論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Robust Predictions with Reservoir Computing
著者(和文)	SENNCHRISTOPH
Author(English)	Christoph Walter Senn
出典(和文)	学位:博士(工学), 学位授与機関:東京工業大学, 報告番号:甲第12704号, 授与年月日:2024年3月26日, 学位の種別:課程博士, 審査員:熊澤 逸夫,山口 雅浩,小尾 高史,篠﨑 隆宏,渡辺 義浩
Citation(English)	Degree:Doctor (Engineering), Conferring organization: Tokyo Institute of Technology, Report number:甲第12704号, Conferred date:2024/3/26, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Robust Predictions with Reservoir Computing

a dissertation presented
by
Christoph Walter Senn
to
The Department of Information and Communications Engineering

in partial fulfillment of the requirements
for the degree of
Doctor of Engineering
in the subject of
Information and Communications Engineering

Tokyo Institute of Technology
Tokyo, Japan
March 2024

Thesis advisor: Professor Dr. Itsuo Kumazawa Christoph Walter Senn

Robust Predictions with Reservoir Computing

Abstract

Reservoir computing as a computation framework has gained increased attention in the last few years. Using a fixed, non-linear dynamical system and a simple readout layer (e.g., a single feed-forward layer) we can map input signals to desired outputs. The aim of this thesis is to improve the robustness of such systems against noise. First we propose a new type of reservoir computing systems using conductive fabrics, showing that it is capable of nonlinear computation and also exhibits memory. Then we show how to apply abstract interpretation on virtual and physical reservoir computing systems using time-series benchmarks. We also show how robust distributed anomaly detection can be realized and implemented with field programmable gate arrays. The significance of this work is the improvement of robustness using a new regularization term during training, and a method for distributed predictions, sharing network states.

Contents

Dedicated to my grandfather, Walter Fassbind, who always supported me in my academic endeavors.

Acknowledgments

This endeavor would not have been possible without, Prof. Dr. Itsuo Kumazawa, who always supported me during the years in his laboratory for my master and doctoral studies. I also could not have undertaken this journey without my defense committee, who generously provided knowledge and expertise. Additionally, this endeavor would not have been possible without generous support from the Japanese Ministry of Education, Culture, Sports, Science and Technology, who financed my research.

I am also grateful to Carmen Mei-Ling Frischknecht-Gruber, Alexander Blaschek, Valentin Zahnd and Alain Lang, who as friends always offered moral support during the past few years. Thanks should also go to Dr. Christian Hilbes, who always had useful insight and moral support and the rest of the safety-critical systems group at the ZHAW. Special thanks go to Prof. Dr. Rudolf Füchslin, who during my bachelor degree introduced me to reservoir computing.

I would also like to give special thank to Kiba and Yukiko for all the entertainment, emotional support and walks in the fresh air they provided.

Kiba (black coat) and Yukiko (red coat)

Listing of figures

1

Introduction

In recent years, reservoir computing has attracted increased interest, especially physical reservoir computing. Reservoir computing as a computational framework has been pioneered by Jaeger [9], Maas [10], and Steil [11] in the early 2000s. It started as a solution to vanishing and exploding gradients in recurrent neural networks but has since evolved beyond artificial neural networks. At its core, we separate the computation into a so-called

reservoir, e.g., a recurrent neural network, and a readout layer, usually a weighted sum of the reservoir state, e.g., the neurons in the case of a recurrent neural network, and only train the readout layer. In this way, temporal dependencies can be removed from the training process, as the reservoir dynamics will take care of integrating information over time. Another advantage is that the reservoir no longer needs to be a computational construct but can be any dynamical system, e.g., physical systems in the real world. For this reason, reservoir computing has attained more attraction over the last few years, as the complicated non-linear calculations that previously were done by the recurrent neural network, can be ”outsourced” to physical systems, potentially at lower cost, compared to using computers. Thus, reservoir computing promises to unlock a more sustainable way of computing. A brief introduction to reservoir computing and physical reservoir computing is given in Chapter 2.

To be able to use this framework in actual applications, we need to train the readout layer. This poses an issue for physical reservoir computing, as no two reservoirs generally have the exact same dynamics, i.e., it is not always possible to use the same readout layer amongst different reservoirs, even if they are supposed to be the same, i.e., the same type of reservoir. Another obstacle is attaining reservoir states for training. Depending on the application, for example a drone, it might not be feasible to get the necessary readings in a safe and repeatable manner. As such, numerical simulations are widely used in the scientific

reservoir computing community. This comes with the drawback that we cannot capture nature completely in simulations and, as such, a system trained with simulations is not generally transferable to the real world. This is also known as the "sim-to-real" gap, and is heavily explored in the reinforcement learning community [12]. Using a method of software verification named "abstract interpretation"[13], we show in Chapter 3, which is based on [7], how we can make such systems robust against some potential sources of noise when moving from simulation to the real world. Namely, building tolerances, that is, noise in initial conditions, and sensor issues, that is, noisy sensors, missing information, and shifted sensor signals.

But noise does not only affect physical reservoir computing systems; systems based on recurrent neural networks will also have to be able to handle noise, when employed in actual scenarios. One such scenario is the prediction of the future state of a system, for example, a facility. Based on this scenario, Chapter 4 explores a distributed approach to inference with inherent robustness against noise. Different reservoirs are assumed to run on their own network and being able to communicate their internal state to other reservoirs over network, with a delay. Using different readout layers and information sharing strategies, we explore different approaches for distributed inference. Information sharing between reservoirs seems to help in dealing with noise, and integration of information over time within the reservoirs hardens against communication errors between reservoir nodes. In addition,

information sharing is shown to be crucial to make predictions about the whole system state on a single node.

2

Reservoir Computing

This chapter introduces the basic notions of reservoir computing for virtual and physical systems. Starting with general recurrent neural networks, I want to motivate this computation scheme by exploring some shortcomings of recurrent neural networks and how reservoir computing solves them. Then, by emphasizing their nonlinear dynamical nature, physical reservoir computing is introduced. Using a mass-spring network, I show how phys-

ical systems can be used for computation, and I also introduce other physical systems that are being used in this manner. The reservoir computing framework was formalized around the same time by Jaeger [9], Mass and Markram[10], and Steil[11].

2.1 Recurrent Neural Networks

For our purposes, we define a general recurrent neural network as a dynamical system of the form shown in Equation 2.1, using a state vector x, representing the neurons of the network often also called the hidden state, an input vector u_t, parameters θ and some function F. In addition, we define a readout as in Equation 2.2, with y representing the output and some function G.

$$x_t = F(x_{t-1}, u_t; \theta) \tag{2.1}$$

$$y_t = G(x_t; \theta) \tag{2.2}$$

In practice we tend to use models of the form as shown in Equations 2.3 and 2.4, the weight matrices W, W_{input} and W_{output} as well as the biases b and b_{input} are the trainable parameters (θ in the general model). The activation function ϕ is usually a sigmoidal function, for example, the hyperbolic tangent or the logistic function, but also the rectified linear unit and its variations have become popular in recent years [14, 15, 16].

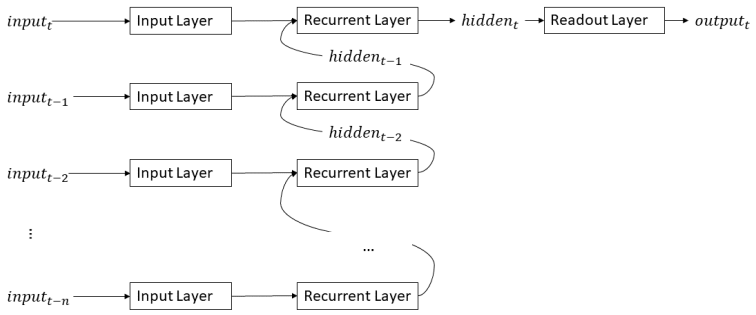


Figure 2.1: Schematic of unrolling a recurrent neural network over time, allows to apply backpropagation to update the parameters as if the network didn't have any recurrent connections. As can be seen the output at timestep t depends on timestep $t - n$, which can make training a general recurrent neural network difficult.

$$x_t = \phi(Wx_{t-1} + W_{input}u_t + b_{input} + b) \tag{2.3}$$

$$y_t = W_{output}x_t \tag{2.4}$$

The common approach to train this kind of system is backpropagation through time [17]. By unrolling the system over time, we can apply backpropagation to update the network parameters; this is reflected in Figure 2.1.

To show how this works in detail, let us introduce a loss function as in Equation 2.5 that calculates the mean of the function l, e.g. the mean squared error between the system output y and the target vector $\hat{y}$, up to the time index T and following the derivation of [18]

we get the following.

$$L(y, \hat{y}; W, W_{output}, W_{input}) = \frac{1}{T}\sum_{t}^{T} l(y_t, \hat{y}_t) \tag{2.5}$$

Then we can use the gradient $\frac{\partial L}{\partial W_{output}}$, assuming a linear activation for the outputs as shown in Equation 2.6.

$$\frac{\partial L}{\partial W_{output}} = \frac{1}{T}\sum_{t}^{T} \frac{\partial l(y_t, \hat{y}_t)}{\partial y_t} \frac{\partial y_t}{\partial W_{output}} \tag{2.6}$$

Similarly, the gradient for W is shown in Equation 2.7 and 2.8.

$$\frac{\partial L}{\partial W} = \frac{1}{T}\sum_{t}^{T} \frac{\partial l(y_t, \hat{y}_t)}{\partial W} \tag{2.7}$$

$$\frac{\partial l(y_t, \hat{y}_t)}{\partial W} = \sum_{i}^{t} \frac{\partial l(y_t, \hat{y}_t)}{\partial x_t} \frac{\partial x_t}{\partial x_i} \frac{\partial x_t}{\partial W} \tag{2.8}$$

As can be seen, due to the recurrence in our system the gradient for W at a time t depends not only on x_t but also on $x_{t-1}, x_{t-2} \ldots x_1$, giving us a term for $\frac{\partial x_t}{\partial x_i}$ as shown in Equation 2.9.

$$\frac{\partial x_t}{\partial x_i} = \prod_{i<j\leq t} \frac{\partial x_j}{\partial x_{j-1}} = \prod_{i<j\leq t} W^T diag(\phi'(x_{j-1})) \tag{2.9}$$

Equation 2.9 elongates with the time interval length T, and similarly to very deep feedforward networks, the gradient for W can explode or vanish, depending on the spectral radius of W and the activation function ϕ, e.g. using *tanh* as the activation function a spectral radius of > 1 lead to an exploding gradient and a spectral radius of < 1 will get us a vanishing gradient. An in-detail derivation of this claim can be found in [18]. In addition [19] showed that significant weight changes occur in a recurrent neural network in the output layer, i.e., W_{output} in Equation 2.4. Thus, training recurrent neural networks is notoriously difficult, but there are approaches that simplify and stabilize training, such as in the case of echo state networks [9], or more generally the reservoir computing framework.

2.2 Echo State Networks

2.2.1 Introduction

The general idea of echo state networks, and reservoir computing, is to have a reservoir mapping inputs to a high-dimensional nonlinear space, and then have a simple readout layer, e.g. a weighted sum, creating the desired output. In the case of echo state networks, the

reservoir is implemented as a random recurrent neural network.

We first augment Equation 2.3 by adding a leaking factor α, giving us Equation 2.10.

$$x_t = \alpha x_{t-1} + (1-\alpha)\phi(Wx_{t-1} + W_{input}u_t + b_{input} + b) \tag{2.10}$$

In addition, we define the washout time $\tau \geq 0$. Then to train such a network, we first drive it with the input signal u and record the state vector x for each time step $t > \tau$, giving us a state matrix X as defined in Equation 2.11.

$$X = \begin{pmatrix} x_{\tau+} \\ x_{\tau+2} \\ \vdots \\ x_T \end{pmatrix} \tag{2.11}$$

Together with the target signal $\hat{y}$ we get the linear system as in Equation 2.12, which we can solve for W_{output}.

$$\hat{y} = XW_{output} \tag{2.12}$$

Usually, this is done using ridge regression [20] giving us a regularized ordinary least squares estimator as in Equation 2.13 with a ridge parameter $\lambda \geq 0$. Naturally, other optimization techniques such as recursive least squares [21] or gradient descent are also used, depending on the application.

$$W_{output} = (X^T X + \lambda I)^{-1} X^T \hat{y} \tag{2.13}$$

Then when we want to perform inference, also called exploitation in the reservoir computing community, we update the reservoir states x_t as in Equation 2.10 and then use Equation 2.14 to create the output y_t.

$$y_t = W_{output} x_t \tag{2.14}$$

Equations 2.10 and 2.14 together with an external input u_t are sometimes also referred

to as open loop. Rewriting the exploit phase in the form of 2.15, by replacing u_t with the previous output y_{t-1}, gives us a closed loop. Adding this feedback loop allows for generative processes, e.g., signal/waveform generation or solving certain control problems.

$$x_t = \alpha x_{t-1} + (1-\alpha)\phi(Wx_{t-1} + W_{input}y_{t-1} + b_{input} + b) \tag{2.15}$$

$$y_t = W_{output}x_t \tag{2.16}$$

2.2.2 Why They Work

An intuitive understanding of why reservoir computing and in particular echo state networks works can be gained by considering a generalization of the Weierstrass approximation theorem[22], the Stone–Weierstrass theorem[23]. The Weierstrass approximation theorem states that any sufficient well-behaving function, i.e., continuous functions, defined in a closed interval $[a, b]$ can be approximated by a polynomial to any degree. The Stone–Weierstrass theorem, applied to our purposes, extends this to continuous functions from polynomials, that is, for any function h, we can find functions $f_1, f_2, \ldots, f_i$ and $g_1, g_2, \ldots, g_i$, so that $||h - \sum_i f_i g_i|| \leq \varepsilon$, for some weights w_i and some $\varepsilon > 0$. To link this with echo state networks, we consider the output of a single neuron as f_i, i.e., a time-dependent nonlinear function, the functions of the form g_i can be rewritten to constant functions $w_i \in W_{output}$,

i.e., the output weights of the readout layer, and finally h is our target filter or time series $\hat{y}$. Thus, echo state networks, and through association reservoir computing, can be thought of as a general approximator framework for nonlinear filters. For more theoretical insights, I refer to Maas and Markram [10].

As physics puts an upper limit on how big a neural network is, i.e., how many neurons it can have, to still be feasible to compute with, we cannot approximate any functions arbitrarily. In fact the memory of an echo state network is bounded by the number of neurons [24], at least in an open loop setup. By adding a feedback loop, we can bypass this limitation by introducing attractors that act as memory states [25].

2.2.3 Properties and Tuning

One of the core properties of an echo state network is the so-called "echo state" first defined by the Jaeger in 2001 [9] and revisited in [26]. It refers to the asymptotic behavior of the reservoir states. It allows the reservoir to forget or wash out past information; i.e., it dampens the reservoir dynamics. To achieve the echo state property, the spectral radius σ of the weight matrix W, is tuned. Independent of the input, the echo state property is attained for $\sigma(W) < 1$; and when also considering input values greater than 1 are also permissible. The relations between input and echo state properties have been further explored in [27]. The echo state directly being dependent on σW, the spectral radius is one of the first things to tune an echo state network for a given task. Generally, one tries to maximize it while still

maintaining stable reservoir dynamics. This maximizes the computational capability of echo state networks [28].

At first, it was believed that connectivity or rather sparsity of the recurrent neural network also has an influence on the capabilities by providing relative decoupling of subnetworks, enriching the reservoir dynamics [9]. But empirical results point towards no effect on the performance of echo state networks - despite that, sparse networks can still be preferable for a more optimal usage of computational resources, e.g., by allowing the usage of more optimized matrix multiplication algorithms.

A third parameter to scale is the input scaling, often denoted as ε, similar to the spectral radius σ, this scaling also has an influence on the reservoir dynamics. Very small values for ε can lead to a linearized reservoir, i.e. the inputs lead to such small changes in the reservoir neurons that it behaves linearly, similar to numerical differentiation. Large values, on the other hand, can lead to a binarized reservoir if a sigmoidal action function is used, that is, the changes in the neurons become so large that the function saturates; for example, we would only get 0s and 1s when using the logistic activation function.

2.3 Physical Reservoir Computing

2.3.1 Introduction

One big advantage of the reservoir computing framework is that it transcends the virtual world. As recurrent neural networks are, in principle, discrete dynamical systems, we can translate the reservoir computing framework to the physical world with a few steps.

Starting with Equation 2.10, we first replace the activation function with a general function f to get Equation 2.17.

$$x_t = \alpha x_{t-1} + (1-\alpha) f(x_{t-1}, u_t; W, W_{input}) \tag{2.17}$$

Ignoring the leaking factor α and examining the equation we can see that $f(x_{t-1}, u_t; W, W_{input})$ is essentially Δx, giving us a discrete dynamical system as in Equation 2.18.

$$\Delta x = f(x_{t-1}, u_t; W, W_{input}) \tag{2.18}$$

By extension, we also get the general continuous dynamical system by considering $\frac{\partial x}{\partial t}$ instead of Δx, giving us a link to physical dynamical systems. Unfortunately, generally it is

not always feasible or very difficult to adjust the parameters, i.e. W and/or W_{input}, of our system during training, making recurrent neural network like training difficult or even impossible. For example, we might want to use a compliant material as a reservoir to control a robot where the actuators act as the output and input to our system [29], effectively outsourcing part of the control to body, also known as morphological computing [30]. As the material properties are fixed and cannot be changed dynamically, we cannot use a back-propagation-through-time analogue for the physical world. What we can instead do is add some sensors to our compliant body, acting as a surrogate for the state x of the compliant body. We can then sample, as in echo state training, these states while forcing the actuators with a teaching signal, i.e., desired control signals, combine them into our matrix X and solve the system described in Equation 2.13. The calculated weights W_{output} can then be used with a small microprocessor to efficiently generate a control signal for the actuators. Such a system can, for example, be used to control a robot arm, as has been shown numerically by Bernhardsgrütter and Myself [1]. We attached a virtual mass spring system to the robot arm, as in Figure 2.2. This network simulated a compliant material, and we used the mass accelerations as sensor readings. The robot end effector was then forced to follow a circular path using precalculated actuator control signals, while the sensor readings were recorded. The actuator teaching signals acted as the training target, giving us sets of weights for each actuator. As can be seen in Figure 2.3, the robot then successfully recreated the

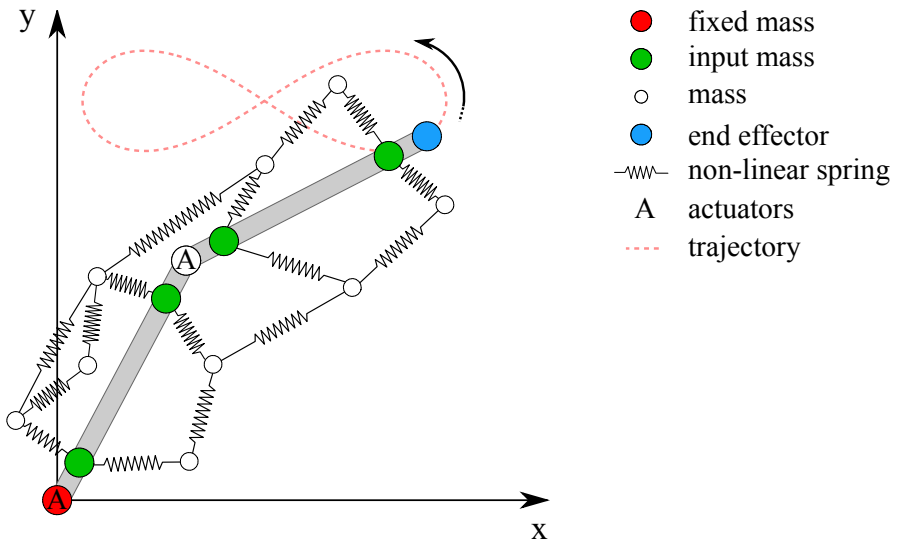


Figure 2.2: Schematic visualisation of the robot arm setup in [1].

circle and even recovered after external perturbations were applied.

2.3.2 Applications

The promise of physical reservoir computing is to enable state-of-the-art computation while requiring limited computational resources in the form of computers. This is done by outsourcing the difficult, non-linear part of the computation to nature itself, the physical reservoir. This also provides free multiprocessing, as one reservoir can be combined with different readouts for different tasks. As readout we can use any kind of computer, e.g., spanning from a microcontroller like an Arduino [31] to more powerful workstations. On the reservoir side, we can let our imagination run wild. In the following, we will explore some of the more interesting or creative variants.

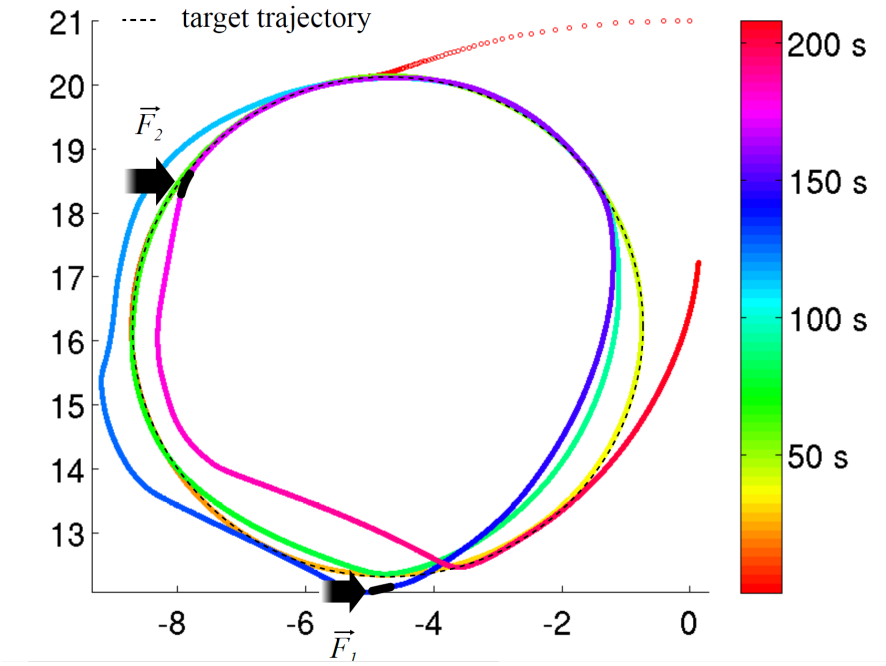


Figure 2.3: The path followed by the robot arm end effector in [1]. The color gradient describes the evolution through time. Note the perturbations F_1 and F_2 from which the system recovered.

Chrisantha Fernando and Sampsa Sojakka used a bucket of water, which they coined "The Liquid Brain" in 2003 as a reservoir [2]. The bucket had two stimuli as input, as shown in Figure 2.4. As input audio recordings of the words "zero" and "one" were used, after preparing their waveforms with Fourier transforms to be suitable as input for the motors that were attached to the Lego contraption. To read the state of the reservoir, a camera was pointed at the surface, and with some filtering and optimal lightning they got images of the form shown in Figure 2.5. The pixels were then fed into a simple perceptron, giving the classification of the waveform, "zero" or "one".

Around 2015, Najakima et al. [3] used an artificial tentacle, made of a compliant material as a reservoir. Figure 2.6 shows how a tentacle with embedded sensors and a linear readout can be used to approximate a desired signal*. They managed to approximate different time series with this setup, e.g., different non-linear NARMA time-series, and also showed that the non-linear dynamics of the reservoir are necessary to attain their results, i.e., a simple linear combination of the input signal was not enough to approximate their target signal as well as when the reservoir states were used.

Bhovad and Li [4], used Origami structures made of paper to create a reservoir. Their structure of choice was the Miura-Ori fold[32] in numerical simulations but also in a real-world proof of concept setup. In their simulations, illustrated in Figure 2.7 they randomly selected some creases as inputs and applied some moment on them, folding/unfolding

*There is also a video available under: https://www.youtube.com/watch?v=rUVAWQ7cvPg

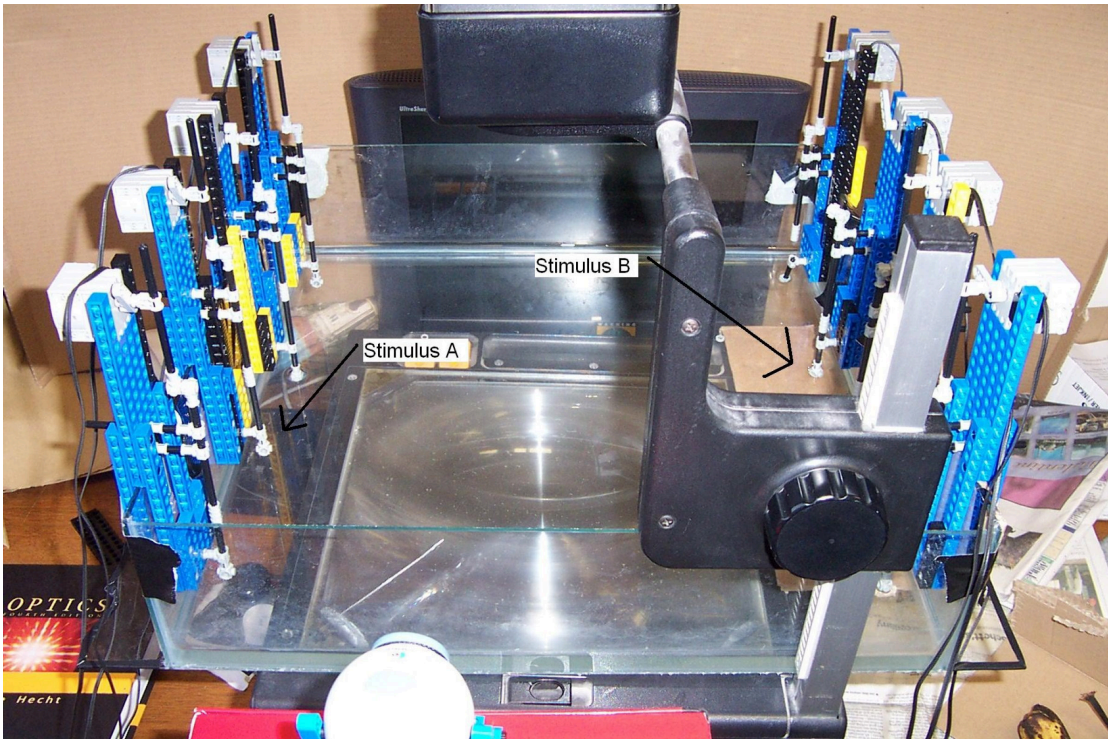


Figure 2.4: Water bucket reservoir from ”Pattern Recognition in a Bucket”[2]. The authors chose the fitting name ”The Liquid Brain”. As can be seen there are two stimuli, i.e., inputs, that lead to ripples on the surface, which are then used in the readout.

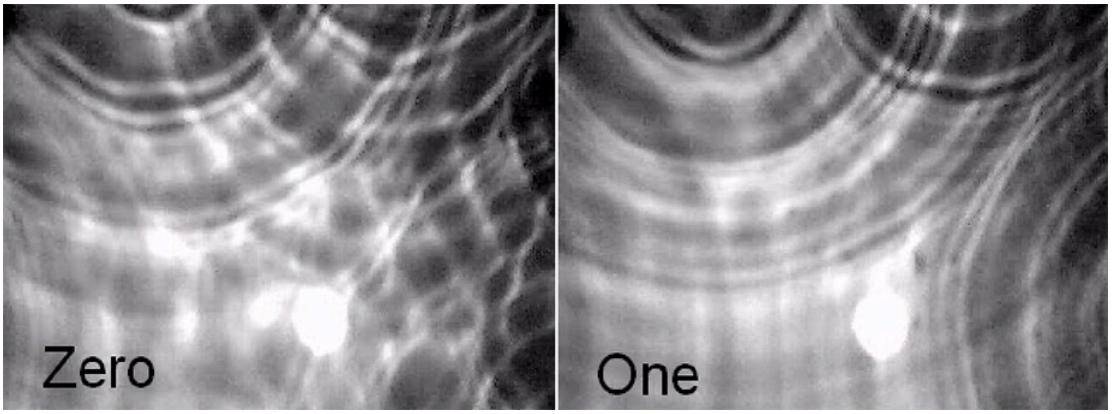


Figure 2.5: Ripples on the water surface from ”Pattern Recognition in a Bucket”[2]. On the left we see the response of the liquid brain to an audio recording of a person saying ”zero” and on the right the response to the word ”one”. The ripples were recorded with a camera and fed into a simple perceptron to classify what word was used as an input.

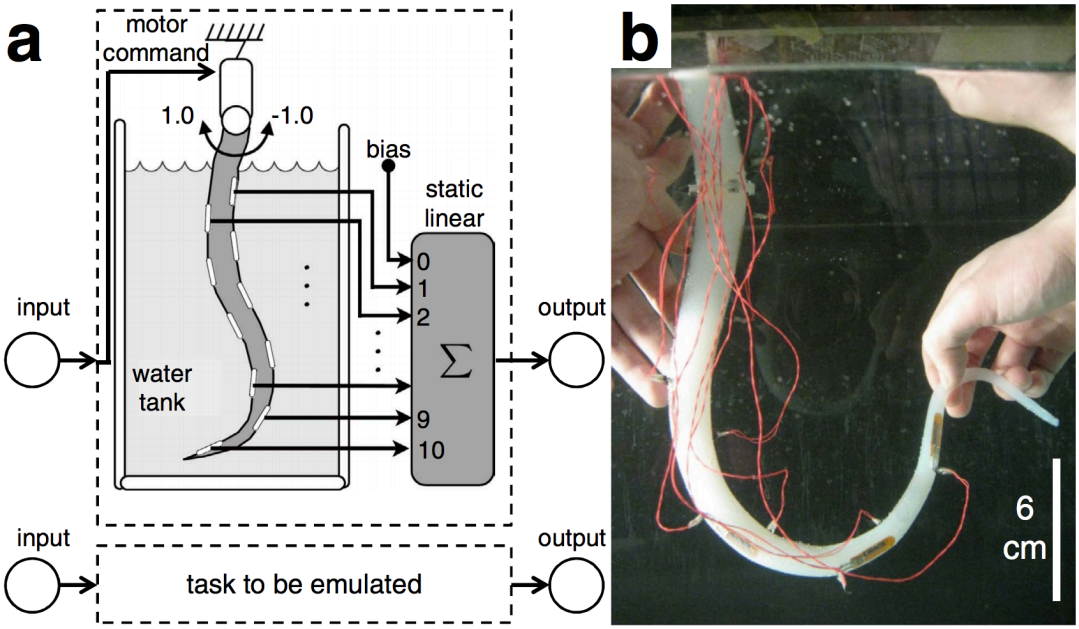


Figure 2.6: Tentacle setup from Nakajima et al.[3] licensed under CC BY 4.0. Sensors were embedded in the body of the tentacle and a single motor acted as an input. The sensors were fed to a single layer perceptron, i.e., a linear sum, to create the desired output, e.g., a NARMA time-series.

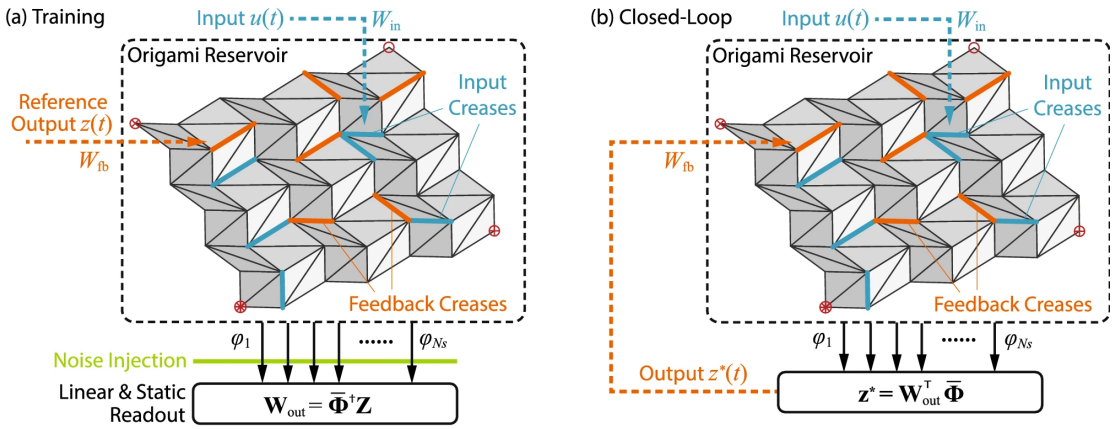


Figure 2.7: Origami setup from Bhovad and Li [4] licensed under CC BY 4.0. Some creases were selected as input, and actuated with an input moment, folding or unfolding the crease. The rest of the Origami then reacted, and the angles of the creases are measured as the reservoir state.

the crease. The whole structure then reacted and the crease angles of the folds could then be used as a reservoir state for further computation. In the case of a closed-loop setup, some creases were selected as additional feedback creases. They then also created an actual Origami reservoir, shown in Figure 2.8, made of think paper. Instead of actuating single folds, they opted for an excitation of the whole system, by placing the reservoir on a plate that could be actuated. To further break symmetries in the reservoir masses were added at random positions. After exciting the reservoir the vertical displacement of the coloured vertices could be recorded with a camera and used to create the desired output.

One very promising branch of physical reservoir computing is the use of light, so-called photonic reservoir computing [33, 34]. There are currently multiple types of reservoir, e.g., silicon photonics networks. Using optical waveguids, splitters, and combiners, photonic

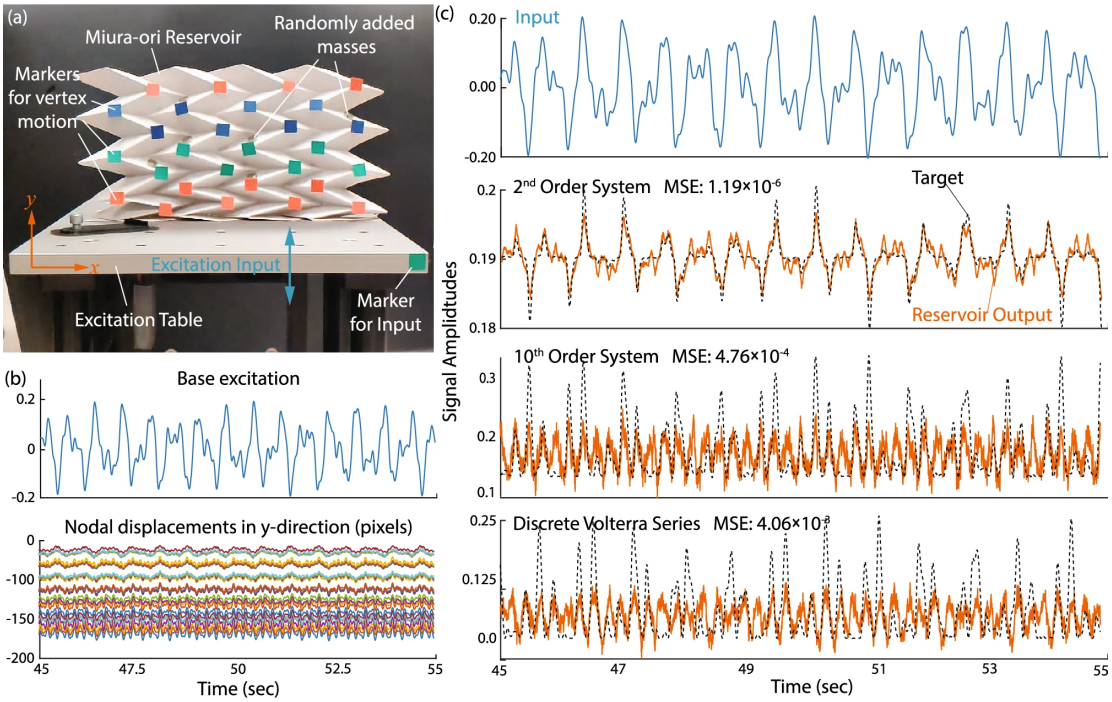


Figure 2.8: Origami reservoir proof of concept by Bhovad and Li [4] licensed under CC BY 4.0. The Origami symmetry was broken by adding additional masses at random positions. The reservoir was fixed to the actuator, i.e., the system was moved in the y-direction as input. Then the vertical displacement of the marked vertices was used as reservoir state.

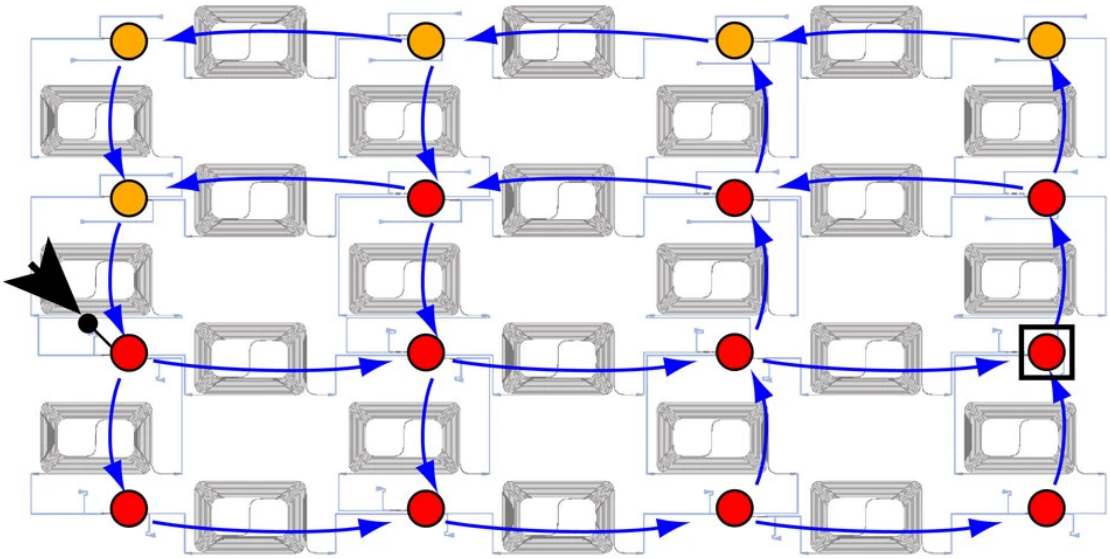


Figure 2.9: Passive 16-node reservoir by Vandoorne et al. [5] licensed under CC BY 4.0. Light as input is added at the input (black arrow). The light then propagates through the network following the blue arrows. The red dots were the nodes used as sensory measurement to measure the reservoir state.

networks can be created (see Figure 2.9)[5]. Vandoorne et al. created a small chip with passive elements through which the light would propagate when added at the input. Nonlinearities were then added at the readout stage, and the authors were successful in solving Boolean operations and digit recognition. A big advantage is the time scale of the network, as it mostly depends on the optical delay, which would allow one to perform Boolean operations at the speed of several 100s of $Gbit^{-1}$.

A final class of reservoirs I want to introduce is mass-spring networks [1, 35, 36, 37], as these systems can act as a general model for different types of materials, most notably compliant materials. Mass-springs systems are networks of masses interconnected with (non-linear) springs. We can exploit such networks for computation by encoding our input

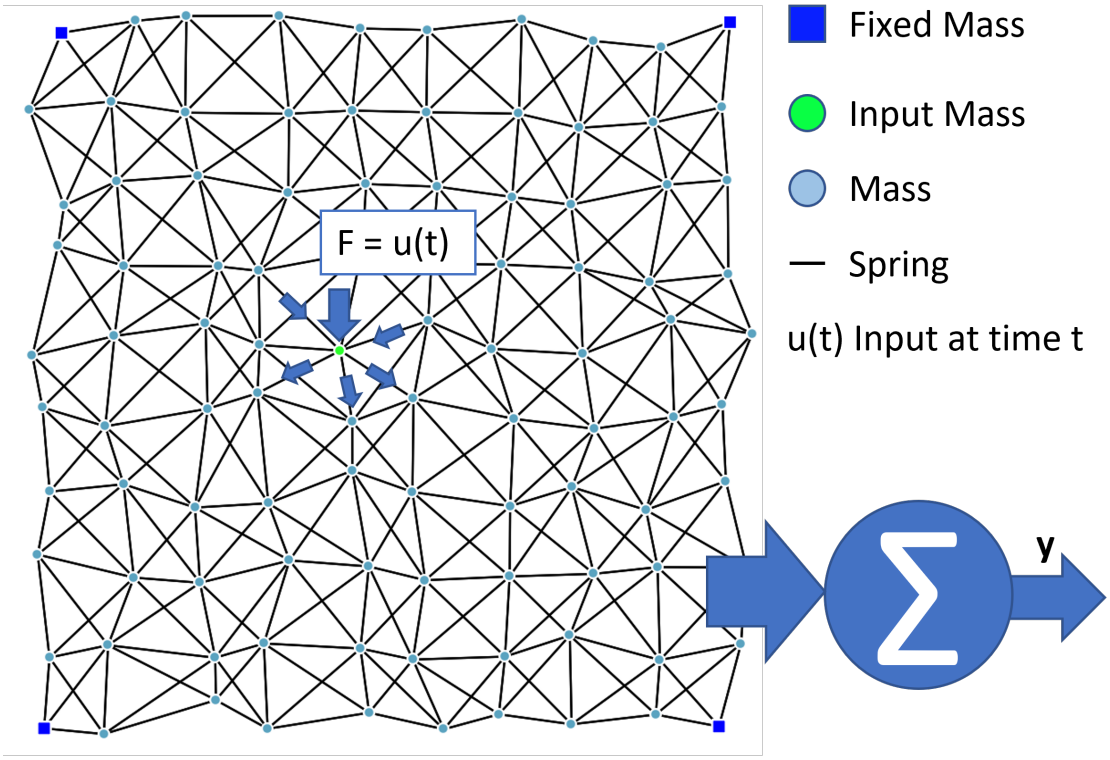


Figure 2.10: Schematic of a mass-spring network. When a force is applied to one of the masses, the springs transfers part of the force to neighbouring masses, which makes the network oscillate based on the input. Using sensors, e.g., elongation sensors along the springs, we can create a representation of the state of the network that can be used with a simple readout.

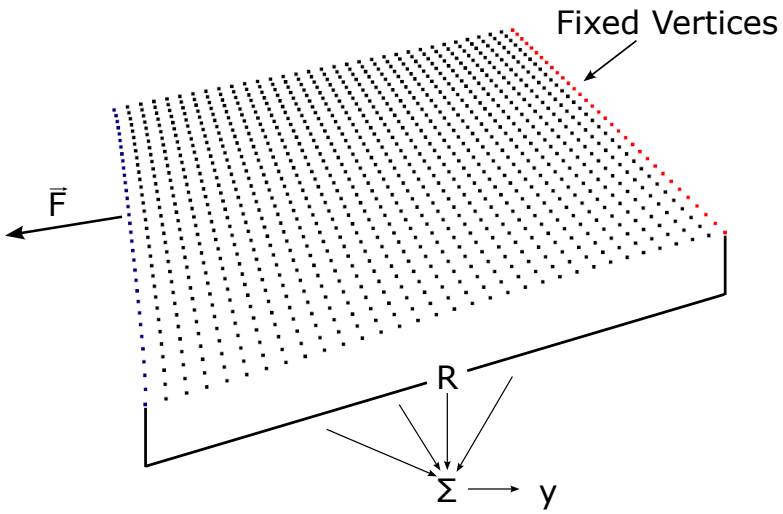


Figure 2.11: Illustration of how our proposed cloth-based reservoir works. The dots represent the vertices of the cloth inside the simulation. One side of the cloth is fixed (red), whereas on the opposite side, an input is applied in the form of force F pulling on the cloth. The cloth dynamics then change the resistance of the conductive threads, which can be measured and used as the sensor readout of the reservoir to generate the output y as a weighted sum. To further improve the performance, the mass of random vertices within the cloth is increased, as published in [6].

signal as a force vector and applying it to the network, either to a single mass or to the entire system; this is illustrated in Figure 2.10. The big advantage of this kind of computational model is that it is universal to some degree, i.e. applicable to a wide range of problems. For example, mass-spring networks are highly used in the numerical simulation of textiles [38, 39, 40], soft-body dynamics [41, 42] or even biological tissue [43]. Chapter 3 is heavily based on mass-spring networks.

Due to their relationship with fabrics in [6], I explored the applicability of conductive fabrics for nonlinear computation and examined their memory capabilities. By attaching probes to measure the resistance along two opposing edges of the cloth, we can build a sim-

ple reservoir computing system, by exploiting the change of resistance when the piece of cloth deforms due to applied forces. This setup is depicted in Figure 2.11. The nonlinear computation capabilities were explored using a NARMA20 timeseries, as seen in Equation ?? with inputs $u(t)$ based on a Mackey-Glass timeseries, shown in Equation 2.20. The system was trained to predict the NARMA20 timeseries $y(t)$ with $u(t)$ being the input force applied to the fabric.

$$y_t = tanh(0.3y_{t-1} + 0.05y_{t-1}(\sum_{i}^{20} y_{t-i}) + 1.5u_{t-20}u_t + 0.1) \tag{2.19}$$

$$u_t = 0.9u_{t-1} + \frac{0.2u_{t-24}}{0.8 + u_{t-24}^{0.9}} \tag{2.20}$$

To further improve the dynamics of the cloth, random vertices in its discresitazion had their weight increased. Figure shows the input and outputs of cloths with different size, and with uniform and nonuniform mass distribution. As can be seen, breaking symmetries by adding additional weight at random positions seems to be a key ingredient, to make the system work to its full potential.

I then further investigated the memory capacity (MC) of such a system, based on the squared correlation coefficient, as described in Equation 2.21. The MC is a measurement for how long information can be retained within the system, by driving the system with an

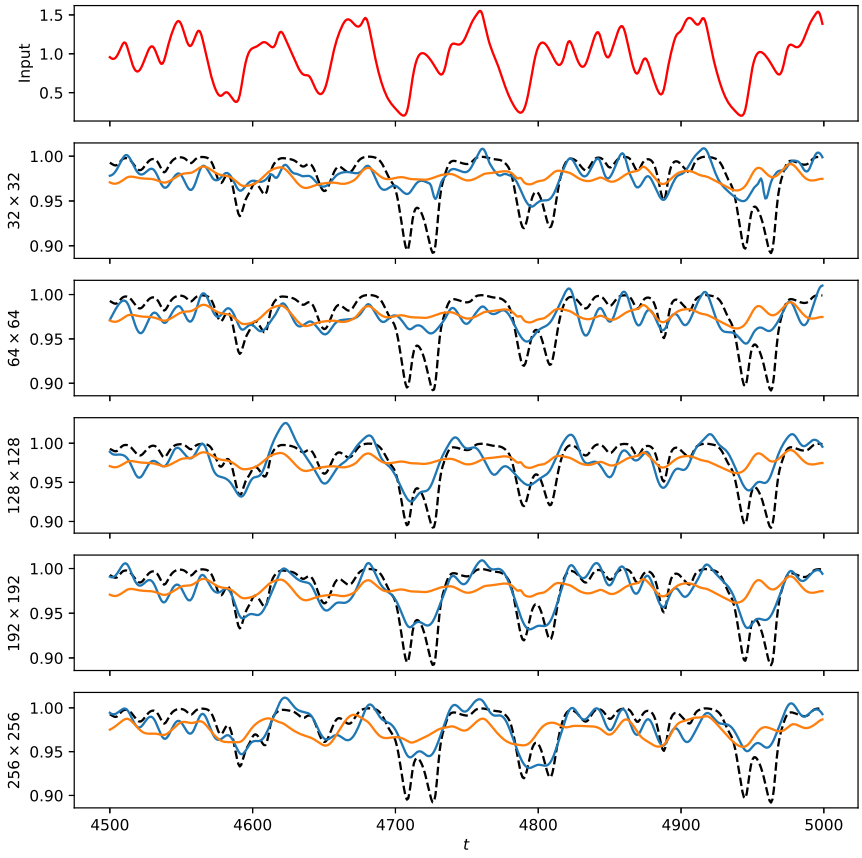


Figure 2.12: Input-output plots of the NARMA20 task for the last 500 time steps. The top plot shows the discrete Mackey-Glass time series used as input for the reservoirs. In the bottom plots for each cloth with vertices 32×32, 64×64, 128×128, 192×192, and 256×256, we have the average output of the system with non-uniform mass distribution (blue) and uniform mass distribution (orange), with the corresponding ground truth (dashed; black), as published in [6].

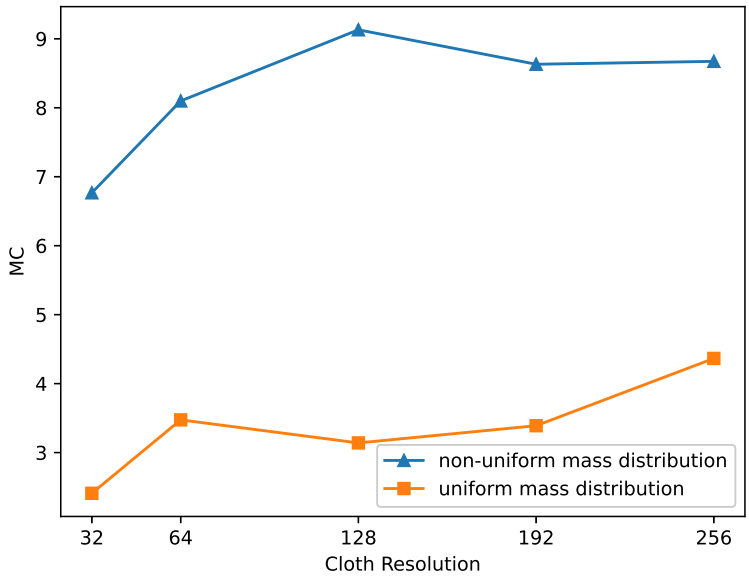


Figure 2.13: Average memory capacity with respect to the cloth resolutions used in the simulation based on 20 trials for each data point. The cloths with non-uniform mass distribution (blue; triangles) perform significantly better than the cloths with a uniform mass distribution (orange; rectangles), as published in [6].

input y and train the system to reproduce the input lagged by τ timesteps.

$$MC = \sum_{\tau=1}^{100} \frac{cov^2(\hat{y}, y_\tau)}{\sigma_{\hat{y}}^2 \sigma_{y_\tau}^2} \tag{2.21}$$

As can be seen in Figure 2.13, using a nonuniform mass distribution again improves the performance, and puts it in the range of echo state networks with roughly ten neurons.

3

Abstract Reservoir Computing

This chapter is based on the publication "Abstract Reservoir Computing"[7] by Senn and Kumazawa. Based on mass-spring networks as introduced in Chapter 2 we introduce a training regime that in theory allows the translation of weights learned from simulations to systems in the real world. In general, this is not always possible as there is always a difference between simulated worlds and the real world, also known as the Sim2Real gap. Using

ideas from abstract interpretation[13], we build a training framework capable of capturing differences in initial conditions, for example, created by changes in mass placements due to building tolerances; this is introduced in Section 3.1. Numerical simulations of different possible sources of errors support our findings, as shown in Sections 3.2 and 3.3. Some further extensions are then discussed in 3.4

3.1 Methods

When physically building mass-spring systems, as introduced in Chapter 2, we have to deal with tolerances due to imperfections in the creation process. Therefore, the initial position of the masses will divert from the positions assumed in the simulation, this is visualized in Figure 3.3. But depending on the application, numerical simulations are inevitable; think of systems that work in difficult-to-access environments, where a loss can be potentially hazardous or expensive, or where building is time and cost consuming. Although such numerical simulations have improved in fidelity, it is not possible to accurately represent all facets of physical systems in simulated environments. This in part leads to a gap between simulations and reality, also called the sim2real gap.

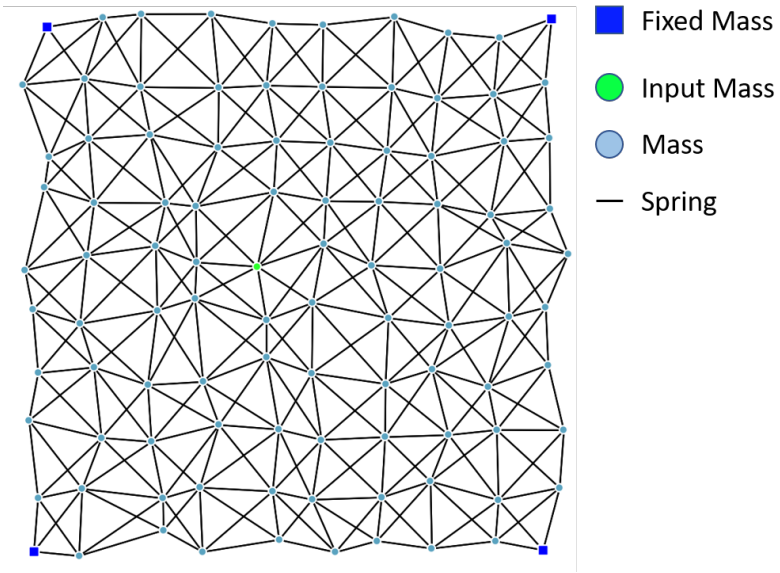


Figure 3.1: Visualisation of a mass-spring system as used in the experiments of [7]. The corner masses in dark blue are fixed, the mass in green acts as an input mass - receiving the input as a force - the masses in light blue react to the input and the (nonlinear) springs are represented as black lines.

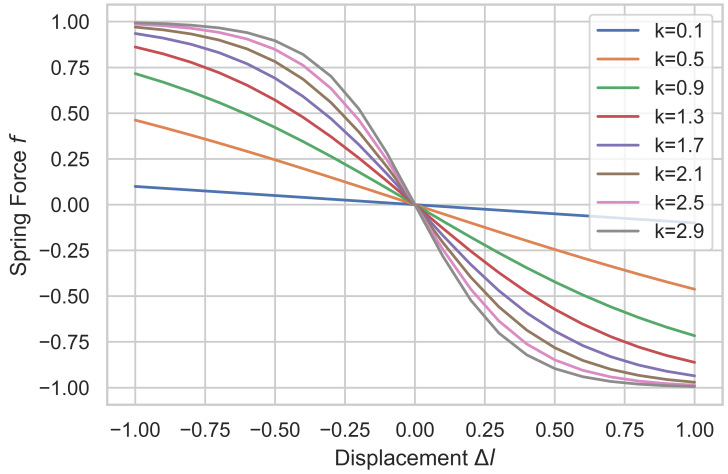


Figure 3.2: Depending on the spring constant k, the springs used in the simulation, from [7], exhibit different force-displacement behavior..

We can formalize this problem in the form of local robustness as in Equation 3.1 for a point $x_0 \in \mathbb{R}^d$, tolerances δ and δ_{target}, a distance function and some function f, e.g., a neural network or a physical system like a mass-spring network. In short, given two points within a distance of δ from each other, we want the output of the function f of both points to be within our target tolerance δ_{target}.

$$\forall x \in \mathbb{R}^d . |x - x_0| \leq \delta \Rightarrow |f(x) - f(x_0)| \leq \delta_{target},\ d \in \mathbb{N},\ \delta \in \mathbb{R}_+ \tag{3.1}$$

We can begin to address this issue by replacing each component p_i of the location vector p of a mass with an interval or ball of the form $(p_{i,center}, p_{i,radius})$, which represents the potential positions of the mass (red rectangle in Figure 3.3). This abstraction can then be used directly in the simulation using ball arithmetic [44, 45]. Instead of concrete numbers for our states x_t, we will have state tuples of the form $(x_{t,centre}, x_{t,radius})$, which we can collect in the matrices X_c and X_r, respectively. In [46] we proposed to use the additional information as constraints for the linear regression and to employ a splitting conic solver [47] to solve:

$$\begin{aligned} &\underset{w}{argmin} \|X_c w - y_c\| \\ &s.t.\ |w| X_r \leq y_r. \end{aligned} \tag{3.2}$$

This approach has the benefit of providing exact upper error bounds encoded in y_{radius},

which signify the maximum allowed divergence from the concrete solution given as y_{centre}. However, the use of a solver significantly reduces the speed of training. By removing the need for an upper error bound, we can reformulate Equation 3.2 as follows:

$$\underset{w}{argmin}\|X_c w - y_c\|^2 + \|X_r w\|^2, \tag{3.3}$$

Reformulating this as the cost function L allows us to derive a closed form solution as shown in Equations 3.4 and 3.5:

$$\begin{aligned} L &= (X_c w - y_c)^2 + (X_r w)^2 \\ \frac{dL}{dw} &= 2X_c^T X_c w - X_c^2 y_c + 2X_r^T X_r w \end{aligned} \tag{3.4}$$

, then setting the derivative $\frac{dL}{dw}$ equal to 0 we can solve for w:

$$\begin{aligned} 0 &= 2X_c^T X_c w - X_c^2 y_c + 2X_r^T X_r w \\ w &= (X_c^T X_c + X_r^T X_r)^{-1} X_c^T y \end{aligned} \tag{3.5}$$

By substituting Equation 3.5 for Equation 3.2, we can achieve a considerable increase in speed, albeit at the cost of sacrificing guarantees of error bounds.

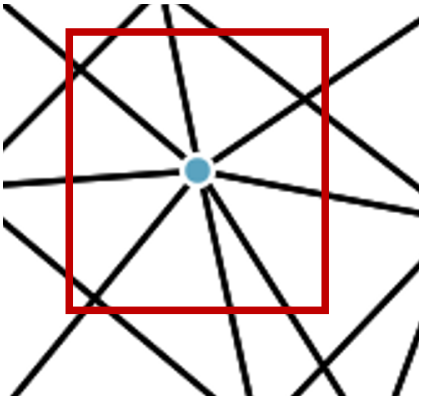


Figure 3.3: Visualisation from [7] of tolerances that can occur when physically building a mass-spring network . The light blue circle represents the exact location of the mass, while the red rectangle indicates the area of potential locations due to horizontal and vertical displacements. These displacements can change the dynamic behaviour of the network compared to its simulated version, making it hard to transfer learned parameters from the simulation to real-world systems.

3.2 Experimental Setup

We evaluated our proposed approach by running a numerical simulation of a mass-spring network using Julia 1.5 [48]. We tested it with three datasets in both open- and closed-loop setups for different types of error. The benchmark datasets were the same as those used by Goudarzi et al. [49]. We precomputed them for 5000 time steps and then repeated each point in each time series 5 times, resulting in a time series with 15′000 data points. We then split them into a training and testing set. The training set consisted of the first 10′000 time steps, while the remaining 5′000 time steps were used for the testing set.

3.2.1 Mass-Spring Network

We use a mass-spring network as depicted in Figure 3.1 and introduced in the previous Chapter 2. The masses are initially arranged in a regular grid and then slightly shifted with a value $\Delta p \in \mathcal{U}^2(-0.25, 0.25)$. Non-linear springs are then used to connect each mass to its 8-neighbourhood, as described in Equation 3.6. Figure 3.1 shows this setup. The four corner masses (blue) are fixed, whereas the input signal is applied as a force to a single input mass (green). The acceleration of each mass in the network is used as a sensor reading.

For the connections between the masses we use nonlinear springs exhibiting a spring force of the following shown in Equation 3.6

$$f(\Delta l) = tanh(-k\Delta l), \tag{3.6}$$

Δl being the spring displacement and k the spring constant. This emulates a compliant, elastic material with a force-displacement curve as shown in Figure 3.2.

To exploit such a system for computation, the input signal u is translated to a force f and applied to predetermined input masses (green in Figure 3.1). The network then starts to oscillate accordingly, and we can record the mass accelerations as the state x and use it to calculate the readout weights, as introduced in Chapter 2.

3.2.2 Hénon Time-Series

The Hénon time-series is based on the Hénon map introduced in 1976 [50]. Equation 3.7 was used to compute the time series.

$$y_t = 1 - 1.4y_{t-1}^2 + 0.3y_{t-2} + \mathcal{N}(0, 0.001) \tag{3.7}$$

The Hénon time-series as used in the experiments is shown in Figure 3.6.

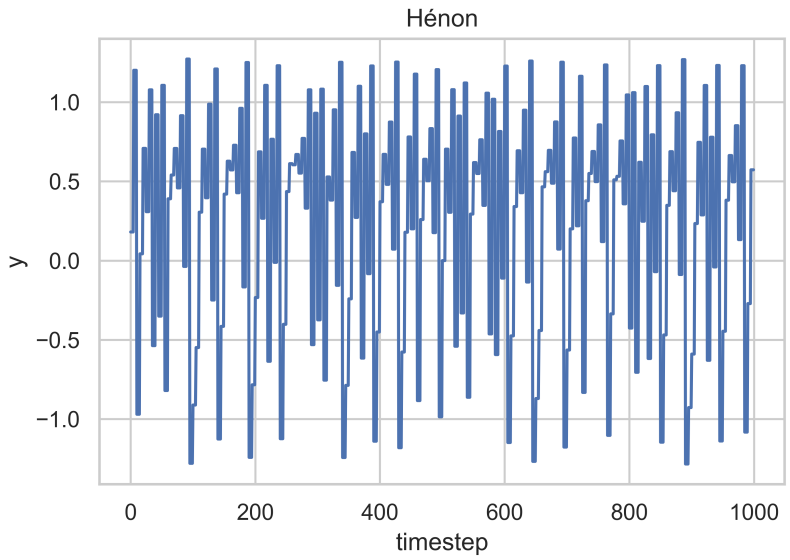


Figure 3.4: Hénon time-series as used for the experiments from [7].

3.2.3 NARMA10 Time-Series

NARMA10 is a widely used benchmark in the reservoir computing community. It is a non-linear autoregressive moving average task and is expressed as follows:

$$y_t = 0.3y_{t-1} + 0.05y_{t-1}\sum_{i=1}^{10} y_{t-i} + 1.5u_{t-10}u_{t-1} + 0.1 \tag{3.8}$$

with $u_t \in \mathcal{U}(0, 0.5)$ being drawn from a uniform distribution.

The NARMA10 time series used in the experiments is shown in Figure 3.5.

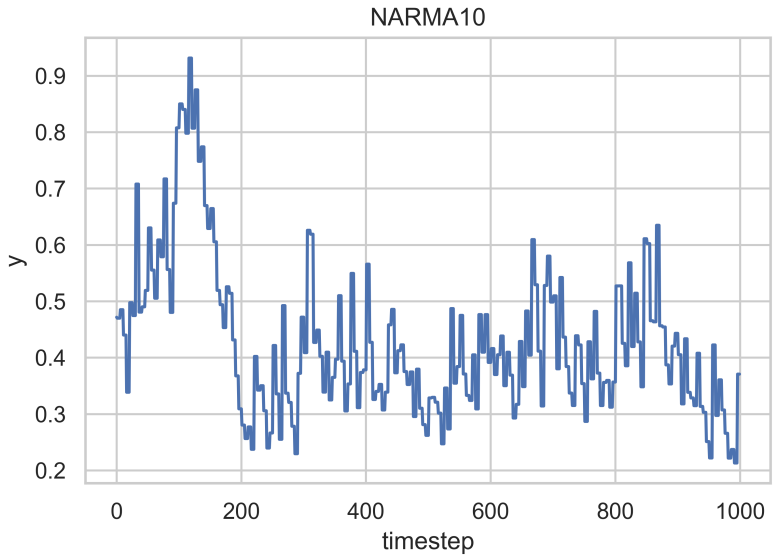


Figure 3.5: NARMA10 time-series as used for the experiments from [7].

3.2.4 NARMA20 Time-Series

The NARMA20 task is similar to the NARMA10, but with a longer time dependence and an extra nonlinearity.

$$y_t = tanh(0.3y_{t-1} + 0.05y_{t-1}\sum_{i=1}^{20} y_{t-i} + 1.5u_{t-20}u_{t-1} + 0.1) \tag{3.9}$$

The NARMA20 time series used in the experiments is shown in Figure 3.6.

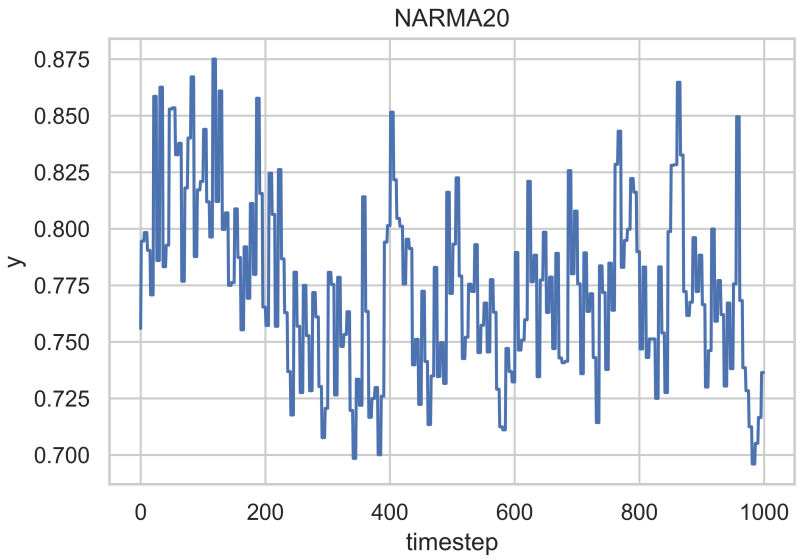


Figure 3.6: NARMA20 time-series as used for the experiments from [7].

3.2.5 Baselines

We compared the results of our proposed approach to the following two baselines:

- Training with ridge regression (classical model)
- Training with linear regression and added noise (noise model)

The sensor readings from the training simulation, $x_{\tau+1}, .., x_T$, are used to train the classical model with Equation 3.10 and the noise model is trained with Equation 3.11 with ε set to the magnitude of the current error parameter.

$$X = \begin{pmatrix} x_{\tau+1} \\ x_{\tau+2} \\ \vdots \\ x_T \end{pmatrix}$$

$$w = (X^T X + 0.001I)^{-1} X^T y \tag{3.10}$$

$$X = \begin{pmatrix} x_{\tau+1} \\ x_{\tau+2} \\ \vdots \\ x_T \end{pmatrix}$$

$$w = (X + \mathcal{U}(-\varepsilon, \varepsilon))^{\dagger} y \tag{3.11}$$

3.2.6 Sensor Augmentations

We examined each model under various types of errors that could occur in a real world situation.

- Sensor Malfunction
  - Before testing, a certain number of masses were randomly chosen with a given probability p and their readings were set to 0 during the testing.
- Sensor Noise
  - Gaussian noise with a mean of 0 and a varying standard deviation σ was added during the testing.
- Fixed Sensor Displacement

  - Sensor readings were shifted by a fixed value z.

- Mass Position Displacement

  - The mass positions were randomly shifted by a random vector $\Delta x \in [-k; k]^2$.

Each potential source of error was tested separately from the other sources. The parameter ranges are listed in Table 3.1.

Table 3.1: The parameter ranges used for each type of simulated error from [7].

Augmentation	Parameter	Range
Sensor Failure	p	$[0, 0.1]$
Sensor Noise	σ	$[0, 0.1]$
Fixed Sensor Displacement	z	$[0, 0.1]$
Mass Position Displacement	k	$[0, 0.1]$

3.3 Results and Discussion

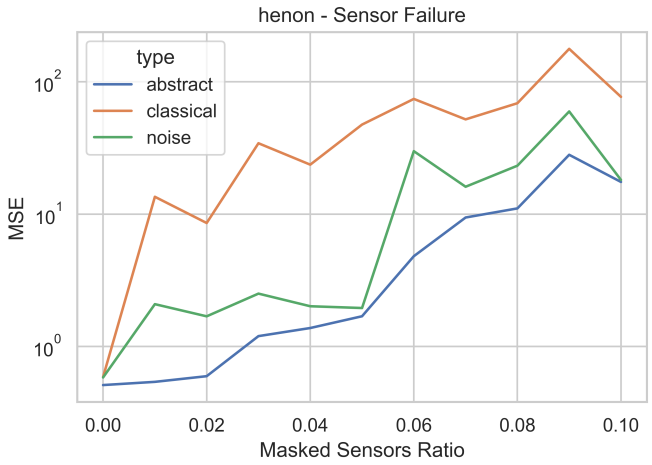
We calculated the mean squared error (MSE) between the generated outputs $\hat{y}$ by the system and the values y in the test sets for each experiment. Figures 3.7 - 3.10 show that our proposed approach outperforms the classical approach in terms of MSE in all experiments. At high noise levels, it either surpasses or performs equally to our proposed training regime

compared to the training regime with noise. Detailed numbers can be found in the Appendix in Tables A.1 - A.4.

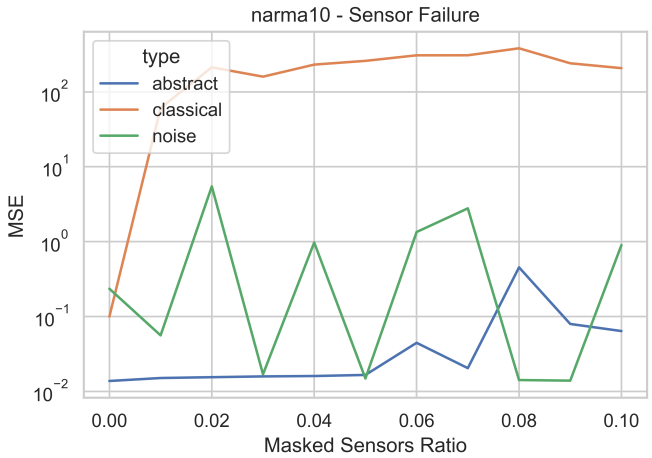
$$MSE = \frac{1}{N}\sum_{i}^{N}(y_i - \hat{y}_i)^2 \tag{3.12}$$

When looking at the results when no noise is present in the input, our proposed surpasses both baselines in any case. This can indicate that the abstract regulariser is generally the better choice than $L2$-regularisation. Considering the results of the experiments with noise present, our proposed approach gives more consistent results over the whole spectrum of noise amplitudes, except for sensor failures, i.e., missing sensor readings. In this case, both baselines also perform poorly.

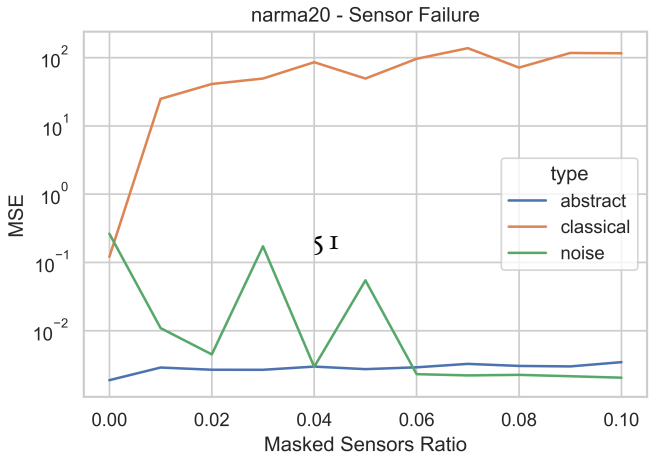
One reason for the better performance, even without noise present, is the fact that ball arithmetic [44] as used in the experiments also captures numerical imprecisions. This can be compared to adding noise to the system, and thus help against overfitting. Looking at the standard deviations of the results (Tables A.1 - A.4), we see that the abstract training regime leads to fewer variations on the output, indicating that our approach is more robust against randomness.



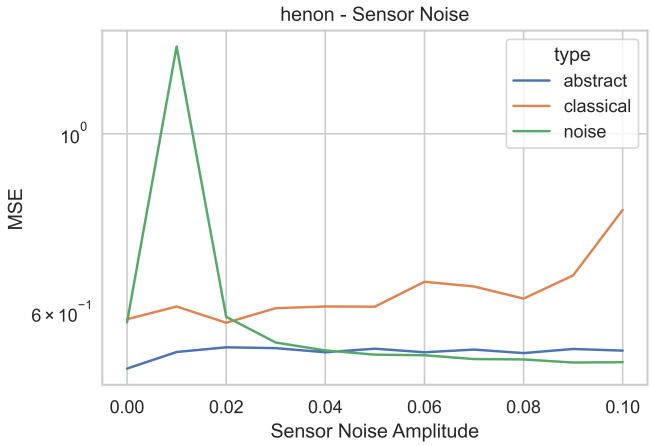
(a) MSE for simulated sensor failures, i.e., the sensor readings were masked, with probability given by the x-axis for the Hénon dataset from [7].



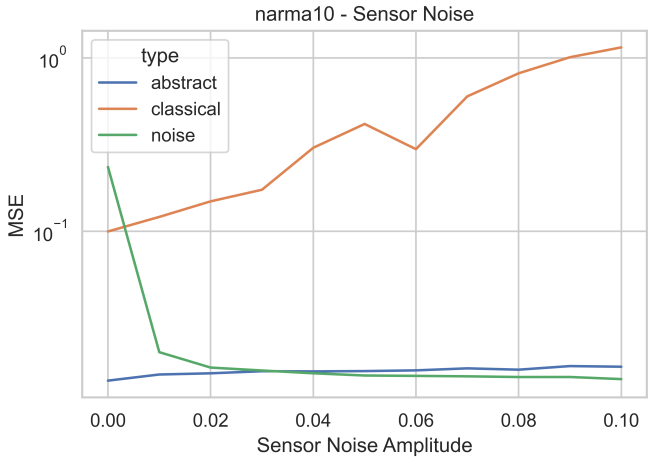
(b) MSE for simulated sensor failures, i.e., the sensor readings were masked, with probability given by the x-axis for the NARMA10 dataset from [7].



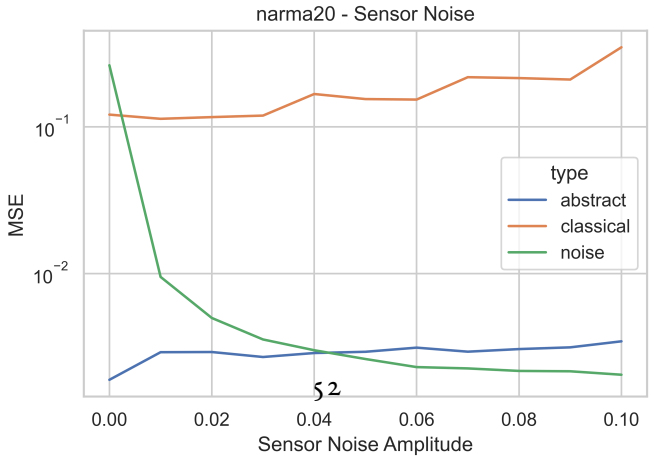
(c) MSE for simulated sensor failures, i.e., the sensor readings were masked, with probability given by the x-axis for the NARMA20 dataset.



(a) MSE for simulated sensor noise with the standard deviation given by the x-axis for the Hénon dataset from [7].

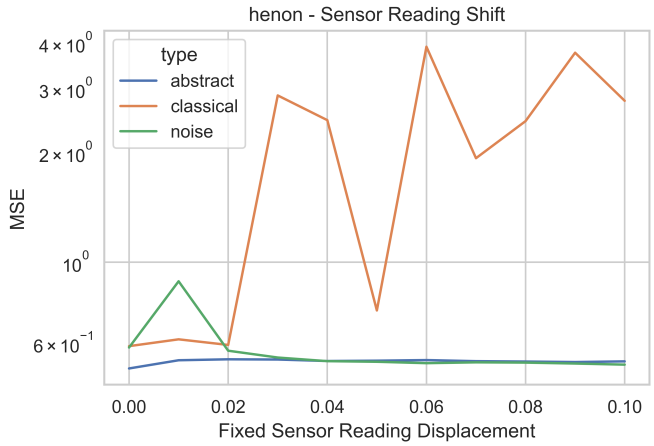


(b) MSE for simulated sensor noise with the standard deviation given by the x-axis for the NARMA10 dataset from [7].

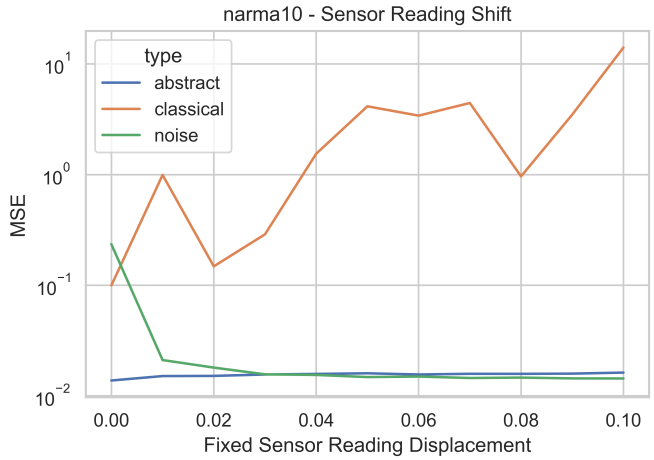


(c) MSE for simulated sensor noise with the standard deviation given by the x-axis for the NARMA20 dataset from [7].

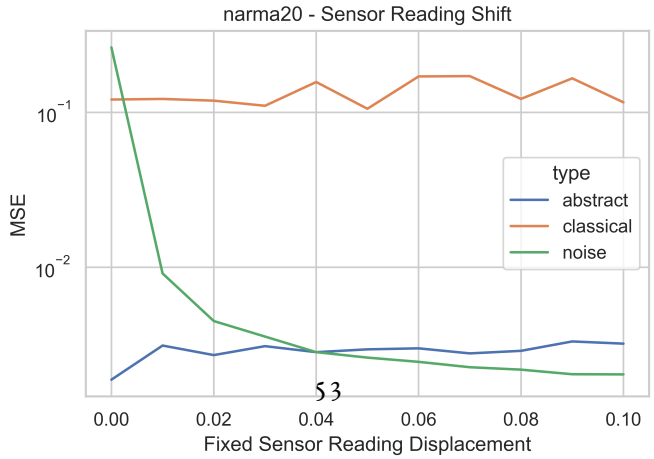
Figure 3.8: The average MSE with standard deviations for simulated sensor noise for the datasets (**a**) Hénon, (**b**) NARMA10 and (**c**) NARMA20 from [7]. Gaussian noise of the given standard deviation with mean 0 was added to the mass sensor readings.



(a) MSE for fixed simulated sensor perturbation with the amplitude given by the x-axis for the Hénon dataset from [7].

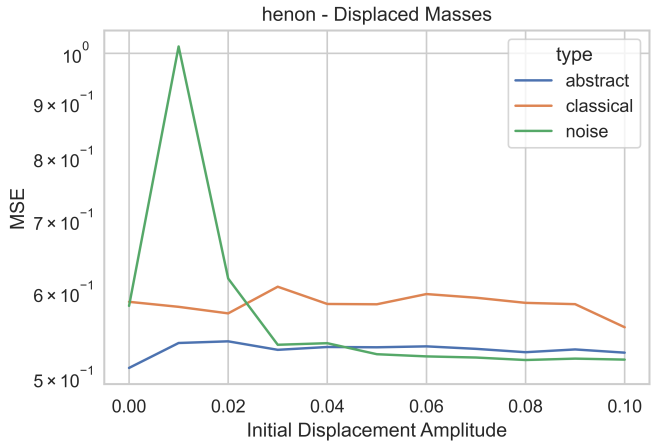


(b) MSE for fixed simulated sensor perturbation with the value given by the x-axis for the NARMA10 dataset from [7].

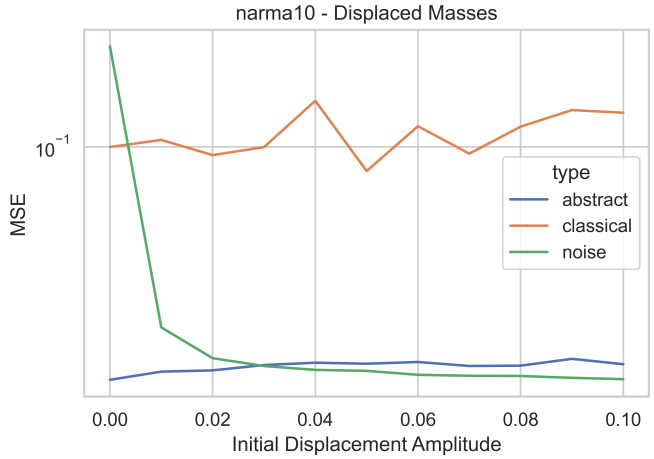


(c) MSE for fixed simulated sensor perturbation with the value given by the x-axis for the NARMA20 dataset from [7].

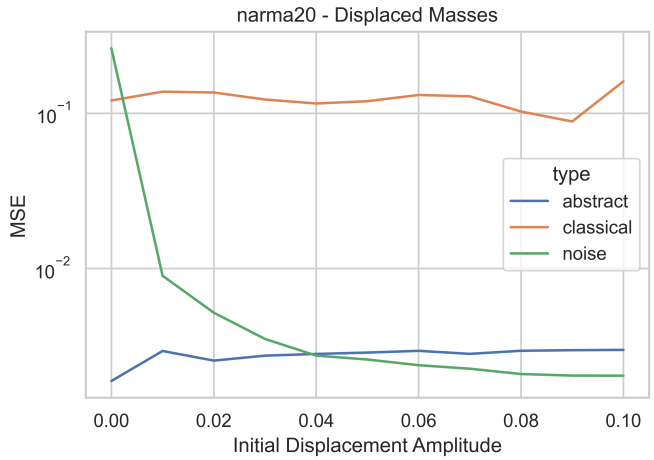
Figure 3.9: The average MSE for fixed simulated sensor reading displacements for the datasets (**a**) Hénon, (**b**) NARMA10 and (**c**) NARMA20 from [7]. The values read from the mass sensor readings were shifted by the given value.



(a) MSE for simulated initial mass displacement with the amplitude given by the x-axis for the Hénon dataset from [7].



(b) MSE for simulated initial mass displacement with the amplitude given by the x-axis for the NARMA10 dataset from [7].



(c) MSE for simulated initial mass displacement with the amplitude given by the x-axis for the NARMA20 dataset from [7].

Figure 3.10: The average MSE for simulated initial mass displacements for the datasets (**a**) Hénon, (**b**) NARMA10 and (**c**) NARMA20 from [7]. The initial position of the masses during test was shifted by uniformly sampled noise based on the amplitude given in the x-axis.

3.4 Conclusion and Outlook

Although physical reservoir computing systems are becoming increasingly important, there has been limited research into transferring trained systems from simulation to the real world. We proposed a new training regime based on abstract interpretation to address potential issues such as building tolerances, sensor defects, and noise. We verified our approach in a series of simulated experiments, which showed that our abstract regularizer achieved lower error rates than the classical approach with $L2$ regularization. This is likely due to the interval arithmetic used in our experiments. When noise was added, the difference between the two approaches became even more pronounced. Interestingly, the training regime with added noise became better the more noise was added. Therefore, in settings with high noise amplitudes, this approach appears to be the better choice.

For future research, we can compare a physical reservoir computing system in simulations and in real life, such as those based on textiles, which are similar to the mass-spring systems used in experiments. Additionally, we can explore the type of noise captured by the abstraction. Currently, we only consider uniformly distributed abstractions, but many problems in nature are normally distributed. Having the ability to use different distributions for the abstractions would give us the ability to customize the training regime to the specific problem. Both of these directions will make physical reservoir computing more accessible, as it will enable rapid iterations in simulations and a direct transfer of results to the

real world.

4

SwarmESN

This chapter is based on the publication "SwarmESN for Robust Distributed Reservoir Computing"[8]. Using echo state networks, introduced in Chapter 2, we introduce a decentralized system that is capable of distributed inference, that is, each echo state network node is capable of inferring information about the global state based on its local information and information it received from other nodes. We show the applicability of this ap-

proach using numerical experiments.

Based on mass-spring networks as introduced in Chapter 2 we introduce a training regime that, in theory, allows the translation of weights learned from simulations to systems in the real world. In general, this is not always possible as there is always a difference between simulated worlds and the real world, also known as the Sim2Real gap. Using ideas from abstract interpretation[13], we build a training framework capable of capturing differences in initial conditions, for example, created by changes in mass placements due to building tolerances; this is introduced in Section 3.1. Numerical simulations of different possible sources of errors support our findings, as shown in Sections 3.2 and 3.3. Some further extensions are then discussed in 3.4

4.1 Introduction

Neural networks have found their way into many facilities in various forms. Whether predictive maintenance [51, 52], intrusion detection [53, 54], quality control [55], fault detection [56]. Being able to analyze nonlinear correlations and/or get a peak of the future, based on numerous sensor readings, is an integral part of modern facilities [57]. Typically, these sensor readings are transferred to a central system where they are processed. As the number of sensors and therefore the amount of data they generate can get big, particularly considering the Internet of Things or specialized research facilities such as accelerators, it is not al-

ways feasible to first move the data to a centralized storage to be processed. Especially when real-time constraints are to be considered. Thus, efforts are being made to move this computation closer to where the data is generated [58], helping not only to reduce bandwidth and latency but also to improve resilience, since the computation is spread over multiple devices instead of a single point of failure.

Handling such distributed decision-making can still be challenging, with many proposed methods going in different directions. Moustafa, et al. [59] used an ensemble of Gaussian mixture models with a correntropy technique to detect cyberattacks in edge networks. Li, et al. [60] partitioned deep neural networks and placed the partitions on different devices for computation. This allows for an efficient use of computational resources but also introduces dependencies between the partitions, limiting the parallelism of the whole process. Stahl, et al. [61] showed that model partitioning and layer fusion can be used to run even demanding models such as YOLOv2 [62] can be run on edge devices with little memory if distributed accordingly.

One big issue with distributed systems is the uncertainty of the medium used to transport information, e.g., packet loss of $1\% - 2\%$ is still considered acceptable with streaming on the Internet, but depending on the environment used medium and protocol, the number can be bigger or smaller. As such, a distributed system should also be able to work under various levels of packet loss if used on edge devices.

A common technique to counter packet loss is retransmission. This is implemented in the TCP/IP protocol [63], but distributed learning approaches also implement this approach. For example, Sapio et al. [64] used retransmission to ensure that all gradient updates are aggregated and propagated correctly. This makes sense for training, as offline training is the norm, as we usually don't have any time constraints on how long one parameter update step can take, and being complete is more important. For inference, on the other hand, e.g., when trying to detect anomalies that require the shutdown of a facility, we might have strong real-time constraints, and cannot rely on the packet retransmission as this would introduce additional delays.

In addition, other possible causes for errors are misconfigurations, e.g., a sensor connected to the wrong inlet, or changes in the environment, e.g., sensors are moved around and plugged into different devices. In such cases, it is beneficial to have a system that is invariant to such changes.

Regarding cheap and robust computation, reservoir computing [65, 66] has gained more attention in recent years. By combining a dynamical system, such as a recurrent neural network [9, 10, 11], with a simple readout layer, we can get a universal approximator [67, 68]. Since the dynamical system does not have to be a neural network, we can implement such a system relatively cheaply in an FPGA or even use a physical system [65, 66, 69]. Furthermore, when properly trained, such systems can encode limit cycles and are thus inherently

resistant to perturbations [70].

Our proposed architecture is able to create local predictions with partial information about the global state, e.g., in case of packet loss, hardware failure, misconfiguration, etc. Although still lightweight enough to work on hardware with limited capabilities due to the use of the reservoir computing scheme [9, 49]. In addition, we show that the addition of a self-attention layer as the readout layer greatly improves the predictive capability, even in environments with high packet loss.

4.2 Architectures

We conducted an experiment to compare four different architectures in a distributed setting. We began with classical echo state networks [9] introduced in Chapter 2 that did not share any internal states, yet still attempted to forecast the global state, i.e., nonlocal outputs. We then tested echo state networks with time-delayed shared states but a classical readout layer. We followed this with a version that had a self-attention readout layer. Lastly, we used a feed-forward network as a baseline to create the output.

4.2.1 SwarmESN with linear readout

Our architecture is based on echo state networks, with the addition of internal states shared between each reservoir, as depicted in Figure 4.1 with a linear sum as readout. This concept allows for local information to be incorporated into the reservoir states and blended with

the states of other reservoirs. Consequently, each reservoir will gain knowledge about the global state over time.

We propose two distinct methods for combining global information. The first is to include external states in the reservoir state vector $\vec{x}$, as seen in equation (4.1), which we refer to as the 'single-vector' approach. The second is to keep track of it in a separate reservoir $\vec{z}$, as seen in equation (4.2), which we call the 'stacked-vector' variant, since the readout combines the local and 'global' vector.

We compute the global information vector $\vec{\lambda}$ in both cases, lagging behind by τ timesteps. For the 'single-vector' variant, we mix $\vec{\lambda}$ directly with the reservoir state $\vec{x}$ as expressed in (4.1) using the randomly initialized matrix W_2. Alternatively, we mix $\vec{\lambda}$ with the existing state $\vec{z}$ as in (4.2).

$$\begin{aligned} \vec{x}_{t+1}^{(i)} &= \alpha \vec{x}_t^{(i)} + (1-\alpha) tanh(W_{input} \vec{u}_t^{(i)} + W \vec{x}_t^{(i)} + W_2 \vec{\lambda}_{t-\tau}) \\ \vec{\lambda}_{t-\tau} &= vec(\{\vec{x}_{t-\tau}^{(j)}, j \neq i\}) \end{aligned} \tag{4.1}$$

$$\vec{x}_{t+1}^{(i)} = \alpha\vec{x}_t^{(i)} + (1-\alpha)tanh(W_{input}\vec{u}_t^{(i)} + W\vec{x}_t^{(i)})$$

$$\vec{z}_{t+1}^{(i)} = tanh(W\vec{z}_t^{(i)} + W_2\vec{\lambda}_{t-\tau})$$

$$\vec{\lambda}_{t-\tau} = vec(\{\vec{x}_{t-\tau}^{(j)}, j \neq i\}) \tag{4.2}$$

For the sake of simplicity, we use the same matrices W_{input}, W, and W_2 for all Echo State Networks (ESNs) and train them in the same way as the ESN variant as shown in equation (2.12). To generate a prediction, $\hat{y}$, we then employ equation (2.14) with one set of output weights W_{output} and use the same set for each ESN to make the system more robust against potential permutations.

4.2.2 SwarmESN with Self-Attention

This architecture is based on the SwarmESN architecture, as shown in Figure 4.1, with modifications to the readout layer. We replaced the linear sum with the self-attention mechanism [71] in (4.3). This mechanism consists of K, Q and V NxN-matrices, W a NxM-matrix, N the length of the reservoir state vector $\vec{x}$, M the dimensionality of the output, ϕ the softmax activation function, *relu* the ReLU activation and $\vec{b}$ a bias vector.

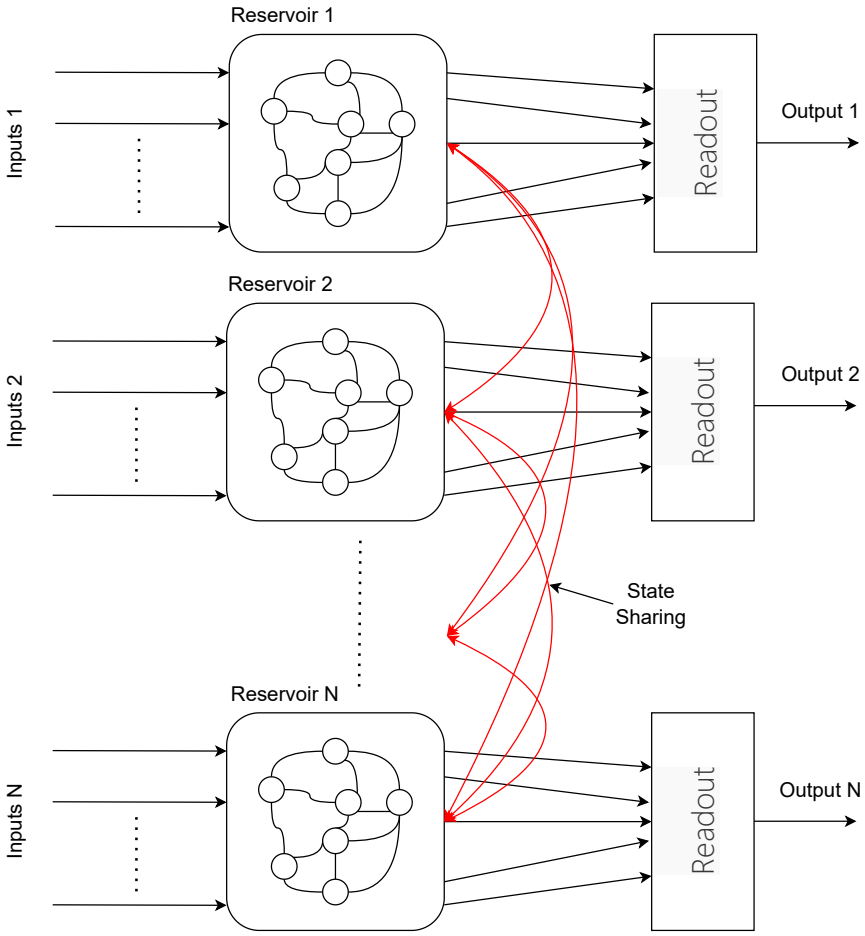


Figure 4.1: Schematic depiction of the SwarmESN architecture from [8]. The readout is either a linear sum, a self-attention layer, or a multilayer feed-forward network.

$$K_{\vec{x}} = K\vec{x}$$

$$Q_{\vec{x}} = Q\vec{x}$$

$$V_{\vec{x}} = V\vec{x}$$

$$\vec{y} = relu(W\phi(K_{\vec{x}}Q_{\vec{x}})V_{\vec{x}} + \vec{b}) \tag{4.3}$$

The purpose of introducing a self-attention mechanism is to enable the system to adapt itself to the information available, such as when there is noise or missing inputs.

We train the the parameters K, Q, and V with gradient descent.

4.2.3 SwarmESN with feed-forward neural network

A basic feed-forward neural network is employed as the readout in this variation, as shown in Figure 4.1.

We train the readout neural network using gradient descent.

4.3 Experimental Setup

4.3.1 WESAD

The WESAD dataset is a collection of data gathered from 15 individuals using a device that is worn on the wrist and chest. It includes acceleration on three axes, body temperature, respiration, electrodermal activity, electromyogram, electrocardiogram and the blood volume pulse [72]. An example of the time series of the dataset is shown in Figure 4.2. The 15 subjects were divided into three datasets: 10 for training, two for validation, and three for testing.

The objective of the experiment was to compare the performance of echo state networks with linear readout and with self-attention to predict the next time step.

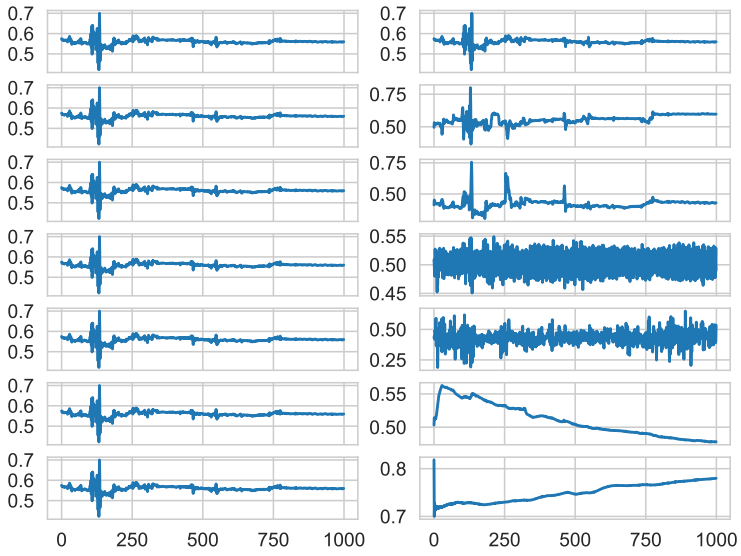


Figure 4.2: Example time-series of the WESAD dataset we used in our experiments in [8] after normalization from [8].

ρ	0.9
c	0.1
α	0.2
N	100

Table 4.1: The hyperparameters of the reservoir used in our experiments. The reservoir parameters were selected using a grid search with a standard ESN on the training dataset.

The hyperparameters chosen are presented in Table 4.1. The traditional ESN was trained with ridge regression, while the augmented ESN was trained with stochastic gradient descent and mean square loss until the model achieved convergence.

4.3.2 Resilience

For training, we set $\tau = 3$ in equations (4.1) and (4.2), meaning that each reservoir was given information from other reservoirs that was delayed by three time steps. This was done to simulate a network delay. During testing, the received states were randomly blocked with a probability of $p_{masked} \in [0, 1]$ to evaluate how the system responds to packet loss.

We tested the resilience of the input system to normal noise by adding a noise vector $X \sim \mathcal{N}(0, \sigma^2)$ with $\sigma^2 \in [0, 1]$ to the inputs, and repeating this process 10 times. To measure the effect of our distributed architectures, we also tested them without any state sharing as an ablation study.

4.4 Results and Discussion

We evaluate the models by calculating the mean squared error, denoted *MSE*, in (4.4) and the mean absolute error, denoted *MAE* in (4.4).

$$MSE := \frac{1}{N}\sum_{n=1}^{N}(\hat{x} - x)^2 \tag{4.4}$$

$$MAE := \frac{1}{N}\sum_{n=1}^{N}|\hat{x} - x| \tag{4.5}$$

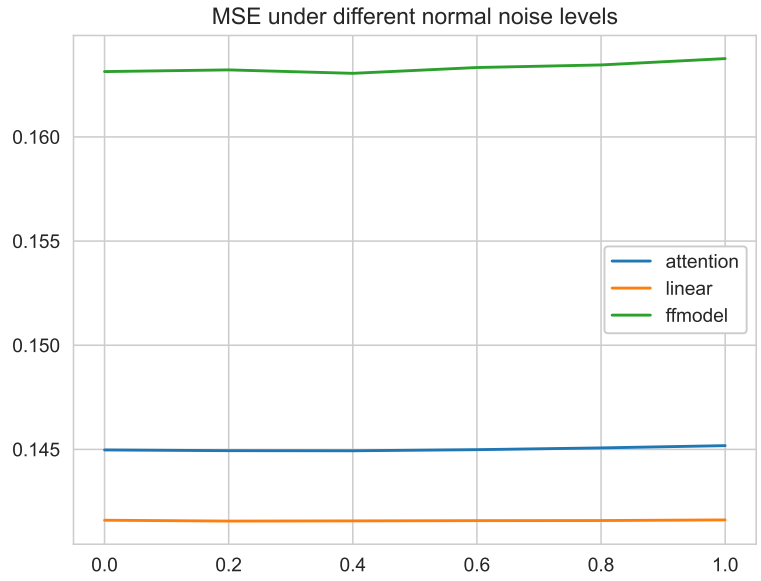


Figure 4.3: MSE of the architectures using the single-vector representation under noise from [8].

By comparing the mean squared error and mean absolute error after introducing normal noise to the input with various standard deviations $\in [0, 1]$, a clear difference is visible in Figures 4.3 and 4.4 and Table 4.2. The single-vector representation is more resilient to noise on the input side, with the traditional linear readout demonstrating the best results.

Examining packet loss yields a similar outcome. The single-vector representation outperforms the stacked-vector representation in terms of MSE and MAE when the loss probability $p_{masking}$ ranges from 0 to 1. The linear readout has the highest overall performance, as shown in Figures 4.5 and 4.6 and Table 4.3.

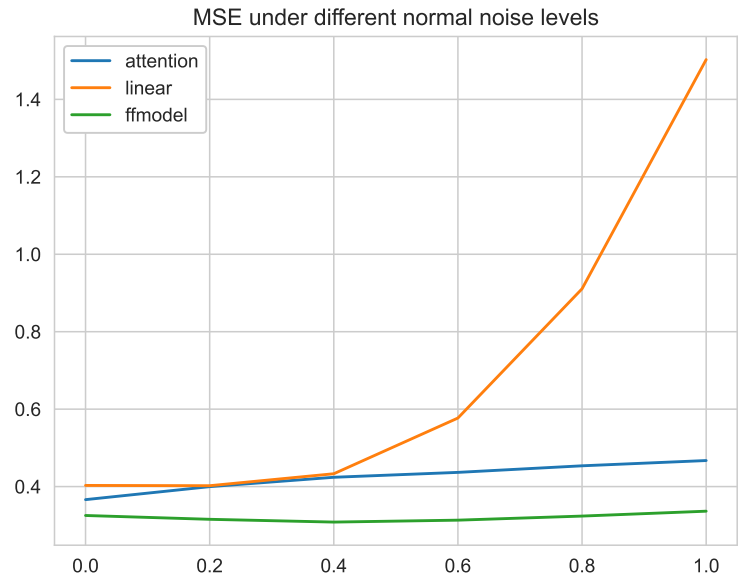


Figure 4.4: MSE of the architectures using the stacked-vector representation under noise from [8].

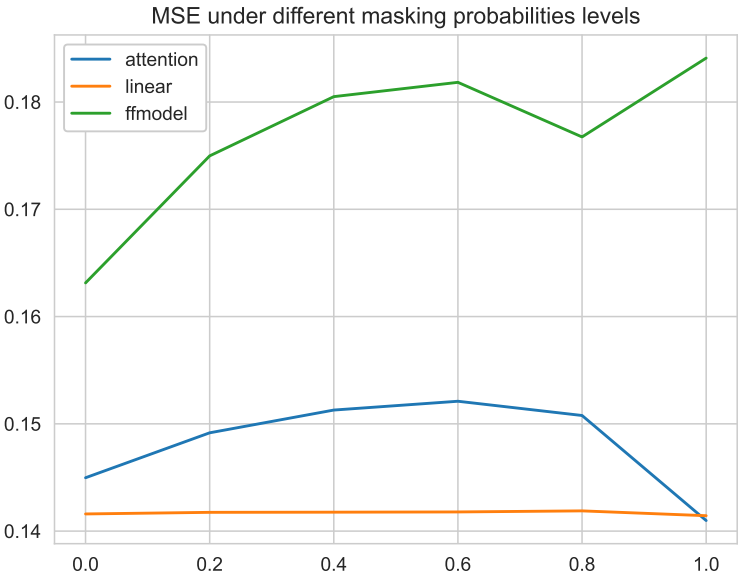


Figure 4.5: MSE of the architectures using the single-vector representation under packet loss from [8].

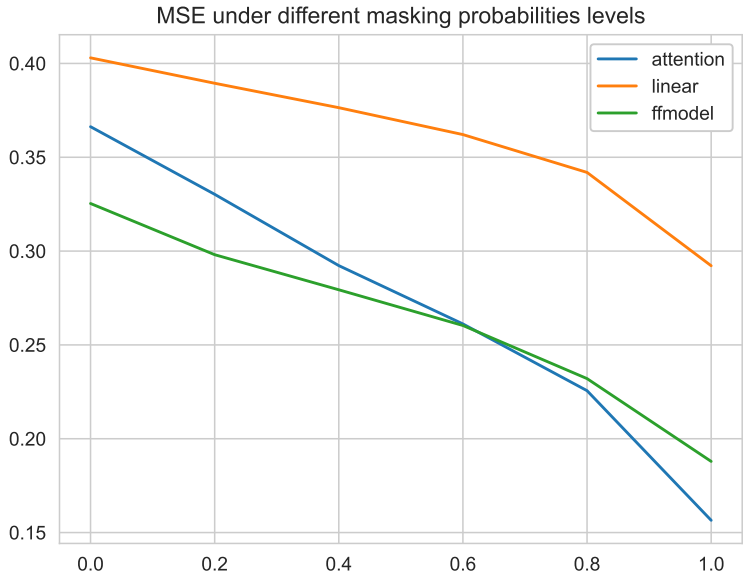


Figure 4.6: MSE of the architectures using the stacked-vector representation under packet loss from [8].

Table 4.2: The mean-squared error and mean-absolute error of our experiments from [8] for different noise σ^2 rounded to 5 significant digits.

		0.0		0.2		0.4		0.6		0.8		1.0	
		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Single Vector	Linear	0.1416	0.22297	0.14156	0.22290	0.14157	0.22291	0.14158	0.22292	0.14158	0.22293	0.14161	0.22295
	Attention	0.14497	0.22538	0.14494	0.22541	0.14493	0.22539	0.14499	0.22544	0.14507	0.22549	0.14518	0.22554
	Feed-Forward	0.16314	0.23456	0.16322	0.23469	0.16305	0.23457	0.16333	0.2347	0.16346	0.2348	0.16376	0.23494
Stacked Vector	Linear	0.40298	0.31194	0.40226	0.31293	0.43315	0.32561	0.57703	0.36746	0.91078	0.44377	1.5027	0.55100
	Attention	0.36629	0.30684	0.39984	0.31458	0.42404	0.32826	0.43668	0.34185	0.45366	0.35335	0.46726	0.36118
	Feed-Forward	0.32536	0.32503	0.31549	0.32276	0.30834	0.32388	0.31329	0.33007	0.32384	0.33802	0.33646	0.34612
Only Local	Linear	1.1218	0.26818	1.2017	0.29491	1.7833	0.42842	3.4959	0.66198	7.1744	0.98834	13.475	1.3921
	Attention	0.16573	0.23358	0.25946	0.2629	0.30258	0.2851	0.33865	0.30937	0.39250	0.33768	0.46012	0.36731
	Feed-Forward	0.17464	0.23727	0.18684	0.24694	0.20356	0.25947	0.23819	0.27992	0.28682	0.30454	0.34490	0.33067

Table 4.3: The mean-squared error and mean-absolute error of our experiments from [8] for different masking probabilities p_{masked} rounded to 5 significant digits. The results of the only-local variants are unaltered by the masking probability, as no information is shared among the reservoirs.

		0.0		0.2		0.4		0.6		0.8		1.0	
		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Single Vector	Linear	0.1416	0.22297	0.14175	0.22303	0.14176	0.22301	0.14179	0.22300	0.14188	0.22305	0.14144	0.22259
	Attention	0.14497	0.22538	0.14916	0.22746	0.15128	0.22831	0.15211	0.22857	0.15078	0.22792	0.14098	0.22365
	Feed-Forward	0.16314	0.23456	0.17498	0.23987	0.18050	0.24196	0.18184	0.24270	0.17674	0.24276	0.1841	0.27609
Stacked Vector	Linear	0.40298	0.31194	0.38944	0.30467	0.37642	0.29736	0.36209	0.28876	0.34192	0.2756	0.29218	0.23654
	Attention	0.36629	0.30684	0.33026	0.29884	0.29226	0.29316	0.26117	0.28771	0.22561	0.27570	0.15653	0.23508
	Feed-Forward	0.32536	0.32503	0.29808	0.31674	0.27938	0.31091	0.26035	0.30394	0.23204	0.29211	0.18792	0.27503
Only Local	Linear	1.1218	0.26818	1.1218	0.26818	1.1218	0.26818	1.1218	0.26818	1.1218	0.26818	1.1218	0.26818
	Attention	0.16573	0.23358	0.16573	0.23358	0.16573	0.23358	0.16573	0.23358	0.16573	0.23358	0.16573	0.23358
	Feed-Forward	0.17464	0.23727	0.17464	0.23727	0.17464	0.23727	0.17464	0.23727	0.17464	0.23727	0.17464	0.23727

To evaluate the effect of our proposed state-mixing schemes, we compared each architecture with a non-sharing version. The results in Tables 4.2 and 4.3 suggest that non-local information is necessary to generate reliable predictions in our experiments, particularly when noise is present in the input. Comparing the stacked- and single-vector state sharing schemes, it appears that the single-vector one performs better than the stacked-vector one. This could be due to the richer, more homogenous dynamics of the single-vector representation across all reservoirs. To investigate this, we reduced the dimensionality of the reservoir states of one experimental run using principal component analysis (PCA) to two dimensions and visualized the latent space in Figures 4.7 and 4.8, with each color referring to a different reservoir. Figure 4.7 shows that the single-vector representation looks like a 2-dimensional Gaussian, while Figure 4.8 reveals that the stacked-vector representation covers multiple different regions. This could make it more difficult to use shared weights across all reservoirs, as in our experimental setup.

4.5 Conclusion and Outlook

We showed how the combination of states from multiple reservoirs can help to predict a global state on a local level with a low computational cost. The classical reservoir computing approach with linear readout had the best performance. We plan to use this system in a multivariate, multimodal setting with tight real-time constraints. Utilising the SwarmESN

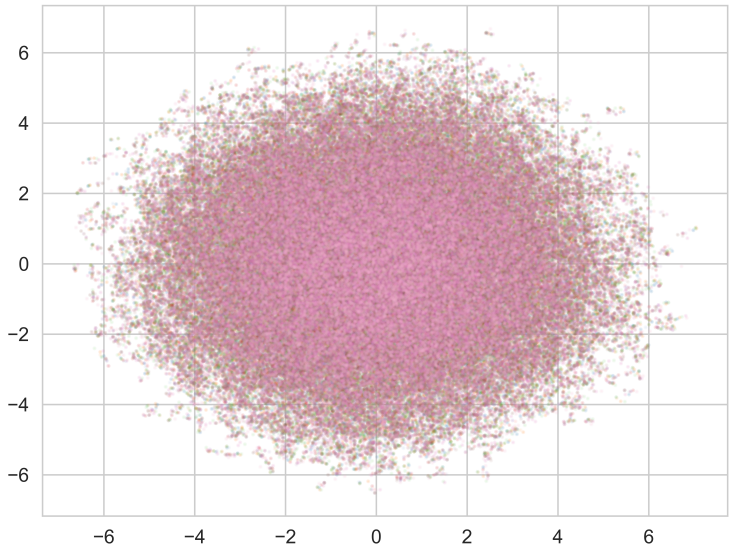


Figure 4.7: A graphical representation of all the reservoir states for one experiment from [8] is shown using the single-vector representation and the first two principal components calculated with PCA. Each of the fourteen reservoirs is represented by a different color. The states appear to be randomly distributed and resemble a Gaussian distribution.

approach with a single readout layer would enable online training even on limited hardware or FPGAs, as the learning rule can be easily derived and hard coded and would not require a lot of resources. To further enhance the attention architecture, one could keep track of the most recent τ reservoir states and then calculate the attention over these vectors. We believe that this would improve the resilience against missing non-local states, as we can directly focus on available information. Additionally, the single-vector mixing scheme is somewhat similar to the message passing algorithm for graph neural networks [73], so an extension in this direction, taking into account the edges between nodes in our distributed

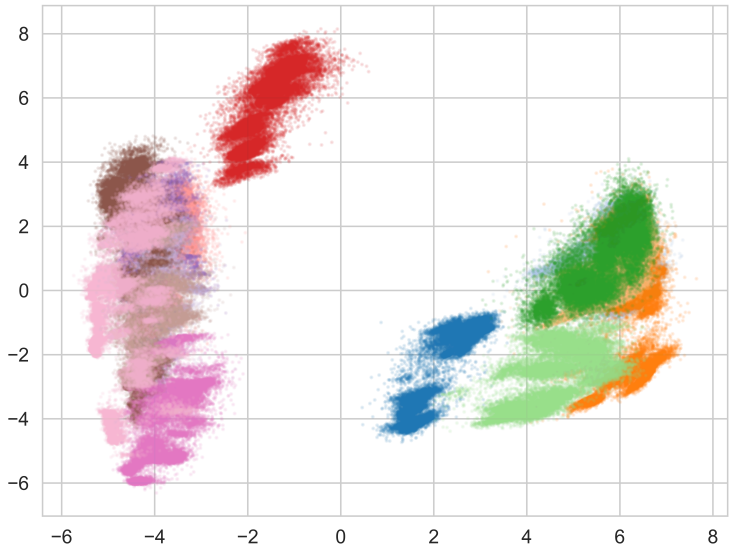


Figure 4.8: A graphical representation of the reservoir states of one experiment from [8] is shown using the stacked-vector representation and the first two principal components calculated with PCA. Each of the fourteen reservoirs is colored differently. Over time, the reservoirs tend to converge to different areas of the latent space, which is in contrast to the single-vector representation.

setting, could be beneficial. Another option would be to train separate weights for each different reservoir, but this would mean giving up on the inherent permutation invariance, i.e., inputs would be tied to a specific reservoir.

5

Conclusion

In the process of this thesis, I explored robust computation, that is, robust against noise, with reservoir computing under different conditions.

As numerical simulations are widely used in physical reservoir computing, in Chapter 3, a computational scheme for simulations was presented that can capture the differences between the simulated and the real world. Using abstract interpretation, tolerances, noise,

etc. can be captured, and using the proposed abstract regularized, we can train robust systems that can not only deal with noise, but in principle should also allow the application of the trained parameters in the real world. Thus, allowing us to train such systems in simulations, potentially reducing the costs and effort for this process, and exploiting the same system in the real world. However, a limitation of the work is that this transfer was not tested; thus, one future research endeavor is the application of abstract interpretation in this setting. Using advances in cloth simulation and cloth parameter estimation[74, 75, 76], it became possible to replicate cloth dynamics based on real-world parameters. Using a woven resistor[77] in simulation and the real world, we can build a system suitable as a reservoir. The fabric dynamics would act as reservoir dynamics and can be approximated using a mass-spring network with estimated parameters from the real world. As the readout, the resistance between selected points on the cloth should be suitable, thus creating a system with applications in movement classification, e.g., accident-detecting smart clothes, wind direction/strength measurements as a flag, or even a tactile layer for (soft) robots, akin to human skin.

A distributed inference scheme using reservoir computing was explored in Chapter 4. The reservoirs acted as time-dependent signal integrators and, combined with a simple linear readout, exhibit resistance against noise on the input side, or communication errors between the different reservoirs. I further investigated different information aggregation

schemes, by either keeping local and nonlocal reservoir states separate or combining them into one single state. Here, the combination into one single state per reservoir seemed superior, and I hypthesis that is due to the richer dynamics in the single vector representation as pictured in Figures 4.8 and 4.7. The small size of the reservoir with linear readout approach also makes it suitable for embedded devices. Thus, a next step is the implementation of this system on top of the existing hardware of the fast beam interlock system[78] for the European spallation source[79], to help detect potential anomalies and protect the facility from damage.

This thesis has shown how reservoir computing can help build robust systems, not only against noise on the input but also system errors, e.g., defective sensors, and changed initial conditions of the reservoir itself. It is the cumulative work of the 5 publications listed in Chapter 6.

6

Related Publications

Publications that cumulated into this thesis in reverse chronological order. The corresponding authors are marked with *.

C. W. Senn*, Memory in Conductive Fabrics for Reservoir Computing, International Journal for Unconventional Computing (2024).

C. W. Senn*, SwarmESN for Robust Distributed Reservoir Computing (Unpub-

lished).

C. W. Senn*, I. Kumazawa: Abstract Reservoir Computing, AI (2022).

C. W. Senn*, I. Kumazawa: Abstract Echo State Networks, ANNPR2020 (2020).

C. W. Senn*, I. Kumazawa: Robust Echo State Networks, JNNS2019 (2019).

R. Berhandsgruetter*, C. W. Senn*, R. M. Fuechslin, C. Jaeger, K. Nakajima, H. Hauser: Employing L-Systems to Generate Mass-Spring Networks for Morphological Computing, NOLTA2014 (2014).

A

Abstract Reservoir Computing

A.1 Supplementary Data

A.1.1 Results

Tables A.1 - A.4 show additional statistics averaged over 10 runs for each experiment conducted.

Table A.1: Average MSE for the experiments simulating failing sensors over 10 experiment runs. The parameter gives the fraction of sensors that were forced to 0 from [7].

Dataset	Model	Sensor Failure MSE±Std Dev@Parameter Value			
		0.00	0.01	0.02	0.03
Hénon	abstract	$5.11E-01 \pm 3.10E-04$	$5.40E-01 \pm 8.56E-03$	$5.96E-01 \pm 1.24E-01$	$1.20E+00 \pm 1.35E+00$
Hénon	classical	$5.88E-01 \pm 3.17E-02$	$1.35E+01 \pm 2.69E+01$	$8.57E+00 \pm 1.54E+01$	$3.43E+01 \pm 4.09E+01$
Hénon	noise	$5.83E-01 \pm 2.38E-02$	$2.09E+00 \pm 2.74E+00$	$1.69E+00 \pm 3.13E+00$	$2.51E+00 \pm 3.56E+00$
NARMA10	abstract	$1.38E-02 \pm 3.88E-05$	$1.51E-02 \pm 8.40E-04$	$1.55E-02 \pm 4.69E-04$	$1.59E-02 \pm 1.66E-03$
NARMA10	classical	$9.99E-02 \pm 4.08E-02$	$6.08E+01 \pm 1.14E+02$	$2.13E+02 \pm 2.20E+02$	$1.60E+02 \pm 1.61E+02$
NARMA10	noise	$2.35E-01 \pm 1.19E-01$	$5.60E-02 \pm 7.01E-02$	$5.46E+00 \pm 1.12E+01$	$1.69E-02 \pm 3.29E-03$
NARMA20	abstract	$1.88E-03 \pm 3.67E-05$	$2.88E-03 \pm 3.20E-04$	$2.68E-03 \pm 1.95E-04$	$2.67E-03 \pm 3.33E-04$
NARMA20	classical	$1.21E-01 \pm 6.16E-02$	$2.49E+01 \pm 3.76E+01$	$4.11E+01 \pm 4.70E+01$	$4.95E+01 \pm 9.20E+01$
NARMA20	noise	$2.63E-01 \pm 1.43E-01$	$1.09E-02 \pm 4.58E-03$	$4.49E-03 \pm 7.36E-04$	$1.72E-01 \pm 5.05E-01$

Dataset	Model	Sensor Failure MSE±Std Dev@Parameter Value			
		0.04	0.05	0.06	0.07
Hénon	abstract	$1.38E+00 \pm 1.77E+00$	$1.69E+00 \pm 2.36E+00$	$4.81E+00 \pm 7.96E+00$	$9.43E+00 \pm 1.33E+01$
Hénon	classical	$2.36E+01 \pm 2.17E+01$	$4.75E+01 \pm 5.20E+01$	$7.42E+01 \pm 5.73E+01$	$5.20E+01 \pm 6.11E+01$
Hénon	noise	$2.01E+00 \pm 4.35E+00$	$1.95E+00 \pm 3.86E+00$	$2.99E+01 \pm 8.65E+01$	$1.61E+01 \pm 4.46E+01$
NARMA10	abstract	$1.61E-02 \pm 1.20E-03$	$1.66E-02 \pm 1.82E-03$	$4.46E-02 \pm 7.64E-02$	$2.05E-02 \pm 4.45E-03$
NARMA10	classical	$2.31E+02 \pm 1.64E+02$	$2.59E+02 \pm 1.51E+02$	$3.07E+02 \pm 2.39E+02$	$3.08E+02 \pm 2.19E+02$
NARMA10	noise	$9.72E-01 \pm 2.87E+00$	$1.49E-02 \pm 8.54E-04$	$1.34E+00 \pm 3.99E+00$	$2.78E+00 \pm 4.31E+00$
NARMA20	abstract	$2.98E-03 \pm 3.64E-04$	$2.73E-03 \pm 3.37E-04$	$2.91E-03 \pm 4.38E-04$	$3.26E-03 \pm 5.69E-04$
NARMA20	classical	$8.58E+01 \pm 7.37E+01$	$4.94E+01 \pm 4.46E+01$	$9.63E+01 \pm 5.72E+01$	$1.38E+02 \pm 1.25E+02$
NARMA20	noise	$2.95E-03 \pm 4.11E-04$	$5.46E-02 \pm 1.55E-01$	$2.31E-03 \pm 1.37E-04$	$2.21E-03 \pm 1.08E-04$

Dataset	Model	Sensor Failure MSE±Std Dev@Parameter Value		
		0.08	0.09	0.10
Hénon	abstract	$1.10E+01 \pm 1.91E+01$	$2.81E+01 \pm 3.49E+01$	$1.75E+01 \pm 3.93E+01$
Hénon	classical	$6.89E+01 \pm 1.30E+02$	$1.77E+02 \pm 2.03E+02$	$7.70E+01 \pm 1.06E+02$
Hénon	noise	$2.32E+01 \pm 6.62E+01$	$5.97E+01 \pm 7.35E+01$	$1.81E+01 \pm 5.20E+01$
NARMA10	abstract	$4.53E-01 \pm 1.26E+00$	$7.96E-02 \pm 1.77E-01$	$6.39E-02 \pm 5.86E-02$
NARMA10	classical	$3.82E+02 \pm 1.75E+02$	$2.41E+02 \pm 1.04E+02$	$2.07E+02 \pm 1.66E+02$
NARMA10	noise	$1.42E-02 \pm 3.38E-04$	$1.39E-02 \pm 5.19E-04$	$9.00E-01 \pm 2.66E+00$
NARMA20	abstract	$3.04E-03 \pm 4.95E-04$	$3.00E-03 \pm 3.61E-04$	$3.46E-03 \pm 4.50E-04$
NARMA20	classical	$7.15E+01 \pm 6.04E+01$	$1.17E+02 \pm 9.58E+01$	$1.16E+02 \pm 9.45E+01$
NARMA20	noise	$2.25E-03 \pm 1.34E-04$	$2.15E-03 \pm 2.44E-04$	$2.05E-03 \pm 1.46E-04$

Table A.2: Average MSE for the experiments simulating sensor noise over 10 experiment runs. The parameter gives the amplitude of the noise added to the sensor readings from [7].

Dataset	Model	Sensor Noise MSE±Std Dev@Parameter Value			
		0.00	0.01	0.02	0.03
Hénon	abstract	$5.11E-01 \pm 3.10E-04$	$5.36E-01 \pm 6.54E-03$	$5.43E-01 \pm 1.42E-02$	$5.42E-01 \pm 1.42E-02$
Hénon	classical	$5.88E-01 \pm 3.17E-02$	$6.10E-01 \pm 5.59E-02$	$5.82E-01 \pm 3.28E-02$	$6.07E-01 \pm 3.15E-02$
Hénon	noise	$5.83E-01 \pm 2.38E-02$	$1.28E+00 \pm 1.35E+00$	$5.92E-01 \pm 5.49E-02$	$5.50E-01 \pm 3.63E-02$
NARMA10	abstract	$1.38E-02 \pm 3.88E-05$	$1.50E-02 \pm 5.35E-04$	$1.52E-02 \pm 4.91E-04$	$1.56E-02 \pm 6.17E-04$
NARMA10	classical	$9.99E-02 \pm 4.08E-02$	$1.21E-01 \pm 2.87E-02$	$1.49E-01 \pm 8.47E-02$	$1.74E-01 \pm 9.33E-02$
NARMA10	noise	$2.35E-01 \pm 1.19E-01$	$2.01E-02 \pm 1.09E-03$	$1.64E-02 \pm 7.40E-04$	$1.58E-02 \pm 5.47E-04$
NARMA20	abstract	$1.88E-03 \pm 3.67E-05$	$2.91E-03 \pm 3.86E-04$	$2.91E-03 \pm 4.49E-04$	$2.70E-03 \pm 2.75E-04$
NARMA20	classical	$1.21E-01 \pm 6.16E-02$	$1.13E-01 \pm 6.94E-02$	$1.17E-01 \pm 4.06E-02$	$1.19E-01 \pm 4.61E-02$
NARMA20	noise	$2.63E-01 \pm 1.43E-01$	$9.48E-03 \pm 2.21E-03$	$4.97E-03 \pm 7.51E-04$	$3.55E-03 \pm 6.72E-04$

Dataset	Model	Sensor Noise MSE±Std Dev@Parameter Value			
		0.04	0.05	0.06	0.07
Hénon	abstract	$5.35E-01 \pm 2.98E-03$	$5.41E-01 \pm 1.04E-02$	$5.35E-01 \pm 6.49E-03$	$5.39E-01 \pm 6.83E-03$
Hénon	classical	$6.10E-01 \pm 3.41E-02$	$6.10E-01 \pm 3.51E-02$	$6.55E-01 \pm 6.32E-02$	$6.46E-01 \pm 8.54E-02$
Hénon	noise	$5.38E-01 \pm 1.55E-02$	$5.32E-01 \pm 1.34E-02$	$5.31E-01 \pm 9.41E-03$	$5.25E-01 \pm 1.00E-02$
NARMA10	abstract	$1.56E-02 \pm 7.24E-04$	$1.56E-02 \pm 7.65E-04$	$1.58E-02 \pm 5.73E-04$	$1.62E-02 \pm 9.39E-04$
NARMA10	classical	$3.03E-01 \pm 2.94E-01$	$4.16E-01 \pm 3.04E-01$	$2.98E-01 \pm 2.25E-01$	$6.00E-01 \pm 3.36E-01$
NARMA10	noise	$1.52E-02 \pm 6.63E-04$	$1.48E-02 \pm 6.42E-04$	$1.47E-02 \pm 2.84E-04$	$1.46E-02 \pm 4.15E-04$
NARMA20	abstract	$2.87E-03 \pm 3.53E-04$	$2.93E-03 \pm 3.31E-04$	$3.12E-03 \pm 5.87E-04$	$2.93E-03 \pm 3.55E-04$
NARMA20	classical	$1.67E-01 \pm 1.07E-01$	$1.55E-01 \pm 7.66E-02$	$1.53E-01 \pm 6.87E-02$	$2.18E-01 \pm 1.51E-01$
NARMA20	noise	$3.00E-03 \pm 3.13E-04$	$2.61E-03 \pm 2.11E-04$	$2.30E-03 \pm 9.01E-05$	$2.25E-03 \pm 1.21E-04$

Dataset	Model	Sensor Noise MSE±Std Dev@Parameter Value		
		0.08	0.09	0.10
Hénon	abstract	$5.34E-01 \pm 8.08E-03$	$5.40E-01 \pm 8.01E-03$	$5.38E-01 \pm 7.52E-03$
Hénon	classical	$6.24E-01 \pm 4.45E-02$	$6.67E-01 \pm 1.12E-01$	$8.04E-01 \pm 2.75E-01$
Hénon	noise	$5.24E-01 \pm 3.89E-03$	$5.20E-01 \pm 2.76E-03$	$5.20E-01 \pm 3.61E-03$
NARMA10	abstract	$1.59E-02 \pm 7.09E-04$	$1.67E-02 \pm 1.62E-03$	$1.66E-02 \pm 7.81E-04$
NARMA10	classical	$8.15E-01 \pm 4.14E-01$	$1.01E+00 \pm 6.59E-01$	$1.15E+00 \pm 8.90E-01$
NARMA10	noise	$1.45E-02 \pm 4.38E-04$	$1.45E-02 \pm 2.60E-04$	$1.41E-02 \pm 2.49E-04$
NARMA20	abstract	$3.05E-03 \pm 7.32E-04$	$3.14E-03 \pm 3.92E-04$	$3.45E-03 \pm 2.25E-04$
NARMA20	classical	$2.15E-01 \pm 8.25E-02$	$2.10E-01 \pm 1.12E-01$	$3.49E-01 \pm 1.67E-01$
NARMA20	noise	$2.17E-03 \pm 7.56E-05$	$2.15E-03 \pm 1.45E-04$	$2.04E-03 \pm 1.05E-04$

Table A.3: Average MSE for the experiments simulating a sensor reading shift over 10 experiment runs. The parameter gives the shift that was added to the sensor readings from [7].

Dataset	Model	Sensor Shift MSE±Std Dev@Parameter Value			
		0.00	0.01	0.02	0.03
Hénon	abstract	$5.11E-01 \pm 3.10E-04$	$5.38E-01 \pm 1.40E-02$	$5.41E-01 \pm 9.89E-03$	$5.40E-01 \pm 1.22E-02$
Hénon	classical	$5.88E-01 \pm 3.17E-02$	$6.14E-01 \pm 8.34E-02$	$5.93E-01 \pm 6.51E-02$	$2.87E+00 \pm 4.44E+00$
Hénon	noise	$5.83E-01 \pm 2.38E-02$	$8.87E-01 \pm 8.11E-01$	$5.71E-01 \pm 5.64E-02$	$5.47E-01 \pm 3.61E-02$
NARMA10	abstract	$1.38E-02 \pm 3.88E-05$	$1.51E-02 \pm 5.03E-04$	$1.52E-02 \pm 4.06E-04$	$1.56E-02 \pm 9.01E-04$
NARMA10	classical	$9.99E-02 \pm 4.08E-02$	$9.94E-01 \pm 2.71E+00$	$1.48E-01 \pm 1.51E-01$	$2.89E-01 \pm 2.88E-01$
NARMA10	noise	$2.35E-01 \pm 1.19E-01$	$2.11E-02 \pm 5.82E-03$	$1.80E-02 \pm 2.44E-03$	$1.57E-02 \pm 6.76E-04$
NARMA20	abstract	$1.88E-03 \pm 3.67E-05$	$3.13E-03 \pm 3.54E-04$	$2.72E-03 \pm 2.57E-04$	$3.10E-03 \pm 6.85E-04$
NARMA20	classical	$1.21E-01 \pm 6.16E-02$	$1.22E-01 \pm 6.48E-02$	$1.19E-01 \pm 5.79E-02$	$1.10E-01 \pm 3.31E-02$
NARMA20	noise	$2.63E-01 \pm 1.43E-01$	$9.13E-03 \pm 1.64E-03$	$4.50E-03 \pm 4.86E-04$	$3.58E-03 \pm 5.00E-04$

Dataset	Model	Sensor Shift MSE±Std Dev@Parameter Value			
		0.04	0.05	0.06	0.07
Hénon	abstract	$5.35E-01 \pm 1.02E-02$	$5.37E-01 \pm 8.50E-03$	$5.39E-01 \pm 1.24E-02$	$5.35E-01 \pm 5.82E-03$
Hénon	classical	$2.46E+00 \pm 5.26E+00$	$7.37E-01 \pm 2.69E-01$	$3.91E+00 \pm 6.66E+00$	$1.93E+00 \pm 3.51E+00$
Hénon	noise	$5.35E-01 \pm 1.81E-02$	$5.33E-01 \pm 1.14E-02$	$5.28E-01 \pm 1.41E-02$	$5.31E-01 \pm 1.88E-02$
NARMA10	abstract	$1.58E-02 \pm 7.38E-04$	$1.60E-02 \pm 7.04E-04$	$1.56E-02 \pm 6.62E-04$	$1.58E-02 \pm 6.64E-04$
NARMA10	classical	$1.55E+00 \pm 3.42E+00$	$4.15E+00 \pm 1.05E+01$	$3.42E+00 \pm 9.81E+00$	$4.44E+00 \pm 8.82E+00$
NARMA10	noise	$1.55E-02 \pm 7.33E-04$	$1.48E-02 \pm 4.89E-04$	$1.50E-02 \pm 5.83E-04$	$1.45E-02 \pm 3.55E-04$
NARMA20	abstract	$2.84E-03 \pm 2.18E-04$	$2.96E-03 \pm 2.84E-04$	$3.00E-03 \pm 2.51E-04$	$2.78E-03 \pm 4.59E-04$
NARMA20	classical	$1.57E-01 \pm 8.50E-02$	$1.06E-01 \pm 5.45E-02$	$1.71E-01 \pm 1.05E-01$	$1.72E-01 \pm 1.20E-01$
NARMA20	noise	$2.84E-03 \pm 2.25E-04$	$2.62E-03 \pm 1.80E-04$	$2.46E-03 \pm 1.32E-04$	$2.27E-03 \pm 1.47E-04$

Dataset	Model	Sensor Shift MSE±Std Dev@Parameter Value		
		0.08	0.09	0.10
Hénon	abstract	$5.34E-01 \pm 7.57E-03$	$5.32E-01 \pm 5.62E-03$	$5.34E-01 \pm 1.06E-02$
Hénon	classical	$2.44E+00 \pm 3.27E+00$	$3.76E+00 \pm 3.88E+00$	$2.78E+00 \pm 3.70E+00$
Hénon	noise	$5.30E-01 \pm 1.14E-02$	$5.27E-01 \pm 9.84E-03$	$5.23E-01 \pm 7.76E-03$
NARMA10	abstract	$1.58E-02 \pm 7.20E-04$	$1.59E-02 \pm 7.41E-04$	$1.63E-02 \pm 5.71E-04$
NARMA10	classical	$9.65E-01 \pm 1.37E+00$	$3.49E+00 \pm 9.69E+00$	$1.41E+01 \pm 2.72E+01$
NARMA10	noise	$1.46E-02 \pm 3.87E-04$	$1.44E-02 \pm 2.95E-04$	$1.44E-02 \pm 3.67E-04$
NARMA20	abstract	$2.89E-03 \pm 2.88E-04$	$3.33E-03 \pm 4.70E-04$	$3.22E-03 \pm 3.43E-04$
NARMA20	classical	$1.22E-01 \pm 5.11E-02$	$1.66E-01 \pm 8.10E-02$	$1.16E-01 \pm 5.58E-02$
NARMA20	noise	$2.19E-03 \pm 9.98E-05$	$2.04E-03 \pm 5.94E-05$	$2.04E-03 \pm 8.45E-05$

Table A.4: Average MSE for the experiments simulating tolerances in mass placements over 10 experiment runs. The parameter gives the amplitude of the noise added to the initial mass positions from [7].

Dataset	Model	Mass Displacement MSE±Std Dev@Parameter Value			
		0.00	0.01	0.02	0.03
Hénon	abstract	$5.11E-01 \pm 3.10E-04$	$5.39E-01 \pm 1.26E-02$	$5.41E-01 \pm 9.56E-03$	$5.31E-01 \pm 6.81E-03$
Hénon	classical	$5.88E-01 \pm 3.17E-02$	$5.82E-01 \pm 3.08E-02$	$5.74E-01 \pm 2.52E-02$	$6.07E-01 \pm 4.02E-02$
Hénon	noise	$5.83E-01 \pm 2.38E-02$	$1.01E+00 \pm 1.10E+00$	$6.18E-01 \pm 7.47E-02$	$5.37E-01 \pm 1.81E-02$
NARMA10	abstract	$1.38E-02 \pm 3.88E-05$	$1.48E-02 \pm 4.17E-04$	$1.49E-02 \pm 2.87E-04$	$1.56E-02 \pm 6.36E-04$
NARMA10	classical	$9.99E-02 \pm 4.08E-02$	$1.06E-01 \pm 4.09E-02$	$9.32E-02 \pm 2.04E-02$	$9.97E-02 \pm 3.41E-02$
NARMA10	noise	$2.35E-01 \pm 1.19E-01$	$2.15E-02 \pm 2.83E-03$	$1.66E-02 \pm 1.03E-03$	$1.55E-02 \pm 6.84E-04$
NARMA20	abstract	$1.88E-03 \pm 3.67E-05$	$2.94E-03 \pm 3.89E-04$	$2.55E-03 \pm 1.75E-04$	$2.74E-03 \pm 5.86E-04$
NARMA20	classical	$1.21E-01 \pm 6.16E-02$	$1.38E-01 \pm 6.91E-02$	$1.36E-01 \pm 5.19E-02$	$1.23E-01 \pm 5.56E-02$
NARMA20	noise	$2.63E-01 \pm 1.43E-01$	$8.98E-03 \pm 1.97E-03$	$5.19E-03 \pm 5.42E-04$	$3.52E-03 \pm 4.68E-04$

Dataset	Model	Mass Displacement MSE±Std Dev@Parameter Value			
		0.04	0.05	0.06	0.07
Hénon	abstract	$5.34E-01 \pm 5.45E-03$	$5.34E-01 \pm 1.00E-02$	$5.35E-01 \pm 5.70E-03$	$5.32E-01 \pm 5.11E-03$
Hénon	classical	$5.86E-01 \pm 2.65E-02$	$5.85E-01 \pm 3.60E-02$	$5.98E-01 \pm 3.42E-02$	$5.93E-01 \pm 5.82E-02$
Hénon	noise	$5.38E-01 \pm 2.55E-02$	$5.26E-01 \pm 1.09E-02$	$5.24E-01 \pm 5.41E-03$	$5.22E-01 \pm 8.28E-03$
NARMA10	abstract	$1.59E-02 \pm 8.17E-04$	$1.58E-02 \pm 7.70E-04$	$1.60E-02 \pm 7.59E-04$	$1.55E-02 \pm 2.96E-04$
NARMA10	classical	$1.48E-01 \pm 4.72E-02$	$8.14E-02 \pm 2.71E-02$	$1.19E-01 \pm 5.17E-02$	$9.44E-02 \pm 2.53E-02$
NARMA10	noise	$1.50E-02 \pm 5.30E-04$	$1.49E-02 \pm 7.80E-04$	$1.44E-02 \pm 3.22E-04$	$1.43E-02 \pm 3.03E-04$
NARMA20	abstract	$2.81E-03 \pm 3.76E-04$	$2.87E-03 \pm 3.02E-04$	$2.95E-03 \pm 4.10E-04$	$2.82E-03 \pm 3.37E-04$
NARMA20	classical	$1.16E-01 \pm 4.46E-02$	$1.20E-01 \pm 6.07E-02$	$1.31E-01 \pm 6.77E-02$	$1.29E-01 \pm 4.15E-02$
NARMA20	noise	$2.75E-03 \pm 2.23E-04$	$2.59E-03 \pm 1.90E-04$	$2.38E-03 \pm 1.45E-04$	$2.26E-03 \pm 1.82E-04$

Dataset	Model	Mass Displacement MSE±Std Dev@Parameter Value		
		0.08	0.09	0.10
Hénon	abstract	$5.28E-01 \pm 4.41E-03$	$5.31E-01 \pm 7.63E-03$	$5.28E-01 \pm 8.50E-03$
Hénon	classical	$5.87E-01 \pm 2.03E-02$	$5.85E-01 \pm 3.31E-02$	$5.57E-01 \pm 1.14E-02$
Hénon	noise	$5.19E-01 \pm 2.33E-03$	$5.21E-01 \pm 7.42E-03$	$5.20E-01 \pm 3.08E-03$
NARMA10	abstract	$1.55E-02 \pm 5.72E-04$	$1.65E-02 \pm 9.19E-04$	$1.57E-02 \pm 9.65E-04$
NARMA10	classical	$1.19E-01 \pm 3.58E-02$	$1.37E-01 \pm 8.37E-02$	$1.34E-01 \pm 7.29E-02$
NARMA10	noise	$1.42E-02 \pm 2.14E-04$	$1.40E-02 \pm 4.53E-04$	$1.39E-02 \pm 2.72E-04$
NARMA20	abstract	$2.95E-03 \pm 5.17E-04$	$2.98E-03 \pm 3.89E-04$	$2.99E-03 \pm 3.90E-04$
NARMA20	classical	$1.03E-01 \pm 3.27E-02$	$8.87E-02 \pm 3.96E-02$	$1.61E-01 \pm 7.64E-02$
NARMA20	noise	$2.09E-03 \pm 1.06E-04$	$2.04E-03 \pm 1.03E-04$	$2.04E-03 \pm 7.46E-05$

References

[1] B. R., C. W. Senn, R. M. Fuchslin, J. C., N. K., and H. H., "Employing l-systems to generate mass-spring networks for morphological computing," *IEICE Proceeding Series*, vol. 46, pp. 184–187, 09 2014.

[2] C. Fernando and S. Sojakka, "Pattern recognition in a bucket," vol. 2801, pp. 588–597, 09 2003.

[3] K. Nakajima, H. Hauser, T. Li, and R. Pfeifer, "Information processing via physical soft body," *Scientific Reports*, vol. 5, May 2015.

[4] P. Bhovad and S. Li, "Physical reservoir computing with origami and its application to robotic crawling," *Scientific Reports*, vol. 11, June 2021.

[5] K. Vandoorne, P. Mechet, T. V. Vaerenbergh, M. Fiers, G. Morthier, D. Verstraeten, B. Schrauwen, J. Dambre, and P. Bienstman, "Experimental demonstration of reservoir computing on a silicon photonics chip," *Nature Communications*, vol. 5, Mar. 2014.

[6] C. W. Senn, "Memory in conductive fabrics for reservoir computing," *Int. J. Unconv. Comput.*, 2024.

[7] C. W. Senn and I. Kumazawa, "Abstract reservoir computing," *AI*, vol. 3, no. 1, pp. 194–210, 2022.

[8] C. W. Senn, "Swarmesn for robust distributed reservoir computing.".

[9] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks," GMD Report 148, GMD - German National Research Institute for Computer Science, 2001.

[10] W. Maass and H. Markram, "On the computational power of circuits of spiking neurons," *Journal of Computer and System Sciences*, vol. 69, no. 4, pp. 593–616, 2004.

[11] J. Steil, "Backpropagation-decorrelation: online recurrent learning with o(n) complexity," in *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, vol. 2, pp. 843–848 vol.2, 2004.

[12] E. Salvato, G. Fenu, E. Medvet, and F. A. Pellegrino, "Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning," *IEEE Access*, vol. 9, pp. 153171–153187, 2021.

[13] P. Cousot and R. Cousot, "Abstract interpretation," in *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages - POPL '77*, ACM Press, 1977.

[14] K. Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biological Cybernetics*, vol. 20, no. 3-4, pp. 121–136, 1975.

[15] A. L. Maas, "Rectifier nonlinearities improve neural network acoustic models," 2013.

[16] D. Hendrycks and K. Gimpel, "Bridging nonlinearities and stochastic regularizers with gaussian error linear units," 2017.

[17] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[18] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," 2012.

[19] U. D. Schiller and J. J. Steil, "Analyzing the weight dynamics of recurrent learning algorithms," *Neurocomputing*, vol. 63, pp. 5–23, Jan. 2005.

[20] D. E. Hilt, D. W. Seegrist, and and, *Ridge, a computer program for calculating ridge regression estimates /*. Dept. of Agriculture, Forest Service, Northeastern Forest Experiment Station„ 1977.

[21] E. C. Ifeachor and B. W. Jervis, *Digital Signal Processing: A Practical Approach*. Pearson Education, 2nd ed., 2002.

[22] K. Weierstrass, *Über die analytische Darstellbarkeit sogenannter willkürlicher Functionen reeller Argumente*, vol. 3 of *Cambridge Library Collection - Mathematics*, p. 1–38. Cambridge University Press, 2013.

[23] M. H. Stone, "The generalized weierstrass approximation theorem," *Mathematics Magazine*, vol. 21, p. 167, Mar. 1948.

[24] H. Jaeger, "Short term memory in echo state networks," 2001.

[25] R. Pascanu and H. Jaeger, "A neurodynamical model for working memory," *Neural Networks*, vol. 24, pp. 199–207, Mar. 2011.

[26] I. B. Yildiz, H. Jaeger, and S. J. Kiebel, "Re-visiting the echo state property," *Neural Networks*, vol. 35, pp. 1–9, Nov. 2012.

[27] G. Manjunath and H. Jaeger, "Echo state property linked to an input: Exploring a fundamental characteristic of recurrent neural networks," *Neural Computation*, vol. 25, pp. 671–696, Mar. 2013.

[28] G. B. Morales and M. A. Muñoz, "Optimal input representation in neural systems at the edge of chaos," *Biology*, vol. 10, p. 702, July 2021.

[29] H. Hauser, "Morphological computation–a potential solution for the control problem in soft robotics," in *Advances in Cooperative Robotics*, pp. 757–764, World Scientific, 2017.

[30] R. Pfeifer and J. Bongard, *How the Body Shapes the Way We Think*. The MIT Press, 2006.

[31] M. Banzi, *Getting started with Arduino*. O'Rei, 2008.

[32] K. MIURA, "Method of packaging and deployment of large membranes in space," *The Institute of Space and Astronautical Science report*, vol. 618, pp. 1–9, 12 1985.

[33] G. V. der Sande, D. Brunner, and M. C. Soriano, "Advances in photonic reservoir computing," *Nanophotonics*, vol. 6, no. 3, pp. 561–576, 2017.

[34] D. Brunner, M. C. Soriano, and G. V. der Sande, eds., *Photonic Reservoir Computing*. Berlin, Boston: De Gruyter, 2019.

[35] H. Hauser, A. J. Ijspeert, R. M. Füchslin, R. Pfeifer, and W. Maass, "Towards a theoretical foundation for morphological computation with compliant bodies," *Biological cybernetics*, vol. 105, pp. 355–370, 2011.

[36] Y. Yamanaka, T. Yaguchi, K. Nakajima, and H. Hauser, "Mass-spring damper array as a mechanical medium for computation," in *Artificial Neural Networks and Machine Learning – ICANN 2018*, pp. 781–794, Springer International Publishing, 2018.

[37] J. C. Coulombe, M. C. York, and J. Sylvestre, "Computing with networks of nonlinear mechanical oscillators," *PloS one*, vol. 12, no. 6, p. e0178663, 2017.

[38] D. Baraff and A. Witkin, "Large steps in cloth simulation," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH '98*, ACM Press, 1998.

[39] T. Stuyck and B. A. Barsky, *Cloth Simulation for Computer Graphics*. Morgan & Claypool Publishers, 2018.

[40] X. Provot, "Deformation constraints in a mass-spring model to describe rigid cloth behaviour," in *Proceedings of Graphics Interface '95*, GI '95, (Toronto, Ontario, Canada), pp. 147–154, Canadian Human-Computer Communications Society, 1995.

[41] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson, "Physically based deformable models in computer graphics," *Computer Graphics Forum*, vol. 25, pp. 809–836, Dec. 2006.

[42] G. Urbain, J. Degrave, B. Carette, J. Dambre, and F. Wyffels, "Morphological properties of mass–spring networks for optimal locomotion learning," *Frontiers in Neurorobotics*, vol. 11, Mar. 2017.

[43] A. Murai, Q. Y. Hong, K. Yamane, and J. K. Hodgins, "Dynamic skin deformation simulation using musculoskeletal model and soft tissue dynamics," *Computational Visual Media*, vol. 3, pp. 49–60, Mar. 2017.

[44] F. Johansson, "Ball arithmetic as a tool in computer algebra," in *Maple in Mathematics Education and Research* (J. Gerhard and I. Kotsireas, eds.), (Cham), pp. 334–336, Springer International Publishing, 2020.

[45] C. Fieker, W. Hart, T. Hofmann, and F. Johansson, "Nemo/hecke: Computer algebra and number theory packages for the julia programming language," in *Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '17, (New York, NY, USA), pp. 157–164, ACM, 2017.

[46] C. W. Senn and I. Kumazawa, "Abstract Echo State Networks," in *Artificial Neural Networks in Pattern Recognition* (F.-P. Schilling and T. Stadelmann, eds.), (Cham),

pp. 77–88, Springer International Publishing, 2020.

[47] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd, "Conic optimization via operator splitting and homogeneous self-dual embedding," *Journal of Optimization Theory and Applications*, vol. 169, pp. 1042–1068, Feb. 2016.

[48] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017.

[49] A. Goudarzi, P. Banda, M. R. Lakin, C. Teuscher, and D. Stefanovic, "A comparative study of reservoir computing for temporal signal processing," 2014.

[50] M. Hénon, "A two-dimensional mapping with a strange attractor," vol. 50, pp. 69–77, Feb. 1976.

[51] O. Serradilla, E. Zugasti, J. Rodriguez, and U. Zurutuza, "Deep learning models for predictive maintenance: a survey, comparison, challenges and prospects," *Applied Intelligence*, vol. 52, pp. 10934–10964, Jan. 2022.

[52] D. Pagano, "A predictive maintenance model using long short-term memory neural networks and bayesian inference," *Decision Analytics Journal*, vol. 6, p. 100174, Mar. 2023.

[53] J. Lansky, S. Ali, M. Mohammadi, M. K. Majeed, S. H. T. Karim, S. Rashidi, M. Hosseinzadeh, and A. M. Rahmani, "Deep learning-based intrusion detection systems: A systematic review," *IEEE Access*, vol. 9, pp. 101574–101599, 2021.

[54] K. KIM, "Intrusion detection system using deep learning and its application to wi-fi network," *IEICE Transactions on Information and Systems*, vol. E103.D, no. 7, pp. 1433–1447, 2020.

[55] N. Banús, I. Boada, P. Xiberta, P. Toldrà, and N. Bustins, "Deep learning for the quality control of thermoforming food packages," *Scientific Reports*, vol. 11, Nov. 2021.

[56] S. Qiu, X. Cui, Z. Ping, N. Shan, Z. Li, X. Bao, and X. Xu, "Deep learning techniques in intelligent fault diagnosis and prognosis for industrial systems: A review," *Sensors*, vol. 23, no. 3, 2023.

[57] D. Mazzei and R. Ramjattan, "Machine learning for industry 4.0: A systematic review using deep learning-based topic modelling," *Sensors*, vol. 22, no. 22, 2022.

[58] Qu, Zhongnan, *Enabling Deep Learning on Edge Devices*. PhD thesis, 2022.

[59] N. Moustafa, M. Keshk, K.-K. R. Choo, T. Lynar, S. Camtepe, and M. Whitty, "DAD: A distributed anomaly detection system using ensemble one-class statistical

learning in edge networks," *Future Generation Computer Systems*, vol. 118, pp. 240–251, May 2021.

[60] Q. Li, L. Huang, Z. Tong, T.-T. Du, J. Zhang, and S.-C. Wang, "DISSEC: A distributed deep neural network inference scheduling strategy for edge clusters," *Neurocomputing*, vol. 500, pp. 449–460, Aug. 2022.

[61] R. Stahl, Z. Zhao, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann, "Fully distributed deep learning inference on resource-constrained edge devices," in *Lecture Notes in Computer Science*, pp. 77–90, Springer International Publishing, 2019.

[62] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *arXiv preprint arXiv:1612.08242*, 2016.

[63] J. Postel, "Transmission control protocol," STD 7, RFC Editor, September 1981. http://www.rfc-editor.org/rfc/rfc793.txt.

[64] A. Sapio, M. Canini, C. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. R. K. Ports, and P. Richtárik, "Scaling distributed machine learning with in-network aggregation," in *18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, April 12-14, 2021* (J. Mickens and R. Teixeira, eds.), pp. 785–808, USENIX Association, 2021.

[65] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose, "Recent advances in physical reservoir computing: A review," *Neural Networks*, vol. 115, pp. 100–123, 2019.

[66] K. Nakajima, "Physical reservoir computing—an introductory perspective," *Japanese Journal of Applied Physics*, vol. 59, p. 060501, May 2020.

[67] L. Grigoryeva and J.-P. Ortega, "Echo state networks are universal," *Neural Networks*, vol. 108, pp. 495–508, Dec. 2018.

[68] L. Gonon and J.-P. Ortega, "Fading memory echo state networks are universal," *Neural Networks*, vol. 138, pp. 10–13, June 2021.

[69] P. Antonik, A. Smerieri, F. Duport, M. Haelterman, and S. Massar, "Fpga implementation of reservoir computing with online learning," 2015.

[70] J. Dong, E. Börve, M. Rafayelyan, and M. Unser, "Asymptotic stability in reservoir computing," 2022.

[71] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[72] P. Schmidt, A. Reiss, R. Duerichen, C. Marberger, and K. Van Laerhoven, "Introducing wesad, a multimodal dataset for wearable stress and affect detection," in *Proceedings of the 20th ACM International Conference on Multimodal Interaction*, ICMI '18, (New York, NY, USA), p. 400–408, Association for Computing Machinery, 2018.

[73] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, p. 1263–1272, JMLR.org, 2017.

[74] K. S. Bhat, C. D. Twigg, J. K. Hodgins, P. K. Khosla, Z. Popović, and S. M. Seitz, "Estimating cloth simulation parameters from video," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, (Goslar, DEU), p. 37–51, Eurographics Association, 2003.

[75] N. E. Anatoliotakis, P. Koustoumpardis, and K. Moustakas, "Cloth mechanical parameter estimation and simulation for optimized robotic manipulation," in *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pp. 2612–2620, 2021.

[76] E. Ju and M. G. Choi, "Estimating cloth simulation parameters from a static drape using neural networks," *IEEE Access*, vol. 8, pp. 195113–195121, 2020.

[77] Y. Zhao, J. Tong, C. Yang, Y. fei Chan, and L. Li, "A simulation model of electrical resistance applied in designing conductive woven fabrics," *Textile Research Journal*, vol. 86, no. 16, pp. 1688–1700, 2016.

[78] S. Gabourin, M. Carroll, S. Kövecses De Carvalho, A. Nordt, S. Pavinato, and K. Rosquist, "The ess fast beam interlock system: First experience of operating with proton beam," *Proceedings of the 31st International Linear Accelerator Conference*, vol. LINAC2022, p. UK, 2022.

[79] R. Garoby, A. Vergara, H. Danared, I. Alonso, E. Bargallo, B. Cheymol, C. Darve, M. Eshraqi, H. Hassanzadegan, A. Jansson, I. Kittelmann, Y. Levinsen, M. Lindroos, C. Martins, Ø. Midttun, R. Miyamoto, S. Molloy, D. Phan, A. Ponton, E. Sargsyan, T. Shea, A. Sunesson, L. Tchelidze, C. Thomas, M. Jensen, W. Hees, P. Arnold, M. Juni-Ferreira, F. Jensen, A. Lundmark, D. McGinnis, N. Gazis, J. W. II, M. Anthony, E. Pitcher, L. Coney, M. Gohran, J. Haines, R. Linander, D. Lyngh, U. Oden, H. Carling, R. Andersson, S. Birch, J. Cereijo, T. Friedrich, T. Korhonen, E. Laface, M. Mansouri-Sharifabad, A. Monera-Martinez, A. Nordt, D. Paulic, D. Piso, S. Regnell, M. Zaera-Sanz, M. Aberg, K. Breimer, K. Batkov, Y. Lee, L. Zanini, M. Kickulies, Y. Bessler, J. Ringnér, J. Jurns, A. Sadeghzadeh, P. Nilsson, M. Olsson, J.-E. Presteng, H. Carlsson, A. Polato, J. Harborn, K. Sjö-

green, G. Muhrer, and F. Sordo, "The european spallation source design," *Physica Scripta*, vol. 93, p. 014001, Dec. 2017.

This thesis was typeset using LaTeX, originally developed by Leslie Lamport and based on Donald Knuth's TeX. The body text is set in 11 point Egenolff-Berner Garamond, a revival of Claude Garamont's humanist typeface. The above illustration, *Science Experiment 02*, was created by Ben Schlitter and released under CC BY-NC-ND 3.0. A template that can be used to format a PhD dissertation with this look & feel has been released under the permissive AGPL license, and can be found online at github.com/suchow/Dissertate or from its lead author, Jordan Suchow, at suchow@post.harvard.edu.