread that is inversely proportional to the original Gaussian. The mean-zero Gaussian is defined:

$$g(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right) \tag{6.22}$$

Its Fourier transform is:

$$\mathcal{F}(g(x))(k) = \exp\left(-2\pi^2\sigma^2 k^2\right) \tag{6.23}$$

Or in terms of angular frequency, $\mathcal{F}(g(x))(\omega) = \exp\left(-\sigma^2\omega^2/2\right)$. Not only is Eq. 6.23 **another Gaussian**, but notably the Fourier transform is **real**.

The square root of the Fourier Gaussian that is relevant to the IMED is:

$$\sqrt{\mathcal{F}(g(x))(k)} = \left[\exp\left(-2\pi^2\sigma^2 k^2\right)\right]^{1/2} = \exp\left(-\pi^2\sigma^2 k^2\right) \tag{6.24}$$

or in terms of angular frequency, $\sqrt{\mathcal{F}(g(x))(\omega)} = \exp\left(-\sigma^2\omega^2/4\right)$.

**ST by Convolution in Fourier Space**

Returning to carrying out convolution in Fourier space, the *convolution theorem* states that it is accomplished using element-wise multiplication of transforms in Fourier space:

$$f * h = \mathcal{F}^{-1}(\mathcal{F}(f) \cdot \mathcal{F}(h)) \tag{6.25}$$

Treating the one-dimensional case for simplicity, we can recreate the IMED as the inner product $\langle z, g(x) * z \rangle$ when $z$ is the image difference $z = y - d$. Convolution is defined:

$$(g * z)(x) = \int_{-\infty}^{\infty} \mathrm{d}x' g(x - x') z(x') = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{\infty} \mathrm{d}x' \exp\left(-\frac{1}{2}\frac{(x - x')^2}{\sigma^2}\right) z(x') \tag{6.26}$$

Then, the one-dimensional IMED is the inner product:

$$\langle z, g * z \rangle = \int_{-\infty}^{\infty} \mathrm{d}x \, z^*(x)(g * z)(x) = \int_{-\infty}^{\infty} \mathrm{d}x z^*(x) \int_{-\infty}^{\infty} \mathrm{d}x' \, g(x - x') z(x') \tag{6.27}$$

We now exploit the fundamental property of the Fourier transform that it is *unitary* on $\mathcal{L}_2$, such that the inner product is conserved, i.e.:

$$\langle z, g * z\rangle = \langle \mathcal{F}(z), \mathcal{F}(g * z)\rangle = \int_{-\infty}^{\infty} \mathrm{d}k \, (\mathcal{F}(z))^* \, \mathcal{F}(g * z) = \int_{-\infty}^{\infty} \mathrm{d}k \, (\mathcal{F}(z))^* \, \mathcal{F}(g) \, \mathcal{F}(z) \tag{6.28}$$

Therefore, the squared IMED can be calculated by determining the Fourier transforms of the image difference $z$ and the kernel $g$, multiplying the transforms element-wise; then summing all elements.

Using $\mathcal{F}(g)$ defined in Eq. (6.23), the standardizing transform can be implemented in Fourier space by taking the square root, noting that $\mathcal{F}(g)(k)$ is real and non-negative:

$$\langle z, g * z\rangle = \langle \mathcal{F}(z), \mathcal{F}(g * z)\rangle = \int_{-\infty}^{\infty} \mathrm{d}k \left(\mathcal{F}(z)\sqrt{\mathcal{F}(g)}\right)^* \left(\sqrt{\mathcal{F}(g)}\mathcal{F}(z)\right) \tag{6.29}$$

In complete analogy to Eq. (6.7). The straightforward separation in Fourier space is not allowed in the real-space version of Eq. (6.27) as $\sqrt{g(x-x')}$ cannot be removed from the rightmost integral w.r.t. $x'$. But in Fourier space, this allows rewriting the standardizing transform:

$$\mathbf{Z}_{\mathrm{ST}} = \mathrm{vec}^{-1}\left(\mathbf{G}^{1/2}\mathbf{z}\right) = \mathcal{F}^{-1}\left(\mathcal{F}(\mathbf{Z}) \cdot \sqrt{\mathcal{F}\left(g_{xy}\right)}\right) \tag{6.30}$$

where $g_{xy}$ is a multidimensional version of the Gaussian.

### Discrete Case and Practical Considerations

The *continuos Frequency Gaussian* can be sampled for the practical, discrete case. This largely avoids numerical issues as seen in Figure 6.3. The image or data $\mathbf{Z}$ is discrete and finite, and can be transformed using the fast Fourier transform (FFT), an efficient implementation of the discrete Fourier transform (DFT), whose forward and backwards transforms are shown below, respectively:

$$H_k = \sum_{n=0}^{N-1} h_n \exp\left(\frac{2\pi i k n}{N}\right) = \sum_{n=0}^{N-1} h_n \left[\cos\left(\frac{2\pi k n}{N}\right) + i \sin\left(\frac{2\pi k n}{N}\right)\right] \tag{6.31}$$

$$h_n = \frac{1}{N}\sum_{k=0}^{N-1} H_k \exp\left(-\frac{2\pi i k n}{N}\right) = \frac{1}{N}\sum_{k=0}^{N-1} H_k \left[\cos\left(\frac{2\pi k n}{N}\right) - i \sin\left(\frac{2\pi k n}{N}\right)\right] \tag{6.32}$$

Here, $n$ is an index to the real space signal $h_n$ with $n \in \{0, 1, ..., N-1\}$.

When the convolution theorem is applied using discrete Fourier transforms, the $N$-periodicity

of the discrete transform implies that convolution is *periodic*, and it is known as **circular convolution**. This is not the case for continuos and non-finite transforms, as the period will be *infinite*.

Up to a normalization constant, the DFT is identical to the Fourier Series coefficients of an $N$-periodic function. The FFT of a signal or an image that violates this assumption can therefore exhibit edge effects that occurs due to the discontinuity between the first and last point in the signal. To avoid the discontinuity, one can extend the signal with the mirror signal, so that it becomes $2N$-periodic.

This is identical to performing a **discrete cosine transform** (DCT), producing **symmetric convolution** whose edge effects are often more natural for image applications including in a regression problem context. In Section 6.6.2 the IMED is therefore extended to DCT representations.

#### Frequency Representation of the IMED *Filter*

In Eq. (6.24) I have derived the analytical frequency filter, which is preferable to application of the DFT. To illustrate the differences, I plot in Figure 6.3 the real-space Gaussian, its analytical and discrete transforms. We now see that the filter will be a Gaussian that narrows as $\sigma$ is increased, the opposite of the real space Gaussian. Further, the DFT of the Gaussian kernel is complex (real and imaginary parts shown separately), which is not a property of the true transform. The analytical filter is already normalized to unity, but this is not the case for the DFT, so some care must be taken if that is applied. The analytical transform is superior, however, and is what is used in the provided IMED software. It is unknown whether authors of e.g. (Bing Sun et al. 2015) have applied the DFT or analytically sampled filter.

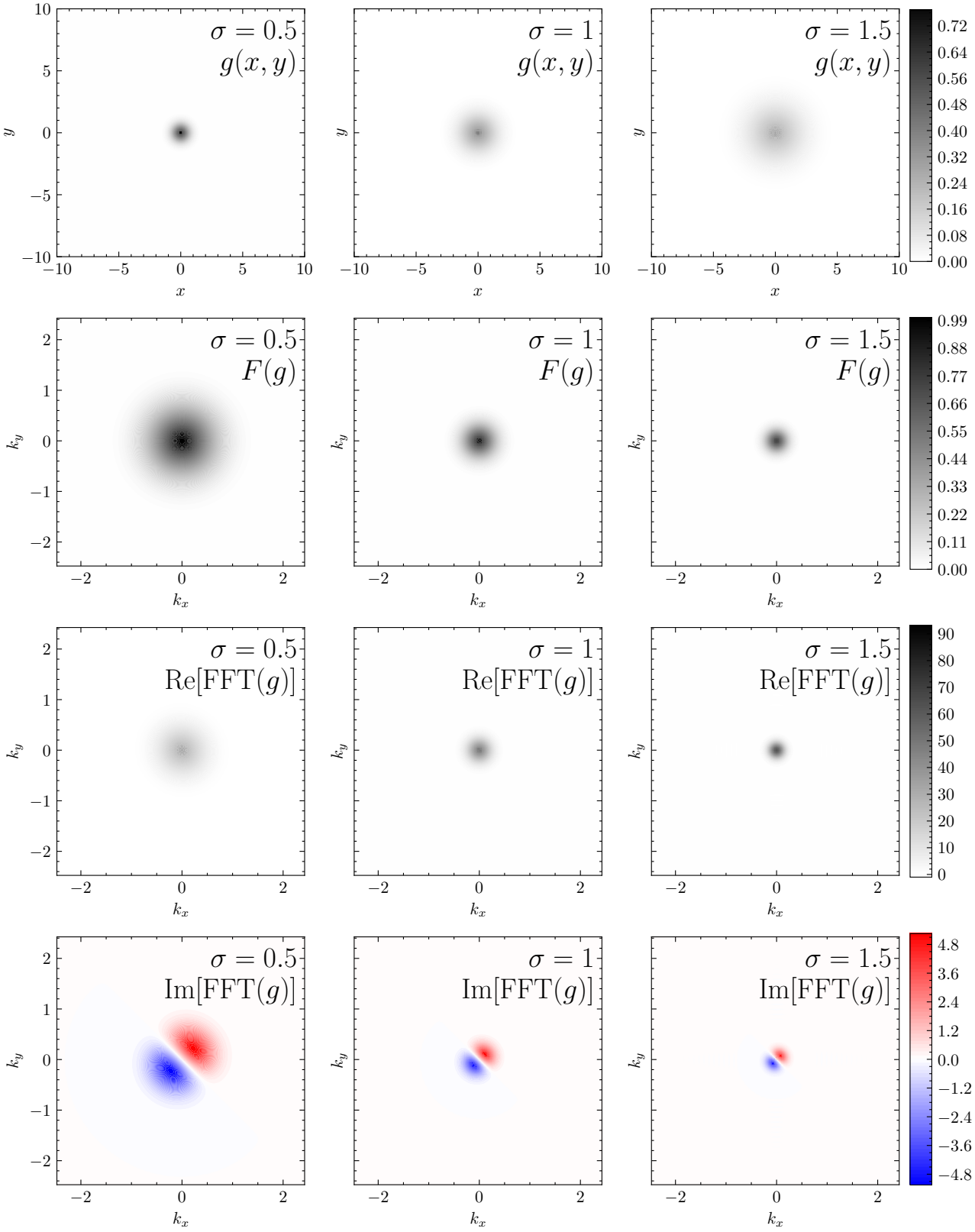



Figure 6.3: Two-dimensional Gaussians. From top row and down: real space Gaussian, corresponding analytical Fourier transformed Gaussian, discrete Fourier transformed Gaussian (real part), discrete Fourier transformed Gaussian (imaginary part); for different values of $\sigma$, as shown. Imaginary part is a property of the numerical method, not of the Fourier Gaussian. Every transform has unique color scale. DFT algorithm used is `scipy.fft2`. In the case of a Gaussian frequency filter, it is both more efficient and accurate to sample directly from the analytical transform. If the DFT version is used as frequency filter, it should be normalized to unity, and I would personally disregard the imaginary part.

### 6.6.2 IMED by Discrete Cosine Transform

**Continuos Cosine Transform**

The cosine transform, in the continuos case is related to the Fourier transform by Euler's formula, and for a real function $h$ is the real part of the Fourier transform, keeping only cosine functions as basis:

$$H^c(k) = \mathcal{F}_c\left(h(x)\right)(k) = \int_{-\infty}^{\infty} h(x)\cos\left(2\pi k x\right)\mathrm{d}x \tag{6.33}$$

And the transform is *even*. The Gaussian of Eq. (6.22) is *also* even, so the cosine transform of the Gaussian is *identical* to its Fourier counterpart, as in Eq. (6.23).

**Discrete Cosine Transform, Type I**

Whereas the Discrete Fourier Transform is equal, up to a normalization constant, to the Fourier series coefficients for an N-periodic signal, the discrete cosine transform type I (DCT-I) $H_k^c$ of a signal $h_n$ is identical to the DFT of a symmetrically extended signal truncated by two points given the definition:

$$H_k^c = h_0 + (-1)^k h_{N-1} + 2\sum_{n=1}^{N-2}\cos\left(\frac{\pi k n}{N-1}\right) \tag{6.34}$$

With the definition in Eq. (6.34), *the DCT-I is its own inverse*.

A benefit of the DCT over the DFT is that it only samples positive frequencies, increasing the resolution. This is due to the evenness of the cosine, i.e. $\cos(-x) = \cos(x)$. The Fourier transform of a real signal instead exhibits *conjugate symmetry* such that negative-frequency terms are the complex conjugates of their positive counterparts, i.e. $H_{-k} = H_k^*$, and half the samples of the DFT are thus spent on these negative-frequency terms.

## 6.7 Deconvolution And the Inverse Standardizing Transform for SESNs

### 6.7.1 An Ill-Conditioned Problem

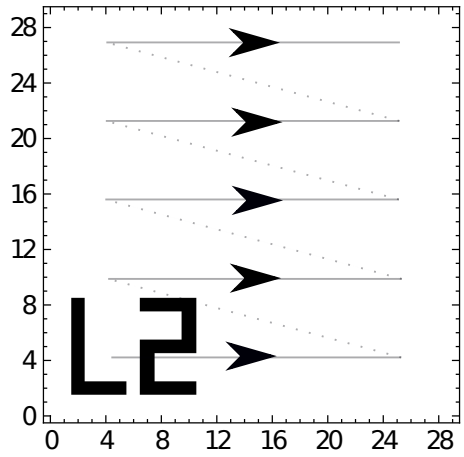
As I have stated, in regression problems like in spatial ESNs the IMED can be applied as a **pre-processing** and **post-processing** step such that regression is carried out in the standardized-transform space, but the inverse transform is used to 'restore' the predictions to the original space. Since the standardizing transform is *convolution*, the inverse

transform is **deconvolution**. In the presence of any noise this is an ill-conditioned problem, and with frequency methods the noise will be catastrophically amplified, as we shall see.

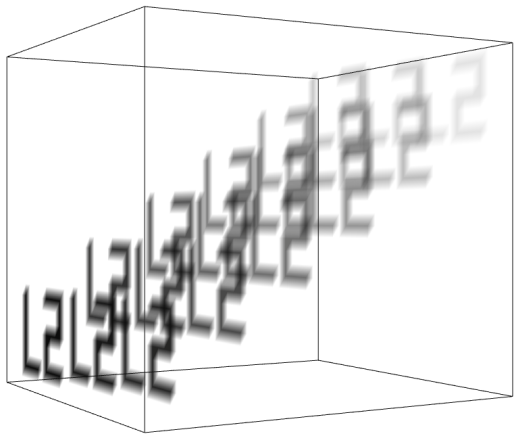
In this section I therefore also develop a simple, but surprisingly robust method for the inverse transform, which exploits that the *frequency filter* is *controllable* (unlike the point-spread function for e.g. cameras).

### *6.7.2 Naive Inverse Transforms*

In Fig. 6.4, I show the 3D volume that I will use throughout this section to demonstrate the deconvolution performance of frequency and matrix implementations of the inverse standardizing transform.




(a) The dataset used to estimate deconvolution restoration errors is a 3D volume in which the 'L2' text (value 1) is sliding horizontally and jumping vertically (arrows for illustration only)



(b) Slices of the 3D simulated dataset in which the 'L2' text slides across the 2D image over time (first horizontally, then vertically). Only every second sample is shown. Created using PyVista.

Figure 6.4: Illustrations of the Dataset for ST and iST restoration errors. I refer to the volume as the tensor $\chi$ with dimensions $(T, N_x, N_y) = (36, 30, 30)$. The 36 images are very discontinuous (only binary values 0 and 1), and Gibbs effects can therefore be expected from the frequency methods.

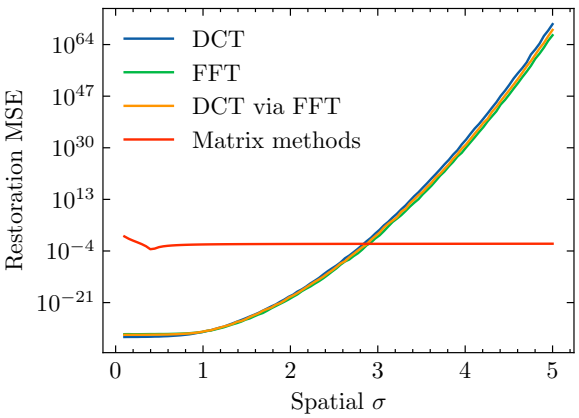
#### Naive Spatial-Only Restoration

In Figure 6.5 I show the performance of the naive inverse transforms on a spatially transformed volume (in Fig. 6.4). I refer to *restoration error* as the pixel-wise *mean squared error* between the original volume $\chi$ and restored volume $\mathrm{ST}^{-1}(\mathrm{ST}(\chi))$. For the sake of

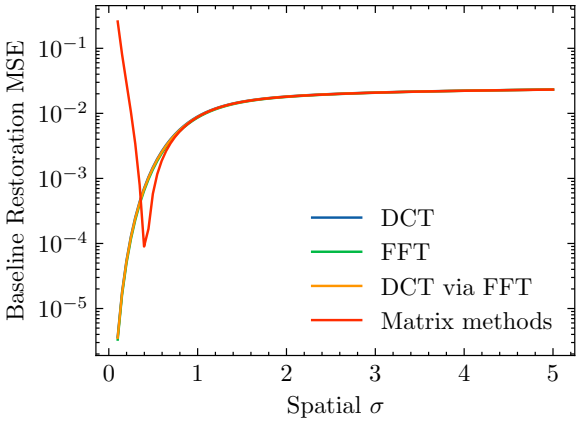
comparison, I also show a 'baseline' restoration error, which is the *MSE* of $\chi$ and to $\mathrm{ST}(\chi)$. All at different values of $\sigma$ (spatial only).

It is interesting to see in the restoration MSE that frequency methods are extremely well-behaved at small values of $\sigma$, but that the restoration MSE explodes as it is increased. In frequency space, $\sigma$ has the inverse interpretation of a Gaussian spread in real space (See Figure 6.3). Therefore, it is reasonable that frequency methods are very well-behaved when $0 \leq \sigma \leq 1$ where the frequency Gaussian (Eq. 6.24) widens, and more frequencies in the transformed volume $\mathcal{F}(\mathrm{ST}(\chi))$ are divided by larger values. This is opposed to when $\sigma > 1$ and the frequency Gaussian very quickly drops towards zero, amplifying numerical noise by extreme amounts.

We see that the inverse matrix methods are more well-behaved, achieving about $10^{-2}$ restoration MSE somewhat independently of $\sigma$. The matrix methods are convolutions in real space, and as such they are more well-behaved when $\sigma \geq 1$. When $0 < \sigma < 1$ the normalization constant of the Gaussian that includes $\sigma^{-1}$ makes the Gaussian as a whole increase dramatically, also amplifying noise. In the baseline restoration MSE we see this effect for the matrix methods.




(a) Restoration MSE computed using original volume and $\mathrm{ST}^{-1}(\mathrm{ST}(\chi))$.

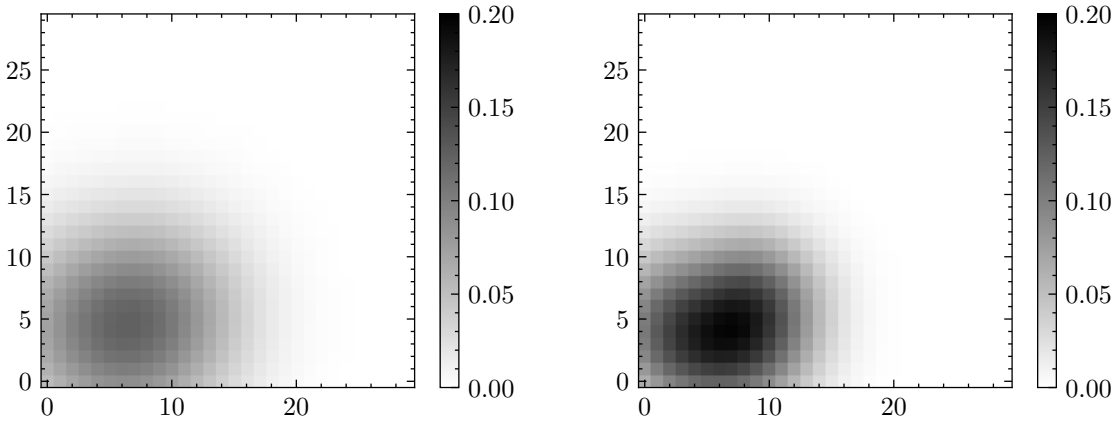



(b) As a baseline measure, I here apply no iST to view the effect of the forward ST on the volume. In a prediction setting, it is still necessary to restore images to the original space to ensure the quality of predictions.

Figure 6.5: Restoration errors based on spatial-only standardizing transforms of a 3D volume illustrated seen in Fig. 6.4. Restoration errors are *MSE* of original volume and transformed volume. A method to avoid noise amplification is needed for the inverse transform of frequency methods.

Even for the well behaved case of the matrix methods, at $\sigma = 5$ a restoration MSE of $2.5 \times 10^{-2}$ is achieved, but the results are not at all sufficient in a prediction setting. Fig.

6.6a shows what the restored sample looks like, compared to the standardizing transform baseline in Fig. 6.6b.




(a) 'Restored' version of the first slice of the volume after spatial standardizing transform followed by inverse transform.

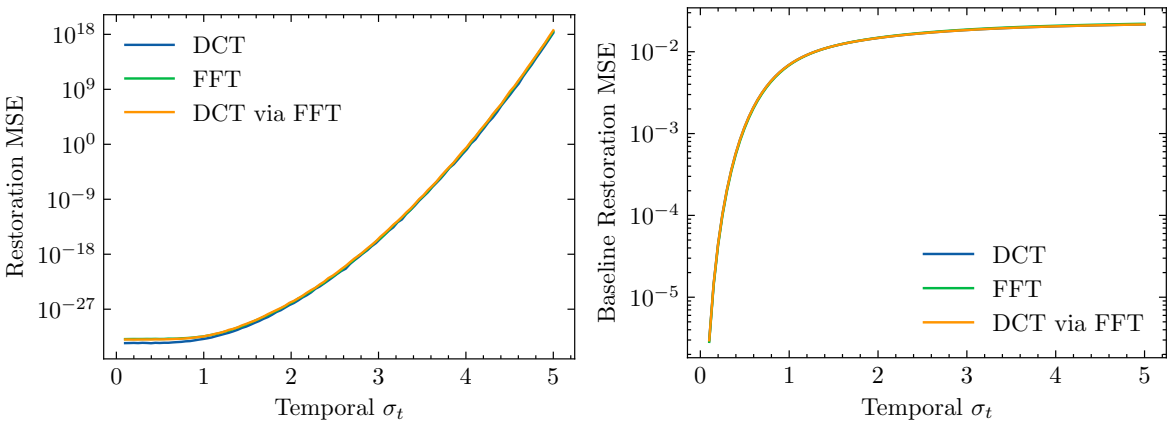
(b) Baseline. First slice of the volume after spatial standardizing transform.

Figure 6.6: While matrix methods achieve better results in the naive spatial ST restoration process with restoration MSE $2.5 \times 10^{-2}$, at $\sigma = 5$ the results, as seen in (a) are hardly sufficient for most applications. The original text 'L2' is illegible.

### Naive Temporal-Only Restoration

As the matrix methods are two-dimensional in nature, they are not applied in this section and when discussing spatio-temporal restoration.

In Figure 6.7a I show the restoration performance of frequency methods for the temporal standardizing and inverse standardizing transforms, and in Fig. 6.7b.

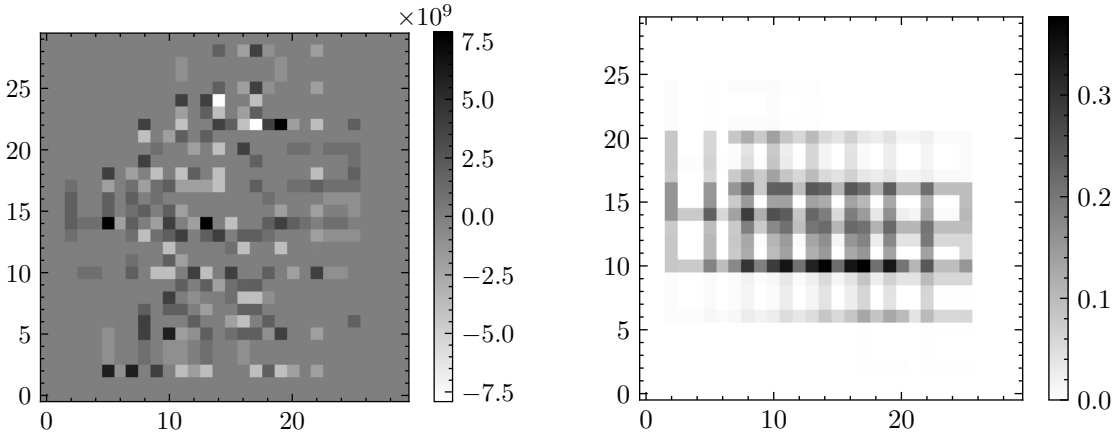



(a) Temporal restoration MSE computed using original volume and $\mathrm{ST}^{-1}(\mathrm{ST}(\chi))$.

(b) Baseline measure as in Fig. 6.5b, only its temporal version.

Figure 6.7: Restoration errors based on temporal-only standardizing transforms of a 3D volume illustrated seen in Fig. 6.4, i.e $\sigma_t > 0$, $\sigma = 0$. Matrix methods not included due to 2D nature.

Again, I provide an example of the restoration at $\sigma_t = 5$ in Fig. 6.8a along with the standardizing transform itself in Fig. 6.8b. Clearly, the frequency restorations continue to cause major restoration issues, although to a smaller degree than the spatial-only transforms. I attribute this difference to the higher dimensionality of the spatial as opposed to temporal transforms.




(a) 'Restored' version of the 15th slice of the volume after temporal standardizing transform followed by inverse transform.

(b) Baseline. 15th slice of the volume after temporal standardizing transform.

Figure 6.8: Example of restoration of temporal standardizing transform ($\sigma_t = 5$) using DCT method. The naive deconvolution approach is clearly not sustainable as $\sigma_t$ is increased.

**The Trouble with Inverse Filtering**

The minimum decreases exponentially with $\sigma$, of course, and the divisions involved in inverse filtering become smaller and smaller.

### *6.7.3 The Wiener Filter*

A well-known, simple method for noisy deconvolution is *the Wiener filter*. It should be considered, as it is not overly computationally intensive in comparison to iterative restoration methods that are more commonly used today.

As we have seen, noisy deconvolution must be handled with care for the Gaussian filter that rapidly approaches nullity. The Wiener filter is:

$$Y(k) = \frac{H^*(k)}{|H(k)|^2 + NSR(k)} \tag{6.35}$$

Where $H$ is the original filter (the frequency representation of the Gaussian) and $NSR$ is the frequency representation of the noise-to-signal ratio. White noise, for instance, has a flat power spectrum of simply $\sigma_N^2$, whereas red noise leads to $\frac{S_0}{\omega^2}$ where $\omega$ are angular frequencies. In both cases, a constant must be estimated.

It is seemingly paradoxical that the signal one is trying to estimate by inverse filtering must be known in advance. We find, however, that an average power spectrum of the *training* set can be somewhat successfully applied when the spatio-temporal data is sufficiently continuous. Results are far from perfect, however, exhibiting artefacts that are known to occur with Wiener deconvolution.
The need to estimate the signal power spectrum also **prohibit long-term prediction in an ESN prediction context**, since the power spectrum of the *prediction set targets* deviate more and more from that of the training set, as time goes on.

The noise amplification is largely dependent on the inverse filter. When $\sigma > 1$ the frequency representation of the Gaussian will be narrower (in k) than in real-space (in x). Therefore, it quickly tends towards zero, and during inverse filtering by division any noise is amplified; sometimes by hundreds of orders of magnitudes.

### *6.7.4 A Simple Solution*

Wiener deconvolution is an imperfect method that does not suffice in a free-running prediction setting. A simple approach, then, is to solve the extreme noise amplification inherent to inverse filtering by **avoiding division by near-zero values**. By comparing Table 6.1 and e.g. Figure 6.5a, we see that even at $\sigma = 2$ at which point the filter has a minimum of $\sim 10^{-7}$, inverse filtering works remarkably well. Therefore, adding a small constant as in Eq. (6.36) to *both* the filter and the inverse filter (in $\mathrm{ST}(\cdot)$ and $\mathrm{ST}^{-1}(\cdot)$), we find that the worst noise amplification issues of deconvolution may be mitigated.

$$\tilde{g}(k) = \sqrt{\mathcal{F}(g)(k)} + \epsilon, \quad \epsilon > 0 \tag{6.36}$$

It is pertinent to stop and ask at this point what ramifications a small constant may have on the IMED: Of course, the distance measure may be altered slightly (recall that only

the forward $\mathrm{ST}(\cdot)$ is necessary for distance evaluation), but IMED authors in (Liwei Wang et al. 2005) required only that the filtering function $f$ must be (1) *dependent of the pixel-distance*, (2) *that it be monotonically increasing as the pixel-distance increases*, and (3) that the function must be applicable independently of the image dimensions. Adding a small constant, say $\epsilon$, therefore violates no assumptions for the distance not to be *an IMED*. In frequency space, adding a small constant implies extending the convolution support increases, making each pixel in the ST-transformed image depend slightly more on distant neighbours. Note however, that the Gaussian is not much different, as it is always non-zero. Adding a constant simply makes the Gaussian approach $\epsilon$ rather than zero as $k \to \infty$ (or at certain $k \neq 0$ with increasing $\sigma$). Still, adding a small $\epsilon$ makes the frequency Gaussian of Eq. (6.24) have values larger than 1 at $k = 0$, and therefore slightly amplifies the frequency (this is at the frequency that is already most emphasized by the Gaussian kernel). An alternative to avoid this is to use:

$$\tilde{g}(k) = \frac{\sqrt{\mathcal{F}(g)(k)} + \epsilon}{1 + \epsilon}, \quad \epsilon > 0 \tag{6.37}$$

Which has the effect of preserving the maximum at the expense of slightly increasing all other values of the original frequency Gaussian such that higher frequencies enter slightly more in the convolution, but lower frequencies are not altered by as much as in the no-normalization approach of Eq. (6.36). In Table 6.1 I have stated the effects of $\epsilon$ along with normalization on the minimally sampled value of the frequency Gaussian for different values of $\sigma$. We see that only the tails at higher values of $\sigma$ are affected.

| $\sigma$ | 0.0 | 0.5 | 1.0 | 1.5 | 2.0 | 2.5 | 3.0 | 3.5 | 4.0 |
|---|---|---|---|---|---|---|---|---|---|
| $\epsilon = 0$ | $10^{0}$ | $10^{-1}$ | $10^{-2}$ | $10^{-3}$ | $10^{-5}$ | $10^{-7}$ | $10^{-10}$ | $10^{-14}$ | $10^{-18}$ |
| $\epsilon = 10^{-10}$ | $10^{0}$ | $10^{-1}$ | $10^{-2}$ | $10^{-3}$ | $10^{-5}$ | $10^{-7}$ | $10^{-10}$ | $10^{-10}$ | $10^{-10}$ |
| $\epsilon = 10^{-5}$ | $10^{0}$ | $10^{-1}$ | $10^{-2}$ | $10^{-3}$ | $10^{-5}$ | $10^{-5}$ | $10^{-5}$ | $10^{-6}$ | $10^{-6}$ |

**Table 6.1:** *Effect of Eq.* (6.37) *($\epsilon$ and normalization) with non-zero $\epsilon$ on minimum of Gaussian filter with different $\sigma$ and $\epsilon$. Found by evaluating $\left(\sqrt{\mathcal{F}(g)(k)} + \epsilon\right)/(1+\epsilon)$ using Wolfram|Alpha. Trivially, the other approach with no normalization (Eq. 6.36) simply changes minima below $\epsilon$ to $\epsilon$ in this order-of-magnitude overview.*

In Fig. 6.9 I further visualize the difference between the two approaches by looking at the impact on the entire spectrum at the relatively high value of $\sigma = 4$. The normalization implies that only the tails of the frequency Gaussian are impacted. As such, the vast effect

of the convolution is preserved, as the low-frequency amplitudes are largely unaffected. This implies that the normalization method is satisfies several criteria:

- Very small amplitudes at tail of the Gaussian are increased to counter noise amplification inherent to the inverse standardizing transform in frequency space
- The forward standardizing transform is largely unaffected since the low-frequency amplitudes are changed relatively little
- No frequency amplification is possible, since the filter has values of at most one.
- Normalization is more forgiving to the end-user, since the (ab)use of high values of $\epsilon$ still largely preserves the transform as intended.

While the simple addition approach (Eq. 6.36) also achieves the first bullet, it lacks the benefit of the remainder. For this reason, **we continue the exploration using the small-$\epsilon$ approach *with normalization* as in Eq. (6.37).**

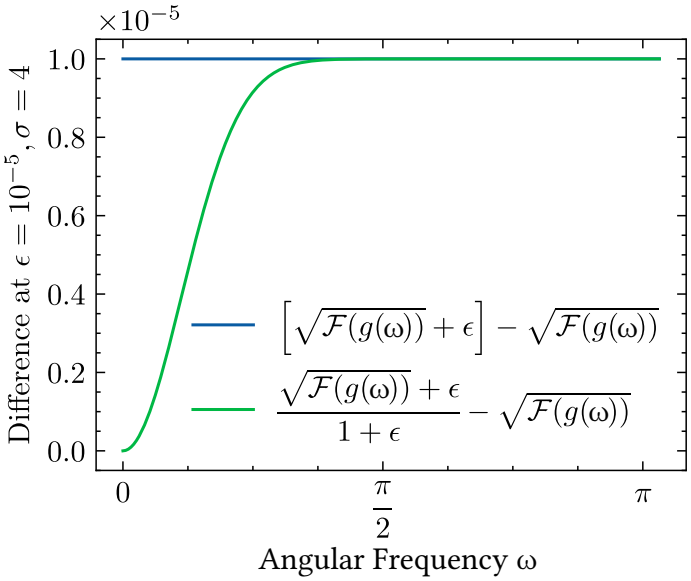
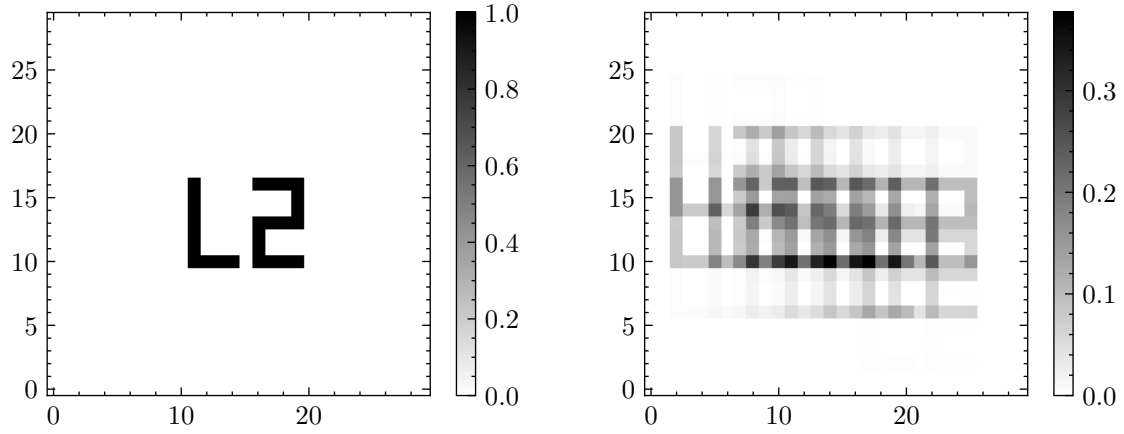



Figure 6.9: Two approaches to adding a small constant $\epsilon$ to mitigate extreme noise amplification of deconvolution. The plot shows the difference of the Fourier space Gaussian $\sqrt{\mathcal{F}(g(k))}$ to each of the two methods in Eq. (6.36) and Eq. (6.37). The normalized approach prevents amplification at $k = 0$, and only has a significant impact on the tails of distribution to counter division by near-zero inherent to inverse filtering.

### Simulation Results

First, let's see how the catastrophic inverse transform of Fig. 6.8a does now that $\epsilon$-ST and $\epsilon$-iST is performed with $\epsilon = 10^{-5}$. The result for the forward and backwards transform is

displaced in Fig. 6.10. Mitigating noise amplification by the addition of a small constant is *remarkably effective*.



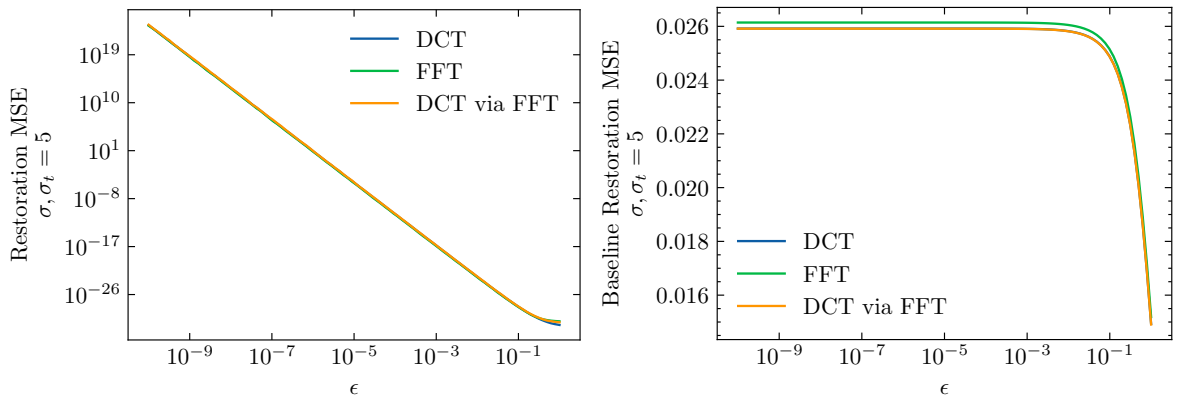

(a) $\epsilon$-restored version of the 15th slice of the volume after temporal standardizing transform followed by inverse transform

(b) Baseline. 15th slice of the volume after temporal $\epsilon$-ST.

Figure 6.10: Example of $\epsilon$-restoration of temporal standardizing transform ($\sigma_t = 5$, $\sigma = 0$, $\epsilon = 10^{-5}$) using DCT method. Compare to Fig. 6.8. The difference a small constant can make to restoration is dramatic. The forward-transformed version (b) is not visibly affected by $\epsilon$ (compare to Fig. 6.8b). Original volume slice not pictured, as it looks identical to (a). Restoration MSE of the volume $\mathrm{ST}^{-1}(\mathrm{ST}(\chi))$ from which one slice is displayed here, is $1.2 \times 10^{-24}$.

The visual restoration from Fig. 6.10a to 6.10a is impressive at the relatively high value of $\sigma_t = 5$, but for a more quantitative summary, see Figure 6.11 in which $\sigma = \sigma_t = 5$ and $\epsilon$ is varied to dampen numerical noise as necessary. For completeness, I display even the case of $\epsilon \sim 1$ (the maximum of the frequency Gaussian), although this is not the intended use of the *small* constant.

In conclusion, the $\epsilon$-ST has the attributes that we like in a deconvolution method in that it is fast, effective, simple, easily adjusted if needed, and does not require estimation of the signal or its Fourier transform. **It can therefore be implemented in the spatial ESN without issue**. If in doubt when using the method for regression problems, I recommend setting the relatively high value of $\epsilon = 1 \times 10^{-2}$.




(a) Restoration MSE as a function of $\epsilon$.

(b) MSE between forward ST and original volume

Figure 6.11: With $\sigma = \sigma_t = 5$, Successful spatio-temporal restoration is achieved when a small constant $\epsilon$ is added to the frequency filter as in Eq. (6.37). With $\epsilon = 0$, we previously had $MSE \sim 10^{120}$, but now achieve e.g. $MSE = 5.1 \times 10^{-6}$ using $\epsilon = 10^{-5}$ and the DCT-based method. In (b) we now see the baseline MSE decreasing as $\epsilon$ is increased to very large values approaching 1. Such use is not intended, but due to the normalization in Eq. (6.37), the $\epsilon$-ST is still sensible. Here, $\epsilon$ has the affect of lifting the tails of the frequency Gaussian, but still retains its maximum. Had normalization not been used, every frequency would be amplified at $\epsilon = 1$.

## 6.8 IMED Benchmarks

The background for this chapter was that the naive IMED prototype implemented in the SESN ((Heim and Avery 2019)) was *slow*, *memory-inefficient* and with no robust inverse method for higher values of $\sigma$.

In this section I show that the frequency methods are indeed faster. I do so with exclusive access to *nodes* of the 'MODI' cluster of the UNICPH High Performance Computing Center. The cluster has 8 *nodes* with identical specifications. Since the SSESN is not *(yet?)* implemented with distributed computing in mind, I use only a single node. It has specifications[4]:

| | |
|---|---|
| **CPU** | AMD EPYC 7501, 2 x 32 cores at 2Gz |
| **Memory** | 256GB Memory |

For measuring runtimes at different resolution, and number of dimensions, I allow my implementations to utilize the multi-core processors when they can. This can benefit

[4]For more on MODI, see the MODI user guide: *https://erda.dk/public/MODI-user-guide.pdf#section.3*

especially the higher-dimensional frequency methods.

In general, I rescale arbitrary data in 1D, 2D and 3D such that the volumes contain up to $10^8$ elements, and I apply forward standardizing transforms.

### 2D Case ('An Image')

Since the prototype implementation of the SESN was only for use on 2D images, we start by observing the difference in performance of the original matrix method, and the Kronecker decomposed version (`fullMat` and `sepMat`) in Fig. 6.12. It is clear that the sepMat method is *dwarfed* by the naive implementation (a good thing).

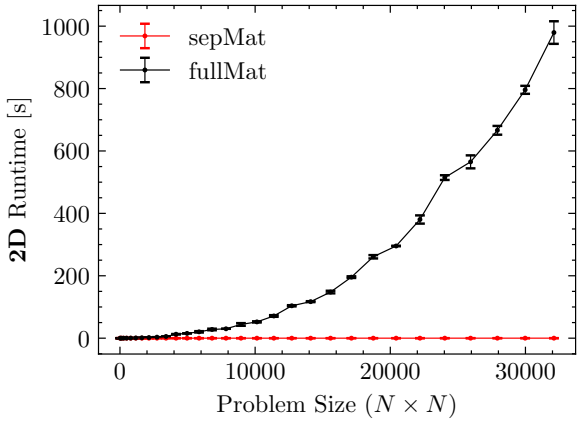
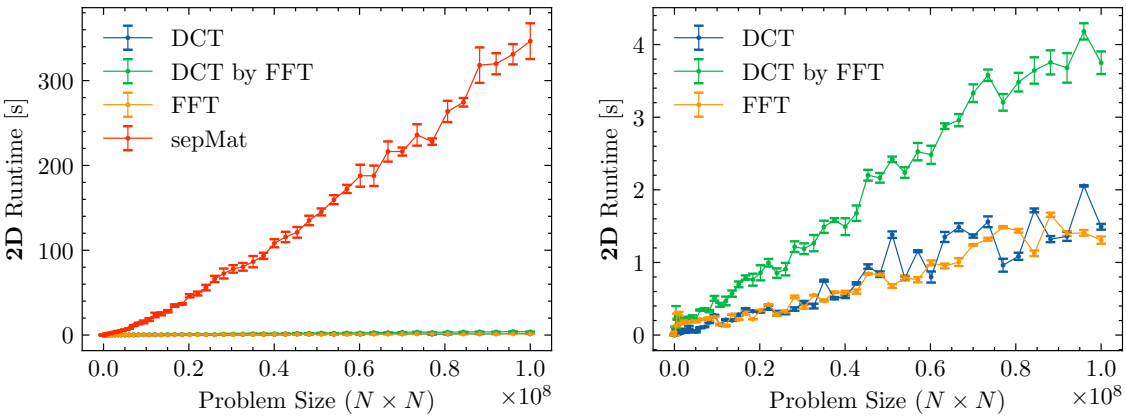



Figure 6.12: Matrix implementations of the IMED. `fullMat` is the *originally proposed* transform, `sepMat` is Kronecker decomposed version. Due to time constraints, only two points are sampled for each average. All CPU cores of the MODI node can be utilized for real-world applicability. Note that problem size is 'small' in comparison due to limits of time and memory consumption.

The frequency methods, of course, are even faster than the `sepMat` method, especially when larger than 1D which allows the implementation to utilize more cores in parallel. In Figure 6.13a we compare the `sepMat` method to frequency implementations. Note the much larger problem size that is possible with these methods. Here, it is the *frequency methods that are dwarfed*. For this reason, I show frequency methods only to the right, in Fig. 6.13b. As expected, they scale better; the IMED journey has been successful!




(a) Non-naive IMEDs

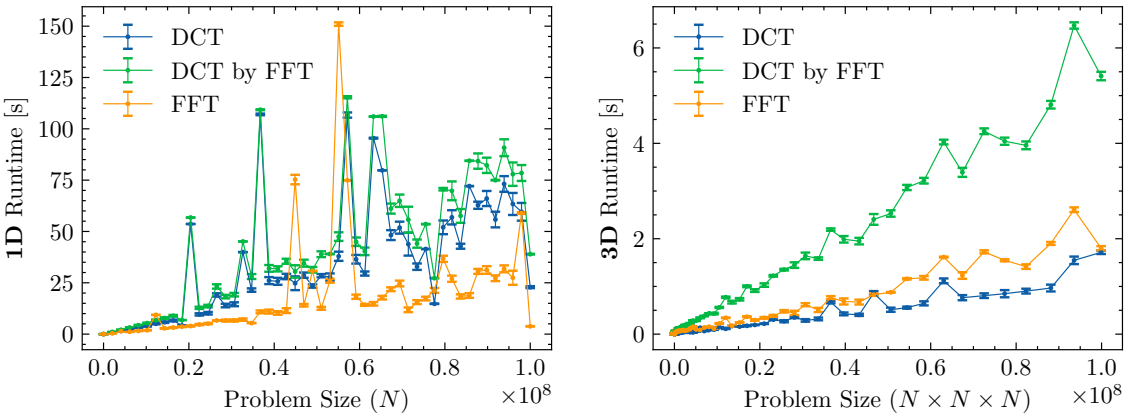
(b) Frequency IMEDs

Figure 6.13: Benchmarks on MODI node for the non-naive methods on 2D volumes. All averages are of 5 observations. All CPU cores, when possible, are utilized in this benchmarks for real-world applicability.

### 1D and 3D Case

The frequency IMEDs are purposefully constructed so that they are applicable to any number of dimensions. In Figure 6.14, I show their runtimes. What may be surprising is that the *1D* case is much slower than the 3D case when the volumes have approximately the same number of elements. This is because higher dimensions allow the implementation to utilize multi-core CPUs.

It is also a general point that the frequency methods are faster for some problem sizes than others, explaining very large 'jumps'. An easy optimization that will be applied in the future is to zero-pad structures to only run transforms for *fast sizes*.




(a) 1D Transform

(b) 3D Transforms

Figure 6.14: Runtimes in 1D and 3D, only implemented for frequency methods. All points are averages of five observations.

# Part III

# Application

# Chapter 7

# *Forecasting with the Scalable SESN*

---

This chapter collects experiments with the SSESN and SESN in the broadest sense. The selection of important hyper parameters is discussed in a prediction setting, and how they can be determined. The SSESN is pushed to its limits with respect to spatial scalability, while exemplifying their predictive performance and stability. This is achieved on the synthetic *orb* data shown in Sec. 1.2 along with a classic *shallow water simulation* that the SSESN can learn to predict without knowledge of the underlying physics.

These synthetic situations warm up to the deployment of the SSESN on the much more complex ocean system (CESM simulations), which the SSESN can make qualitatively plausible predictions of in high spatial resolution – in a few minutes.

Lastly, benchmarks are also provided that show the fruits of the labour of this thesis project.

## 7.1 Reading Guidance

Many different SSESN/SESN experiments are presented in this chapter.
In Section 7.2 I discuss the issue of hyperparameter optimization, and why their automatic tuning are not yet reliably producing for high-quality spatial predictions.
In Sections 7.3-7.7 I showcase the prediction capabilities of the SSESN on a number of spatio-temporal systems, including two very different areas of the ocean.
In Section 7.8, I compare the time complexities of the 'old' SESN, and the 'new' SSESN by timing at increasingly large problem sizes on fixed hardware setup (spatial scaling). Here, I also get into many considerations of CPU versus GPU utilization.

## 7.2 The Unsolved Issue of Hyperparameter Optimization

### *7.2.1 Training, Validating, and Testing*

An almost obvious approach to tune important hyper parameters of the SSESN is a type of *grid search* on reasonable hyperparameter combinations. This can be achieved by *training* the ESN on a 'training' set, and evaluating predictions with that particular hyperparameter configuration on a temporally subsequent 'validation' dataset.
*Validation* implies that the *best hyperparameters* may be chosen by comparing prediction errors on the validation set, and choosing the configuration that minimizes it. The *validation error*, however, is then *not* an unbiased estimate of generalization performance (on new data).

Even so, several approaches are possible:
For ocean data, for example, a training and validation *area* of the world may be used to select hyperparameters for searching in other areas of the world, and the *test set* can then be considered the application of the model to *other parts of the world*. Even then, different areas will be less *predictable* than others, and the dynamics can vary.
Another approach than a spatial division, since the ESN model is sequential in nature, is to use a temporal subset of data (of the same spatial area) for training, validation, and testing respectively. That way, parameters may be tuned better to the problem, but at the expense of 'wasting' data for validation if temporal observations are limited. Of course, the readout matrix of the ESN will have to be refitted on that validation data in order to

produce test-time predictions.

Comparing the test-time *predictions* to the test-time *targets* will provide a less biased estimate of generalization (out-of-sample) error. If the data were *non-sequential* and independent and identically distributed, the testing error would in fact be *unbiased* with deviations from the testing error expected to deviate less when the test set has more observations. If the former approach, i.e. *spatial subdivision* of a spatio-temporal series, samples can be considered less dependent, although the ocean system, for example, is of course deeply connected.

With these considerations in mind, I will list the hyperparameters that are worth optimizing.

### *7.2.2 Essential Hyperparameters*

From my own experience, as well as literature on *tricks of the trade* e.g. (Lukoševičius 2012), the parameters of Table 7.1 are most influential to the final prediction quality, and therefore candidates for a relatively limited grid search. In the table I describe examples of reasonable setting for each.

It is common to select hyperparameters on *smaller problems*, and it is my experience that this does hold for spatio-temporal series that are *up* or *downscaled* within reason. Therefore, a more exhaustive grid search can often be performed on spatially smaller versions of the problem that are faster to compute.

| Hyperparameter | Description | Reasonable Settings |
|---|---|---|
| Spectral Radius, $\rho(\mathbf{W}_{\text{hh}})$ | Scales non-linearity of $\tanh(\cdot)$, Memory capacity, Max. amplification of $\mathbf{h}_t$ | 0.8, 1.0, 1.5, 2.0 |
| Input Scaling, $\kappa$ | Factor to scale input mapping, non-linearity of $\tanh(\cdot)$ | $\left(10^{-3}; 10^{1}\right)$ |
| Principal Components, $N_{\text{PC}}$ | Effectively, the number trained weights per image pixel at time $t$. | 300, 500, 1000 or $\min(N_{\text{train}}, N_{\text{hidden}})$ as $\mathbf{H} \in \mathbb{R}^{N_{\text{train}} \times N_{\text{hidden}}}$ |
| Spatial and Temporal IMED settings $\sigma_t, \sigma_{xy}$ | When resolution is increased, but spatial/temporal distances are equal, $\sigma_{xy}, \sigma_t$ should be conserved in units of distance. $\sigma = 0$ implies that Euclidean distance is used as opposed to IMED. | Depends on distance unit. In pixel units, $\sigma_{xy} \in (0; 10)$ is often reasonable. $\sigma_t$ should be low due to temp. edge effects. |
| Input Map specs; including dimension $N_{\text{hidden}}$ | Different input maps as explained in Table 7.5. **Note:** SESN often needs more input maps to perform well, and on smaller spatial problems. Dimension reduction of SSESN may explain this change. | Combination of bilinear rescaling, convolutions, gradient, DCT with $(N_{\text{k1}}, N_{\text{k2}}) = (20, 20)$ |

**Table 7.1:** *Hyperparameters that are essential to successful prediction using the SSESN. An example of an often non-essential parameter is the sparsity of $\mathbf{W}_{hh}$ — in my own experience and that of (Lukoševičius 2012).*

### *7.2.3 Optimization with Competing, Imperfect Error Measures*

In Chapter 6 I implemented the *standardizing transform* and its *inverse* with more efficient frequency implementations, and the more natural boundary effects of the DCT implementation, especially.
I admit, however, that the simplicity of the IMED/ST (along with the standard Euclidean) does *not* provide a perfect spatial similarity measure that is perfect for optimizing predictions of chaotic systems.
What this means is that *minimizing* the IMED, or rather, the MSE-IMED (squared error averaged over space and prediction times) does *not* necessarily provide the *best* qualitative prediction. For instance, if the hyperparameters in Table 7.1 are *not tuned properly*, a decently low *IMED-MSE* of predictions can often be obtained with a 'still' prediction that has *frozen* in time.

This is a major obstacle to automatic hyperparameter optimization, as the result of hyperparameter selection must often be *judged qualitatively*. Luckily, I find that different random initializations of the reservoir, $\mathbf{W}_{\text{hh}}$, does not tend to lead to the shift between *frozen* or *dynamic* predictions, whereas tuning of e.g. $\rho\left(\mathbf{W}_{\text{hh}}\right)$ *can*. This implies that qualitatively probable predictions of ocean systems, for example, are possible, though they must often be judged qualitatively due to their immense complexity, and the intrinsic property of *chaotic systems being unpredictable* despite the determinism of the system.

The difficulties of quantitatively comparing predictions do not end there, however. There is no universally *perfect* IMED-configuration, i.e. with $\sigma_{xy}, \sigma_t$ fixed, and these parameters *change the loss function itself*. For instance, higher $\sigma$ tends to blur out both predictions and targets, and will often yield a lower *IMED-MSE* than with lower $\sigma$; not due to higher quality after iST has been applied, but due to *excessive blurring*.
One option is to vary $\sigma$ parameters in a grid search setting, but compare predictions and targets *after* the inverse transform using the *standard* Euclidean MSE (over time and space).
Alternatively, a *fixed* IMED can be applied as a post-processing step in order to compare all predictions and targets in the same way across hyperparameters. As discussed, neither is a perfect, and I proceed with comparing the Euclidean *MSE* such that IMED is

only applied *internally* in the ESN to optimize the spatial coherence of predictions, and approximately minimize the Euclidean distance in the original space (after iST).

Note that the original-space comparison of prediction and targets does not *necessarily* need to be an $\mathcal{L}_2$ metric, which is, on the other hand, necessary when least squares is applied internally in the ESN.

## 7.3 Synthetic Data: Predicting an Orb with Lissajous Curve Centre

In this section, I show SSESN prediction examples on the *orb with a periodic path* seen in a subfigure of Fig. 1.2. To be specific, the periodic orb, implemented by (Heim and Avery 2019) has its centres over time in a Lissajous curve, see e.g. the Wikipedia article (*Lissajous Curve* 2021) for a brief introduction.

### *7.3.1 Grid Search on a Spatially Smaller Problem*

In accordance with the approach described in Section 7.2, I perform a more exhaustive grid search on a spatially smaller version of the synthetic data, and a less exhaustive grid search on spatially larger (and more computationally intensive) problems from a subset of the hyperparameters that worked best on the smaller problem.
The 'exhaustive' grid search is performed on a $49 \times 49$ spatial problem. The training set consists of 1200 temporal samples of which the first 200 are used for construction of the first 'reliable' $\mathbf{h}_t$ that is initialized from $\mathbf{h}_{t=0} = \mathbf{0}$ ($N_{\text{trans}} = 200$). Therefore $N_{\text{train}} = 1000$. The 'validation set' used to compare prediction errors is another $N_{\text{pred}} = 100$ steps that directly proceed the training set.

The hyperparameter grid is described in Table 7.2.

**Note:** This section does *not* seek to provide unbiased generalization performance estimates, but to provide the reader with example output.

| Hyperparameter | Small Problem<br>Hyperparameter Grid |
|---|---|
| Spectral Radius,<br>$\rho(\mathbf{W}_{\text{hh}})$ | $\rho(\mathbf{W}_{\text{hh}}) \in \{0.8, 1.0, 1.5, 2.0\}$ |
| Input Scaling, $\kappa$ | $\kappa \in \{1, 1.5, 2\}$<br>Note that input scaling applies to *input maps* of<br>image input at any time $t$, after the volumes have been<br>scaled such that the training input volume has maximum value 1<br>and minimum -1. |
| Principal Components,<br>$N_{\text{PC}}$ | $N_{\text{PC}} \in \{300, 500, 1000\}$<br>1000 is the maximum sensible PCA-dimension as<br>$\mathbf{H}$ has minimal dimension $N_{\text{train}} = 1000$ |
| Spatial and Temporal<br>IMED settings $\sigma_t, \sigma_{xy}$ | Temporal $\sigma_t = 0$, but $\sigma_{xy} \in \{0, 1, 2.5, 5\}$<br>in artificial, but consistent units when resolution is varied.<br>$\sigma = 0$ implies no ST-convolution. |
| Input Map specs;<br>including dimension<br>$N_{\text{hidden}}$ | Always the following input maps, but<br>with different uniform scaling $\kappa$:<br>• *Bilinear Rescaling* to $1/2$ size in each dimension.<br>• $5 \times 5$ *Convolution* with random kernel<br>• $9 \times 9$ *Convolution* with random kernel<br>• *Image Gradient*<br>• *DCT* with $(20 \times 20)$ lowest frequency bins |
| Spatial Dimensions<br>(for grid search only) | $49 \times 49$ |

**Table 7.2:** *Selected Grid search values for hyperparameter optimization through grid search for the Lissajous orb prediction problem. And later, in Section 7.4, for the chaotic Mackey-Glass orb. Lowest Euclidean MSE (over space and time) in the original, non-IMED, space is chosen as an admittedly imperfect measure of prediction quality. IMED is still utilized internally in the ESN, except when $\sigma = 0$. The models that obtain lowest prediction MSE are* ***manually evaluated*** *to make sure that predictions are spatially coherent, and e.g. not a 'frozen' prediction that happened to minimize MSE, as can sometimes happen.*

### Grid Search Results

Running the $49 \times 49$ Lissajous problem, excluding creation of the artificial data, takes approximately **9 seconds** on the 16 core CPU-only configuration described in Section 7.8.2. This is an estimate without guarantee of exclusivity on the CPU usage by the script in question.

After hyperparameter optimization on the $49 \times 49$ Lissajous orb problem, the lowest Euclidean MSEs was obtained using the different configurations listed in Table 7.3. Clearly, the in-between value of $\rho \approx 1.5$ produced good results, and $N_{\text{PC}} = 500$ principal components was a good model complexity trade-off. Further, $\sigma_{xy} = \{2.5, 5\}$ produced good results in terms of Euclidean MSE, and so did input scaling $\kappa \in \{1.5, 2.0\}$. I proceed with these *four configuration* options when increasing the Lissajous problem size.

In the table, please note the hyperlinks to prediction animations with comparison to unseen targets, in all model instances. Further, I show a subset of prediction frames in Figure 7.1. Crucially, the predictions show a *coherent* orb, which is *more important* than a low-MSE score (optimally, both are true).

The Lissajous predictions are *qualitatively* good. However, the simple periodic Lissajous task is actually one where the SESN without PCA 'regularization' can outperform the SSESN (on small problems). This is because *over-fitting* is not an issue for the periodic signal, as long as the validation sequence is *also* entirely contained in the training set. The SSESN predictions, however, exhibit less over-fitting in general, which is valuable for more complex problems.

| **Euclidean Validation MSE Over Space and 100 Predictions (Ascending Order)** | **Lissajous Hyperparameter Combination** |
|---|---|
| $3.60 \times 10^{-5}$, see GitHub[1] | $\rho \approx 1.5$<br>$\kappa = 2$<br>$N_{\text{PC}} = 500$<br>$\sigma_{xy} = 5$ arb. units (2.45 in pixel units) |
| $4.21 \times 10^{-5}$, see GitHub[2] | $\rho \approx 1.5$<br>$\kappa = 1.5$<br>$N_{\text{PC}} = 500$<br>$\sigma_{xy} = 5$ arb. units (2.45 in pixel units) |
| $5.15 \times 10^{-5}$, see GitHub[3] | $\rho \approx 1.5$<br>$\kappa = 1.5$<br>$N_{\text{PC}} = 500$<br>$\sigma_{xy} = 2.5$ arb. units (1.225 in pixel units) |
| $7.83 \times 10^{-5}$, see GitHub[4] | $\rho \approx 1.5$<br>$\kappa = 1.0$<br>$N_{\text{PC}} = 500$<br>$\sigma_{xy} = 2.5$ arb. units (1.225 in pixel units) |
| $1.11 \times 10^{-4}$, see GitHub[5] | $\rho \approx 1.5$<br>$\kappa = 1.5$<br>$N_{\text{PC}} = 300$<br>$\sigma_{xy} = 0$ arb. units (0 in pixel units) |

**Table 7.3:** *A subset of 'best' performing configurations of the Lissajous Hyperparameter Grid Search in terms of Euclidean MSE (for comparison across internally used IMED settings). The Lissajous prediction problem is not a challenge, and many configuration lead to good result (that are all spatially coherent when inspected).*

[1]Validation Set Video: *https://github.com/jfelding/esn/blob/thesis_assets/lissajous_blob_dim49/esn011/comparison.mp4?raw=True*

[2]Validation Set Video: *https://github.com/jfelding/esn/blob/thesis_assets/lissajous_blob_dim49/esn010/comparison.mp4?raw=True*

[3]Validation Set Video: *https://github.com/jfelding/esn/blob/thesis_assets/lissajous_blob_dim49/esn009/comparison.mp4?raw=True*

[4]Validation Set Video: *https://github.com/jfelding/esn/blob/thesis_assets/lissajous_blob_dim49/esn008/comparison.mp4?raw=True*

[5]Validation Set Video: *https://github.com/jfelding/esn/blob/thesis_assets/lissajous_blob_dim49/esn007/comparison.mp4?raw=True*

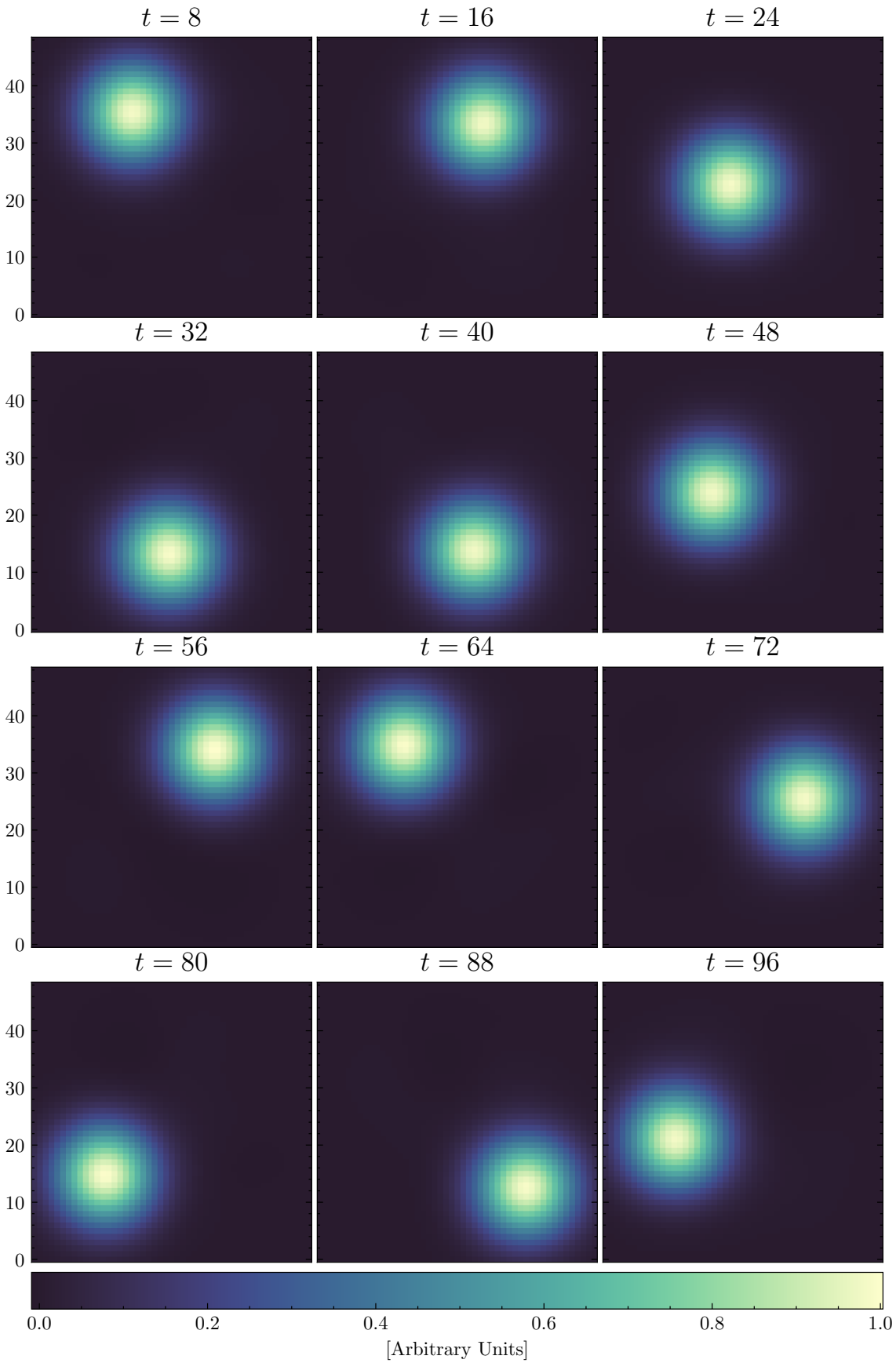



Figure 7.1: A subset of prediction frames of the 49 × 49 Lissajous orb problem, with the lowest MSE-model of Table 7.3. Crucially, the orb remains coherent throughout the predictions. Full animation with comparison to target sequence is available at GitHub.

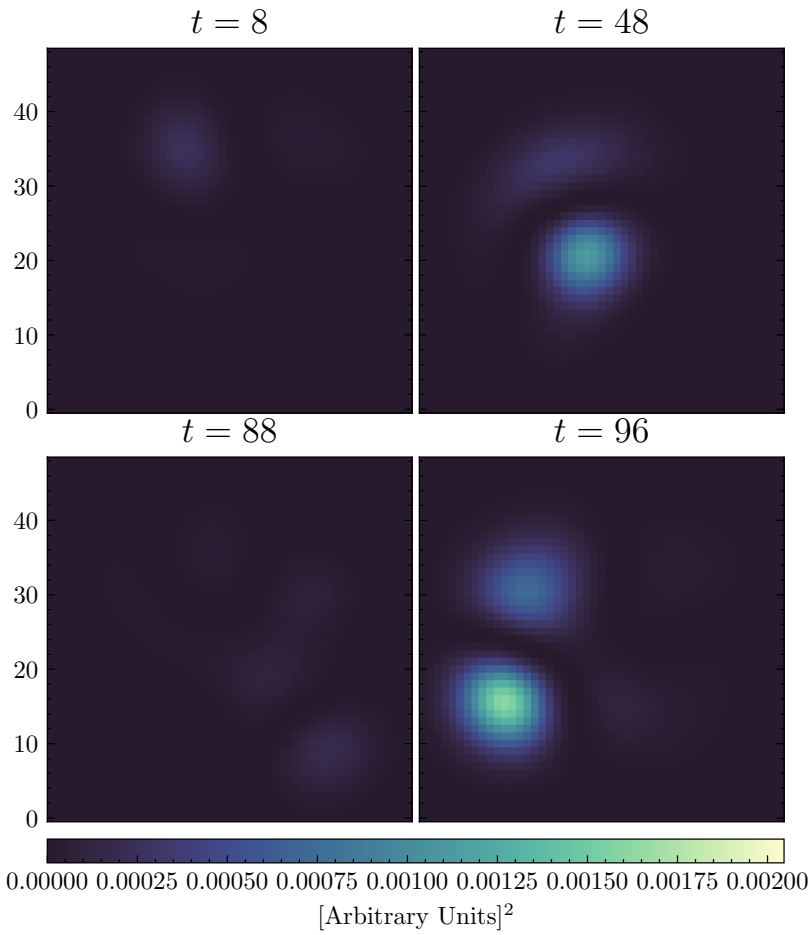



Figure 7.2: Spatial Mean Squared Errors from an interesting subset of frames from Figure 7.1. Note that their colour scales are not identical. Comparison of predictions and unseen target frames.

### *7.3.2 High-Dimensional Application: 500x500 Pixels*

With the four optimal configurations for the $49 \times 49$ Lissajous problem, I proceed to high-resolution version of the Lissajous orb prediction problem.

In the interest of brevity, I present only a $(500 \times 500)$ pixel version, run on the same parameters as that of the smaller problem version. While the input itself has $N_{\text{input}} = 250,000$ individual pixels, which would make up quite a large hidden space, and much higher than in traditional ESN applications, the input maps used, described in Table 7.2 imply that the hidden dimension of the problems are slightly more than *four times larger* than the input dimension. This is *quite excessive* for such a simple prediction problem. In fact, successful higher-dimensional predictions can even be achieved on subsampled (rescaled) versions of the raw input by calling the *Bilinear Rescaling* input map only.

Running this model, which has more than 100x the number of pixels as the smaller-version problem, takes approximately 471.7$s$, or about 8 minutes.

While this could be *much* faster with reasonable prediction quality if a smaller hidden state were used, I try only to be consistent in this section, and to provide the reader with output examples.

The best runs of the $500 \times 500$ resolution Lissajous sequence had the configuration:
$\rho \approx 1.5$, $\kappa = 1.5$, $N_{\text{PC}} = 500$, $\sigma_{xy} = 2.5$ arb. units (12.5 in pixel units).
These models produced MSEs (over space and time) of order $10^{-5}$ with the best run achieving $4.25 \times 10^{-5}$. The MSE may also be determined as a time series by averaging spatial errors only. The result is visible in Figure 7.3, which however displays what I refer to as the **IMED-MSE**, i.e. the MSE over time with the IMED metric of the particular model.

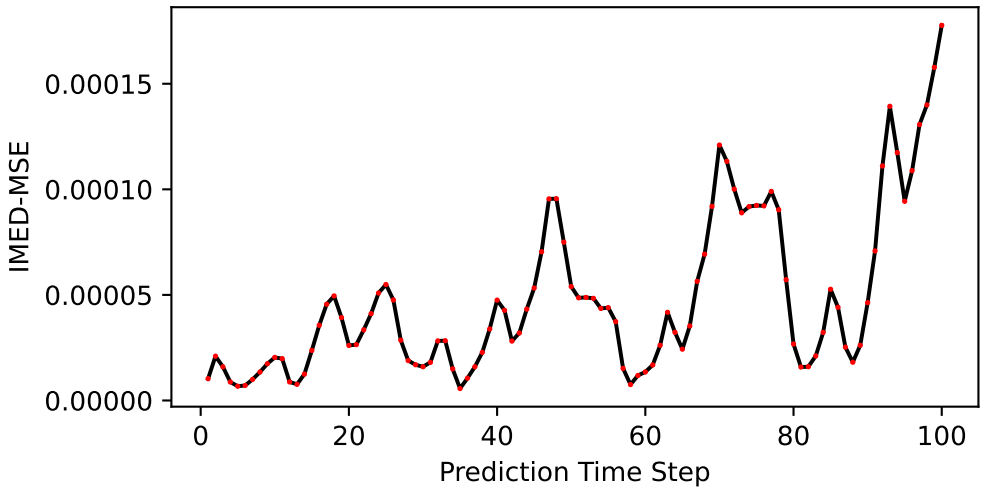



Figure 7.3: In the standardizing transform space, the MSE is determined, yielding an IMED metric MSE, or 'IMED-MSE'. While errors remain low, we see an increase at the end of the arbitrarily chosen length of 100 prediction frames. Qualitatively, the error towards the end is not easily visible in the prediction frames, see Figure 7.4. That is one issue of using the Euclidean MSE: A small displacement of the orb can significantly increase the MSE, although the orb is coherent and qualitatively satisfactory.

Like for the smaller problem size, I also show prediction frames of the $500 \times 500$ problem in Figure 7.4. A full animation comparison is available at GitHub[6].

[6] *https://github.com/jfelding/esn/blob/thesis_assets/lissajous_blob_dim500/esn002/comparison.mp4?raw=True*

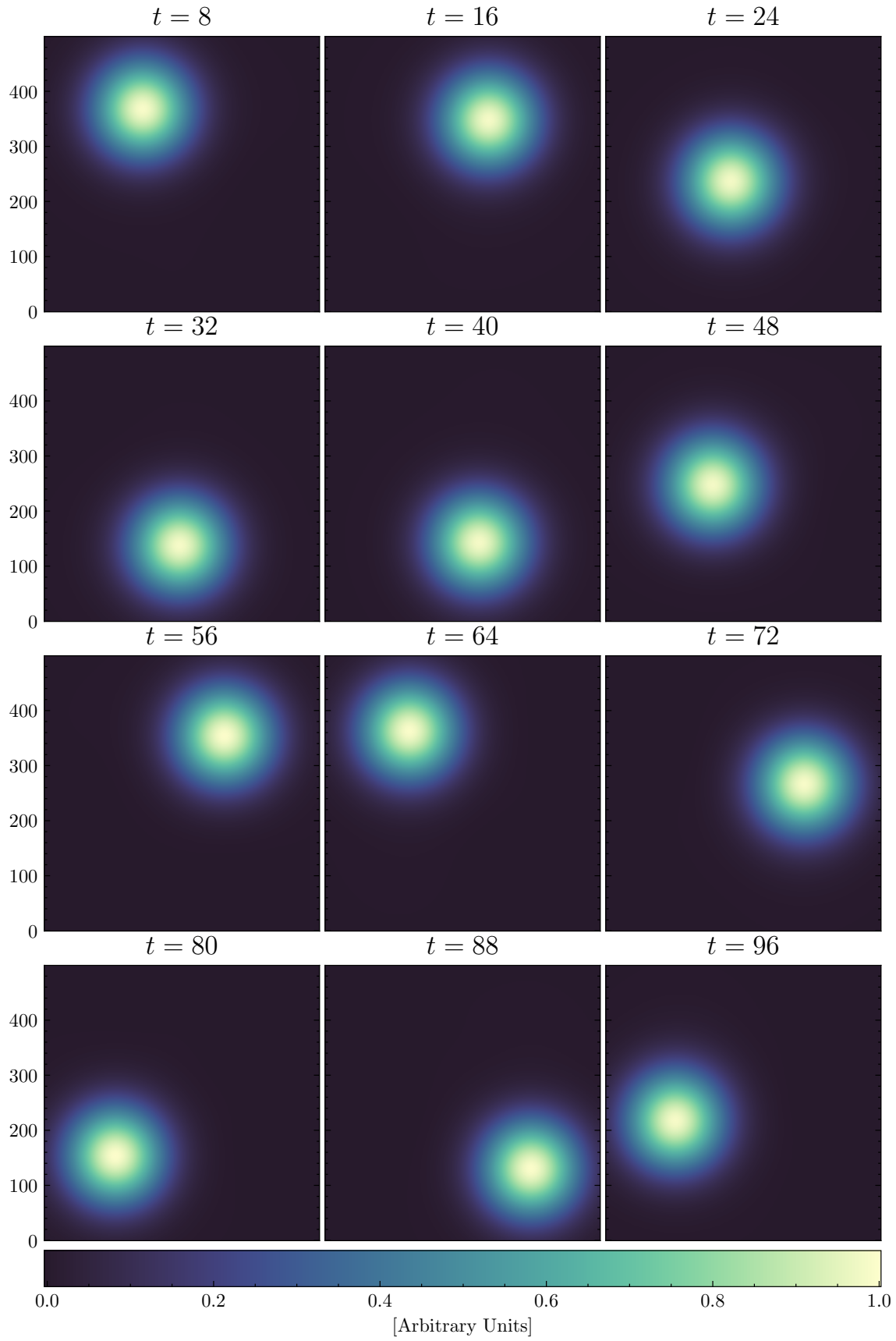



Figure 7.4: Prediction time frames of the 500×500 Lissajous orb problem. Full animation with comparison to target sequence is available at GitHub.

### 7.3.3 *A* $1500 \times 1500$ *Lissajous For the Front Page*

As I have stated in the previous section without evidence, it is *easy* to go to higher spatial dimensions by reducing the hidden state dimension, at least for the simple Lissajous problem. In this section, I briefly will show that a $1500 \times 1500$ prediction is achievable in a shorter time than the $500 \times 500$ problem of the previous section. This is possible when utilize a smaller hidden state, with simpler input mappings. $1500 \times 1500$ is about 92% of *1080p* resolution, and with *nine times* $\left(3^2\right)$ as many pixels as the $500^2$ problem.
’ Without further ado, I choose from experience and intuition the hyperparameters $\rho = 1.5$, $N_{\text{PC}} = 1000$, $\sigma = 0$, $\kappa = 1.5$.
I use the *bilinear rescaling* input map alone, and downscale the $1500 \times 1500$ images to $100 \times 100$ for a hidden state of $N_{\text{hidden}} = 10^5$.

The runtime for this setup, excluding data generation, is **214.2 seconds**, and the MSE is $1.33 \times 10^{-8}$ (do note that *more zero-valued background pixels do influence the average*). The MSE plot (no IMED, as $\sigma = 0$) is displayed in Figure 7.5, and a target-prediction comparison is once again available at GitHub.

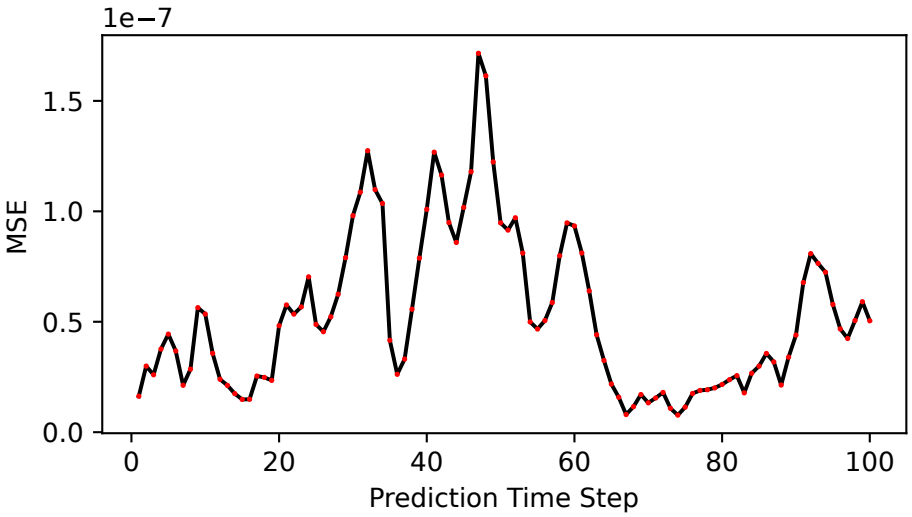



Figure 7.5: MSE over time for the $1500 \times 1500$ Lissajous orb problem, produced with a hidden state dimension of only $N_{\text{hidden}} = 10^5$. Full animation with comparison to target sequence is available at GitHub.

For brevity, I present in the report only the last prediction frame in with $1500 \times 1500$ pixels. Behold the high-resolution glory of the Lissajous orb in Figure 7.6.

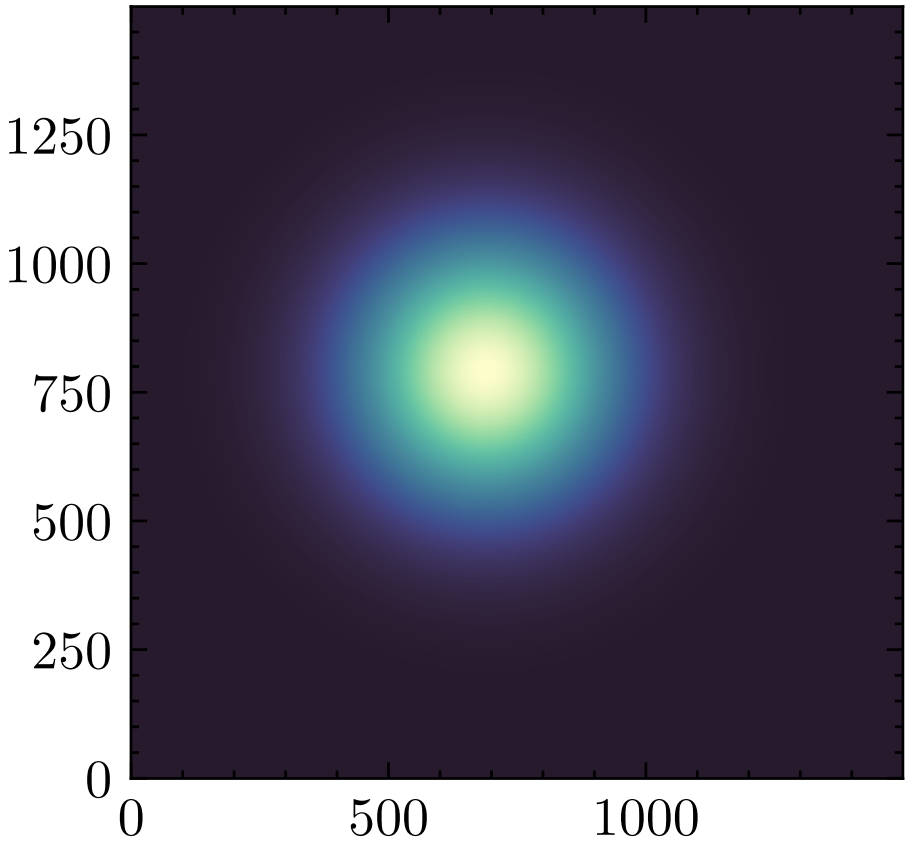



Figure 7.6: The 100th prediction frame of the 1500 × 1500 Lissajous orb prediction. And the **front page** of this thesis!

## 7.4 Synthetic Data: Predicting an Orb with Mackey-Glass Centre

In this section, I keep studying predictions of the synthetic *orbs*, but for a more complex prediction problem that is also visualized in Figure 1.2. The problem, here, is an orb with a centre determined by chaotic Mackey-Glass equations with parameter $\tau = 17$ that governs the volatility of the system, see e.g. (*Mackey-Glass Equations* 2021) for a brief introduction.

Testing the SSESN on this problems brings us closer to prediction on *chaotic ocean currents*, and helped the analysis of what the implementation that I have made *does*. Before the application of the PCA dimension reduction, I typically experienced over-fitting, i.e. a machine precision level training, but with the orb quickly falling apart with no *spatial coherence*. Especially for a non-periodic orbit like the sequence used in this section, I once again will stress that *MSE is not all*. For chaotic systems, it is *much* more important to have qualitatively probable predictions rather than a low-MSE predictions that have less spatial coherence or *freeze* over time. This is, of course, because a small change in *initial*

*conditions* can propagate to produce a much different outcome later on in *chaotic systems*.

I proceed with the approach of the Lissajous orb experiment, with the same grid search on the smaller $49 \times 49$ pixel problem. That is, the search space is described in Table 7.2.

The results of the optimization on the grid search on the smaller problem is shown in Table 7.4, with a subset of different best-performing models in terms of MSE over space and time. It is quite clear, that $\rho \approx 1.5$ is the better parameter, and that $N_{\text{PC}} = 1000$ is a good choice, too. For $\sigma_{xy}$ I will try both $\{2.5, 5\}$, and $\kappa \in \{1.5, 2.\}$. It is interested, but not unexpected, that the Mackey-Glass orb problem performs better with *more trained parameters per pixel* ($N_{\text{PC}} = 1000$) than the simple Lissajous problem did ($N_{\text{PC}} = 500$). However, the optimum may be somewhere in-between for stronger regularization to prevent over-fitting – a major challenge for a non-periodic system where the validation set is not approximately equal to a sequence in the training set.

Since the Mackey-Glass orb is more chaotic, it is now sensible to assume that the approximately one order of magnitude larger MSE value, compared to Lissajous, is caused mainly by the prediction of the position of the chaotic orb to no longer be *perfect*. I visualize, therefore, the frame-to-frame differences in Figure 7.8 of the predictions and unseen targets.
A video comparison of prediction and target is available at GitHub[7].
Lastly, I also show the IMED-MSE over time of the minimal MSE model in Figure 7.7.

[7] *https://github.com/jfelding/esn/blob/thesis_assets/chaos_blob_dim49/esn017/comparison.mp4?raw=true*

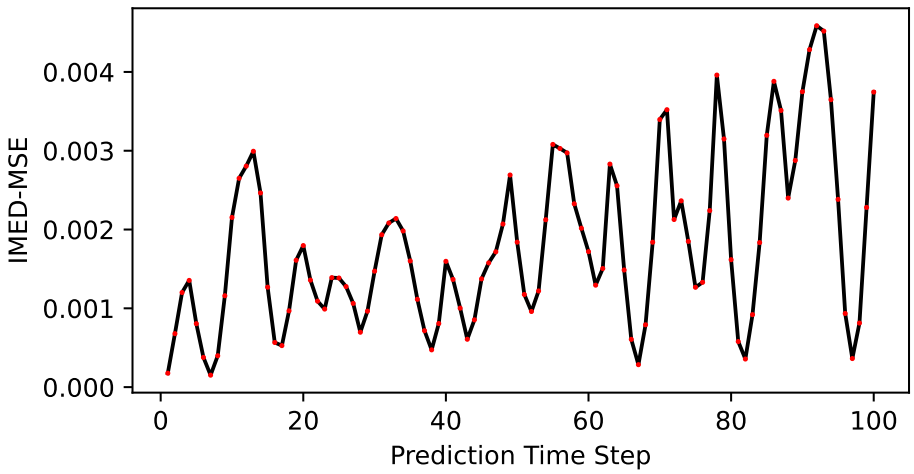



Figure 7.7: IMED-MSE over time for the best-case Mackey-Glass orb in a $49 \times 49$ grid. While the orb qualitatively *remains on path* of the unseen target, the IMED-MSE seems to trend towards increasing values over time. This is generally expected as the predictions are *free-running* so that early missteps propagate to later frames as well.

| **Euclidean Validation MSE Over Space and 100 Predictions (Ascending Order)** | **Mackey-Glass Hyperparameter Combination** |
|---|---|
| $4.20 \times 10^{-4}$, see GitHub[8] | $\rho \approx 1.5$<br>$\kappa = 2$<br>$N_{\text{PC}} = 1000$<br>$\sigma_{xy} = 2.5$ arb. units (1.225 in pixel units) |
| $4.41 \times 10^{-4}$, see GitHub[9] | $\rho \approx 1.5$<br>$\kappa = 2$<br>$N_{\text{PC}} = 1000$<br>$\sigma_{xy} = 5$ arb. units (2.45 in pixel units) |
| $4.80 \times 10^{-4}$, see GitHub[10] | $\rho \approx 1.5$<br>$\kappa = 1.5$<br>$N_{\text{PC}} = 1000$<br>$\sigma_{xy} = 5$ arb. units (2.45 in pixel units) |
| $8.33 \times 10^{-4}$, see GitHub[11] | $\rho \approx 1.5$<br>$\kappa = 2$<br>$N_{\text{PC}} = 500$<br>$\sigma_{xy} = 1.0$ arb. units (0.49 in pixel units) |
| $9.40 \times 10^{-4}$, see GitHub[12] | $\rho \approx 1.5$<br>$\kappa = 2$<br>$N_{\text{PC}} = 1000$<br>$\sigma_{xy} = 0$ arb. units (0 in pixel units) |

**Table 7.4:** *A subset of 'best' performing configurations of the Mackey-Glass Hyperparameter Grid Search in terms of Euclidean MSE (for comparison across internally used IMED settings). The complexity of the Mackey-Glass orb problem is significantly greater than the Lissajous orb problem, since it is more erratic, and not periodic.*

[8] Validation Set Video: *https://github.com/jfelding/esn/blob/thesis_assets/chaos_blob_dim49/esn017/comparison.mp4?raw=True*

[9] Validation Set Video: *https://github.com/jfelding/esn/blob/thesis_assets/chaos_blob_dim49/esn016/comparison.mp4?raw=True*

[10] Validation Set Video: *https://github.com/jfelding/esn/blob/thesis_assets/chaos_blob_dim49/esn015/comparison.mp4?raw=True*

[11] Validation Set Video: *https://github.com/jfelding/esn/blob/thesis_assets/chaos_blob_dim49/esn014/comparison.mp4?raw=True*

[12] Validation Set Video: *https://github.com/jfelding/esn/blob/thesis_assets/chaos_blob_dim49/esn013/comparison.mp4?raw=True*

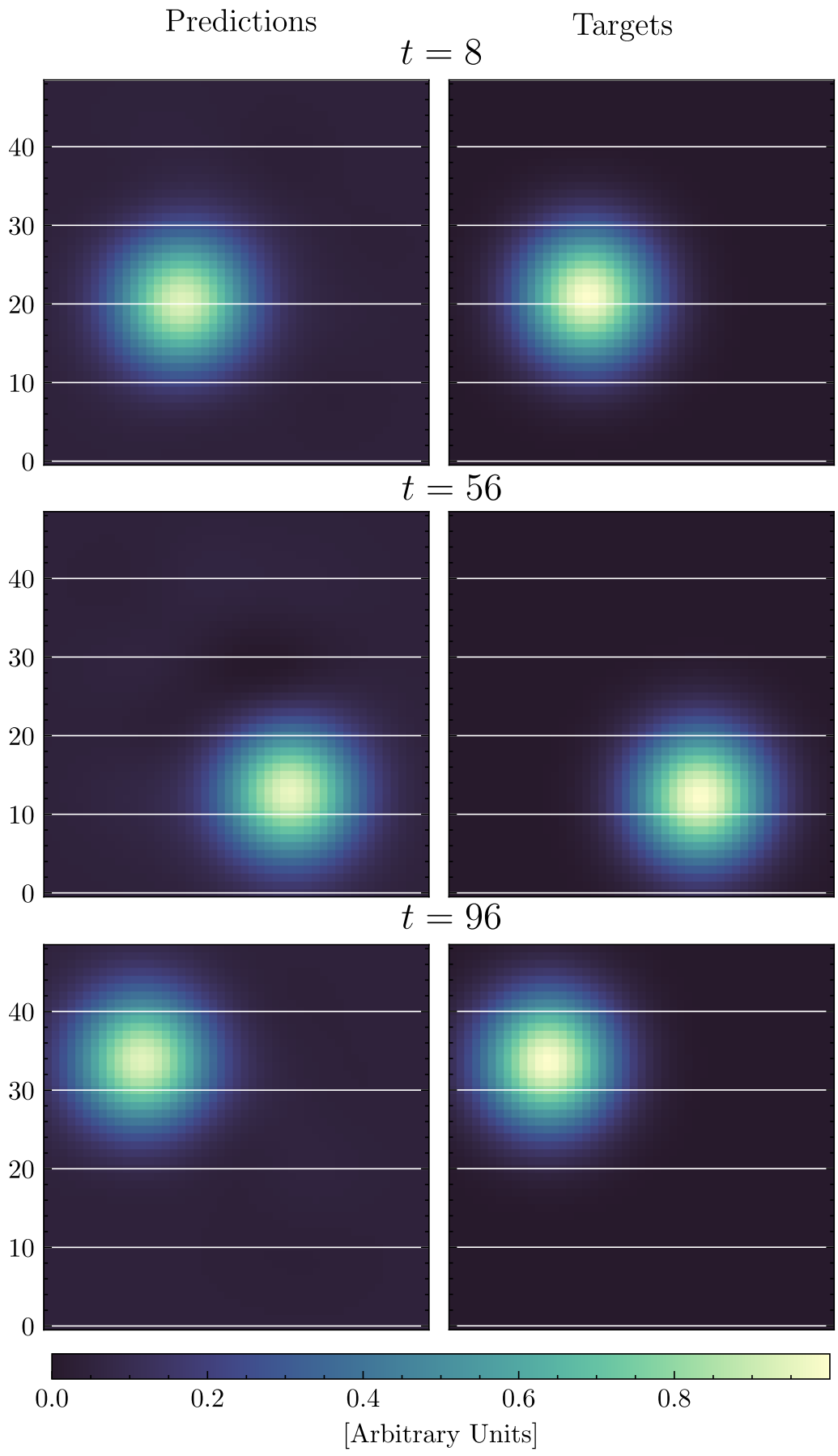



Figure 7.8: Frame-to-frame comparisons for the optimized 49 × 49 Mackey-Glass Orb. Due to chaotic behaviour, the alignment of the predictions and targets orbs are no longer *as good* as for the Lissajous orb.

### 7.4.1 An Example of a Poor Choice of Hyperparameters

During the hyperparameter search *many* poor predictions were made! This is important to stress, and it underscores the importance of understanding, or at least tuning, the behaviour of the ESN dynamical system, and applying regularization through the number of trained weights through $N_{\text{PC}}$. The showcasing of 'best-case' results should not leave the reader with the impression that nothing can go wrong.

The spectral radius, especially, is an easy way to render the approximation performance of an ESN instance *useless*. This is true when it is set too high *or* too low.

To display this, I have selected one of the first hyperparameter combinations on the search grid, with settings $\rho \approx 0.8, N_{\text{PC}} = 300, \kappa = 1, \sigma_{xy} = 0$.

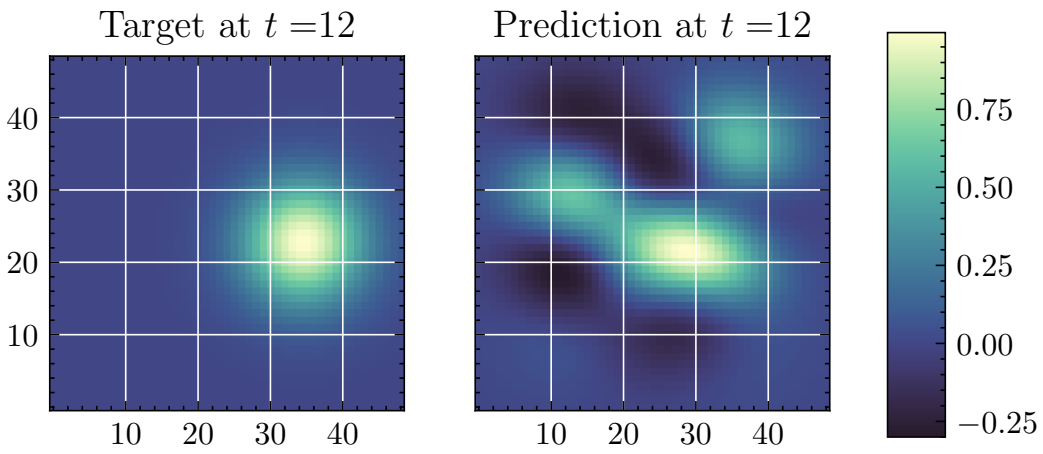



Figure 7.9: The purpose of this section is to showcase just how wrong predictions can be, when the selected subset of hyperparameters are *off*. In this particular case, $\rho \approx 0.8, N_{\text{PC}} = 300, \kappa = 1, \sigma_{xy} = 0$. It is widespread in ESN research to consistently use $\rho \leq 1$ because this is thought to satisfy the Echo State Property more often, see Section 3.4.2. But not trying out higher spectral radii can also significantly impair learning. In fact, there is no general $\rho < 1$ ESP requirement, but only for the trivial case of *zero*-inputs, which rids the ESN system of its designation as *input-driven*.

I have also made available an animation comparing the predictions and targets at GitHub[13] along with more specific hyperparameters and settings, data and an illustration of MSE over time.

### 7.4.2 $500 \times 500$ *Mackey-Glass Orb*

I have found, generally, that predicting the Mackey-Glass sequence requires a higher number of trained parameters – set indirectly using $N_{\text{PC}}$ – than the Lissajous sequence

[13] https://github.com/jfelding/esn/tree/thesis_assets/chaos_blob_dim49/esn002

did. Therefore, proceed with a narrower grid search on the higher-resolution version of the Mackey-Glass sequence, with $\rho \approx 1.5, N_{\text{PC}} = 1000, \sigma_{xy} \in \{2.5, 5\}, \kappa \in \{1.5, 2\}$.
All 500 × 500 Mackey-Glass models initialized with these parameters had qualitatively good results.

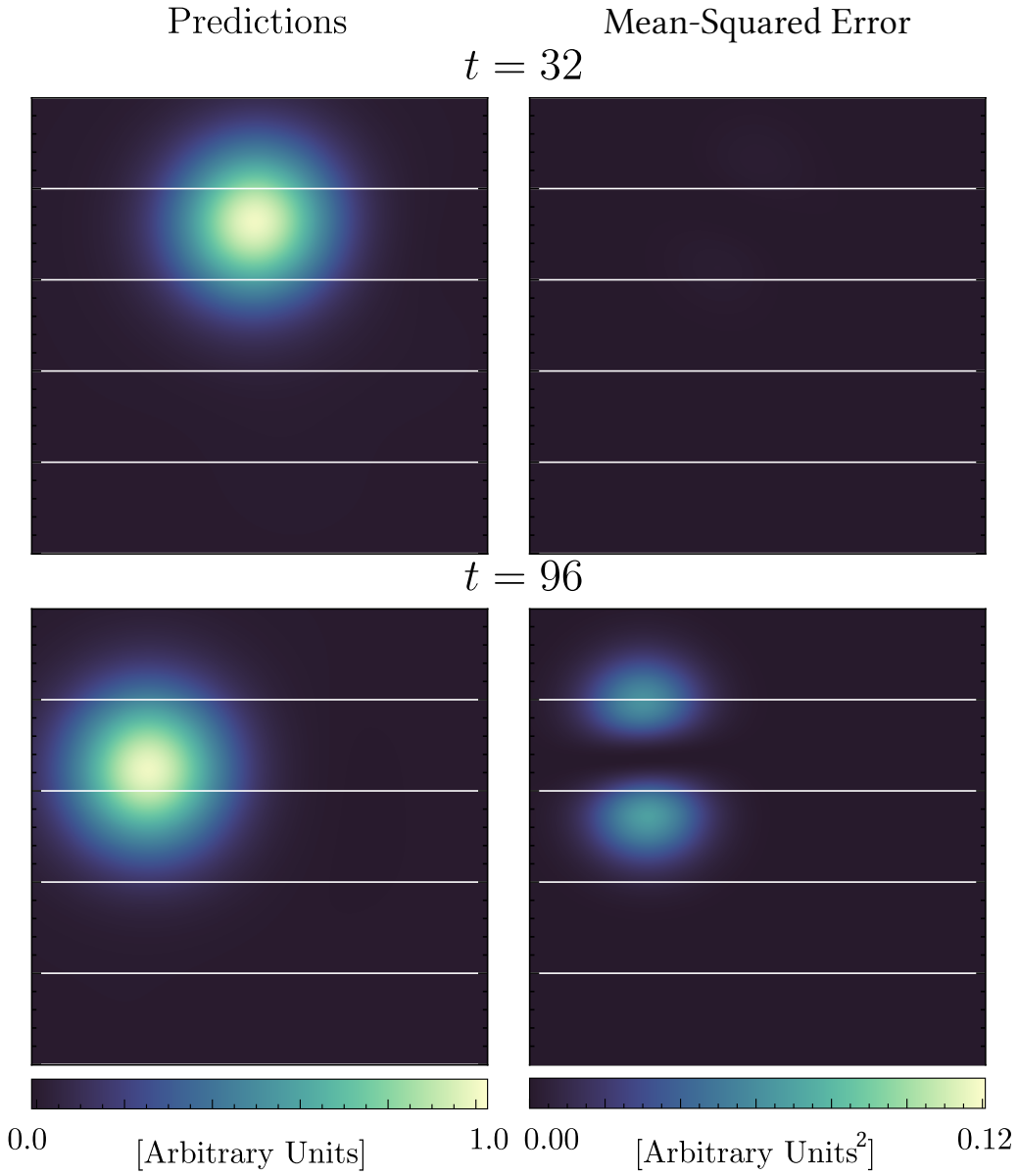



Figure 7.10: The 500 × 500 chaotic Mackey-Glass orb remains structurally coherent throughout predictions. In the first frame, at prediction time $t = 32$ the prediction is all but perfect, as the spatial MSE is small. Later, at time $t = 96$ two artefacts are seen. In fact, one has positive (absolute) error, and one has negative error. This implies, that although the predicted orb is coherent, its position is *shifted* in comparison to the unseen targets. For a chaotic system where small deviations can have large effects later on, this is *all* that one can hope for. Unfortunately the loss measure is still imperfect for recognizing that fact. This is especially relevant for the more complex ocean simulations.

## 7.5 Shallow Water Simulation

A *classic* example of a physical system modelled by simple partial differential equations is the **Shallow Water Model** (*Shallow Water Equations* 2021), and can be considered an *extremely simplified ocean model*.
In this section I (again) demonstrate the capabilities of the SSESN as a **spatio-temporal integrator**. I provide the SSESN with a training set that contains solutions to a 2D shallow water model over time. I here demonstrate the ability of the SSESN to make accurate predictions of the future sequence for a significant number of time steps.

This classic, simple model starts with a droplet dropping in the midst of a walled 'aquarium'. This creates waves that reflect back and forth in the aquarium due to gravitational forces. An animation of the training set, animation of the predictions, hyperparameter settings, and simulation code by James Avery can be found at GitHub[14]. Additionally, I show a frame-by-frame comparison of unseen targets and predictions in Figure 7.13.

We see that the SSESN is quite successful at predicting the evolution of the shallow water model. Eventually, the dynamics *blur out*, but even then the dynamics seem to capture the shapes of the targets.
To make accurate predictions, it was crucial to use a large enough $\sigma_{xy}$ such that the SSESN did not base its predictions on overly noisy covariates. $\sigma_{xy} = 3$ (pixel units) was found to be a good level in-between overly noisy, and overly blurry. $\kappa$ had a large influence and was successful at low values - in this case at $\kappa = 1 \times 10^{-3}$ and $\rho \approx 1$ worked best, avoiding amplification issues present at $\rho > 1$ for longer-term stability of predictions. I also used $N_{\text{PC}} = 1000$ and a low temporal $\sigma_t = 0.2$ that did not seem to have significant influence on the predictions.

To me, it is impressive that spatio-temporal dynamics can be approximated *without knowing* the equations that govern the system. While the shallow water model is simple, this demonstrates the utility in computational physics where unknown equations may govern measurements. The SSESN allows predicting the future of a spatio-temporal system in spite of that fact, although it does not directly lead to *interpretable solutions*. That is, I

[14] Shallow Water Model Materials: *https://github.com/jfelding/esn/tree/thesis_assets/shallow_water*

could *not* tell you the equations that govern the shallow water model by inspecting the SSESN (this is left as an exercise to the reader!).

### *7.5.1 Input Scaling Dramatically Impacts ESN Expressiveness*

In this case, $\kappa$ dramatically impacted the approximation capabilities, with values of order $10^0$ leading to poor prediction performance. While this is certainly a surprise to me, it is *not hard* to imagine why that might sometimes be the case:
When the training data (in this thesis always scaled from $(-1;1)$) is used to drive the system using Eq. 3.1 the non-linear activation function $\tanh(\cdot)$ can *saturate quickly*.

*Saturation* refers to the case where many or all outputs of the element-wise activation are *close to the endpoints* of the range of the function. For $\tanh(\cdot)$ the range is $(-1;1)$. At the end points of the inputs before input mapping, we see that $|\tanh(\pm 1)| \approx 0.762$, and at $\pm\pi$ the activation is approximately saturated $|\tanh(\pm\pi)| \approx 0.9963$. Above these values in magnitude *every output* becomes similar to $\pm 1$, and *information about the input is lost*. Please refer to Fig. 7.11 for an illustration of the saturation.

Beyond *saturation* $\kappa$ can also be thought of as modifying (along with $\rho$) the non-linearity of the ESN dynamical system, and therefore the transformation of the *regressors*. In Figure 7.11 one can imagine that close to the origin, a *straight line* can approximate the activation function well, while at larger magnitudes, the input *bends* more non-linearly (the straight line does not approximate this part well).

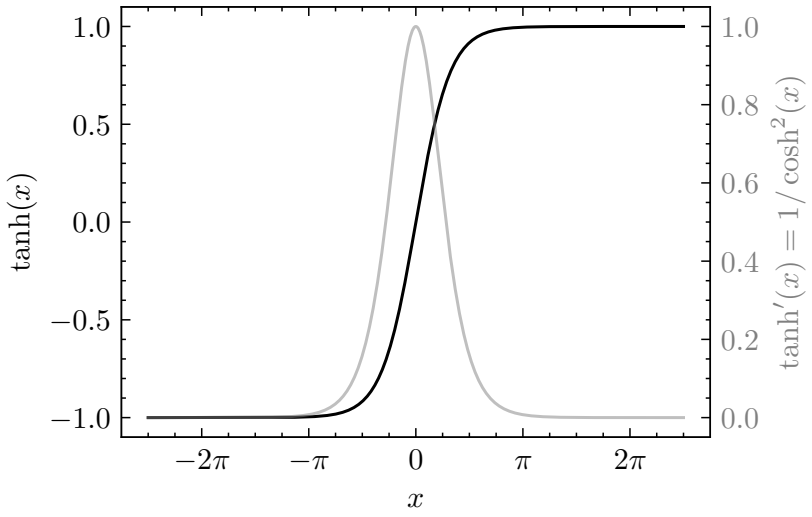



Figure 7.11: The Activation function $\tanh(x)$ and its derivative, to illustrate the change as a function of the input.

Prior to my observation that small values of $\kappa$ may sometimes benefit predictions, I *did* inspect the outputs of each input map, and I implemented further normalization of input maps to make the *output* of *input mapping* more reasonable (lower) in magnitude given the standard $(-1; 1)$ scaling of training data. I also implemented a simple warning if However, not all input maps guarantee a reasonable scaling, and some of them are based on random distributions, complicating matters.

Admittedly, the tuning of the SSESN (and ESNs in general) should be companied by plots of the *hidden state* to make sure that the states are not *overly saturated* (if only a few out variables are saturated — out of thousands according to $N_{\text{hidden}}$ it is *not* an issue).

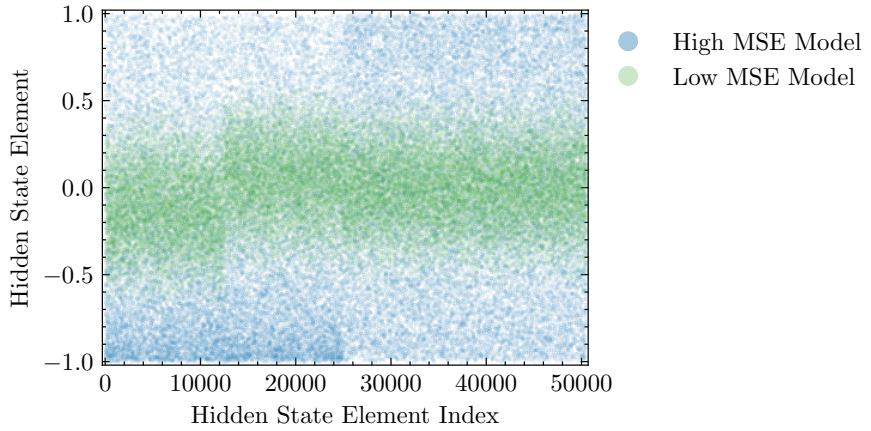



Figure 7.12: The last *training* set *hidden state* of two SSESN models that approximate the shallow water sequence. The model that has poor performance, and high MSE, is more saturated that the one that performs better, in this case. Saturation implies the *loss* of information, as the original input becomes indistinguishable, and uninteresting when applied as regressors. The four slight 'jumps' that are visible in the figure are *different input maps* that are applied

This particular dataset could be approximated so well that automatic hyperparameter optimized functioned properly, with MSEs low enough to correlate well with a qualitative evaluation, as one can see for yourself in Figure 7.13. The starting point of the predictions, $t_0$, is at 573.5s or at time step 3700.

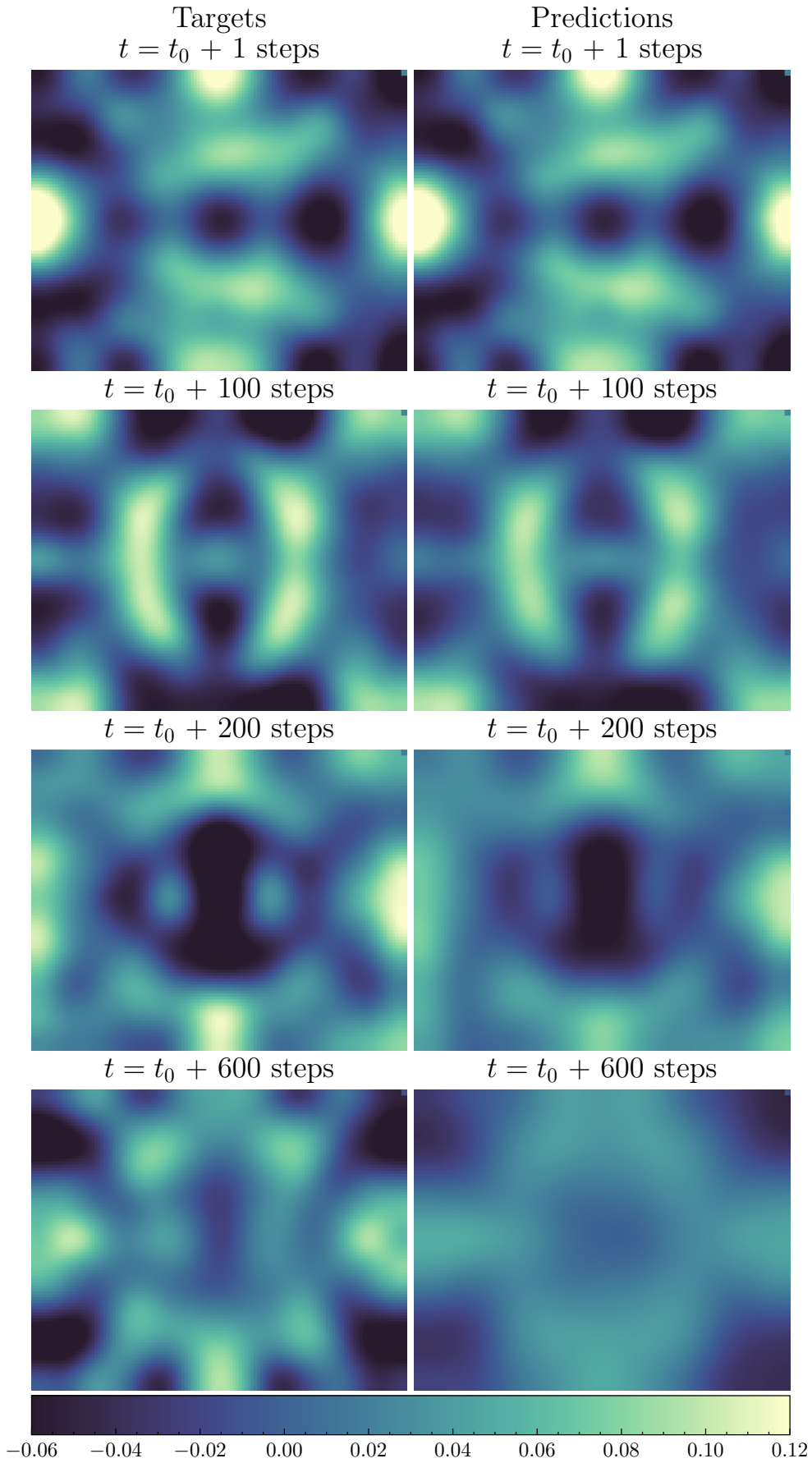



Figure 7.13: Sea surface height [m] in a $125 \times 100$ shallow water model. Each time step integrates the system forward by 0.155s. Hyperparameters: $\rho \approx 1$, $\sigma_{xy} = 3$, $\sigma_t = 0.2$, $N_{\text{PC}} = 1000$, $\kappa = 0.001$.

## 7.6 Full-Resolution Ocean Predictions on the Kuroshio

As I have alluded to in Section 7.2, I do not generally obtain better Ocean SSH predictions by applying automatic hyperparameter optimization. I will emphasize once again, that in a chaotic system, it is my conviction that *predictions that qualitatively capture the dynamics* are preferable to **predictions that minimize simple error measures** but *fail* to capture the dynamics by 'opting' for *on-average-okay* predictions that do not develop in time like the ocean. For truly chaotic systems, it is *an impossibility* to perfectly predict its future as even numerical precision limits the representation of initial conditions. Therefore that should not be one's aim. Predicting a likely scenario in the near-future, or getting large-scale features correct over longer time scales is an achievement in itself.

By a weather forecasting *analogy*, a more *bold*, if somewhat incorrect, prediction is also often preferable: That it '*will rain* tomorrow in cities *A, B* and *C*' is more useful than a prediction of: 'across the country, it will not, on average, rain tomorrow'. Even if the first prediction failed to predict that precipitation actually did occur in, say, cities $B, C, D$, and not $A$.

To obtain dynamically plausible predictions, I turn to the rather *tedious* task of *manually tuning* the hyperparameters to get qualitatively good results – an approach that is admittedly more *art* than *science*. I suspect that geophysicists of Team Ocean of the Niels Bohr Institute would be able to provide better evaluations of the models than I could.
I do so on the *Kuroshio* and *Kuroshio* extension CESM simulation data discussed in Section 1.3. After a day's worth of manual inspection of hyperparameter configurations and prediction animations, I find that:

- The predictions of the current is very sensitive to spectral radius $\rho$, input scaling $\kappa$, number of trained parameters (set using $N_{\text{PC}}$). $\sigma_{xy}$ also changes the dynamics of the predictions. The effects of the hyperparameters are of course intertwined.
- $\rho \approx 1$ produces more interesting dynamics than even slightly above or below that number. I use $\rho \approx 0.98$ to stay below $\rho = 1$ with high probability.
- A conservative input scaling of $\kappa = 0.5$ produces more interesting dynamics

- $\sigma_{xy} = 0.1°$ produced more interesting results than both $\sigma_{xy} = 0$ or values higher than 0.1°.

- Above $N_{\text{PC}} \sim 350$ the SSESN seems to be over-fitting, approximating the training set well, but generalizing somewhat poorly.

- Utilizing the *bilinear resampling* input map only gives good results with original resolution (no resizing). Adding the *gradient* and (20,20) *DCT* maps also provides good results, but it is *difficult* to manually evaluate whether it is better with certainty.

- With other parameters 'slow' dynamics are obtained, that often provide lower MSE: While they do not exhibit very likely dynamics, they **may be useful to anomaly detection** as they give *on average* better approximation of the stream, and since anomaly detection should most likely be based on the comparison of CESM simulations and SSESN predictions in the *short-term* rather than prediction months or years ahead of the training set.

To give the reader a sense of the differences that emerge from hyperparameter tuning, I show on the next pages several figures, each with a different configuration. I show both short-term and long-term predictions, the latter of which of course *cannot generally be accurate*. The figures all show the same image sequence, but with different predictions, and all plotted on the same color scale for the sake of comparison.

**A First Try**

First, in Figure 7.17 I have chosen hyperparameters naively from the intuition gained from the Mackey-Glass orb hyperparameter optimization, and I choose a training set with the same number of observations to better compare the two (I use the same dataset onwards). The chosen parameters are: $\rho \approx 2$, $N_{\text{PC}} = 800$, $\kappa = 1.5$, $\sigma_{xy} = 0.5°$ These parameters provide *very uninteresting predictions*, qualitatively. From the onset, predictions are blurred out, with very little change from frame-to-frame, although it is *not completely frozen*.

This model had runtime **267.2s** and MSE over 450 days and all space of **145cm**$^2$. An animation is available atGitHub[15]

### After Many Tries

Next, I show the **best predictions**, qualitatively, that I find to have more interesting dynamics, and continue to do so for the entire prediction period, set somewhat arbitrarily to 450 days in this instance. These are seen in Figure 7.18 and are the result of manually evaluating at 130 hyperparameter configurations in the long and short term. The parameters are drastically changed: $\rho \approx 0.98$, $N_{\text{PC}} = 350$, $\kappa = 0.5$, $\sigma_{xy} = 0.1°$. At no point does this system cease to develop dynamically over time.
This model had runtime **196.8s** and MSE over 450 days and all space of **163cm**$^2$. An animation is available at GitHub[16]

### Hyperparameter Sensitivity

Finally, to illustrate the sensitivity to the parameters, I show in Figure 7.19 prediction for which I have changed the parameters to $N_{\text{PC}} = 500$ and $\sigma_{xy} = 0.3°$. At first, the system is *remarkably similar* to the former, but the dynamics slow down over time to almost freeze. Compare the last frames to the former model. This model had runtime **236.9s** and MSE over 450 days and all space of **126cm**$^2$. An animation is available at GitHub[17]

### Spatial Mean Squared Errors

For the hand-tuned model, i.e. that seen in Figure 7.18, it is interesting to see which areas the SSESN gets right and wrong, and how quickly large errors occur. In Figure 7.16 we see that errors are generally *very small* in the first predictions, and as expected deviate more and more over time. However, not all parts of the area around Japan is error-prone. Instead, prediction errors occur at the SSH front of the Kuroshio and its extension that is *most dynamic*. Due to the dynamics of both the target and prediction,

[15]Animation: *https://github.com/jfelding/esn/blob/thesis_assets/kuro_fullres/esn131_verybad/comparison.mp4‘raw=True*
[16]Animation: *https://github.com/jfelding/esn/blob/thesis_assets/kuro_fullres/esn130_tuned/comparison.mp4?raw=True*
[17]Animation: *https://github.com/jfelding/esn/blob/thesis_assets/kuro_fullres/esn132_goodshortterm/comparison.mp4?raw=True*

the errors also *oscillate* after the initial error drift. Errors do not necessarily increase in magnitude after that point, but oscillate and eventually reach some kind of equilibrium error.

In addition to the qualitative evaluation of all SSESN instances, it is always relevant to evaluate whether the hidden states are overly saturated, as also visualized in Section 7.5.1. Like in that section, I display in Figure 7.15 the *last hidden state based on training observations* for the 'best' and 'worst' Kuroshio models from Figures 7.18 and 7.17, respectively. We see that saturation indeed does play a role for the model that *poorly approximates* the Kuroshio dynamics, while the better model does not overly saturate the $\tanh(\cdot)$ activation.

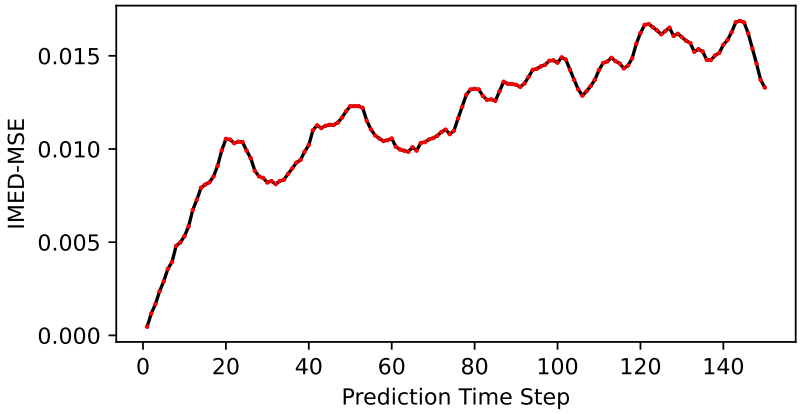



Figure 7.14: IMED-MSE over time for the hand-tuned model to predict the Kuroshio current, averaged over spatial dimensions. An initial error drift is always seen. Time resolution: 3 days. Parameters of the model: $\rho \approx 0.98$, $N_{\text{PC}} = 350$, $\kappa = 0.5$, $\sigma_{xy} = 0.1°$

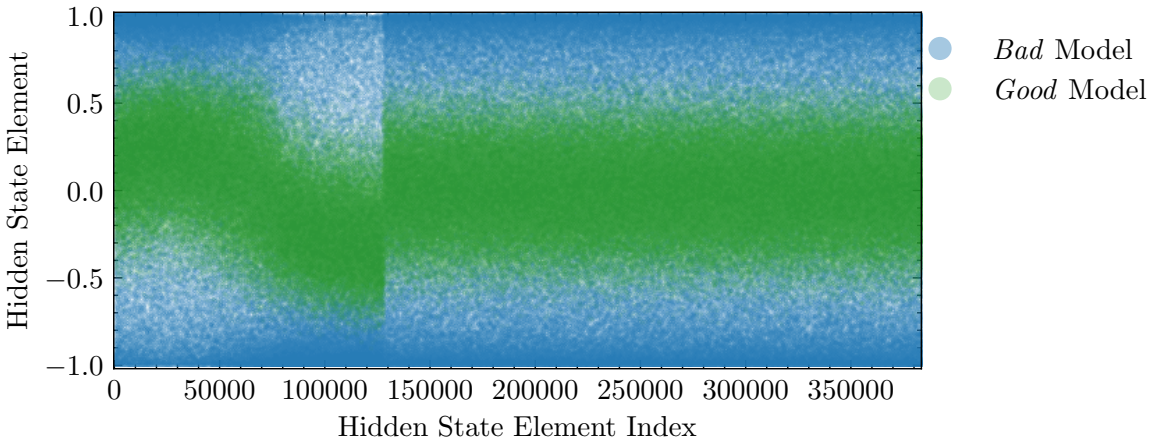



Figure 7.15: The 'bad Kuroshio model' in Figure 7.17 has saturated *hidden states*. Here, the last hidden state produced using training set data is shown.

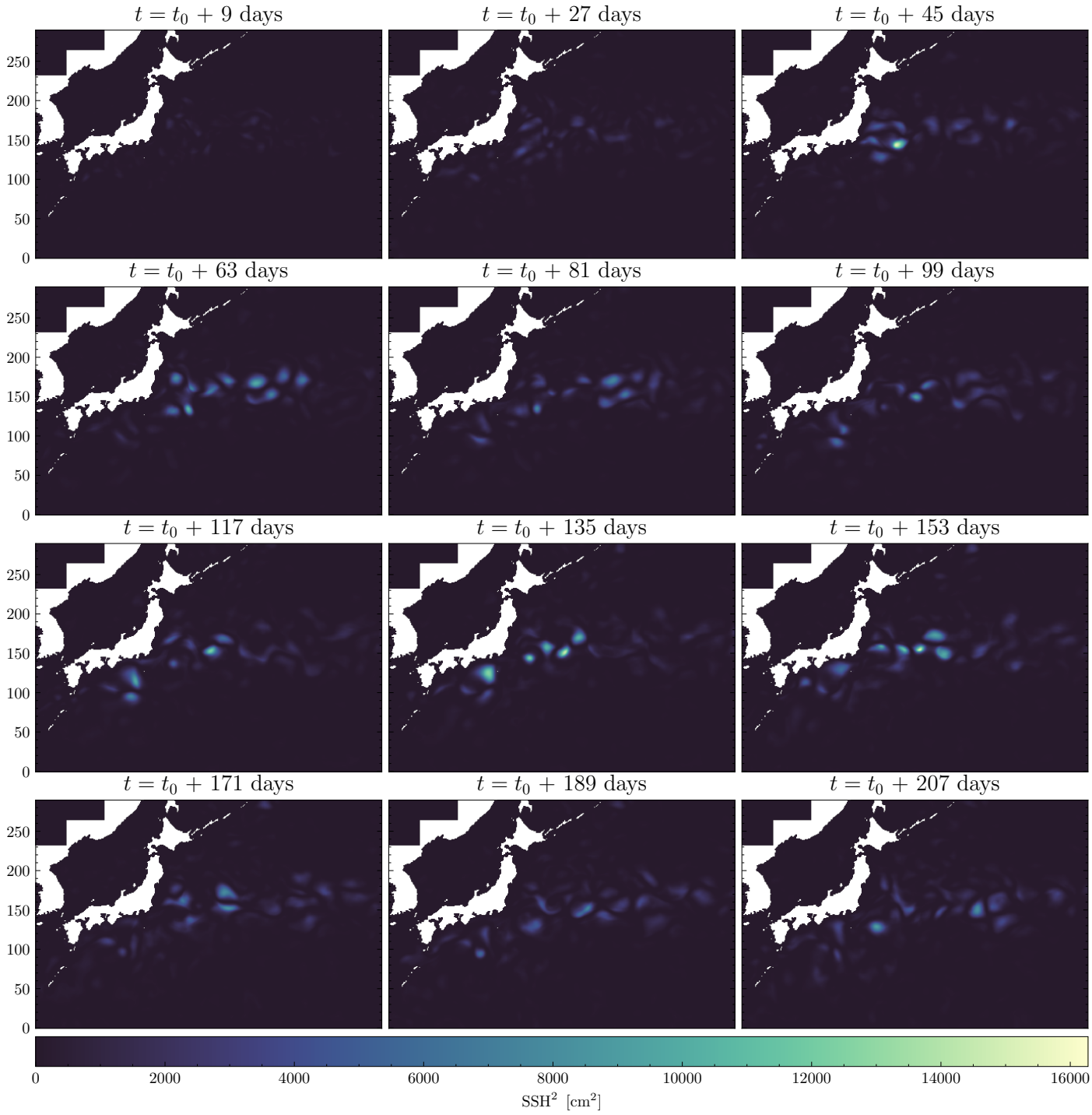



Figure 7.16: Prediction/Target MSEs for the Model seen in Fig. 7.18 with parameters $\rho \approx 0.98$, $N_{\text{PC}} = 350$, $\kappa = 0.5$, $\sigma_{xy} = 0.1°$.

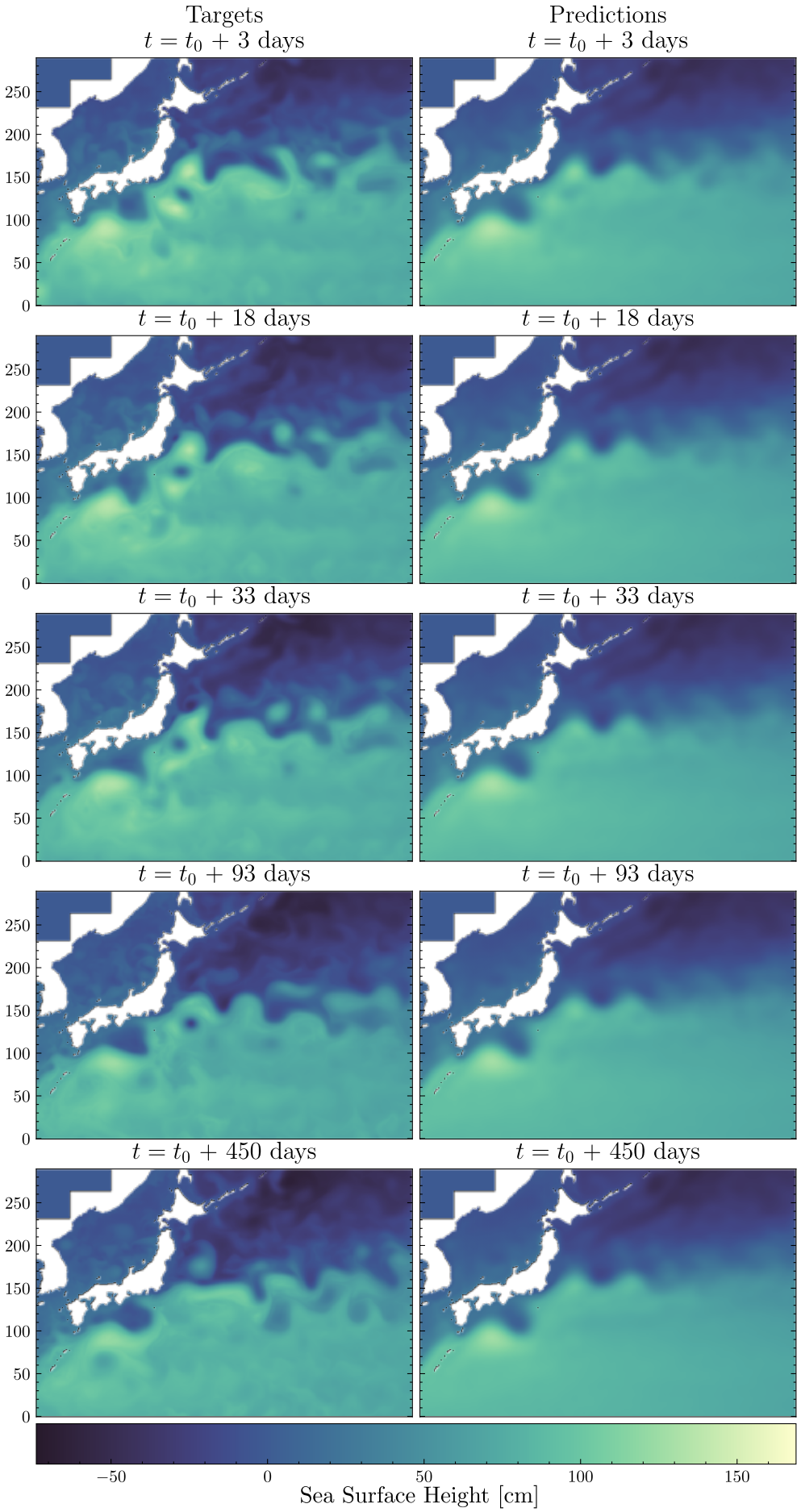



Figure 7.17: An initial configuration gives uninteresting dynamics:
$\rho \approx 2$, $N_{\text{PC}} = 800$, $\kappa = 1.5$, $\sigma_{xy} = 0.5^\circ$

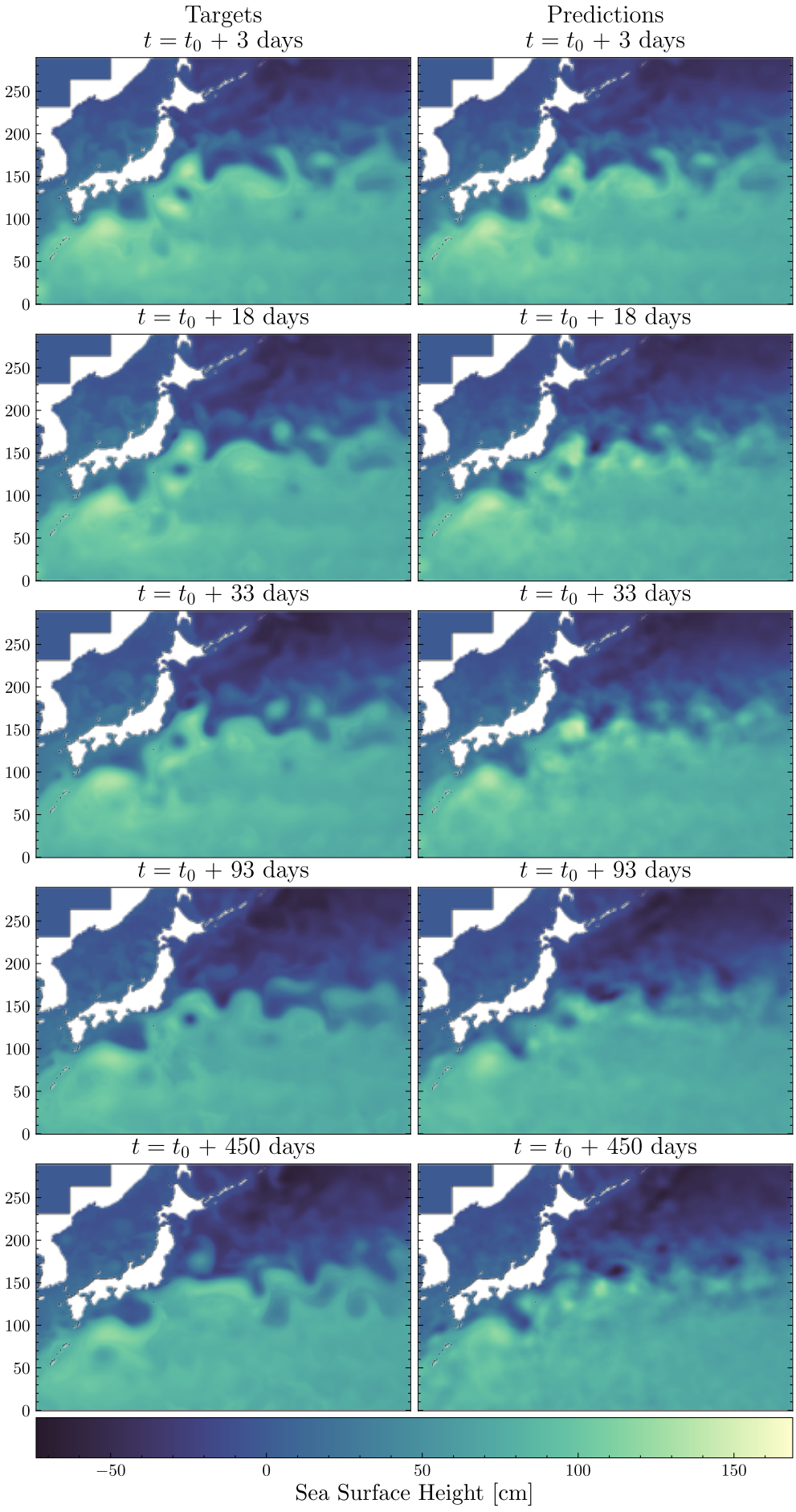



Figure 7.18: After manual tuning from 130 model outcomes, more interesting dynamics develop: $\rho \approx 0.98$, $N_{\text{PC}} = 350$, $\kappa = 0.5$, $\sigma_{xy} = 0.1°$

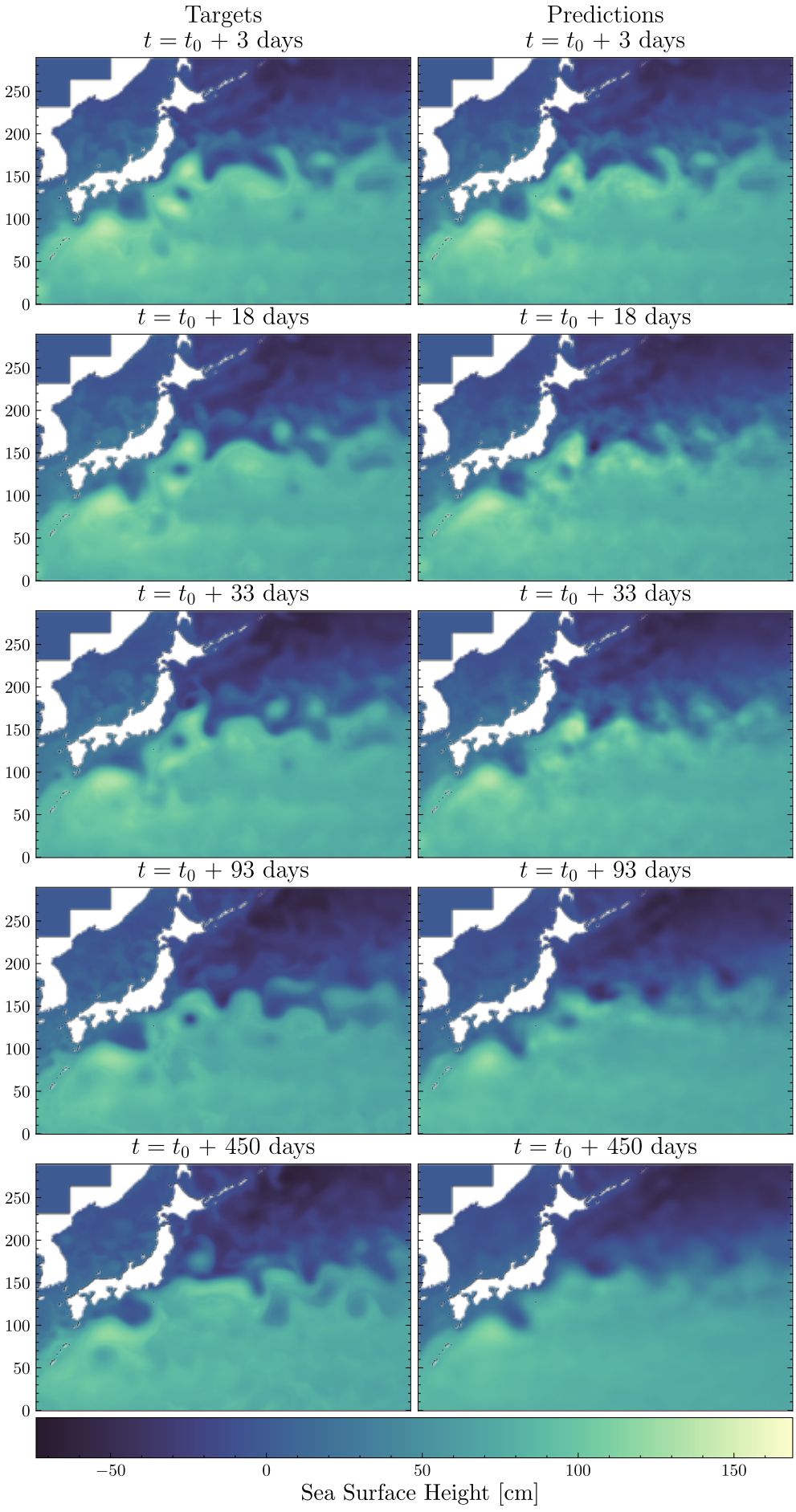



Figure 7.19: Changing a few parameters from Fig. 7.18 slightly gives a similar outcome at first, but dynamics freeze over time in this case: $\rho \approx 0.98$, $\mathbf{N_{PC} = 500}$, $\kappa = 0.5$, $\sigma_{\mathbf{xy}} = \mathbf{0.3}^\circ$.

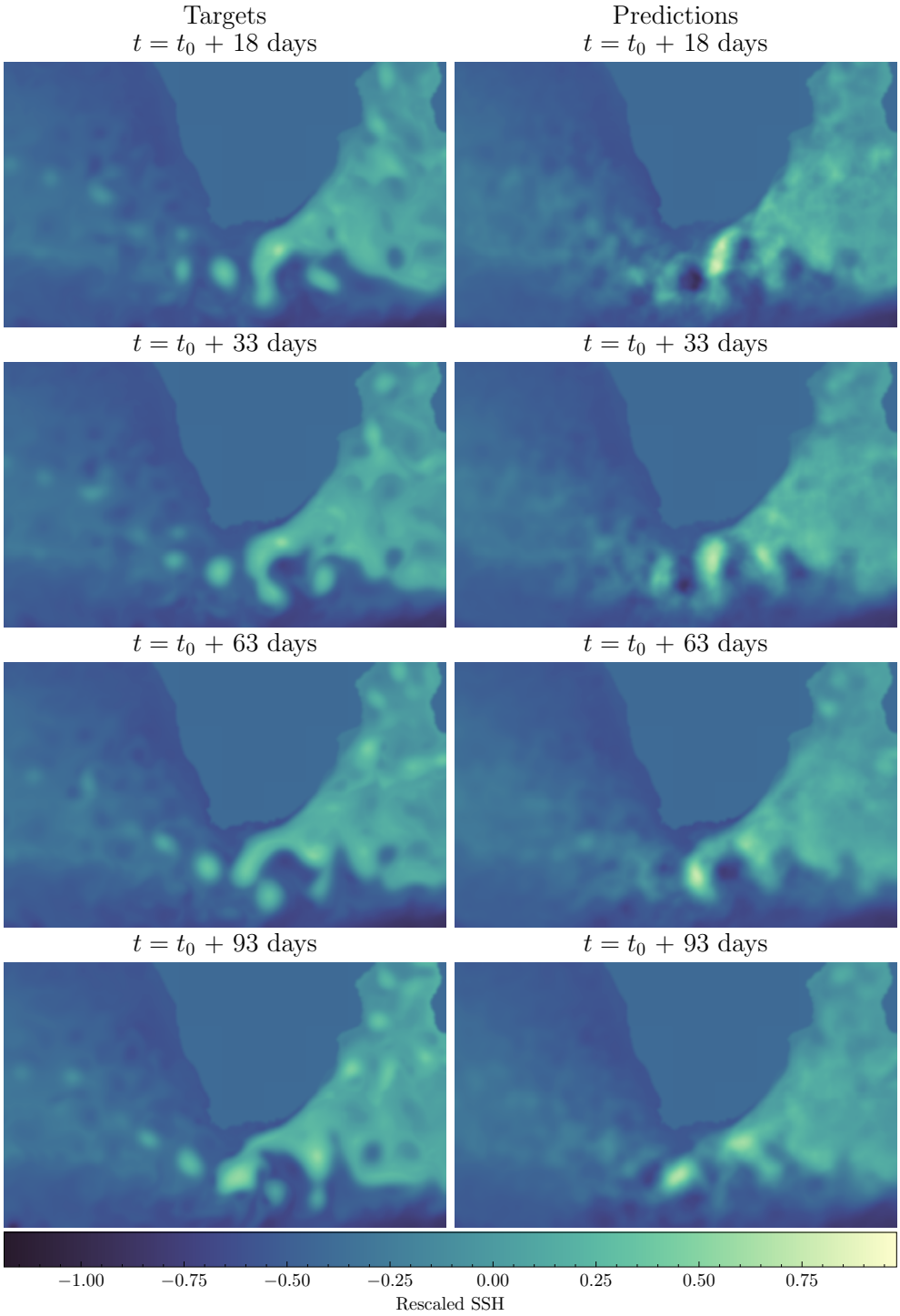



Figure 7.20: Predictions of the Agulhas Current with $\rho \approx 1$, $N_{\text{PC}} = 800$, $\sigma_{xy} = 0.2°$, $\sigma_t = 0.6$days.

## 7.7 The Agulhas Current

A quite different ocean system from the Kuroshio is *the Agulhas* at the coast of South Africa. It is the *strongest* current in the world, which leads to other dynamics than the Kuroshio. Therefore, it is interesting to see if the SSESN is capable of learning the dy-

namics in a qualitatively plausible manner.

I show here the first attempt at forecasting the Agulhas with parameters $\rho \approx 1$, $N_{\text{PC}} = 800$, $\sigma_{xy} = 0.2^{\circ}$, and $\sigma_t = 0.6$days. At that point $\kappa$ was set to 1, but it had a slightly different interpretation prior to some additional work on normalization of input maps.

The $500 \times 300$ frames at different time points (the series are 3-day means) are seen in Figure 7.20.

## 7.8 Time Complexity Benchmarking (16 Core CPU)

This section will explore the runtime, and scaling, of the Scalable Spatial Echo State Network versus the *Spatial* Echo State Network of (Heim and Avery 2019). The section does not *directly* describe the *memory complexity* of the ESNs, which is also a *major* selling-point for the SSESN. This is mainly due to the difficulties in profiling *jax* programs that the SESN applied.

However, for both the SESN and SSESN I run on a machine used exclusively by the script and *run until memory has lapsed*, which can be seen as an *indirect* gauge of memory consumption.

Before getting to the benchmark results, I will describe the setup, and considerations behind it.

### *7.8.1 Computational Considerations*

#### The Prospect of GPU Acceleration

Today, it is a common assumption that *deep learning* is carried out on GPU. This is especially true for the heavy gradient-based optimization methods.

Recurrent neural networks are not *embarrassingly parallel*, however, as the evolution of the hidden state $\mathbf{h}_t$ is an inherently *sequential* one. This is in contrast to e.g. pattern recognition learning on images using convolutional neural networks where the processing (convolution, mainly) of the images in a (mini-)batch can in principle be completely parallelized, only hampered by a sequential gradient update after, say, each completion

of an *epoch*, the entirety of the training set.

The ESN and SSESN operations that can be reasonably carried out on a GPU are:

1. Matrix-vector products of the dynamical system
2. Least squares optimization of the readout matrix
3. Principal component analysis for dimension reduction of hidden states

First **(1.)**, the GPU may be used for evaluating matrix products in parallel. This is true for RNNs, too, but they typically apply a lower-dimensional hidden state where reservoir parameters are, however, optimized. The sequential data transfer of the hidden state from SSD to dedicated graphics memory will lower the achievable speed-up, however.
**(2.)** With a linear ESN readout, least squares optimization is also distributable, as it consists of vector-vector and matrix-vector products in its most basic form. For the SSESN, dimension reduction is necessary for the matrix of harvested training states, $\mathbf{H} \to \mathbf{H}_r$, when the hidden state is very high-dimensional, or with a large number of training observations. This is due to the relatively limited dedicated GPU memory.
**(3.)** PCA can be thought of as mainly requiring diagonalization of the *Gram* Matrix — in the ESN case $\mathbf{H}^\top\mathbf{H}$ — but it is more efficient to an SVD algorithm with another approach (though still iterative), typically involving a QR decomposition. It is possible to parallelize PCA algorithms, and python implementations include:

- `h2o4gpu.solvers.pca.PCAH2O` (H2O.ai 2021), using a GPU-accelerated truncated singular value decomposition,
- `skcuda.linalg.PCA` (scikit-cuda 2021) using an iterative Gram-Schmidt orthogonalization algorithm, *'GS-PCA'*, introduced by (Andrecut 2009).

It is my, admittedly superficial, understanding that both implementations currently require the data matrix $\mathbf{H}$ to be transferred to the dedicated GPU memory before computations begin, which can be prohibitive.
Note that the aim of PCA for dimension reduction is an $(N_{\text{hidden}} \times N_{\text{r}})$ matrix to acquire the map $\mathbf{H} \to \mathbf{H}_r$, where $N_{\text{r}}$ is the number of dimensions to keep, limited from above by the minimal dimension of $\mathbf{H}$.
The memory consumption of the PCA transform matrix may also be prohibitive if partial

results are not iteratively transferred from GPU to main memory.

For future reference, perhaps to a new master student, I will also recommend the use of the library `cupy`/`cupyx` for drop-in replacement functions of `NumPy` and `SciPy`. The (old) **SESN is based on `JAX`**, whose current state have methods that are not always optimized for multi-core CPU usage out of the box. `JAX` allows implementations with just in time (JIT) compilation, but such implementations are not always feasible with the current state of `JAX` (beyond working with primitives).

**Memory, and Why CPUs Are Generally Preferable**

The current state of the SSESN implementation (see my GitHub) is built for CPU usage. Beyond the prospect of limited speed-up on GPU the main reason is that *reservoir computing is memory intensive*. Increased memory consumption over RNNs is *the price ESNs pay* for not optimizing the elements of the hidden-to-hidden matrix $\mathbf{W}_{\text{hh}}$ (ESN reservoir). To increase performance, and memory capabilities, ESNs instead employ a larger hidden dimension, and of course we *select* the spectral radius of the reservoir to induce stability. However, the discussion of applying accelerators is not extremely necessary for the SSESN, as we shall see. This is because the ESN optimization strategy is a single least squares optimization step, and is *much* faster than comparable gradient-based methods for optimizing RNNs, even if RNNs have much lower hidden dimensionality.

The main consideration for the SSESN is *memory*. Generally, it is advisable to increase the hidden dimension when the dimension of (spatial) input is increased, which comes at a cost. This allows the dynamical system to increase its *memory* of previous states. I again encourage the reader imagine ESNs as two completely separate parts:
(i) the dynamical system, and (ii) the readout (prediction generator).
For a number of reasons, as explained in Chapter 5, I introduce *PCA in the readout part only*, which allows the dynamical system to take advantage of high-dimensional hidden states, while ensuring stability and extremely fast training of the readout layer. It does, however, introduce a relatively large PCA transformation matrix with dimensions $(N_{\text{hidden}} \times N_{\text{r}})$ where $N_{\text{r}}$ is the number of dimension to keep, and that matrix must be kept in memory throughout the optimization and prediction setting.

The SSESN has *many* memory optimizations over the ESN:

- The frequency space implementation of the IMED does not require storing an $\left(N_x N_y \times N_x N_y\right)$ matrix, where $N_x, N_y$ are dimensions of the spatial inputs (images).
- No unused hidden states are stored unnecessarily in the transient or prediction state of the ESN.
- We allow representing data structures with lower precision than `float64` using a single `dtype` parameter, while this is not recommended due to numerical instability
- PCA makes the linear output matrix $\mathbf{W}_{\text{ho}}$ much smaller, $(N_{\text{train}} \times N_{\text{r}})$ instead of $(N_{\text{train}} \times N_{\text{hidden}})$ (later in this section I apply $N_{\text{hidden}} \sim 4 \times 10^6$, so the issue is real), but another $(N_{\text{hidden}} \times N_{\text{r}})$ where $N_{\text{r}}$ PCA transformation matrix is required, so the optimization is mostly one of time complexity wrt. optimization.

Either way, for high-resolution spatial inputs, a lot of memory can be consumed. Especially if the selected *input maps* (thoroughly discussed in Heim and Avery 2019) produce a hidden dimension that is a multiple of the high-dimensional (flattened) input. I find, however, that the SSESN with PCA dimension reduction is less sensitive to the number of spatial input maps: PCA finds *another* correlation than the differential operator interpretation of $\mathbf{W}_{\text{ho}}$ in the SESN.

To *allow* large memory consumption without excessive data transfers from main memory and to GPU, the ESN is primarily designed for CPU use, and I proceed with a CPU-only benchmark.

### *7.8.2 Hardware*

The main specifications of the machine used to benchmark are the following.

| | |
|---|---|
| **CPU** | AMD Ryzen Threadripper 1950X (16 cores, 3.4 GHz) |
| **GPU** | Nvidia GeForce RTX 2080Ti, 11GB GDDR6<br>(Not used for computations in this benchmark) |
| **Memory** | 128GB DDR4 2933MHz (quad channel) |
| **Storage** | 2x Samsung 970 EVO Plus NVMe M.2 2TB SSD (on Raid 0) |

For the SSESN, the memory capacity is especially important, as it effectively limits the problem size that can be computed.

The machine is running `Ubuntu 20.04.2 LTS (GNU/Linux 5.8.0-59-generic x86_64)` and uses `OpenBLAS` as `NumPy` back-end. On the AMD CPU there are *minor* speed-ups to cash in on by replacing `OpenBLAS` with the `AMD BLIS`[18] BLAS back-end as well as the `libflame`[19] library, which is *not* utilized for simplicity of this benchmark.
According to benchmarks on the hardware by PhD student Carl Johnsen about a 5% speed-up is realistic with sparse (CSR) reservoir-dense vector products that I have implemented in the SSESN, when dimensions are $1 \times 10^6$.

### *7.8.3 Setup, Hyperparameters and Problem Size*

The scripts that compute the benchmark of SESN/SSESN are available at GitHub[20] for inspection.

#### Input Series

As spatio-temporal inputs, the *orb*-like synthetic data is used, as seen in Sec. 1.2. This artificial data allows spatial rescaling to probe *spatial scalability* of the SESN/SSESN implementations in terms of time complexity. The benchmarking is *not* a parallelization benchmark i.e. *strong* or *weak* scaling. My SSESN implementation takes advantage of multi-core CPUs when possible, so to me, it is most interesting to take full advantage, and see what spatial scaling is feasible, especially limited by memory.

#### Problem Size and SESN Input Maps

This raises the question *what is the problem size?* It is natural for the end-user to see the spatio-temporal input as the problem size, i.e. the dimensions of the input volume. However, for the ESN, it is more natural to see the problem size as the chosen *hidden*

[18] *https://developer.amd.com/amd-aocl/blas-library/*
[19] *https://github.com/flame/libflame*
[20] Benchmarking scripts:
*https://github.com/jfelding/esn/tree/thesis_assets/scripts/benchmarks*

*dimension*, since the most time consuming operations are carried out in the *space of the hidden states*.

For the SESN introduced by (Heim and Avery 2019), the concept of *input maps* is introduced to the world of ESNs. The *traditional* ESN approach to transforming $N_{\text{input}}$-dimensional input at time $t$ to the *hidden space*, is the application of a *dense* random matrix $\mathbf{W}_{\text{ih}}$, which necessarily has dimensions $\left(N_{\text{hidden}} \times N_{\text{input}}\right)$.
The *ingenuity* of the SESN is to replace $\mathbf{W}_{\text{ih}}$ with with **linear maps** (functions) that provide spatial information to the hidden state, while being computationally more efficient than matrix-vector multiplication with a very large dense matrix $\mathbf{W}_{\text{ih}}$.
By the linearity of the *input maps*, they *could* be represented in matrix form (true as long as the input transform is *real-to-real*).

**Table 7.5:** *Input Maps of (Heim and Avery 2019) used by the SESN and SSESN in general.*
*SSESN has further improvements to scaling of outputs wrt. activation function* $\tanh(\cdot)$.
***Factor:*** *Scaling factor of map output to tune non-linearity of* $\tanh(\cdot)$ *(more linear behaviour closer to the origin)*
***Kernel Type:*** *"random" uses discrete convolution kernel shaped* $(x, y)$ *with random* $(-1; 1)$ *entries. "gauss" uses normal distribution.*

| **Map Description** | **Output Shape** | **Additional Arguments** |
|---|---|---|
| *Random Projection:*<br>Traditional ESN $\mathbf{W}_{\text{ih}}$<br>Mat.-Vec. Product | Any Integer | *Factor* |
| *Spatial Convolution* | Flattened Input Image | *Factor*,<br>*Kernel Shape*: (x,y)<br>*Kernel Type*: ("random"/"gauss") |
| *2D Image Gradient* | Flattened Input Image | *Factor* |
| *Discrete Cosine Transform:*<br>Low Frequency Amplitudes | $N_{\text{k1}} \times N_{\text{k2}}$,<br>at least $(2 \times 2)$ | *Factor*,<br>*Shape*: $(N_{\text{k1}}, N_{\text{k2}})$, number of sampled frequencies of each dimension; ascending order |
| *Bilinear Resampling:*<br>Up or Down | Any integer | *Factor* |

The properties of the input maps are summarized in Table 7.5.

Returning to the question of problem size, then, the table shows approximate dimensions of the input mappings, and we see that *convolution, random projection and gradient* each transform the input image to a flatted vector of the same size, whereas 'random weights', 'discrete cosine transform' and 'bilinear resampling' can be adjusted to ones liking. When several maps are applied, as recommended in (Heim and Avery 2019), the flattened outputs are all concatenated to a single vector that **defines the dimension of the hidden state**.
For the particular input map configuration that I use, see instead Table 7.6. Here, we see that the **total hidden dimension** is $N_{\text{hidden}} = 4N_{\text{input}} + 3725$.
My approach, then, is to create square spatial inputs $N_{1\text{D}} \times N_{1\text{D}}$ starting from $N_{1\text{D}} = 20$, providing $N_{\text{hidden0}} = 4925$ and increasing $N_{1\text{D}}$ *until memory is full* on the hardware, as described.
When showing timing results in diagrams, I have *hidden dimension* on the first axis, and one may simply compute $N_{\text{input}} = N_{\text{hidden}}/4 - 3725$ to see the 'spatial problem size', if need be. The hidden dimension is *relatively small* compared to the suggestions of (Heim and Avery 2019), which is more of an issue for the SESN than the SSESN due to PCA.

On the SSESN, I have taken additional steps to make the input maps well-behaved by ensuring that *rescaling* much beyond the range of the element-wise activation function, $\tanh(\cdot)$ when `factor=1`.

### Hyperparameters and Configuration

The interested reader can study the other hyperparameters used for the benchmarks in Table 7.6. One may notice that not all settings are identical. This is due to the vast differences of the SESN and SSESN prediction stabilities. The spectral radius is often increased above 1 for the SESN with no stability guarantees for the hidden state evolution, whereas the SSESN often thrives with a spectral radius around 1.
Additionally, different versions of the *IMED* (loss function) transformations are used. The SESN applied the original IMED implementation, which requires diagonalization and storing a large matrix, as described in Chapter 6. With my frequency implementations,

this is no longer an issue. For reference, in my tests of the SESN, the IMED transformations took roughly the same amount of time as the least squares optimization itself. In my results, I include IMED transformation in the 'optimization' category.

**Input Map Configuration**

| *Map Type* | *Dimension* |
|---|---|
| Bilinear Sampling: | $N_{\text{input}}$ |
| DCT: | $(15 \times 15)$ |
| Gradient | $2N_{\text{input}}$ |
| Random (3x3) Convolution | $N_{\text{input}}$ |
| Random Projection (Traditional ESN Map) | 3500 |
| Total Hidden Dimension $N_{\text{hidden}}$: | $4N_{\text{input}} + 15 \times 15 + 3500$ |

| **Contrasting Parameters** | **SSESN** | **SESN** |
|---|---|---|
| Spectral Radii, $\rho(\mathbf{W}_{\text{hh}})$: | 1.3 | 2.0 |
| Trainable Parameters per input pixel: | 500 (499 PCs) | $N_{\text{hidden}}$ |
| IMED Method: | FFT | Matrix |

| **Shared Parameters** | | |
|---|---|---|
| *Time Steps* | | |
| Transient Set: 100 | Training Set: 500 | Predictions: 100 |
| IMED Settings: | Spatial IMED $\sigma = 2$ | $\epsilon = 10^{-2}$ |
| Random Non-Zero Elements in all rows of $\mathbf{W}_{\text{hh}}$ : | 10 | |
| Data Structures & input data type: | `float64` | |

**Table 7.6:** *Hyperparameters of the Benchmark. For details on the input maps, see (Heim and Avery 2019)*

### Repetition, and Extreme Outliers

I find that it is not generally necessary to rerun the SESN/SSESN many times to get accurate wall time benchmark. I.e., *cache warming* does not have a major impact. For the

SESN, however, diagonalization is utilized two times: **(i)** when building the reservoir and rescaling to acquire specified $\rho$. This is sparse matrix diagonalization, performed using ARPACK, as discussed previously. Second **(ii)** the Matrix IMED implementation requires diagonalization of a dense matrix, performed using LAPACK.

The sparse eigenvalue decomposition (ii) starts from a random vector initialization, and convergence is *not guaranteed*, and not consistent when repeated. Slight differences in the end result are to be expected, too. For this reason alone, I use **3 timings per problem size**. A few times, the sparse diagonalization *did not converge at all*, which happens increasingly when the problem size is increased. One might argue that such timings should count as *infinite*, but that would not make nice plots. The SESN timings are therefore 'optimistic', as outliers with extremely high timings are *excluded* (perhaps the only option, as the program *never* completes).

The SSESN does away with diagonalization, and differences over the 3 repetitions are minimal (a low number of *per mille*).

With the setup completely laid out for the reader, we proceed to empirical timing results on the described hardware.

### *7.8.4 Benchmarks of Time Complexity of Spatial Dimensions (CPU)*

Finally, timing results may be viewed. Recall that I choose to run the SESN and SSESN with increasingly large problem sizes, until memory consumption exceeds the hardware configuration. With that in mind, Figure 7.21 (left) shows the total runtime of the ESNs from start to finish — training to prediction. Here, the SESN was limited by memory consumption at hidden state dimension above approximately $N_{\text{hidden}} > 60000$ or input $120 \times 120$. The SSESN was not, but I show them on the same scale for clarity.

I show the SSESN running both *with* and *without* the PCA dimension reduction layer for a perhaps more 'fair' comparison. The SSESN+PCA model uses a *fixed number of features* independent of the problem size, which the SESN does not. The PCA layer, however, does not only increase computational efficiency, but also often the prediction performance, as has been discussed in Section 5.1. The SSESN without PCA was limited by memory at $N_{\text{hidden}} = 243825$ corresponding to spatial input $245 \times 245$. Going forward, I refer to the "SSESN+PCA" model as "SSESN" unless I explicitly state otherwise.

Obviously, the SSESNs are much faster than the SESN, with their time consumptions peaking at 25s and 27s compared to 754s of the SESN, a **30x speed-up** at the same problem size. The SESN displays *super-linear time complexity* (not preferable) due to diagonalization (LAPACK: $\mathcal{O}\left(N_{\text{hidden}}^3\right)$), and least squares $\mathcal{O}\left(N_{\text{hidden}}^2\right)$ with $N_{\text{hidden}}$ number of trained parameters *per input pixel* (compared to 500 per pixel for the SSESN). The SSESN is obviously *faster*, but the time complexity is not clear from this plot.

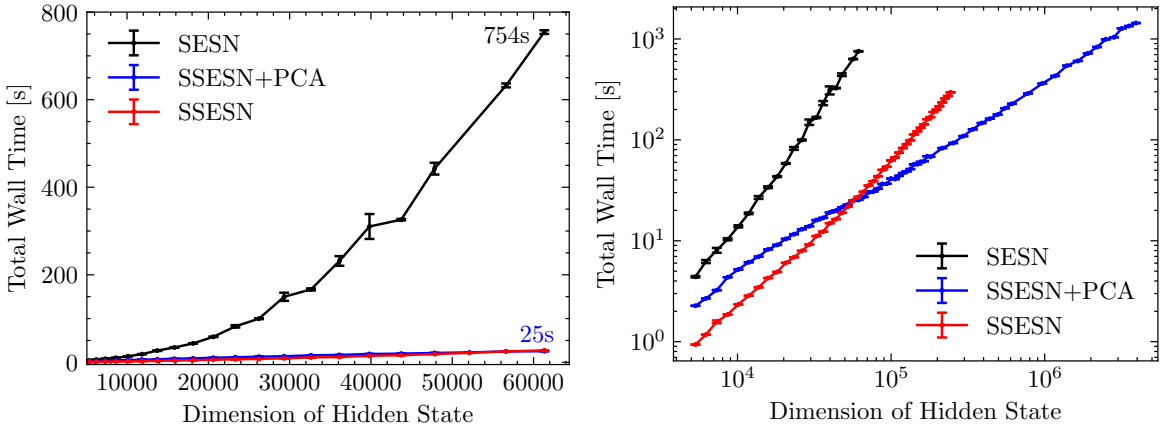



Figure 7.21: Runtime from start to finish of the ESNs as a function of hidden state dimension, the best descriptor of the RNN/ESN problem size. The SESN requires two diagonalizations of (increasingly) large matrices, whereas the SSESN does not, giving linear time complexity, as seen in Fig. 7.22 In this figure, the hidden state dimension was increased until the SESN needed more memory consumption than available on the system. Largest spatial input size was $120 \times 120$ pixels. I show two versions of the SSESN. One where the number of covariates is *fixed* using the PCA layer. And, for better comparison, one without PCA that has fits the same number of covariates of the SESN, although this is not generally beneficial to prediction performance.

It is, however, clear from Figure 7.22, that the SSESN exhibits *linear time complexity* wrt. the hidden dimension, a major improvement over the SESN. With the time consumption split in categories, we see that evolution of the hidden state (for initial transient evolution and *harvesting* prior to optimization) are very clearly linear in the hidden state dimension. Optimization has some *quirks*. This is due to the operations included in the overall 'optimization' category. The distinctions are clear from Table 7.7.

**Table 7.7:** *'Optimization' category of wall time plots (in orange)*

| **SESN** | **SSESN** |
|---|---|
| • Least Squares on hidden state covariates<br>• IMED (matrix) transform contained in least squares | • Unsupervised training of PCA transform<br>• Least Squares on dimension reduced hidden states<br>• IMED (frequency) transforms on input/output sequence |

The 'quirks', then, are a product of the frequency IMED method (here, *FFT* is chosen). FFT methods are quite optimized, and for *certain* problem sizes (the frequency IMEDs are applied to *input* and *target* data, not hidden states). When the transformed axis has length of *a power of two*, for instance, it is faster. In the future, the IMED implementations should be extended with a method that *resizes* the transformed axis according to the *next fast FFT length*.

The only part of the SSESN that is *not* faster than the SESN is the *prediction* phase, because the PCA transformation requires each hidden state at prediction time $t$ to be dimension reduced by matrix multiplication. The difference is more than outweighed by the optimization time savings (and prediction quality improvement) caused by utility of PCA.

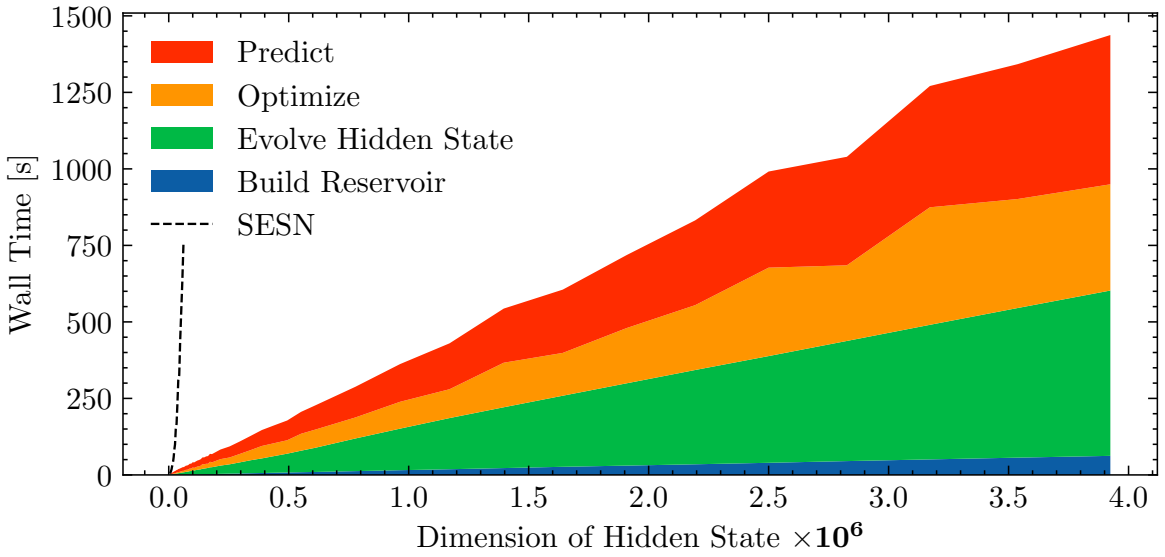



Figure 7.22: SSESN runtime at hidden state dimension (in millions) that was increased until execution was limited by its memory consumption on the system. While the SESN ran out of memory around $N_{\text{hidden}} \sim 6\times10^4$, the SSESN keeps going until $N_{\text{hidden}} \sim 4\times10^6$, a drastic increase. Total SESN runtimes are plotted for comparison. The SSESN exhibits linear time complexity across the range of problem sizes. Largest spatial input size was $990 \times 990$ pixels, compared to the $120 \times 120$ achievable for the SESN. $N_{\text{hidden}}$ is a better problem size indicator, however, as larger spatial predictions are achievable with the SSESN using fewer or lower dimensional *input maps*.

Finally, in Figure 7.23, I use a logarithmic first axis to compare the SESN and SSESN runtimes. This way, we may also see what processes are taking up most time for the SESN (shown with hatched area). Optimization (least squares, and IMED) is clearly taking up most of the time when pushing the SESN to its $N_{\text{hidden}}$ limit of $6 \times 10^6$. In my later tests, I find that the IMED and least squares steps took approximately the same amount of time, with IMED taking the lead at higher dimensions due to its diagonalization requirement for the matrix implementation.

It is interesting to see that the time consumed by the SESN at its limits corresponds to a $\sim$ 33-fold increase of the problem size for the SSESN. Note: The logarithmic plot should not be used for judging the time complexity, which is why I use the previous figures for those considerations.

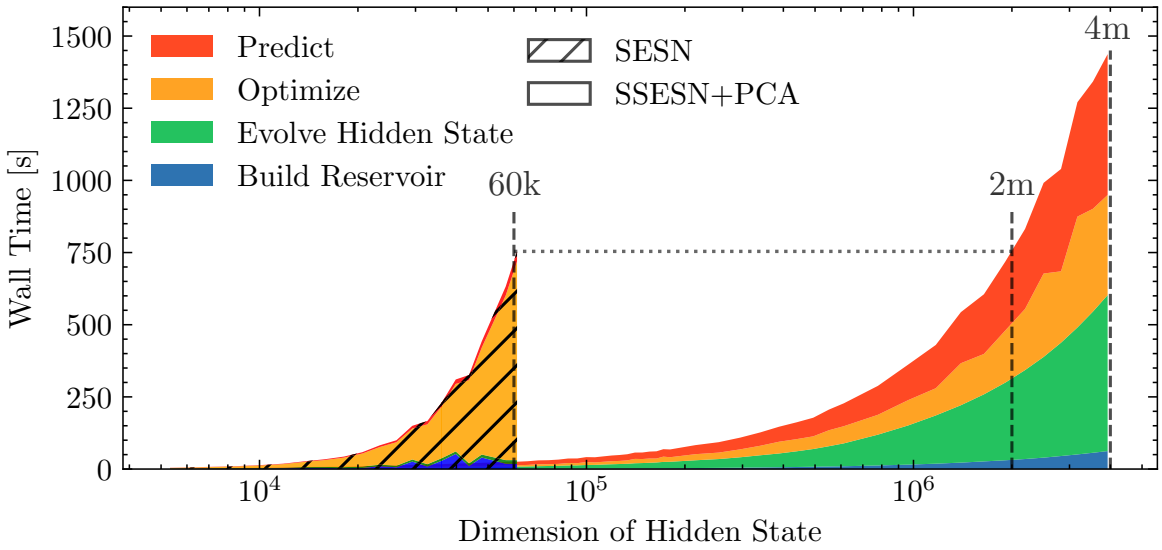



Figure 7.23: With a logarithmic first axis it is easier to compare the processes that take up time in the SESN and SSESN while noting that the runtime of the SESN at its $N_{\text{hidden}} \sim 6 \times 10^4$ limit is comparable to the SSESN at $N_{\text{hidden}} \sim 2 \times 10^6$.
Clearly, the optimization of the SESN dominates the wall time as the spatial dimensions are increased. This is no surprise, as the IMED method is prohibitively expensive in both time and memory, while the least squares approach (without a PCA step) train $N_{\text{hidden}}$ weights *per input pixel*, compared to a fixed number of weights per pixel for the SSESN with PCA.

# Chapter 8

# *Anomaly Detection*

---

Anomaly detection in chaotic time series, or image sequences, is *not easy*. But this chapter will demonstrate that anomalies in ocean surface topography can be pinpointed in time and space by recasting the hard anomaly detection problem as a prediction problem. One that the SSESN can handle.

This avoids the problem of *defining normality* in every ocean current of the world let alone its dependence on large-scale ocean and climate trends. The adaptivity of machine learning predictions, and the spatial scalability of the SSESN allows the global exploration of such ocean anomalies.

## 8.1 Method of Detecting Anomalies from Prediction-Target Comparison

Recall Figure 1.4 in which I alluded to the approach of anomaly detection taken by this thesis. The approach is *not an original idea* of mine, but the approach of (Heim and Avery 2019) who in turn apply the work of (Ahmad et al. 2017) in combination with their SESN.

To rid ourselves of *hard definition problems* of describing *normality* over time and space in e.g. oceans, we recast the anomaly detection problem as a *prediction problem* that provides an *error sequence* $\mathbf{E}(t)$ based on the comparison of *predictions* and *targets*.
I utilize climate model simulations to create machine learning (SSESN) *predictions* of the future outcome of the Sea Surface Height variable, and I then *compare* the simulations – prediction targets – to the predictions to achieve an *error measure*.

An attempt to visualize the entire anomaly detection procedure on the Kuroshio current SSH data is seen in Figure **8.1**.

### Moving Averages of the Prediction-Target Error Sequence

With an error measure describing how well the future climate simulations sequence is approximated by the SSESN predictions on the short term, **moving averages** on the error sequence can be utilized to construct an **normality score sequence**, $\mathcal{N}(t)$. The latter is the insight of (Ahmad et al. 2017) while the application to ML predictions is work of (Heim and Avery 2019).
The normality score sequence will have range $(0, 1($ with 1 being considered *completely normal*, and values very close to zero considered *anomalous*. (Ahmad et al. 2017) recommend a general **anomaly threshold** (a significance level) of $\mathcal{E} = 10^{-5}$ to trigger a binary **anomaly classification** at a particular time in the normality score sequence when $\mathcal{N}(t) < \mathcal{E}$.

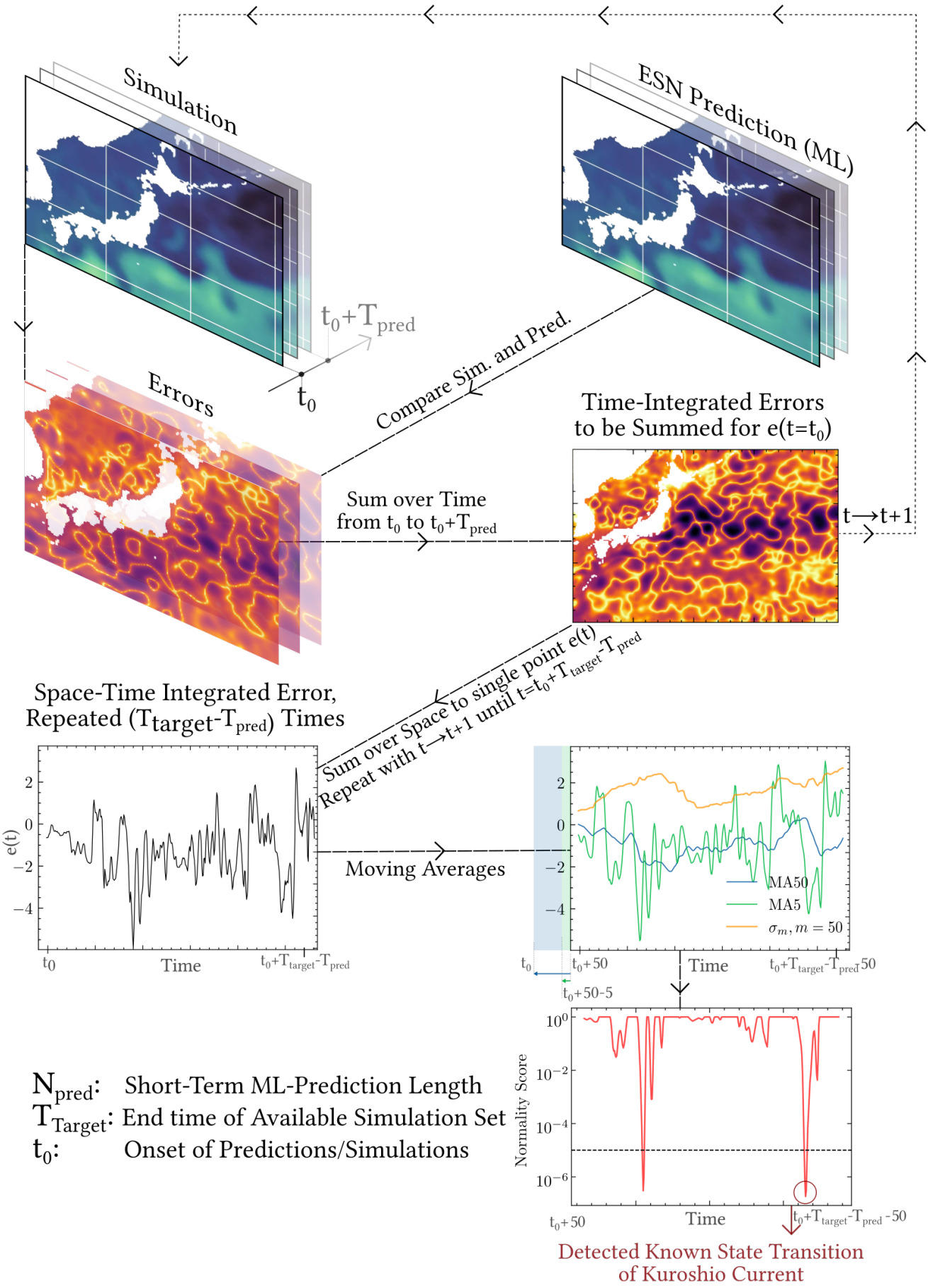



Figure 8.1: Anomaly detection procedure.

### 8.1.1 Online ESN Learning for Error Sequence Generation

Obviously, we expect predictions on chaotic oceans to deviate more and more from the simulation targets over time, and as such, we cannot robustly find anomalies years ahead of time (with the training set endpoint determining the starting point of predictions $t = t_0$). Instead, we take an **online learning** approach. The step-by-step construction of the *error sequence* using online learning (retraining once new information is made available) is seen in Table 8.1. Again, please compare to Figure 8.1 which seeks to illustrate the process.

**Procedure for Online Learning Predictions and Error Sequence Generation**

1. Initialize the hidden state and evolve through transient period from time step $t = 0$ to $t = N_{\text{trans}}$, e.g. $N_{\text{trans}} = 200$
2. Set $t' = 1$
3. Train SSESN on $N_{\text{train}}$ observations from time $t = N_{\text{trans}} + t'$ to $t = N_{\text{trans}} + t' + N_{\text{train}}$
4. Make free-running predictions from $t = t_0 + t'$ until $t = t_0 + t' + N_{\text{pred}}$ with $t_0 = N_{\text{train}} + N_{\text{trans}}$
5. Determine error (a single scalar) of the short-term predictions of $T_{\text{pred}}$ time steps by comparing with targets. Save this *space-time integrated* error in error sequence at time $t = t_0 + t'$, i.e. $\mathbf{E}(t = t_0 + t')$
6. Set $t' = t' + 1$ and go to 3 until $t' = T_{\text{target}} - N_{\text{pred}}$

**Variables**

$t$: Time step $t_0$: End time of 'training set' $t'$: Iterator
$T_{\text{target}}$: End of target seq. $N_{\text{trans}}$: Initial Transient Length $N_{\text{train}}$: Training Set Length
$N_{\text{pred}}$: Short-Term Prediction Length

**Table 8.1:** *'Online' learning is used for the SSESN to generate the error sequence. That sequence will later be processed to acquire a normality score.*

### 8.1.2 Smoothing the Error Sequence using Moving Averages

A *moving* average takes into account a number of recent observations in order to *smooth* a volatile time series, and is often used in financial analytics. I refer to $\mu_\tau = \text{MA}\tau[\mathbf{E}(t)]$ as the value of the *smoothed* error sequence created by averaging the $\mathbf{E}(\tau)$ and the $\tau - 1$

prior observations, i.e.

$$\mu_\tau(t) = \mathrm{MA}\tau[\mathbf{E}(t)] = \frac{1}{\tau}\sum_{i=0}^{\tau-1}\mathbf{E}(t-i) \tag{8.1}$$

The idea, then, is to make *two* moving averages on the error sequence $\mathbf{E}(t)$. One, very smooth, with a large window size $\tau_1$, and another $\tau_2 \ll \tau_1$ that changes more rapidly, and may even go as low as 1, meaning that it is no moving average at all. These settings depend on how volatile the series is. The smaller parameter $\tau_2$ should not smooth out major errors, while $\tau_1$ *should*. The idea is, that there should be a *significant* difference between the two, when something *unpredictable* happens.

Along with $\tau_1, \tau_2$ an important setting is $N_{\text{pred}}$, i.e. the number of time steps that are predicted ahead at each time $t$. When the system is very *chaotic*, like the ocean, it needs to be quite small so that turbulence does not dominate the error sequence.
Note that $N_{\text{pred}}$ influences the error values themselves (long-term predictions will give larger errors sums!). On the other hand, it only changes the *length of the error sequence* by cutting off at the last $N_{\text{pred}}$ points of the prediction dataset which ends at $T_{\text{target}}$.

Note that the first time point of the MA-sequences must be $t = t_0 + \max(\tau_1, \tau_2)$ where $\tau_1$ is the size of the largest MA-window.

In order to speak of *normality*, we also determine the *sliding window variance* $\sigma^2_{\tau_1}$:

$$\sigma_{\tau_1}(t) = \frac{1}{\tau_1 - 1}\sum_{i=0}^{\tau_1-1}\left(\mathbf{E}(t-i) - \mu_{\tau_1}(t)\right)^2 \tag{8.2}$$

At this point, we have acquired an (integrated) error sequence $\mathbf{E}(t)$, and moving averages of it: $\mu_{\tau_1} = \mathrm{MA}\tau_1[\mathbf{E}(t)]$ and $\mu_{\tau_2} = \mathrm{MA}\tau_2[\mathbf{E}(t)]$, and for the largest-window moving average, the sliding variance sequence $\sigma_{\tau_1}(t)$ We are therefore ready to construct a normality score by *comparing the MAs*.

## 8.2 Anomaly Score from Moving Averages of Error Sequence

Authors of (Ahmad et al. 2017) suggest the following *anomaly score* sequence:

$$\mathcal{A}(t) = 1 - Q\left(\frac{\mu_{\tau_1}(t) - \mu_{\tau_2}(t)}{\sigma_{\tau_1}(t)}\right) \tag{8.3}$$

Which is nicely normalized from $(0, 1)$ and can be interpreted as an anomaly probability. Here, the $Q$-function is the Gaussian *tail distribution function* and can be expressed using the Gaussian error function $\mathrm{erf}(\cdot)$:

$$Q(x) = \frac{1}{2} - \frac{1}{2}\mathrm{erf}\left(\frac{x}{\sqrt{2}}\right) = \frac{1}{2}\mathrm{erfc}\left(\frac{x}{\sqrt{2}}\right) \tag{8.4}$$

The $Q$-function has the interpretation that it expresses the probability $Q(x) = \Pr(X > x)$ for a Gaussian random variable $Y$ with mean $\mu$, variance $\sigma^2$ such that $X = \dfrac{Y-\mu}{\sigma}$ and $x = \dfrac{y-\mu}{\sigma}$, see (*Q-Function* 2021).

As mentioned, the anomaly score can then be *thresholded* using the suggested $\mathcal{E} = 10^{-5}$ (or another fitting significance level) for a binary normal/abnormal classification.

In the particular application of comparing predictions and targets, as in (Heim and Avery 2019), they made a small correction to the anomaly score:

$$\mathcal{A}(t) = 1 - 2Q\left(\frac{\max\left(0, \mu_{\tau_1}(t) - \mu_{\tau_2}(t)\right)}{\sigma_{\tau_1}(t)}\right) \tag{8.5}$$

The idea is that when $\mu_{\tau_1} < \mu_{\tau_2}$ the prediction error is larger than normally. We are not interested in the case $\mu_{\tau_1} > \mu_{\tau_2}$ where the prediction error is smaller than usual, as we consider that *normal*, which explains the utility of $\max(\cdot)$. An additional factor of 2 allows the probability interpretation since now $x > 0$ for $Q(x)$, with $2Q(x > 0)$ having range $(0, 1)$.

With online learning predictions in hand, this allows us to determine anomalies in ocean systems by means of comparing SSESN predictions to simulation targets.

## 8.3 Anomalies of the Kuroshio: A Proof of Concept

With the method of (Heim and Avery 2019) described, anomaly detection with the SS-ESN is *ready* to be showcased. The exact same data is applied here for demonstration purpose – only this time on a larger spatial area: on the 'full-resolution' CESM Kuroshio Current and the Kuroshio Extension, that we have encountered in Section 7.6. The data

are CESM three-day-means resampled to 5-day-means such that a year is defined as exactly $365/5 = 73$ days.

To make online-learning predictions, I use the following hyperparameters:

- $N_{\text{train}} = 780$ (in units of 5 days)
- $N_{\text{trans}} = 350$ (in units of 5 days)
- $N_{\text{pred}} = 30$ (in units of 5 days)
- $\left(\sigma_t, \sigma_x, \sigma_y\right) = (1 \text{ day}, 2^\circ, 2^\circ)$
- $N_{\text{PC}} = 150$
- $\rho = 1.0$
- Input maps: Bilinear resampling ($10\% \times 10\%$), random convolution, gradient, and random weights (traditional $\mathbf{W}_{\text{ih}}$ resulting in 10000 elements in $\mathbf{h}$).

I apply a cyclic buffer in that the 'training set' contains the same number of samples throughout such that conditions are the same at each round of predictions. That implies that the oldest training sample is removed once a new sample is made available (in the online learning approach). To speed up training (refitting $\mathbf{W}_{\text{ho}}$ at each step) the same PCA transformation is applied throughout, as the transformation is not expected to significantly differ as training progresses.

In Figure 8.2 I display the error sequence across simulation days in the prediction set (after the initial training period) along with moving average determined from the error sequence. These functions will be the basis for anomaly detection. To utilize the IMED, the samples (targets and predictions) are data saved from the SSESN internals. That is, the standardizing transform is applied to the volume, the training set is then rescaled to the interval $(-1; 1)$ (and prediction set accordingly), and predictions are made after these transforms. This provides a means of universal error measure that does not depend on the units of SSH (cm) or other variables that may be explored.

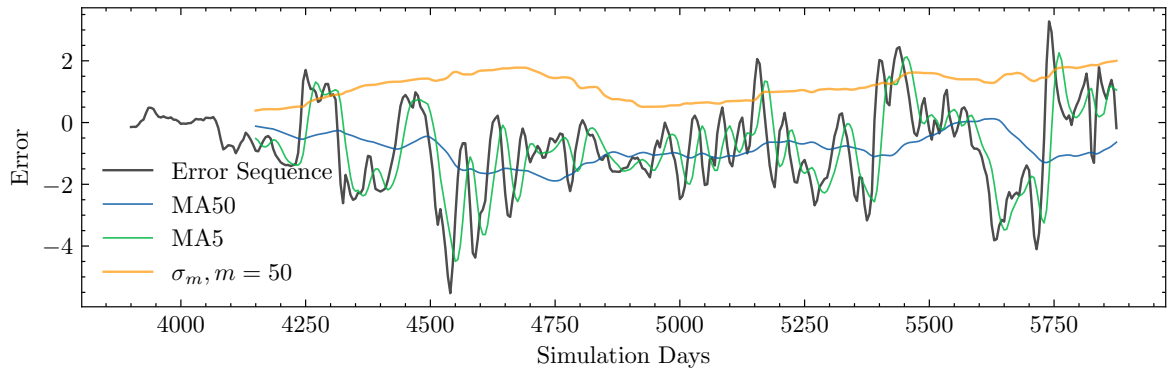



Figure 8.2: The error sequence and moving averages applied for determining a normality score as a function of the simulation days.

As described, these time series enter into the anomaly or normality score. In Figure 8.3 the *normality score* found on the entire Kuroshio area is displayed. The simulation anomaly is known to occur around simulation day 5500, as also displayed in (Heim and Avery 2019).

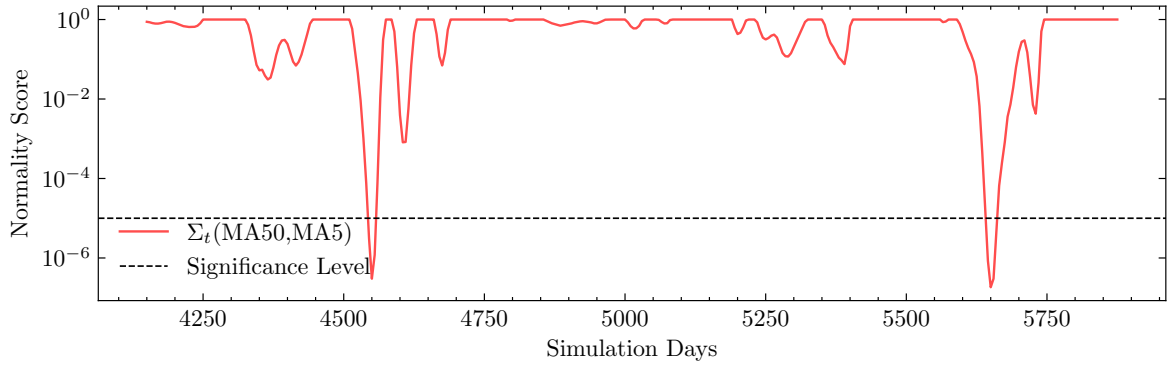



Figure 8.3: Normality scores based on Fig. 8.2 displaying the detection of two significant anomalies

The SSESN recovers the *known anomaly* detected in (Heim and Avery 2019) as it should, and also finds something anomalous about three years prior. From this visualization, it is not possible to pinpoint the area that is found to be anomalous. For this purpose, the anomaly detection system can work on *subsets* of the simulation/prediction series, and the approach can be repeated across the Kuroshio map.

With very little time allowed for the work in this chapter, further analysis is left as future work.

# Chapter 9

*Conclusion and Outlook*

## 9.1 Wrapping It Up

This project has sought to identify and eliminate all major bottlenecks in terms of memory and time consumption of the *Scalable Echo State Network* of (Heim and Avery 2019) *without* compromising its approximation capabilities. The development work, described in Part II of the thesis, can be summed up in three main contributions:

1. The further development and implementation of efficient spatially sensitive transformations and metrics;

2. A method to avoid diagonalization of the reservoir matrix $\mathbf{W}_{\text{hh}}$ to allow them to scale beyond traditional dimensions;

3. A dimension reduction layer to stabilize and optimize training using information from the high-dimensional hidden state.

In Part III of the thesis, I demonstrated the improvements by applying the Scalable Spatial echo state network to the same domains as (Heim and Avery 2019) and (Heim 2018) did, and more. That is, I used *synthetic training data* to familiarize myself with the SSESN, and I was able to apply the insights to CESM climate model simulation data to make *qualitatively plausible* predictions in a chaotic ocean system, specifically the Kuroshio current, the Kuroshio Extension and the Agulhas current, flowing along the southern coast of Japan and South Africa, respectively. The SSESN allows training and forecasting the future of individual grid points (pixels) for such systems **in a matter of minutes** while taking contextual information into account (the benefit of high-resolution). In doing so, the project has allowed scaling up the echo state network dynamical system to dimensions that are *orders of magnitude* larger than traditional ESNs while delivering the highly efficient optimization that is intrinsic to the *reservoir computing* approach of ESNs.
In Chapter 8, I demonstrated the utility of the SSESN in an online learning setting to pinpoint a *known* Kuroshio ocean surface topographic transition that last took place in 2017, and affects the climate of Japan to this day.

Despite many applications left to explore using the tools developed in this thesis, the thesis project has come to an end. Perhaps the most obvious application is the combination of the SSESN with the anomaly detection system that can be applied on other parts of the world to discover similar anomalous behaviour, allowing the exploration of real oceans

through simulation data, or the prediction of Kuroshio anomalies *with forewarning*. Since the CESM sea surface height variable is observable by satellite and available from NASA at e.g. (Zlotnicki et al. 2019), SSESN predictions and anomaly detection can be made on *real* measurements, and almost in real-time, as NASA frequently updates the datasets.

While some applications are straightforward at this point, and others are easily reachable everything is not perfectly finished! The next sections will discuss a number of problems that are regrettably left as future work.

## 9.2 Further Venues of Research

### *9.2.1 An Imperfect Loss Function*

**The Necessity of Manual Hyperparameter Optimization**

What perhaps bugs me the most is that, despite my implementation and development of the Image Euclidean Distance, it is *not* a perfect spatial measure. We see this especially with the complexity of predicting the ocean system where *automatic hyperparameter optimization* is not feasible. This is true for systems that are not *almost perfectly* approximated by the SSESN. By its nature, the chaos inherent to the ocean system *does not allow* perfect long-term approximation. When automatic grid search is tried, error measures often prefer predictions that *freeze* over time, or are qualitatively *wrong*.

That leaves only automatic hyperparameter optimization on the short-term, which does *not* guarantee the long-term behaviour. Therefore, much manual inspection has to be done, as I have tediously experienced first-hand.
Note: The loss function applied in the ESN (that has to be compliant with demands of least squares) need not necessarily be the same as the one that evaluates quality of predictions. For instance, it is probably not hard to detect when *dynamics freeze*.

One option is to apply *perceptual similarity measures* arising from pre-trained deep neural networks used in image classification problems. However, it remains to be seen whether these metrics can easily be used on input that is *unlike* what they are trained on *ImageNet*[1]

[1]ImageNet Database: https://www.image-net.org/

RGB camera images (Zhang et al. 2018).

### 9.2.2 GPU Utilization

As noted in Chapter 7, GPU utilization is straight-forward with mostly a few drop-in libraries (I have successfully cut about half of the runtime this way, but with an earlier code revision). However, memory is the most important thing to spatio-temporal ESN applications, and this often limits the use of GPUs. The sequential nature of ESNs *also* limit the gains that can be expected. Still, it can be useful in the context searching through the vast (simulated) oceans for which speed is of essence. Recently, marketed GPUs have gained dedicated memory, which is helpful to the ESN venture.

### 9.2.3 A Closer Look at Spatial Input Maps

The SESN of (Heim and Avery 2019) has introduced input mapping through *functions* rather than explicit matrix multiplication that is traditional to ESNs. This is great for efficiency!
Since the application of PCA, the readout matrix $\mathbf{W}_{\text{hh}}$ has a less *spatial* interpretation than previously. In particular, it should be investigated whether the information of the spatial input maps can be made *denser* without compromising prediction quality.

Options include:

- Dimension reduction of the input maps (or raw input) by means of PCA.
- Interpolation on the spatial input maps to reduce dimensions.
- The point above can be combined with a staggered-grid approach such that different input maps do not perform interpolation by 'removing' the same points.

### 9.2.4 Automatic Hyperparameter Optimization from ESN Dynamics

A very interesting perspective on ESNs is found in (Bianchi, Livi, et al. 2018) where *recurrence analysis* is applied to ESNs. By means of analysing the dynamics of the ESN, the authors find that an *edge of stability* at which approximation power is particularly effective, can be determined by gradually adjusting hyperparameters. The fast runtime of the SSESN would allow such an approach in many circumstances.

### 9.2.5 Distribution of Training using Online Learning

If the SSESN is used for anomaly detection on many, or very large systems, training *can* take a long time. This is because the *online learning setting* requires the constant re-fitting of $\mathbf{W}_{\text{ho}}$ when new information is made available. Time consumption also depends strongly on the number of short-term predictions that must be made. For the Kuroshio system, predictions 1-year ahead of each time step $t$ in the 'validation set' could still be accomplished in a matter of a few hours. This is *without* additional methods to incrementally update the least squares estimate, but with re-fitting at every step.
If necessary, this task is distributable to several machines, as the re-fitting of $\mathbf{W}_{\text{ho}}$ is independent of the predictions made at time $t$, depending only on the available training and target datasets.

### 9.2.6 Applying More Variables in the SSESN

The CESM model is one example of a system that has *many more variables* than the single SSH variable that this project has applied. The Team Ocean simulations contain about 90GB of information for a *single time step* (3-day-mean). The benefits of applying more available variables is obvious, and sometimes necessary to have enough information to provide accurate predictions of complex systems.
The extension of the spatial ESN to more variables should be straightforward, as it was *designed* that way by (Heim and Avery 2019).
More variables, of course, exponentially increase the size of hidden states. For small systems, this is certainly feasible. Otherwise, this change must be combined with more 'dense' input mapping, as suggested in Section 9.2.3.

Indeed, there are many interesting routes to take from this point on. I personally can't wait to see what the future holds for *spatial echo state networks!*

# *Bibiliography*


[1] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. “Leaning from Data : A Short Course”. In: *Leaning From Data - A Short Course*. California: AML-book.com, 2012. ISBN: 978-1-60049-006-4.

[2] Subutai Ahmad et al. “Unsupervised Real-Time Anomaly Detection for Streaming Data”. In: *Neurocomputing*. Online Real-Time Learning Strategies for Data Streams 262 (Nov. 1, 2017), pp. 134–147. ISSN: 0925-2312. DOI: *10.1016/j.neucom.2017.04.070*. URL: *https://www.sciencedirect.com/science/article/pii/S0925231217309864* (visited on 08/09/2021).

[3] E. Anderson et al. *LAPACK Users' Guide*. 3rd ed. PA: Society for Industrial and Applied Mathematics, 1999. ISBN: 0-89871-447-8. URL: *https://www.netlib.org/lapack/lug/* (visited on 04/20/2021).

[4] M. Andrecut. “Parallel GPU Implementation of Iterative PCA Algorithms”. In: *Journal of Computational Biology* 16.11 (Nov. 1, 2009), pp. 1593–1599. DOI: *10.1089/cmb.2008.0221*. URL: *https://www.liebertpub.com/doi/10.1089/cmb.2008.0221* (visited on 07/27/2021).

[5] Filippo Maria Bianchi, Lorenzo Livi, and Cesare Alippi. “Investigating Echo-State Networks Dynamics by Means of Recurrence Analysis”. In: *IEEE Transactions on Neural Networks and Learning Systems* 29.2 (Feb. 2018), pp. 427–439. ISSN: 2162-2388. DOI: *10.1109/TNNLS.2016.2630802*.

[6] Filippo Maria Bianchi, Simone Scardapane, et al. *Bidirectional Deep-Readout Echo State Networks*. Feb. 13, 2018. arXiv: *1711.06509* [cs]. URL: *http://arxiv.org/abs/1711.06509* (visited on 10/20/2020).

[7] *Black Swan Theory*. In: *Wikipedia*. May 7, 2021. URL: *https://en.wikipedia.org/w/index.php?title=Black_swan_theory&oldid=1021886041* (visited on 07/06/2021).

[8] Nicolò Cesa-Bianchi and Gábor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2009 (Print: 2006). ISBN: 978-0-511-54692-1. URL: *https://doi.org/10.1017/CBO9780511546921* (visited on 07/07/2021).

[9] Hans Dembinski and Piti Ongmongkolkul et al. "Scikit-Hep/Iminuit". In: (Dec. 2020). DOI: *10.5281/zenodo.4310361*. URL: *https://doi.org/10.5281/zenodo.4310361*.

[10] Raffaele Ferrari. *Ocean Turbulence*. URL: *http://ferrari.mit.edu/research/ocean-dynamics/ocean-turbulence/* (visited on 07/24/2021).

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. 1st ed. MIT Press, 2016. ISBN: 978-0-262-03561-3.

[12] H2O.ai. *H2o4gpu.Solvers.Pca — H2O4GPU 0.3.2 Documentation*. Version 0.3.2. URL: *https://docs.h2o.ai/h2o4gpu/latest-stable/h2o4gpu-py-docs/html/_modules/h2o4gpu/solvers/pca.html#PCA* (visited on 07/27/2021).

[13] Barbara Hammer. *Learning with Recurrent Neural Networks*. Lecture Notes in Control and Information Sciences 253. London ; New York: Springer, 2000. 148 pp. ISBN: 978-1-85233-343-0. URL: *https://link.springer.com/content/pdf/10.1007%2FBFb0110016.pdf*.

[14] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning - Data Mining, Inference, and Prediction*. 2nd ed. Springer Series in Statistics. Springer Science, Jan. 2017. URL: *https://web.stanford.edu/~hastie/ElemStatLearn/printings/ESLII_print12_toc.pdf*.

[15] Niklas Heim. "Automated Anomaly Detection in Chaotic Time Series". Master's thesis. Copenhagen: Niels Bohr Institute, University of Copenhagen, Sept. 4, 2018. URL: *https://github.com/nmheim/thesis* (visited on 07/23/2021).

[16] Niklas Heim. *Nmheim/Esn: Echo State Networks in JAX!* Jan. 27, 2021. URL: *https://github.com/nmheim/esn* (visited on 06/24/2021).

[17] Niklas Heim and James E. Avery. *Adaptive Anomaly Detection in Chaotic Time Series with a Spatially Aware Echo State Network*. Sept. 2, 2019. arXiv: *1909.01709* [cs, stat]. URL: *http://arxiv.org/abs/1909.01709* (visited on 10/01/2020).

[18] Masaki Kawabe. “Sea Level Variations at the Izu Islands and Typical Stable Paths of the Kuroshio”. In: *Journal of the Oceanographical Society of Japan* 41.5 (Nov. 1985), pp. 307–326. ISSN: 0029-8131, 1573-868X. DOI: *10.1007/BF02109238*. URL: *http://link.springer.com/10.1007/BF02109238* (visited on 07/24/2021).

[19] Cindy Kuiphuis. *Black Swan Photograph*. Dec. 23, 2018. URL: *https://commons.wikimedia.org/wiki/File:Zwarte_zwaan_black_swan.jpg* (visited on 07/06/2021).

[20] “LAPACK Benchmark”. In: *LAPACK Users’ Guide*. In collab. with E. Anderson et al. 3rd ed. Aug. 22, 1999. URL: *https://www.netlib.org/lapack/lug/node71.html* (visited on 08/02/2021).

[21] Jongwon Lee et al. “From O(k2N) to O(N): A Fast Complex-Valued Eigenvalue Solver for Large-Scale on-Chip Interconnect Analysis”. In: *2009 IEEE MTT-S International Microwave Symposium Digest*. 2009 IEEE MTT-S International Microwave Symposium Digest. June 2009, pp. 181–184. DOI: *10.1109/MWSYM.2009.5165662*.

[22] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK Users’ Guide: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. Oct. 8, 1997. URL: *https://www.caam.rice.edu/software/ARPACK/UG/ug.html* (visited on 04/15/2021).

[23] *Lissajous Curve*. In: *Wikipedia*. June 10, 2021. URL: *https://en.wikipedia.org/w/index.php?title=Lissajous_curve&oldid=1027858127* (visited on 08/04/2021).

[24] Liwei Wang, Yan Zhang, and Jufu Feng. “On the Euclidean Distance of Images”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.8 (Aug. 2005), pp. 1334–1339. ISSN: 1939-3539. DOI: *10.1109/TPAMI.2005.165*.

[25] Mantas Lukoševičius. “A Practical Guide to Applying Echo State Networks”. In: *Neural Networks: Tricks of the Trade*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Vol. 7700. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 659–686. ISBN: 978-3-642-35288-1 978-3-642-35289-8. DOI: *10.1007/978-3-642-35289-8_36*. URL: *http://link.springer.com/10.1007/978-3-642-35289-8_36* (visited on 07/29/2021).

[26] *Mackey-Glass Equations*. In: *Wikipedia*. July 6, 2021. URL: *https://en.wikipedia.org/w/index.php?title=Mackey-Glass_equations&oldid=1032294316* (visited on 08/05/2021).

[27] Sergiu Oprea et al. "A Review on Deep Learning Techniques for Video Prediction". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (Apr. 14, 2020). arXiv: *2004.05214*. URL: *https://doi.ieeecomputersociety.org/10.1109/TPAMI.2020.3045007* (visited on 10/01/2020).

[28] Mustafa C. Ozturk, Dongming Xu, and José C. Príncipe. "Analysis and Design of Echo State Networks". In: *Neural Computation* 19.1 (Jan. 2007), pp. 111–138. ISSN: 0899-7667, 1530-888X. DOI: *10.1162/neco.2007.19.1.111*. URL: *https://direct.mit.edu/neco/article/19/1/111-138/7142* (visited on 04/14/2021).

[29] Mads B. Poulsen, Markus Jochum, and Roman Nuterman. "Parameterized and Resolved Southern Ocean Eddy Compensation". In: *Ocean Modelling* 124 (Apr. 1, 2018), pp. 1–15. ISSN: 1463-5003. DOI: *10.1016/j.ocemod.2018.01.008*. URL: *https://www.sciencedirect.com/science/article/pii/S1463500318300258* (visited on 07/24/2021).

[30] *Q-Function*. In: *Wikipedia*. May 23, 2021. URL: *https://en.wikipedia.org/w/index.php?title=Q-function&oldid=1024742403* (visited on 08/09/2021).

[31] scikit-cuda. *Skcuda.Linalg.PCA — Scikit-Cuda 0.5.2 Documentation*. Version 0.5.2. URL: *https://scikit-cuda.readthedocs.io/en/latest/generated/skcuda.linalg.PCA.html* (visited on 07/27/2021).

[32] Yevgeny Seldin. *Machine Learning Lecture Notes*. Mar. 25, 2021. URL: *https://drive.google.com/open?id=1FYkrlmtNM5LJM5bydWy7uolSC7zVeW5y*.

[33] *Shallow Water Equations*. In: *Wikipedia*. May 12, 2021. URL: *https://en.wikipedia.org/w/index.php?title=Shallow_water_equations&oldid=1022755772* (visited on 08/09/2021).

[34] H. T. Siegelmann and E. D. Sontag. "On the Computational Power of Neural Nets". In: *Journal of Computer and System Sciences* 50.1 (Feb. 1, 1995), pp. 132–150. ISSN: 0022-0000. DOI: *10.1006/jcss.1995.1013*. URL: *https://www.sciencedirect.com/science/article/pii/S0022000085710136* (visited on 06/21/2021).

[35] Steven H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. Second edition. FL, USA: CRC Press, 2015. 513 pp. ISBN: 978-0-8133-4910-7.

[36] Shusaku Sugimoto, Bo Qiu, and Niklas Schneider. "Local Atmospheric Response to the Kuroshio Large Meander Path in Summer and Its Remote Influence on the Climate of Japan". In: *Journal of Climate* 34.9 (May 1, 2021), pp. 3571–3589. ISSN: 0894-8755, 1520-0442. DOI: *10.1175/JCLI-D-20-0387.1*. URL: *https://journals.ametsoc.org/view/journals/clim/34/9/JCLI-D-20-0387.1.xml* (visited on 07/24/2021).

[37] B. Sun and J. Feng. "A Fast Algorithm for Image Euclidean Distance". In: *2008 Chinese Conference on Pattern Recognition*. 2008 Chinese Conference on Pattern Recognition. Oct. 2008, pp. 1–5. DOI: *10.1109/CCPR.2008.32*.

[38] Bing Sun, Jufu Feng, and Guoping Wang. "On the Translation-Invariance of Image Distance Metric". In: *Applied Informatics* 2.1 (Nov. 25, 2015), p. 11. ISSN: 2196-0089. DOI: *10.1186/s40535-015-0014-6*. URL: *https://doi.org/10.1186/s40535-015-0014-6* (visited on 10/29/2020).

[39] Terence Tao and Van Vu. *Random Matrices: The Circular Law*. Feb. 28, 2008. arXiv: *0708.2895* [math]. URL: *http://arxiv.org/abs/0708.2895* (visited on 04/20/2021).

[40] UCAR. *Community Earth System Model - CESM®*. URL: *http://www.cesm.ucar.edu* (visited on 07/23/2021).

[41] Philip Matchett Wood. "Universality and the Circular Law for Sparse Random Matrices". In: *The Annals of Applied Probability* 22.3 (June 1, 2012). ISSN: 1050-5164. DOI: *10.1214/11-AAP789*. arXiv: *1010.1726*. URL: *http://arxiv.org/abs/1010.1726* (visited on 04/20/2021).

[42] Izzet B. Yildiz, Herbert Jaeger, and Stefan J. Kiebel. "Re-Visiting the Echo State Property". In: *Neural Networks* 35 (Nov. 2012), pp. 1–9. ISSN: 08936080. DOI: *10.1016/j.neunet.2012.07.005*. URL: *https://linkinghub.elsevier.com/retrieve/pii/S0893608012001852* (visited on 09/28/2020).

[43] Richard Zhang et al. "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recog-

nition (CVPR). Salt Lake City, UT: IEEE, June 2018, pp. 586–595. ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00068. URL: https://ieeexplore.ieee.org/document/8578166/ (visited on 10/13/2020).

[44] Victor Zlotnicki, Zheng Qu, and Joshua Willis. *SEA SURFACE HEIGHT ALT GRIDS L4 2SATS 5DAY 6THDEG V JPL1609. Ver. 1812. PO.DAAC, CA, USA*. 2019. URL: https://doi.org/10.5067/SLREF-CDRV2 (visited on 08/09/2021).