Reservoir computing with complex network topologies

Lukas Manuel Rösel

September 16, 2020

Matrikelnummer: 315225
Erstgutachterin: Prof. Dr. Kathy Lüdge
Zweitgutachter: Dr. Serhiy Yanchuk

Eidesstattliche Versicherung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

_____

Datum, Ort

_____

Unterschrift

Summary

In this work the reservoir performances in hybrid networks of delay-coupled Stuart-Landau oscillators with broken symmetries were investigated and compared to related symmetric topologies like uni-directional rings. While narrowly defined benchmarks were affected by these changes more thoroughly the more general combined performances like the total memory capacities of different degrees only changed slightly. The reservoir performances also appear to be intertwined with another aspect of networks which is the amount of connections. Connectivity greatly determines how often information encoded as perturbations can interact with each other and therefore mix and dilute. A difficulty emerged in the form of much more complex dynamics that made it difficult to categorize and distinguish different synchronization states in order to investigate them precisely.

Deutsche Zusammenfassung

In dieser Arbeit wurden Reservoir-Performance in hybriden Netzwerken von verzögerungsgekoppelten Stuart-Landau Oscillatoren mit gebrockenen Symmetrien untersucht und mit verwandten symmetrischen Topologien, wie uni-direktionalen Ringen verglichen. Während eng definierte Tests von diesen Veränderungen stärker beeinflusst waren variierten allgemeinere Tests, wie die totale Memory-Kapazität verschiedenen Grades, geringfügiger. Die Reservoir-Performance scheint auch vermischt zu sein mit einem anderen Aspekt von Netzwerken, welcher die Anzahl von Verbindungen ist. Sie bestimmt, wie schnell Informationen, die als Störungen kodiert sind miteinander interagieren und sich dadurch vermischen und verdünnen können. Eine Schwierigkeit tauchte auf in Form von deutlich komplexerer Dynamik, die die Kategorisierung und Unterscheidung von unterschiedichen Synchronisierungszuständen in Konsequenz die genaue Untersuchung erschwerte.

Danksagung

Mein Dank geht an Prof. Kathy Lüdge für die Möglichkeit, über dieses interessante und komplexe Thema eine Arbeit schreiben zu können. Mein Dank gilt den viele wichtige Anregungen, der vielen Geduld und der Bereitstellung einer angenehmen und rundherum freundlichen Arbeitsumgebung. Des weiteren geht mein Dank an Dr. Serhiy Yanchuk, der zur rechten Zeit ein paar wichtige Denkanstöße lieferte, welche den sprichwörtlichen Groschen weiterrollen (und fallen) ließen. Außerdem möchte ich meinen beiden Betreuern Dr. André Röhm und Felix Köster danken, mit denen ich viele ausschweifende Diskussionen führte und denen ich viele umständliche Fragen stellen durfte. Zuletzt möchte ich den anderen Mitgliedern der Arbeitsgruppe danken, meinen zwischenzeitlichen Mitbewohnern vom "Kinderzimmer" - Andrej, Clara, Lasse und Giulia - und den liebern Nachbarn aus EW632, Stefan und Jan.

Contents

1 Introduction

Reservoir Computing describes the field of machine learning in which the capacity to store and transform input data is investigated in a wide variety of dynamical systems. These systems are referred to as reservoirs and encompass many different types. Reservoir Computing (RC) can be understood as a generalization of earlier recurrent neural network architectures (RNNs), echo state networks (ESNs) and liquid state machines (LSMs). RC is particularly interesting as computations can be performed by the physical systems directly. Today "classical" computers of the van-Neumann type (or more generally of the Turing-machine-type) rely on the existence of a digital space in which everything is represented by a combination of discrete values (e.g. *On*/*Off* or 0 and 1). To create such a virtual space of "stable states" the usual unpredictability that inhabits the scales in which modern computer circuits exist in has to be tamed in order to establish clear distinctions between different states. As the scales of modern transistors shrink they rapidly approach the scales in which quantum effects become problematic and have to be accounted for by sophisticated error-correction mechanisms. But even without quantum tunneling to worry about the production of transistors at tiny scales becomes increasingly difficult: the scales of modern silicone-transistor-based CPUs have become so small that adequate light sources needed to imprint the conduction paths on silicone wafers are becoming rare and hideously expensive to operate. We are in the waning years of Moore's Law.

At the same time more and more of our daily lives rely on machine learning applications. Artificial neural networks come in many shapes and sizes and they make heavy use of convolutions paired with nonlinear transformations. The inputs of a neural network are transformed by convolving them with specialized (learned) kernels that extract certain features. On digital computers these convolutions are realized through matrix multiplications which modern GPUs are heavily optimized for. But nonlinearities can still be expensive to utilize (*sigmoid* or *tanh* functions especially). But almost everywhere the discrete data that is fed into ML systems can be regarded as originally being continuous in time and/or space. A digital photo is made of discrete pixels with discrete color intensities although the original scene would be regarded as a continuous intensity-function over a continuum of spatial positions. Digital computers have to perform most computations in serial (one after another) and are only able to perform these huge amounts of operations through their great speeds.

Reservoir computing could circumvent several of today's difficulties by exploiting the nonlinear dynamics of physical systems directly without the need of discrete values of digital computation. Analog systems can also perform convolutions of inputs in truly parallel fashion. RC similar to neural information processing is not dependent on the existence of stable states. A great benefit of RC paradigm is that the reservoir does not have to be trained in any way. Instead only a linear transformation of the output layer has to be trained. As the vanishing gradient problem makes training normal RNNs a huge and complicated topic on its own [HOC98] RC is a very attractive alternative. A wide variety of systems can be used as reservoirs e.g. a literal "bucket of water" can act as a reservoir that performs computations [FER03]. Albeit the most interesting applications lie in exploiting optical systems as they can be operated on shorter timescales than electric circuits. RC offers a way of utilizing the highly complex dynamics of optical systems in order to perform computation with them. Optical computers in the form of semiconductor lasers appear to be an ideal application of RC, because of the timescales that these systems

operate in and their intrinsic nonlinear behavior. Optical reservoir computers already perform classification tasks [BRU13a, ARG18] on timescales unmatched by modern silicon electronics. Another great benefit is the lower power consumption of such systems [VAN14, VIN15, VAN08a].

In recent years reservoir computing has received a lot of attention [SAN17a] as a bridge between machine learning and physics. As the end of "Moore's Law" is slowly encroaching upon us after an age of staggering performance leaps in silicon electronic circuits researchers are searching for the next step in computing. Reservoir Computing as a way of utilizing the much smaller timescales at which optical systems operate offer a way of building drastically faster computers that consume a fraction of te energy. and might therefor be a great candidate.

1.1 A generalization

All models are wrong, but some models are useful - attributed to *George Box*

Natural Science can be understood as an exercise to create models. Models are derived from what we see around ourselves in the real world. A good model resembles our world (or aspects thereof) closely enough so that it enables us to predict outcomes. In order to predict the motion of a thrown stone physicists have a model called *"projectile motion"*. This model, when customized to closely fit the gravitational constant g and the angle α and speed v of ejection, is able to predict the real-world destination of the stone.

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} vt\cos\alpha \\ vt\sin\alpha + \frac{g}{2}t^2 \end{pmatrix} \tag{1}$$

Clearly, the model is wrong. It does not account for the weight, size, shape or density of the stone. It does not account for aerial friction, wind, the stone's rotation (magnus effect, inertia tensor) and/or that of the earth (coriolis effect). We might extend the model in order to get a more accurate prediction (e.g. "ballistic motion" for air-resistance). But even then, the model would lack details as it does not account for velocities close to the speed of light (that would result in relativistic effects and/or even nuclear fusion and fission of air and stone molecules colliding with each other [WIF12]). But still: the model is *useful*, as it is able to approximate the stone's destination *closely enough* in most cases. If we need more detail, we can usually use a more detailed model.

Scientists are usually trying to build models that represent certain aspects of the real world in order to infer underlying states or predict outcomes. Meteorologists have models that approximate the weather, astronomers have models that predict planetary motions and were in the past able to infer the existence and positions of new planets years before final confirmation by telescope. All these models were originally derived by careful consideration, expressed through mathematical equations and could be calculated analytically or numerically.

In recent years though machine learning went about the problem from a different angle. By mimicking the way we humans or animals solve problems machine learning has made enormous progress. Whereas a physicist can predict or approximate where a stone lands by solving a differential equation, a dog can demonstrate the same capacity by catching a thrown stone (or rather: a piece of cheese) in a fraction of a second. The dog can even introduce aerodynamics when learning to catch a frisbee. The dog has built an internal model of the world which he uses to predict the trajectory of thrown objects – at least

objects of interest. Humans act the same way. We are building an internal model of the world through which we are capable of adequately predict real outcomes. We usually do this without solving equations. We build our models as detailed as needed: A talented soccer player will have more emphasis on dynamics of soccer balls. The player's model is more accurate by accounting for wind, the Magnus effect, surface friction and even ball-grass interactions in dry and wet conditions. A talented player's internal model will even include other players and their immediate behavior in order to shoot the ball somewhere the other player does not expect. In the same way *"being funny"* – i.e. the ability of making people laugh and having a good sense of humour – can be understood as a well working internal model of other people's thought processes.

By feeding a dynamical system weather data and using its response in order to approximate target values we actually build a model that resembles our real world to a degree. If the model is *useful*, we can predict the weather (e.g. the temperature tomorrow at 5 pm) *sufficiently* well.

One great benefit of machine learning models is often that they need less resources in order to make predictions. We don't need to build complete digital earth with an accurate-to-the-atom weather system in order to predict if it will rain tomorrow. Models can be vastly more effective in describing one very small but relevant part of the whole system. Although ML's biggest advantage is that one does not have to understand the task completely. Instead example data of inputs (the problem) and outputs (solutions) is used to fit a model that can successfully solve problems it hasn't seen before. In a way ML systems are simply mimicking processes that solve given examples of problem without completely understanding them. This can be an enormous benefit in cases, where it is difficult to actually formulate what the problem is, or what is needed to solve it.

Analyzing timeseries data is an important part of machine learning tasks. Many datasets can be regarded as a timeseries. Temperature, pressure and wind evolve over time and are fed into complicated climate prediction models in order to continue this timeseries into the future i.e. predicting tomorrow's weather patterns. The applications of extracting information from timeseries are vast. Predicting the stock market, driving an autonomous car based on a video sensor data and other vehicle metrics or recognizing types of heart diseases and infections through analysis of electrocardiography by machine learning algorhithms. The most apparent timeseries prediction application to this date: voice recognition algorithms are extracting the meaning of information encoded in sound waves everytime we utter the magic words "Hi, Siri", "Ok, Google" or "Alexa...". More classical prediction tasks that do not involve time-conscious datasets like images fed into one directional convolutional feed-forward networks can actually be seen as a rather remote abstraction from the way humans will look at an image. We humans tend to investigate a still image by looking at different parts not at once, but one after another, concentrating on details one after another. In areas where meaning is not immediately obvious we tend to "take another look" (i.e. a feedback mechanism). So even classical machine learning tasks e.g. image recognition tasks can be adapted as timeseries prediction tasks.

1.2 An analogy for reservoir computing

Let us imagine a pond. If we throw stones into it, they will make splashes and create concentric waves that propagate over the surface. If we look away, miss the splash and therefor don't know the position of one stone's impact, could we still estimate where the stone landed two seconds prior? Could we estimate it three seconds prior? Or 10

seconds prior? The answers would be "yes", "probably yes", "possibly yes, but with less accuracy" and our confidence would decrease further the longer we let the pattern of the waves progress. So for short amounts of time the pond has some kind of "memory" about previous events. This memory will decay as the waves' amplitudes get smaller and smaller – the memory is fading away. Additionally, throwing similar stones at comparably similar positions of the pond will create similar wave patterns. So the response of the pond to the stone's impact is not random. Theoretically other parameters might be extracted from the wave-patterns as well - provided we have seen enough stones thrown. We might also be able to estimate the size or weight, velocity, rotational velocity, angle and even shape of the stone just by looking at the waves long enough. The pond can therefor provide linear memory, as we can reconstruct the original values attached to the stones previously thrown into it. But another (counter-intuitive) possibility is the extraction of information that was *not* inserted via the stones directly. Can we extract the product of the weights of two consecutively thrown stones by analyzing the interfering waves that originated at each of the impact regions? In a nonlinear medium like water we can. We can create nonlinear functions whose arguments are the masses of stones (or other attached properties of the stones). In other words we can use the pond's response to inputs in order to construct nonlinear transformations.
Reservoir computing provides a more formal description of this pond-example. It treats the stones as a stream of inputs and the state of the pond over time as a source for outputs. These outputs can be used in a weighted sum (the output layer) in order to predict the values of some desired (nonlinear) function of the inputs.

2 Used symbols

Table of used symbols

$s(t)$ / $\mathbf{s}(t)$	the state /states of a dynamical system
$z(t)$ / $z_i(t)$	state of stuart landau oscillator (global state / individual)
$r(t)$ / $\phi(t)$	radius / phase of Stuart-Landau oscillator
Ω	frequency of a rotating wave solution
λ	bifurcation parameter (or 'pump current')
ω	intrinsic frequency
γ	nonlinearity $Im(\gamma)$ - amplitude-phase coupling)
N / N_r	number of (real) nodes or vertices in network
κ / $\kappa_{i\leftarrow j}$	coupling strength (global / individual)
ϕ / $\phi_{i\leftarrow j}$	coupling phase (global / individual)
τ / $\tau_{i\leftarrow j}$	coupling delay (global / individual)
N_v / vN	virtualization factor - virtual nodes per real node
θ	virtual node time $\theta = T/N_v$
T	input period (often called clockcycle cc). $T = N_v \times \theta$
$\mathbf{x}$ / $\mathbf{x}(nT)$	readout state / readout state at nth input period
D	read-out dimension of $\mathbf{x}$. $D = N_r \times N_v$ (or $N_r \times N_v + 1$)
K / K_{train} / K_{test}	length of input sequence and target/prediction sequence
$\mathbf{U}$	uniform distribution (usually $[-1, 1]$, $[0, 1]$ for NARMA)
$\mathbf{u}^{-h}$ / $\mathbf{u}^{-h}(t)$	left sequence of h previous inputs at time t
$\mathbf{u}^{-\infty}$	left-infinite input sequence
u / $u(t-nT)$ / u_{-n}	a random value / value of a sequence $\mathbf{u}$ at position of n input periods before a time t
$\mathbf{u}_{-n}$	the sequence/vector $\{u(-nT), u(1-nT), \ldots, u(K-nT)\}$
$o(t)$ / $\mathbf{o}$	the *true* target value or sequence thereof
$\hat{o}(t)$ / $\hat{\mathbf{o}}$	the reservoir's prediction of $o(t)$ or sequence thereof (gained through a weighted sum over $\mathbf{x}(t)$)
$\epsilon(t)$ / ϵ	error or error vector. ($\epsilon = \hat{o} - o$)
$L_d(u)$	a Legendre polynomial of degree d
$\mathbf{L}_d(\mathbf{u})$	sequence given through Legendre polynomial mapped over $\mathbf{u}$
$L_{d_1,d_2,\ldots}(u_{-n_1}, u_{-n_2}, \ldots)$	a product of several Legendre polynomials with degrees $d_1, d_2, \ldots$ and arguments $u_{-n_1}, u_{-n_2}, \ldots$
$\mathbf{L}_{d_1,d_2,\ldots}(\mathbf{u}_{-n_1}, \mathbf{u}_{-n_2}, \ldots)$	sequence given through element wise multiplication of sequences $\{\mathbf{L}_{d_1}(\mathbf{u}_{-n_1}), \mathbf{L}_{d_2}(\mathbf{u}_{-n_2}) \ldots\}$
f / $f(\mathbf{u}^{-\infty})$	benchmark transformation. The transformation we try to reconstruct through the dynamics of our reservoir.
C	the capacity of performing a certain benchmark, $C \in [0, 1]$ ($C = 1$ means best performance)
MC^d / $MC^{\{d\}}$	total memory capacity of degree d (total) or d (only).

3 Theory

3.1 Reservoir computing

Reservoir computing (RC) is an area of machine learning. It grew out of a generalization of recurrent neural networks (RNN) and/or echo state networks (ESNs). The methodology was first introduced by Herbert Jaeger and contemporaries [JAE01, JAE02, MAA02] which was in part inspired by the way biological neurons process information. Whereas RNNs today are usually still networks simulated in a digital computer, RC extends to physical systems that can be exploited in order to perform calculations. RC is able to utilize physical systems directly and thereby avoiding the necessity of stable (discrete) states needed in a digital computer.

A reservoir computer needs to exhibit a few important properties in order to perform computations in a meaningful, consistent i.e. useful manner. These properties are:

- the echo state property
- fading memory
- it has to contain a non-linearity
- similar inputs result in similar outputs

(Some of these properties can be defined as consequences of one another, but for clarity we shall mention them individually). The echo state property means that the state of the reservoir $z(t)$ is only defined by the previous inputs and nothing else. The reservoir state z thereby "echoes" the input u. In theory the state $z(t)$ is determined completely only by the left-infinite sequence $\mathbf{u}^{\infty}$ which represents an infinite number of inputs fed into the reservoir before time t. In practicality we have to replace ∞ with a finite length K of input. Fading memory is a property that lets perturbations decay over time. This means also that previously fed inputs do not affect the reservoir indefinitely. An input should not irreversibly change the dynamics e.g. by kicking the state from one attractor into different one. Instead its effects should after a while evaporate completely. The next important property is that a reservoir has to posses a nonlinear element. In order for the reservoir to actually *process* given inputs these inputs have to interact with each other. A completely linear system could be compared to a transparent medium which lets inputs through unaffected. A linear system could at most be used to remember previous inputs i.e. an optical system with a delay line through vacuum is capable of reproducing previous inputs. These nonlinearities can be realized in different ways and depend on the reservoir at hand. In [FER03] the reservoir is realized by a small water container and input is given as excitations (waves) of the surface. Water waves are subject to a number of nonlinear phenomena: The travel speed of waves is different depending on wavelengths and shallowness of the body of water. Semiconductor lasers are highly nonlinear systems with many nonlinear effects to exploit.

A reservoir has to respond predictably to a given input. If a dynamical system is chaotic, the respective trajectories of similar (but not identical) inputs will diverge. If a dynamical system has two of more attractors in pathological cases slightly differing inputs could also put it into completely different attractors basins resulting in largely different system responses. This should not occur.

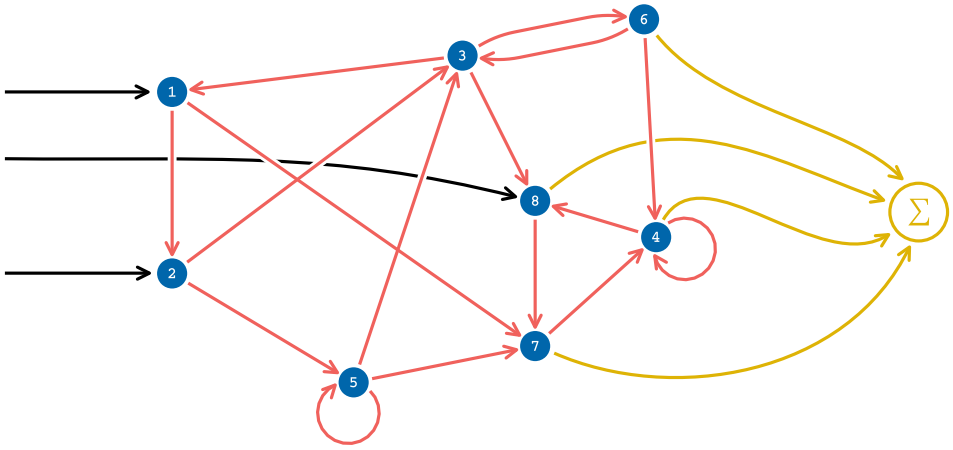

Figure 1: A simple network that can serve as a reservoir. It has some self-coupled nodes ($\{4\}$ and $\{5\}$) as well as sets of nodes that are connected in a circle ($\{1,2,3\}$,$\{1,2,5,3\}$, $\{3,4\}$ and $\{8,7,4\}$) enabling earlier inputs to remain in the network. A set nodes is receiving external input (black). Simultaneously a set of weighted read-outs are being summed up (yellow) in order to predict values of a certain task. Usually the networks are much larger in order to model more complex tasks.

3.1.1 Reservoir computing tasks

We can measure how well a dynamical system performs computations by testing it in a variety of benchmarks. Dynamical systems are continuously evolving in time, thus reservoir computation performance is often tested on time-structured data. Typically reservoir computing tasks involve transformations on data sequences rather than truly continuous time series even when the data sequences can be regarded as discretely sampled continuous data. There are mostly 2 areas that are investigated in a reservoir: benchmarks try to quantify the capacity to remember data fed in at earlier times and the ability to make nonlinear transformations of the data. In reservoir computing there exists a trade-off between these 2 types of capacities [DAM12, BUT13b, INU17]. A reservoir that excels at remembering input is worse at performing nonlinear transformations on that input. This fundamental restriction forces us to carefully think about how a reservoir can perform a given task.

$$f : \mathbb{R}^h \to \mathbb{R} : u^{-h} \to f\left[u^{-h}\right] \tag{2}$$

The different benchmarks can be defined as functions $f_1, f_2, \ldots$ that map a sequence of h previous inputs $u^{-h}(t) = (u(t-h+1), u(t-h+2), u(t-h+2), \ldots u(t))$ onto some value o 2. For a specific function or task f an *input sequence* $\mathbf{u} = (u(1), u(2), \ldots u(K)$ of length K returns a specific *target sequence* $\mathbf{o} = f(u^{-h}(1)), f(u^{-h}(2)), \ldots, u^{-h}(K)$. We can refer to the k-th element of $\mathbf{o}$ as $o(k)$. Note that $u^{-h}(t)$ is the sequence of only length h for a specific time t and is a subset of $\mathbf{u}$. In order to define $u^{-h}(0)$ we need $\mathbf{u}$ to have values for times $t < 0$ as well (It is practical to regard $\mathbf{u}$ as simply an infinite sequence of random numbers in which we can take elements at any desired position). Later $\mathbf{u}$ can be seen as an arbitrarily long sequence that we feed into our dynamical system. As a specific entry of the sequence $\mathbf{u}$ is later associated with a specific time t we write $u(t)$ for the entry at time t with $t \in \mathbb{Z}$ counted in input periods T. For convenience we write input u_0 to denote the *current* input $u(t)$ for any time t. This way we can write inputs relative to t

i.e. $u_{-1} = u(t-1), u_{-2} = u(t-2), \ldots, u_{-n} = u(t-n)$. That way the sequence of previous h inputs at any time t becomes simply $\mathbf{u}^{-h}$ and the full sequence of all inputs n steps before relative to a time t becomes $\mathbf{u}_{-n}$.

These input periods T (also referred to as "clockcycles/cc") are later referred to as the time during which one data point u (also called a "sample") is fed into a dynamic system. For an input period T the reservoir also offers D outputs which are combined into the readout vector $\mathbf{x}(t) \in \mathbb{R}^D$. This can be interpreted as reading or measuring states or dynamic variables of the dynamical system a total of D times per input period T. This means we sample the dynamical system's response on $\mathbf{u}^\infty$ at D different spatio-temporal positions (i.e. D times at one place, 1 time at D different places or at D different positions and times). The resulting readout vector can be linearly transformed in order to reach some *predicted value* $\hat{o}(t) = x^T(t)\mathbf{w}$. Ideally $\hat{o}(t) = o(t)$ which would represent a perfect reconstruction of the target value o through the system. Usually machine learning systems give imperfect predictions $\hat{o}$ that vary from the *true* value o. In Reservoir Computing, the linear transformation of the read-out vector is a key feature, because it makes the training fast and easy. More precisely: The only "learning" in reservoir computing is the fine-tuning of the weights $\mathbf{w}$ which can be done by the least squares method. Restricting the training to solely the output layer (yellow in Fig. 1) makes the training fast. The reservoir computing performance of a given reservoir can be quantified by testing its predictions for certain tasks. The word "prediction" not necessarily means to predict the future value of something e.g. extend a timeseries into the future. Instead it often refers to the estimation of the *true* value (usually gained by an exact calculation or buy waiting until a new measurement can be made) or some hidden value. For example it is possible for humans to roughly estimate if water is of boiling temperature (a hidden value) by the sound (time-structured data) that it makes when pouring a cup of tea. A weather model which has been fed temperature data of the past can be tasked to "predict" (i.e. approximate) the corresponding humidity values which must not necessarily be in the future. In this work the word prediction will be used in the same ML-typical fashion. In ML the task is usually to predict a certain value or set of values from a set of inputs. Ideally the prediction can then be compared to the ground truth and the difference between prediction and ground truth quantifies the error. The closer the predictions match the ground truth over a large dataset, the better the system performs a given task. In our case the size of the dataset is the number of inputs K that we feed into the system.

It is important to note that usually these predictions are not of discrete values. Instead of binary truth values (*true* / *false*) ML systems often operate in the realm of "fuzzy logic" i.e. *partial truths*. For example an image classification system will usually not return a definite answer like "this picture shows an elephant". Instead answers are given as a very high-dimensional vector of probabilities. This vector will have entries quantifying the 'dog-ness', the 'tree-ness', the 'car-ness' of an input image (among many other "thing-ness" values). The actual decision is made later by choosing the entry with the highest probability and omitting the rest. For predictions based on timeseries data the same applies. They are represented by continuous values that the system puts out. Even if the desired output is of a discrete nature e.g. "yes" or "no" the system will usually output a real number and thereby express a tendency rather than a clear-cut answer.

In this work a sequence of inputs $u \in [-1, 1]$ is drawn from a uniform distribution. This sequence is used as input of a transformation f which maps the sequence $\mathbf{u}$ onto its sequence of *true* target values $\mathbf{o}$. Each entry o is gained by evaluating the function f with

argument u^{-h} (2). In order to calculate the system's prediction the same input sequence u is fed into the dynamical system. The dynamic response $\mathbf{x}$ is sampled/measured for each input period T. With an input sequence $\mathbf{u}$ of length K we gain a list of K readout vectors $\mathbf{x} \in \mathbb{R}^D$. We combine all readouts into the *statematrix* $\mathbf{S} \in \mathbb{R}^{K \times D}$ (also referred to as the *design matrix*) by writing the k-th readout into the k-th row. Ideally we want that $K >> D$.

Once we have $\mathbf{S}$ we gain $\mathbf{w}$ by solving the matrix equation $\mathbf{Sw} = \mathbf{o} + \epsilon$ through the least squares method which minimizes the errors ϵ. Once we obtained the optimal weight vector $\mathbf{w}$ we can calculate the actual predictions $\hat{\mathbf{o}} = \mathbf{Sw}$ (see Fig. 5 for a plot of the predictions of Legendre polynomials $L_d(u_{-n})$ with different degrees d and at different numbers of steps in the past n).

When simply testing reservoir performance we are not really interested in the predictions $\mathbf{o}$, but rather the deviations between them and the targets i.e. the errors $\epsilon = \hat{o} - o$. We can quantify the quality of the predictions as the *capacity* of a dynamic system to perform a certain task f. If we take the sequence of errors ϵ and targets $\mathbf{o}$ we can calculate the mean square error **MSE** (3), normalized mean square error **NMSE** (4) and normalized root mean square error **NRMSE** (6) with the help of the variance (5) of a sequence of predictions.

$$MSE(\hat{\mathbf{o}}) = \frac{1}{K} \sum_{k=1}^{K} (\hat{o}_k - o_k)^2 \tag{3}$$

$$NMSE(\hat{\mathbf{o}}) = \frac{\frac{1}{K} \sum_{k=1}^{K} (\hat{o}_k - o_k)^2}{\sigma^2(\mathbf{o})} \tag{4}$$

$$\sigma^2(\mathbf{o}) = \frac{1}{K} \sum_{k=1}^{K} (o_k - \langle \mathbf{o} \rangle)^2 \tag{5}$$

$$NRMSE(\hat{\mathbf{o}}) = \sqrt{NMSE(\hat{\mathbf{o}})} \tag{6}$$

Some benchmark performances are measured using the **NRMSE**. Others are expressed by calculating the systems *capacity* to reconstruct the benchmark's targets. A system's capacity C to reconstruct targets of a certain *task* is usually defined as

$$C_{task} = 1 - NMSE(\hat{\mathbf{o}}_{task}) \tag{7}$$

Following [DAM12] we will use a different method for calculating the capacity

$$C = \frac{\mathbf{o}^T \mathbf{S} (\mathbf{S}^T \mathbf{S})^{-1} \mathbf{S}^T \mathbf{o}}{\langle \mathbf{o}^2 \rangle} \tag{8}$$

with $\mathbf{o}^T$ denoting the transpose of $\mathbf{o}$. The term $(\mathbf{S}^T \mathbf{S})^{-1} \mathbf{S}^T$ – commonly known as *Moore-Penrose-Pseudoinverse* of matrix $\mathbf{S}$ – generalizes the matrix-inverse to non-square matrices (under restrictions) and is often used to solve the least squares problem.

3.1.2 Linear Memory Recall

The simplest task a reservoir can perform is to simply reproduce the information that was fed into it at a previous point in time. If we test the capacity of reproducing the information fed into it $n = 2$ input periods T before we will write our target sequence $\mathbf{o}_2 = \mathbf{u}_{-2}$ with $\mathbf{u}_{-n} = \{u(-n), u(-n+1), u(-n+2), \ldots, u(-n+K)\}$ as described before. The number K is defining the length of the input sequence that we use in the final calculation of the

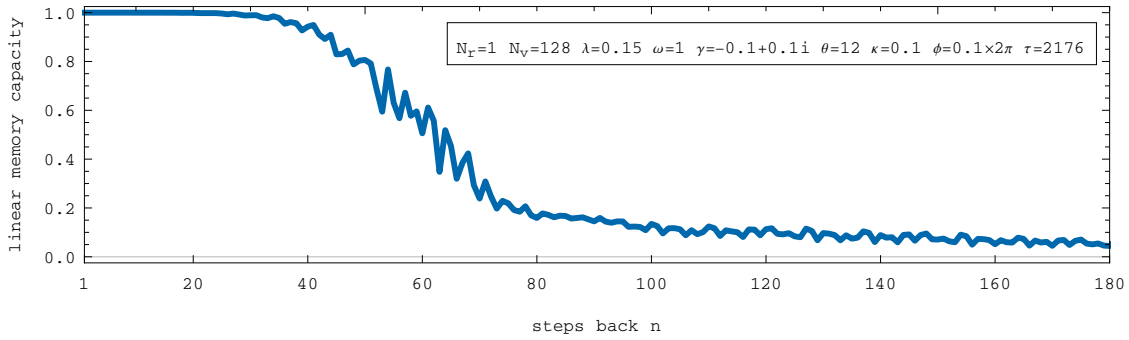

Figure 2: The linear memory capacities C_n^1 for varying steps into the past n. The single Stuart-Landau oscillator is able to almost perfectly reproduce inputs up until 30 steps into the past by combining the $D = 128$ readouts.

capacities. We gain the *linear* recall capacity C_2^1 through 8 with the targets $\mathbf{o}_2$. The superscript 1 explicitly denotes linearity as we will also measure nonlinear memories (see next section).

$$\begin{aligned}
\mathbf{o}_1 &= \mathbf{u}_{-1} = (u(-1), u(-1+1), u(-1+2), \ldots, u(-1+K)) \\
\mathbf{o}_2 &= \mathbf{u}_{-2} = (u(-2), u(-2+1), u(-2+2), \ldots, u(-2+K)) \\
\mathbf{o}_3 &= \mathbf{u}_{-3} = (u(-3), u(-3+1), u(-3+2), \ldots, u(-3+K)) \\
&\vdots \\
\mathbf{o}_n &= \mathbf{u}_{-n} = (u(-n), u(-n+1), u(-n+2), \ldots, u(-n+K))
\end{aligned} \tag{9}$$

Having calculated individual recall capacities $C_1^1, C_2^1, C_3^1, \ldots$ we can visualize the relationship of C_n^1 and n (see Fig. 2). Typically the recall capacity is best for more recent inputs (small n) and degrades for inputs long since gone (large n). This is a typical example of the *fading memory* property of a reservoir computer. Oftentimes instead of recall capacities at individual steps n in the past we are rather interested in the total sum $\sum_{n=1}^{\infty} C_n^1$. Certainly we cannot calculate the sum over an infinite amount of capacities. Though in reality capacities tend to decay against 0 for inputs many steps in the past. *Dambre et. al.* also showed that the sum over all capacities of linear independent tasks cannot exceed the readout dimension D [DAM12]. All linear recall tasks in (9) are linear independent of one another as the variables in the sequence $\mathbf{u}$ are completely random. This means that that no correlation between entries at relative positions can be found. In practice this condition depends on the implementation and has to be taken care of. In computer science there are no *true* random number generators, only *pseudo-random* number generators. Sequences drawn from pseudo-random generators have a finite period at which they repeat. In the context of this work though, we assume perfect randomness as the repetition period of modern pseudo-random number generators vastly exceeds the number of training inputs K.

In Fig. 3 for some reservoir weights are plotted that are used in order to reconstruct previous inputs u_{-n} from read-outs x. They are grouped together by task (one new recall step n for each new color). One can see that the magnitudes of these weights become very large when n increases. Beyond a certain point even with these large weights it is impossible to reconstruct the previous inputs with small errors.

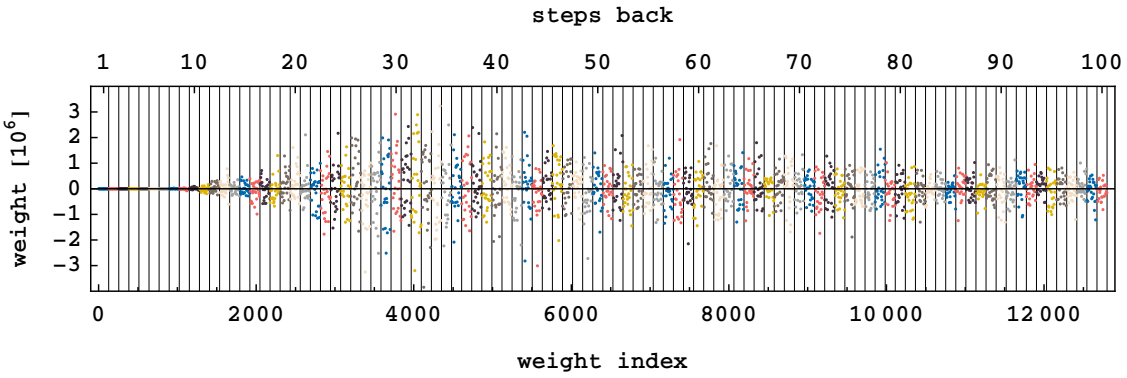

Figure 3: All weights $W_{i,s}$ attained through linear regression of the linear memory recalls $s \in [1, 100]$ steps back. The system was a unidirectional ring with of $N_{real} = 8$ and $N_{virtual} = 16$ nodes. The total read-out dimension is 128. For reconstruction of more recent inputs the inputs are small, but become enormous for inputs further in the past. This is the equivalent of "grasping at straws" as the system tries to extract information by multiplying microscopic fluctuations in the system state to linearily combine them to values between $[-1, 1]$.

3.2 Legendre polynomials as benchmarks for nonlinear transformations

$$\begin{aligned} L_0(x) &= 1 \\ L_1(x) &= x \\ L_2(x) &= \tfrac{1}{2}(3x^2 - 1) \\ L_3(x) &= \tfrac{1}{2}(5x^3 - 3x) \\ L_4(x) &= \tfrac{1}{8}(35x^3 - 30x^2 + 3) \end{aligned} \tag{10}$$

We can measure capacities for individual transformations $f_1(x), f_2(x), \ldots$, but measuring specific benchmarks - though it may be interesting - cannot tell us much about the total capacity or versatility of a dynamic system. In order to investigate all the linear and nonlinear transformation capabilities one can use Legendre polynomials (10) as transformations of inputs $u \in \mathbf{u}$ into a sequence of targets $\mathbf{o}$. A Legendre polynomial $L_{d_1}(x)$ has the useful property of being orthogonal to every other Legendre polynomial $L_{d_2}(x)$ if $d_1 \neq d_2$ and $x \in [-1, 1]$. This makes them highly useful for measuring linearly independent linear or nonlinear transformation capacities of an input u which has been shown by Dambre et. al. [DAM12]. Legendre polynomials $L_d(x)$ for degrees $d \in \{1 - 5\}$ are shown in Fig.: 4. Depending on the definition they are scaled so that $L_d(1) = 1$, which was also made sure of in this work. The Legendre polynomial $L_1(x)$ is of course simply the identity, which means that it is possible to investigate the linear memory capacity this way as well. We can express any specific transformation function $f(u)$ with $u \in [-1, 1]$ as a linear combination

$$f(u) = \sum_{i=1}^{\infty} a_i \, L_{d_i}(u) \tag{11}$$

with a_i being the coefficients of the individual Legendre polynomials L with degree d_i. This means instead of measuring capacities of certain functions $f_1, f_2, \ldots$ (which might not be orthogonal) we can simply measure capacities of all Legendre polynomials $L_1, L_2, \ldots, L_\infty$. Luckily we don't need to measure an infinite number of capacities as the the capacities are usually decaying quickly towards zero as the degree goes up. Similar to

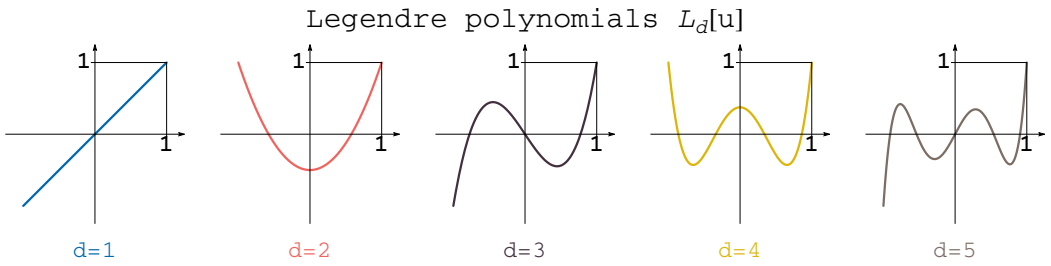

Figure 4: Legendre polynomials $L_d(u)$ for degrees $d \in \{1-5\}$ each shown for $u \in [-1, 1]$.

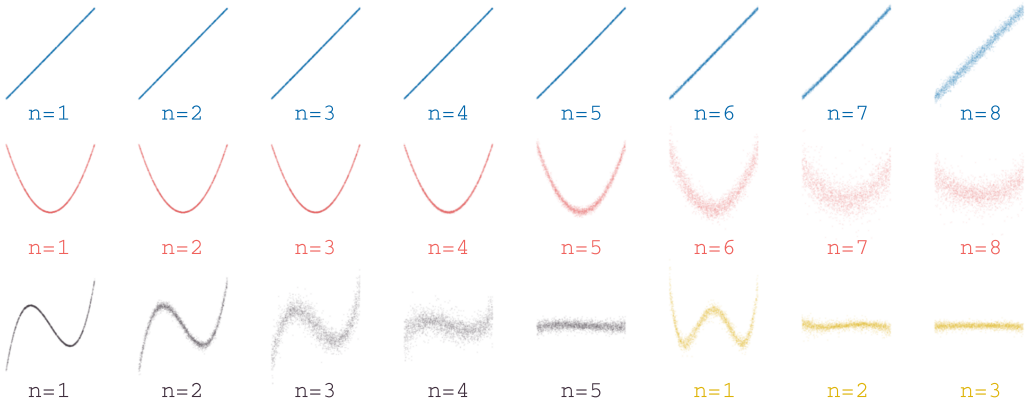

Figure 5: Using only the read-out state $\mathbf{x}$ of a reservoir with some task-specific weight vectors $\mathbf{w}_{task}$ we can - to an extent - predict the values $L_d(u_{-n})$ with n being the number of steps into the past. Tasks $L_d(u_{-n})$ with different d and different n are plotted and colored by degree as given in Fig 4. Each plot contains 2500 dots representing a pair of an input value u_{-n} on the horizontal axis with the the corresponding prediction $\hat{o}$ on the vertical axis with. Reconstructing transformations of inputs fed in the reservoir much earlier (high n) results in higher errors as can be seen in the increasing spread of the distributions. For higher degrees d the results' qualities are also decaying faster. With e.g. $L_4(u_{-n})$ (in yellow) the reconstruction only really works for $n = 1$.

the way that the linear recall capacity decays (Fig. 2). In Fig. 5 the actual predictions $\hat{o}$ are shown for a number of tasks consisting of a single Legendre polynomial. On the horizontal axis there are the values $\mathbf{u}_{-n}$ and on the vertical axis are the predictions $\mathbf{o} = \mathbf{L}(\mathbf{u}_{-n})$. Each image contains 2500 predictions. It is apparent that the ability to recall u_{-n} for small n (in the recent past) is almost perfect. But as n grows the system's predictions become less and less correlated with the true target o. For higher degrees d the predictions degrade much faster as n increases.

3.2.1 Products of Legendre polynomials

As we do not just feed a single input u in our dynamic system, but a long input sequence $\mathbf{u}$ we want to measure capacities of orthogonal transformations over a whole input sequence $\mathbf{u}^{-h}$. This means we also want to test capacities for products of polynomials that combine different inputs i.e u_{-1} and u_{-5}. As an example $o = L_2(u_{-n_1}) \times L_1(u_{-n_2})$ would be the product of two functions L_2 and L_1 with arguments given n_1 and n_2 steps previously.

In general we describe the product of Legendre polynomials through

$$L_{\{d\}}(u_{\{n\}}) = \prod_{i=1} L_{d_i}(u_{-n_i}) \tag{12}$$

with $\{d\} = \{d_1, d_2, \dots\}$ containing the different degrees of the subsequent Legendre polynomials and $\{n\} = \{n_1, n_2, \dots\}$ being the list of steps in the past of the respective arguments $u_{-n_1}, u_{-n_2}, \dots$. The list of individual degrees $\{d\}$ is also called the *powerlist* sometimes. For simplicity we call any function $L_{\{d\}}(u_{\{n\}})$ a Legendre-task for short. We will denote the specific capacity of predicting values of the specific Legendre-task $L_{\{3,2\}}(u_{\{-n_1,-n_2\}})$ by $C^{\{3,2\}}_{\{n_1,n_2\}}$.

3.2.2 Adding together capacities

Our aim is to calculate not just the total capacity of all Legendre transformations MC, but instead the selective total capacities of combined degree or total power p which we will denote by $MC^1, MC^2, MC^3, \dots$. If we calculate the total memory capacity of *only* degree d i.e. the specific *powerlist* $\{d\} = \{2\}$ we will write $MC^{\{2\}}$ where we explicitly write it as a powerlist $\{2\}$ i.e. a list containing only a single degree of 2. For MC^p we add all capacities of Legendre-tasks with powerlists that have a sum total of p. E.g. for MC^3 we have to add all capacities that come from Legendre-tasks with powerlists $\{\{3\}, \{2,1\}, \{1,2\}, \{1,1,1\}\}$. For the total linear memory capacity $MC^1 = MC^{\{1\}}$.
If we add (theoretically *all*) Legendre-tasks together then we will count some capacities multiple times or will count some capacities that are not linear independent of others. In order to avoid this we have to adhere to some restrictions for the steps in $\{n\}$:

1. $n_i \neq n_j$ for $i \neq j$
2. $n_i < n_j$ for $i < j$.

We need restriction (1), because a product $L_{d_1}(u)L_{d_2}(u)$ is linear dependent on $L_{d_1+d_2}(u)$. When we test all possible powerlists $\{\{d\}\}$ we will also independently count $L_{d_1+d_2}$. Therefor we want to avoid counting the former. We need restriction (2), because e.g. $L_{d_1}(u_{-n_1}) \times L_{d_2}(u_{-n_2}) = L_{d_2}(u_{-n_2}) \times L_{d_1}(u_{-n_1})$ – the order of multiplication does not matter.

Often we do not want the total memory capacity MC, but rather the total memory capacities MC^p with total degree or total power p. The total degree p is the sum total of $\{d\} = \{d_1, d_2, \dots, d_n\}$ and is sometimes written $||\{d\}||_1 = \sum_{i=1}^{n} d_i$ (i.e. the 1-norm of a vector).
In practice we want to reverse the approach and start by setting the total power of a powerlist p and find all powerlists with $||\{d\}|| = p$. For each powerlist we calculate the capacities with different combinations of steps $\{n\}$ and add them together to gain the value MC^3.

In general we perform the following steps to find the capacity MC^p:

1. set maximum degree p
2. find all powerlists with $||\{d\}||_1 = p$
3. for each powerlist iterate over individual steps in $\{n\}$
4. add capacity C to MC^p

In reality we cannot calculate an infinite amount of different capacities and will use the fact that C is decaying with recall step n and degree d or combined degree p. We will typically stop calculating capacities after a while when increasing steps n.

As we are not (unfortunately) working in the limit of $K \to \infty$ we can only use input sequences $\mathbf{u}$ of finite length. This means that our capacities are subject to noise and do not converge. Especially for large n (or $\{n\}$) we would sum over a large amounts of (positive) noisy values. This makes our total memory capacities MC or MCp diverge. Especially for high combined degrees the number of individual capacities $C^{\{d\}}_{\{n\}}$ explodes and causes us to mainly add noise. In order to fix this we define a threshold C_{th} and only sum over capacities C where $C > C_{th}$. The higher we set K the lower C_{th} can be, as the noise-floor is decreasing. To find the optimal value of C_{th} it is helpful to plot the actual values C_n or $C^{\{d\}d}_{\{n\}}$. In Fig. 6 the noise distributions of capacities $C^{3}_{\{n\}}$ are shown for 5000 combinations of recall steps $\{n_1, n_2, n_3\}$. More precisely: The capacities $C^{\{1,1,1\}}_{\{n_1,n_2,n_3\}}$ for 5000 combinations of large recall steps $\{n_1, n_2, n_3\}$ (which are almost exlusively noise, because the capacities have decayed quickly towards 0) are shown. The capacity values are expected to be very very small and indistinguishable from the noise. We can see these distributions become narrower and move towards zero as we increase K. In practice we want to set C_{th} so that the various sums MC^d do *not* contain values plotted in Fig. 6. In order to achieve this one has to set this threshold in a way as to not count the distributions shown in Fig. 6.

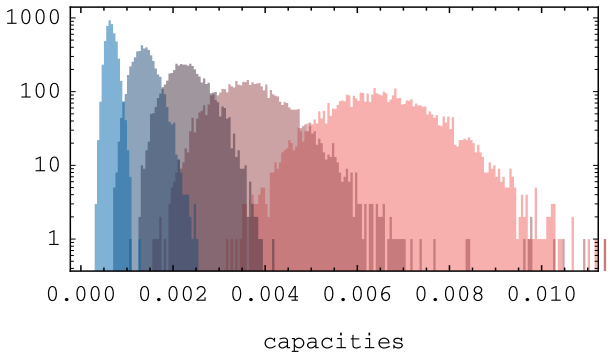

Figure 6: Varying distributions of the last 5000 capacities $C^{\{d\}}_{\{n\}}$ (noise) are shown (high number of steps n). From right to left the distributions resulted from $K = \{10^4, 2 \cdot 10^4, 3 \cdot 10^4, 5 \cdot 10^4, 10^5\}$.

3.2.3 NARMA10 - the Nonlinear Autoregressive Moving Average Task

$$A_{n+1} = 0.3A_n + 0.05A_k \left(\sum_{i=0}^{9} A_{k-9} \right) + 1.5u_{k-9}u_k + 0.1 \tag{13}$$

The performance was also investigated by measuring its capacity to compute the NARMA10 task. The Nonlinear Autoregressive Moving Average Task [HER12] is used in many publications as a benchmark. The sequence $\mathbf{o} = \mathbf{A}$ is calculated using an average of its last 10 steps while also being fed a product of a sequence u taken at two different positions. In order to calculate the NARMA10 sequence one needs memory of 10 steps (hence the "10") into the past as well as nonlinear transformation capacities. Recently it has been shown that the task is not ideal as its difficulty depends non-trivially on the shape of the distribution used [KUB19]. In certain situations A will suddenly quickly

grow to ∞ if the sequence u exhibits a particularly large moving average for a limited amount of time. In order to avoid this A_{n+1} has to be capped at 1 which might be seen as a band aid rather than an elegant solution. The NARMA10 sequence is created by the iterative formula given through (13). The inputs u are drawn from a sequence uniform distribution $U \in [0, 0.5]$. In order to use the same distribution as for the Legendre-tasks the values can simply be mapped from $u \in [-1, 1]$ onto $\hat{u} \in [0, 0.5]$ by the means of (30). We can investigate memory capacities MC^d and NARMA10 $NRMSE(\hat{\mathbf{A}})$ together by only integrating one time but using the mapped values $\hat{u}$ for the NARMA10 sequence $\mathbf{A}$.

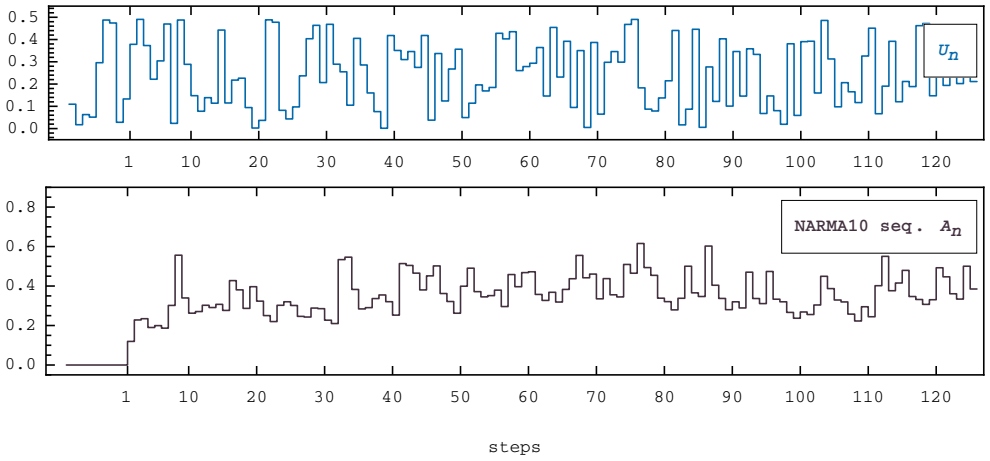

Figure 7: Example for a uniformly drawn random sequence $\mathbf{u} \in [0, 0.5]$ (top, blue) that is used to create a NARMA10 sequence A_n with it (bottom, black). Here $A_n = 0$ for $n < 1$ as an initial condition.

3.3 Networks

Mathematics defines a network as a set of *vertices* (also *nodes*) $i = \{1, 2, \ldots, N_r\}$ (with $i \in \mathbb{N}$) together with a set of *edges* (also: *connections*, *links*) describing connections between them. An edge $i \to j$ is pointing *from* node i *towards* node j. When defining edges $i \to j$ values i and j are always taken as $\mod N_r$ as an edge cannot point from or towards a node that is not in the network. There are different ways of denoting the set of edges. More compactly they can be defined as an *adjacency matrix* $\mathbf{A} \in \mathbb{R}^{N_r \times N_r}$ in which any element $a_{ij} = 1$ denotes an existing edge $i \to j$ (with $a_{ij} \in \mathbf{A}$). There are many ways of investigating networks through the properties of $\mathbf{A}$ such as eigenvalues, eigenvectors, or the detection of cycles through multiplication of $\mathbf{A}$ with itself, to name only a few. In this work though we will not consider adjacency matrices, but instead describe networks through their set of edges $\mathbf{e}$. We will also reverse the direction of the arrows $i \leftarrow j$ when describing individual parameters of edges within a network in order to have the first index i denote the node that is *receiving* input from other nodes j. These individual parameters can be coupling strength $\kappa_{i \leftarrow j}$, coupling phase $\phi_{i \leftarrow j}$ and coupling delay $\tau_{i \leftarrow j}$. If parameters are given without subscripts i.e κ, ϕ, τ they will simply define the default value of all parameters that are not specifically defined.

3.4 Rings

$$\mathbf{e}_{\rightarrow} = \mathbf{e}_{+1} = \{(i \to i + 1, \kappa, \phi, \tau) \,|\, i \leq N_r\}, \tag{14}$$

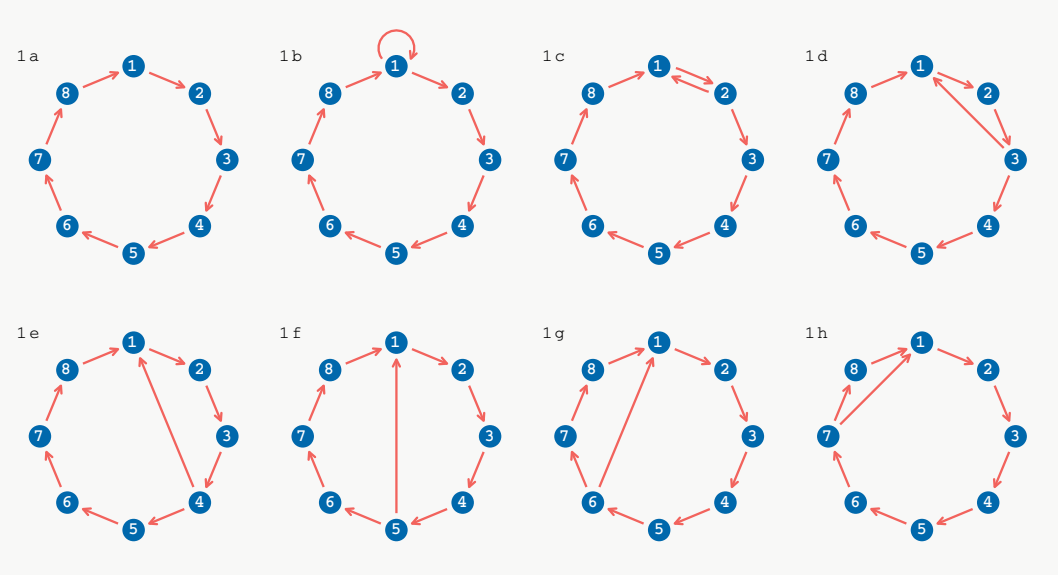

Figure 8: A uni-directional ring topology with $N_r = 8$ nodes (**1a**) and all 7 ways of placing one additional node (**1b-h**).

$$\mathbf{e}_{\leftrightarrow} = \mathbf{e}_{\pm 1} = \{(i \to i \pm 1, \kappa, \phi, \tau) \,|\, i \leq N_r\} \tag{15}$$

Many networks that we consider here are derived from or similar to ring networks. The edges $\mathbf{e}$ of a *uni-directional* ring network are defined as (14) and are visualized in Fig.8 (**1a**) for a network with $N_r = 8$. In most cases values κ, ϕ and τ are the same for all edges and are not mentioned for every edge.

In this work various network topologies of delay-coupled oscillators were considered. We describe topologies in this context as "symmetric" if the vertices/nodes can be shifted around cyclically without changing the topology. A unidirectional ring network (shown with $N_r = 8$ in Fig. 8 (**1a**) is defined by having nodes $i = \{1, 2, \ldots, N_r\}$ connected through the set of edges $\mathbf{e} = \{1 \to 2, 2 \to 3, \ldots, N_r \to 1\}$. If we change the labels i of nodes to labels $i_{+1} = i + 1 \mod 8$, but keep the edges as is, the topology remains the same. When visualizing the network (e.g. as in Fig 8 (**1a**) we can also understand this as rotational symmetry. Then the network is invariant under rotations with integer multiples of angle $2\pi/8$. In general we will use the term "symmetry" when referring to rotational symmetry of the 2-dimensional embedding of the network. This way of understanding symmetries is of course limited. There exist other symmetries in networks that become apparent only if we embed them in 3 dimensions e.g. a dodecahedral skeleton graph embedded in 3-dimensional space exhibits symmetries that we cannot intuitively understand from its 2-dimensional embedding. There are network topologies with even higher degree symmetries where we have no way of intuitively understand them, as we would need to understand their embeddings in very high dimensions. A complete graph (all-to-all coupling) is invariant under any transformation of the node-labels.

In other previous works [ROE18b, ALB19] several kinds of ring topologies have been investigated. We can have rings with uni-directional coupling, bi-directional coupling and difference-coupling which is also called diffuse coupling. Diffuse coupling in a ring network can be understood as bidirectional coupled ring together with self-links at each node that counteract the connections from other nodes.

3.5 Rings with jumps

Other networks that are not as symmetric as rings but still retain *some* amounts of symmetry are ring networks that have a set $\mathbf{j}$ periodically placed additional edges that we will call *jumps*. We define the *jump distance* k as the distance along the ring between origins of additional jump edges. The *jump offset* l is the offset between origin node and node target of a jump. The set of additional jump edges $\mathbf{j}$ can be written as (16) where the node values i, j in edges $i \to j$ are taken as $\mod N_r$.

$$\mathbf{j}_{l,k} = \{m + ik \to m + ik + l \,|\, i < N_r/k\} \tag{16}$$

This way we describe a unidirectional ring with extra jumps simply as the union of $\mathbf{e}_{\to} \bigcup \mathbf{j}$. A unidirectional ring with $N_r = 8$ and jump *length* $l = 4$ and jump *distance* $k = 4$ has 2 additional edges $\mathbf{j}_{4,4} = \{1 \to 5, 5 \to 1\}$ (network **4c** in Fig. 9). With jump length $l = 2$ and distance $k = 2$ we get 4 additional edges $\mathbf{j}_{2,2} = \{1 \to 3,\ 3 \to 5, 5 \to 7, 7 \to 1\}$ (network **3a** in Fig. 9). There are of course many more ways of designing networks with jumps in them. Instead of setting length l and distance k of jumps to the same value we can set them independently. In Fig. 9 all possible placements of jumps (excluding self-links) that retain some degree of rotational symmetry are shown. Here jump length l is set independently of jump distance k. Networks (**2** in Fig. 9) with 8 additional jumps of variing length l can all be rotated freely with $\pm 4(2\pi/8)$.

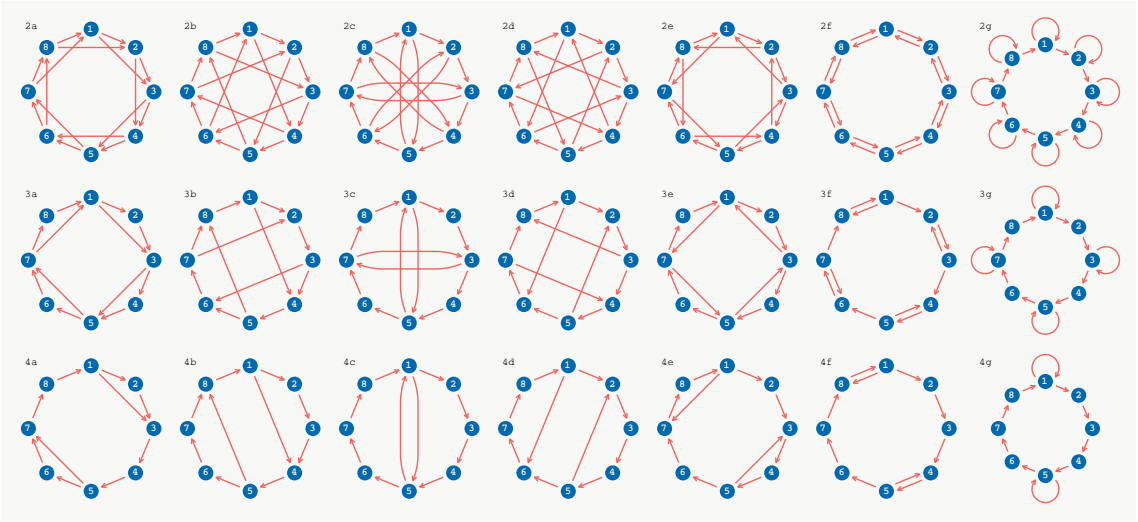

Figure 9: Adding $8, 4$ or 2 (top, mid, bottom) symmetrical jumps to a unidirectional ring with $N_r = 8$ we gain a multitude of new networks that retain some rotational symmetry. Networks (**2a-g**) with jumps $\mathbf{j}_{2,1}, \mathbf{j}_{3,1}, \ldots \mathbf{j}_{8,1}$ remain invariant under rotations $\pm(2\pi/8)$, (**3a-g**) with $\mathbf{j}_{2,2}, \mathbf{j}_{3,2}, \ldots \mathbf{j}_{8,2}$ remain invariant under rotations $\pm 2(2\pi/8)$ and (**4a-g**) with $\mathbf{j}_{2,4}, \mathbf{j}_{3,4}, \ldots \mathbf{j}_{8,4}$ remain invariant under rotations $\pm 4(2\pi/8)$. The bidirectional ring (**2f**) can also be regarded as a ring with $N_r = 8$ nodes that has one additional jump for each node that points one step back.

The benefit of networks with high amounts of symmetries is that there are fewer of them. Once we break symmetries by adding, removing or changing edges the possibilities explode. In a unidirectional ring network with $N_r = 8$ we have have 7 new networks if we place one additional edge as is shown in Fig. 8 (**1b-h**). If we add multiple jumps in a symmetrical manner as shown in Fig. 9 we get a many more possibilities just for $N_r = 8$ nodes.

3.6 The Stuart-Landau oscillator

The Stuart-Landau oscillator is a dynamical system often used to model basic class 1 lasers. These are laser systems that do not exhibit pulsing or chaotic behavior or have relaxation oscillations. It can be written either as a single complex differential equation (17) or a set of two equations written in polar coordinates (18). From the equation in polar coordinates it is easy to see that the equation has rotational symmetry as the radial differential equation does not depend on the dynamical variable ϕ.

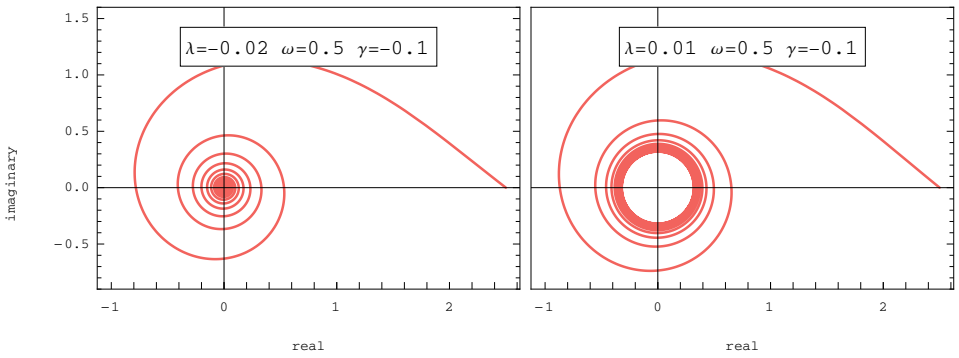

Figure 10: 2 very basic scenarios of the Stuart-Landau oscillator: Decay towards a single fixed point (off-state, left) or towards a stable oscillating state (limit cycle, right).

$$\dot{z} = (\lambda + i\omega + \gamma|z|^2)\, z \tag{17}$$

$$\begin{aligned} \dot{r} &= \lambda r + \mathrm{Re}(\gamma)\, r^3 \\ \dot{\phi} &= \omega + \mathrm{Im}(\gamma)\, r^2 \end{aligned} \tag{18}$$

For the radial dynamical variable the Stuart-Landau oscillator has two fixed points where the derivative $\dot{r}$ vanishes: $r = 0$ and $r = \sqrt{-\lambda/\,\mathrm{Re}(\gamma)}$ whose stability depends on λ and $\mathrm{Re}(\gamma)$. In practicality only values γ with $\mathrm{Re}(\gamma) < 0$ are used (supercritical case). In Fig. 11 on the left the different solutions are shown depending on the bifurcation parameter λ. For $\lambda < 0$ any initial state z_0 decays towards $z = 0$. For $\lambda > 0$ the focus becomes unstable and a new stable limit cycle solution LC appears. When $\mathrm{Im}(\gamma) > 0$ the phase velocity $\dot{\phi}$ is dependent on the amplitude or radial variable r. In this case the Stuart-Landau oscillator has an amplitude-phase coupling in which an oscillator with higher amplitude oscillates faster.

3.6.1 Stuart-Landau with delay coupling

$$\dot{z} = (\lambda + i\omega + \gamma|z|^2)\, z + \kappa e^{i\phi} z(t-\tau) \tag{19}$$

Introducing a coupling delay - an optically coupled "feedback loop" (additional term in (19) - drastically increases complexity of the dynamic system. The phase-space dimension for a single delay-coupled Stuart-Landau oscillator (Fig. 25 with $G = 0$) becomes infinite. The reason for this instantaneous growth of complexity is that the state z now becomes a function defined over the the interval $[t-\tau, t]$. The state of all possible functions over an interval $[-\tau, 0]$ is infinite.

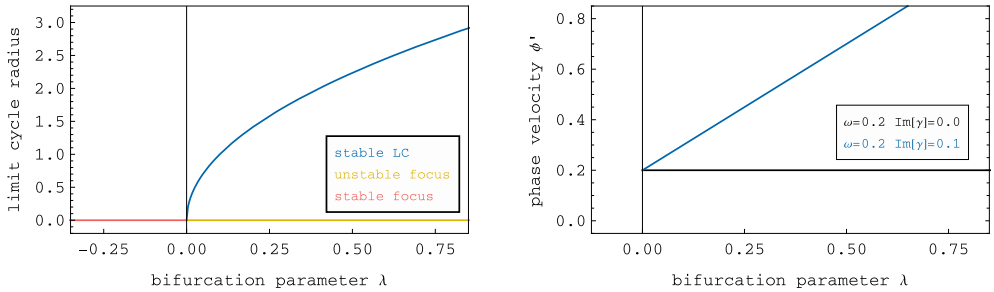

Figure 11: Dependency of limit cycle and phase velocity on parameters λ, ω and γ. (left) $\mathrm{Re}(\gamma) = -0.1$

Finding periodic solutions and investigating their stability becomes complex and is its own field of research. One particular complication that results from introducing delay is that of growing multistability when increasing the delay time τ. In [YAN09, CHO09] multistabilities in delay systems are investigated, specifically for Stuart-Landau oscillators (and Kuramoto phase-oscillators). The method is adapted to fit in this work and to account for the value $\mathrm{Re}(\gamma) < 0$ and for coupling strength κ and coupling phase ϕ. With $G = 0$, $Im(\gamma) = 0$ let us consider the rotating wave solution $z(t) = re^{i\Omega t}$ of eq. 25:

$$z(t) = Ae^{i\Omega t}. \tag{20}$$

Ω is the frequency of the rotating wave solution.
substituting z into (25) we gain:

$$i\Omega = (\lambda + i\omega + \gamma r^2) + \kappa e^{\phi - i\Omega\tau} \tag{21}$$

For a whole network of delay-coupled Stuart-Landau oscillators the investigation of multistable solutions becomes much more complicated as does the investigation of the stability of these solutions. Often times only numerical methods yield useful results. Unstable solutions are also important to investigate, as they inform us about which transitions between stable solutions (attractors) are possible and/or likely. They effectively separate portions of the phase state and act similar to a drainage divide i.e. a mountainrange that separates areas based on the ocean in which streams of water end up in.

The existence of multiple solutions has to be taken into account when investigating transformation capacities. For a set of parameters the different solutions (synchronous and various splay-states) can yield different reservoir capacities.

3.7 Multiple delay-coupled Stuart-Landau oscillators

The work focuses on networks of Stuart-Landau oscillators. Investigating all of the emerging dynamic behavior is too big of a subject and its own field of research. Some examples shall be given in order to discuss implications for the way information (later: perturbations) are propagating through the network.

In Fig. 13 the initial dynamic evolution(s) of $|z_i(t)|$ of two systems is shown. Both systems are set to state(s) $z(t < 0) = 0$ with only the first node being set to $z_1(t = 0) = 0.5$. This "kick" is propagating through both systems. It is widening until it results in the whole system approaching a (different) stable solution in the form of a limit cycle. This widening or diffusion of the initial kick is representative to all perturbations that are introduced in the system. This process also applies to information that is introduced and propagating through the system as this information exists as perturbation.

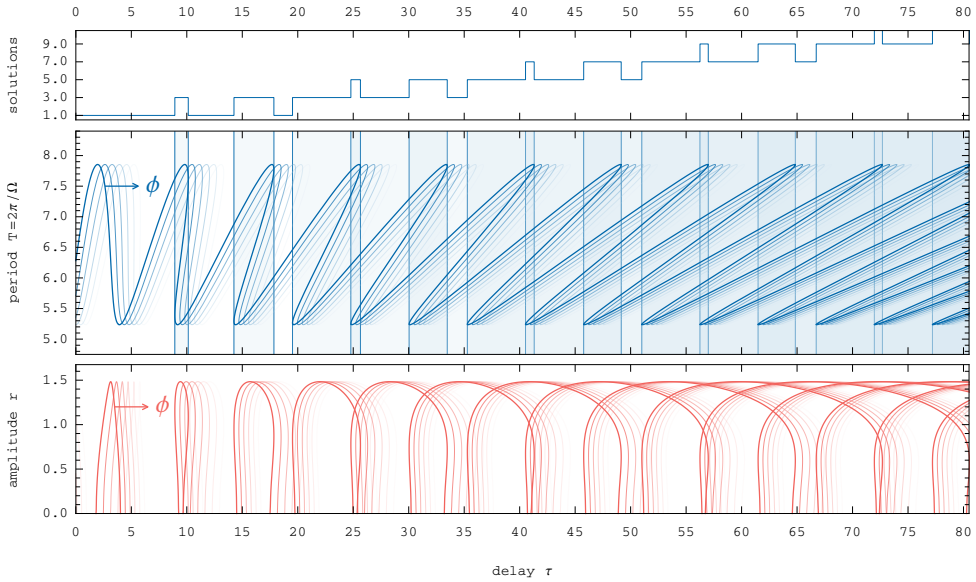

Figure 12: Solutions of (25) with $\lambda = 0.02, \omega = 1, \gamma = -0.1, \kappa = 0.2, \phi = 0$ (solutions with $\phi > 0$ are hinted). (top) Larger τ result in greater number of solutions. (middle:) relationship of τ and period T of a rotating wave solutions. Areas of multiple solutions are colored lightly blue. For greater τ these areas increasingly overlap which results in a higher number of solutions for a given τ. (bottom) the amplitude of solutions at parts where solutions $r \in \mathbb{R}^+$ exist (physical solutions with $r > 0$). For higher τ multiple amplitude levels are depending on the system's state.

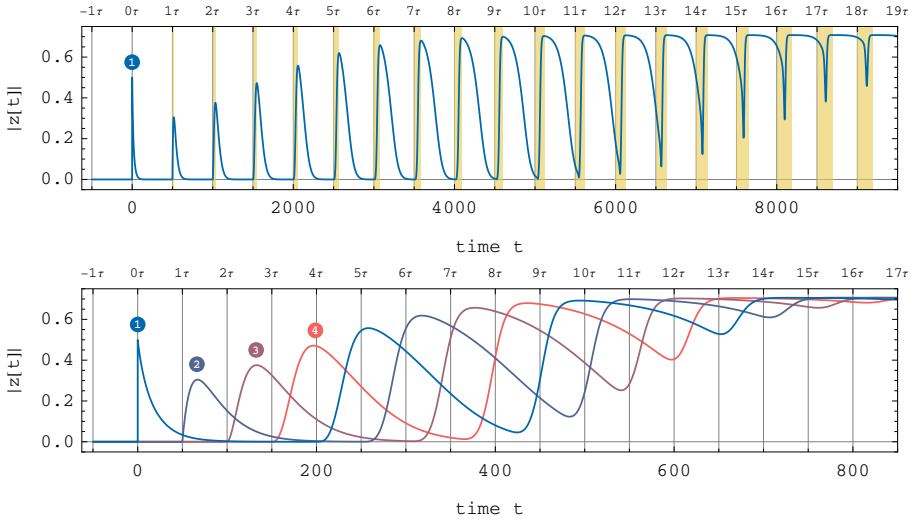

Figure 13: Two systems of Stuart-Landau oscillators with $\lambda = -0.05, \omega = 1, \gamma = -0.1, \kappa = 0.1, \phi = 0$. (top) a single self-coupled oscillator with delay $\tau = 500$, (bottom) a unidirectional ring with $N_r = 4$ oscillators and delay $\tau = 50$. The oscillators are initialized with $z(t < 0) = 0$, but node 1 (blue) is kicked/perturbed to $z_1(t = 0) = 0.5$. In the absence of feedback this perturbation is decaying (as expected for $\lambda < 0$) towards $z(t) = 0$ (off-state). But after τ it is re-introduced again and again later through the delayed coupling. Every time the perturbation propagates to another node it is widening until adjacent repetitions overlap and the node states $z_i(t)$ no longer decay against the focus. The peaks of each of the newly introduced perturbations are also delayed slighty *beyond* τ as is indicated through the yellow areas (top). This is caused by the perturbations only affecting the next coupled oscillator indirectly through the rate of change.

3.7.1 Splay-states and phase synchronization

The term "splay-state" (in this work) is used for coupled oscillators if they are oscillating out of phase, but with constant phase-differences and with the same amplitudes $\|z(t)\|$. N_r coupled Stuart-Landau oscillators in a splay-state can be described through

$$z_j(t) = Ae^{i(\Omega_m t + j\frac{2\pi m}{N_r})+\phi_0} \tag{22}$$

where $j \in 1, ..., N_r$ is the oscillator's index, A the amplitude of the oscillations, Ω_m the splay-state-specific phase-velocity and ϕ_0 a general phase offset. The number $m \in \{1, ..., 7\}$ defines the phase-difference between adjacent oscillators. It is also $mod\, N_r$. If $m = 0$ or $m = N_r$ (or in general $m = nN_r$ with $n \in \mathbb{Z}$) the oscillators would be in-phase and they would therefor not be in a splay-state.

We can measure the amount of phase synchronization of oscillators through the "order parameter"

$$P_o(\mathbf{z}(t)) = \frac{1}{N_r}\left|\sum_{i=1}^{N_r} e^{i\arg(z_i(t))}\right| \tag{23}$$

and the "generalized order parameter"

$$P_g(\mathbf{z}(t)) = \frac{1}{N_r}\left|\sum_{i=1}^{N_r} e^{iN_r\arg(z_i(t))}\right|. \tag{24}$$

The order parameter P_o is 1 if all oscillators are perfectly in-phase. For little or no synchronization of the oscillators $0 \leq P_o < 1$. The normal order parameter P_o is incapable of detecting a splay state, as $P_o = 0$. In order to fix this the "generalized" order parameter P_g can be used which multiplies the phases of oscillators $\arg(z_i(t))$ by the number of oscillators N_r. For all states in Fig. 14 the generalized order parameter $P_g = 1$ whereas the normal order parameter $P_o = 1$ only for the synchronous state (top).

Unfortunately neither the normal nor the generalized order parameters can detect synchronization patterns that occur naturally in networks when the network-topology or the individual edge-parameters $\kappa_{i\leftarrow j}, \phi_{i\leftarrow j}, \tau_{i\leftarrow j}$ change. In Fig. 15 several examples of these irregular, "splay-like" states are shown in a unidirectional ring network in which the edge parameters (κ, ϕ, τ) have been randomized slightly. They all exhibit constant amplitudes (note that delay-randomization in networks more complicated than unidirectional rings - i.e. networks in which nodes receive input from more than one node - often results in states of oscillating amplitude(s). In a symmetry-broken networks like a ring with an additional edge between nodes like Fig. 8 (**1b-h**) the resulting splay-like states can be similar or a combination of the examples in Fig. 15. They are naturally occurring states with fixed relative phase-differences between oscillators (phase-locked) and should therefor be regarded the same as a splay-state. Unfortunately though it is difficult to detect these states as both order parameters P_o and P_g are unsuited for the task.

Even worse: It is not enough to identify individual irregular splay-like states like those in Fig. 15 for one particular irregular topology: the exact configuration of one of these splay-like states can depend on coupling parameters itself even when all edges have the same parameters κ, ϕ and τ. This makes the splay-like states much harder to detect. As an example there are 2 sets of these splay-similar states shown with 2 examples respectively

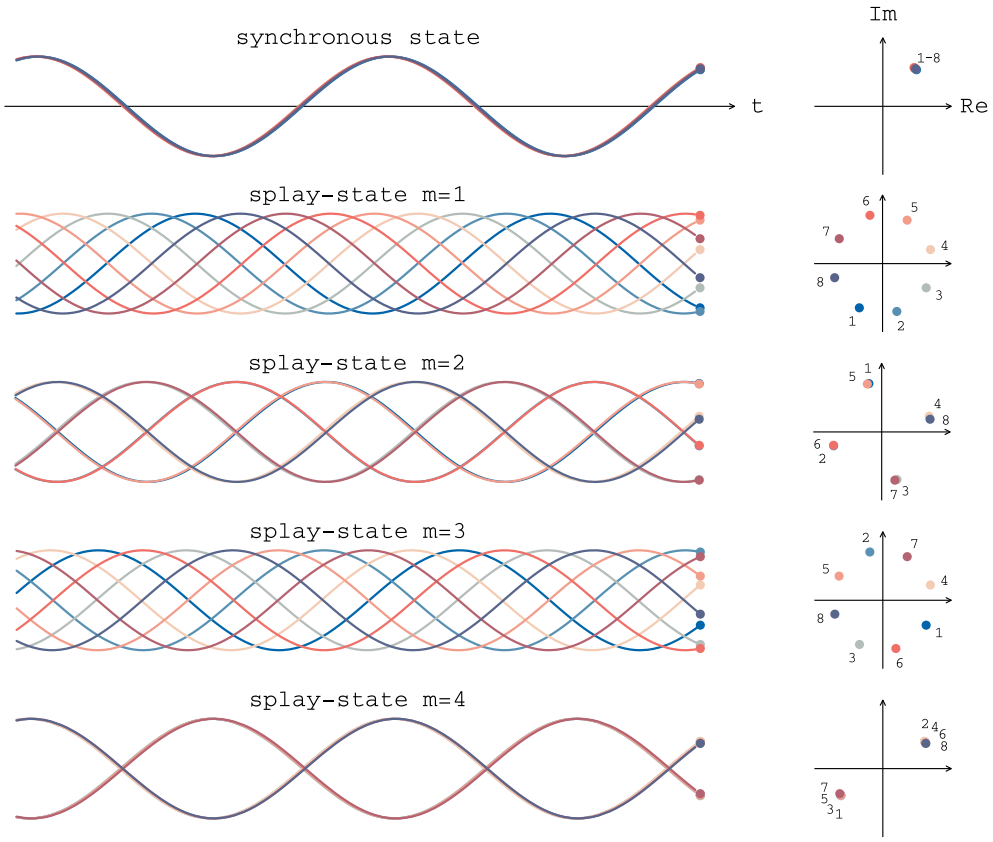

Figure 14: Synchronous and several splay-states $m \in \{1-4\}$ for a unidirectional ring with $N_r = 8, \lambda = 0.02, \omega = 1, \gamma = -0.1, \tau = 100, \kappa = 0.1, \phi = 0$. The various states were obtained merely by setting the initial conditions. The synchronous state results in an order parameter of 1, the other states in 0. All states result in a general order parameter of 1

in Fig. 16. The individual oscillators have slightly different relative positions although both were a result of the same topology (**1c**). On the left the circles were the result of coupling phase $\phi = -0.2 \times 2\pi = 0.8 \times 2\pi$, the disks a result of $\phi = 0.1 \times 2\pi$. On the right the circles were the result of coupling phase $\phi = -0.15 \times 2\pi = 0.85 \times 2\pi$, the disks a result of $\phi = 0.27 \times 2\pi$. It is possible to continuously interpolate between both sets of states i.e. between the circles and the discs by changing the (global) coupling phase ϕ. This means that both states belong to the same "family" (or continuum / category) of splay-like states. This shows that in order to identify different splay-like states it is not enough to simply compare them with each other as several found states might vary slightly but actually belong to the same family of states. Another way of understanding this problem is to realize that even a normal splay-state in a unidirectional ring of Stuart-Landau oscillators is actually a whole continuum of states with varying amplitudes depending on the value λ or the overall coupling strength κ (and to a degree a combination of the coupling phase ϕ and the delay τ). We group these states together into one particular splay-state that we can identify through a value m (22), because we identify these states merely by their relative phase-differences. These splay-like states now require us to find new ways to group them together. Inconveniently we cannot group these continua together along simple directions in parameter space (e.g. along the λ-axis).

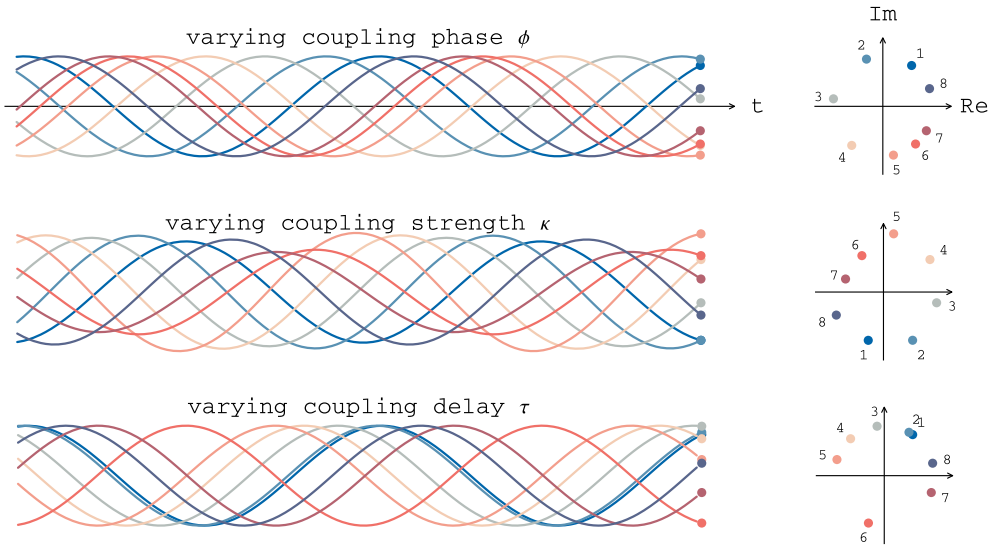

Figure 15: A unidirectional ring with $N_r = 8$ with parameters as in Fig. 14, but (slightly) randomized individual edge parameters. Varying individual coupling phases $\phi_{i\leftarrow j}$ result in unequal relative phase differences (top). Changing the coupling strengths $\kappa_{i\leftarrow j}$ results in unequal oscillator amplitudes (mid) and randomizing the coupling delays $\tau_{i\leftarrow j}$ (bottom) slightly has a similar effect as changing the coupling phases (in a simple unidirectional ring). In all three cases the oscillators' relative phase positions are locked in place - the naturally occurring synchronization states change according to the network parameters. Neither the normal nor the generalized order parameters can detect these states.

3.7.2 Varying pump current

$$\dot{z} = (\lambda + GJ(t) + i\omega + \gamma|z|^2)\, z + \kappa e^{i\phi} z(t-\tau) \tag{25}$$

The limit cycle **LC** which is shown in Fig. 10 is depending on the ratio or λ and $\mathrm{Re}\,[\gamma]$. As can be seen in (17), the equation has a linear and a nonlinear term regarding the absolute value of z. In this work we use a Stuart-Landau oscillator that has a varying bifurcation parameter $\hat{\lambda}(t)$ and a delayed feedback term $\kappa e^{i\phi} z(t-\tau)$. In an experiment this can be a simple laser that is driven by a varying pump-current. In our case $\hat{\lambda}(t)$ will always be a constant offset with only slight variations. We will therefore simply replace $\hat{\lambda}$ with $\lambda + GJ(t)$ where $G \in \mathbb{R}$ is in the order of or exactly equal to 0.01 (if not stated otherwise). G is called the "scaling factor". The input signal $J(t)$ will be the actual variation that is used to feed data into the system (see Fig. 21). The factor G which scales the perturbation $J(t)$ has a large influence on memory capacities MC^d. In Fig. 17 the total memory capacities MC^1, MC^2, MC^3 are shown for a single Stuart-Landau oscillator with delayed feedback. The best linear memory capacity was reached for the smallest values G. Although we can expect for the capacity to get worse for *very* small values. At least for $G = 0$ there can be no memory capacity, because no input $J(t)$ is perturbing the system anymore. In a system that is subjected to noise we can expect MD^1 to reach a maximum and then decay towards 0 if we let $G \to 0$. In this work there was no noise implemented which results in this paradoxical behavior. The quadratic memory capacity MC^2 starts at 0 for $G = 0$, grows quickly and reaches a maximum after which it decays again slightly. The cubic memory capacity MC^3 also starts at 0 for $G = 0$, but only slowly increases in an upward slope until it slows down. If the with higher values G the cubic memory MD^3 will probably also reach a maximum after which it decreases again. This was not explicitly tested however. The exact dependence of the

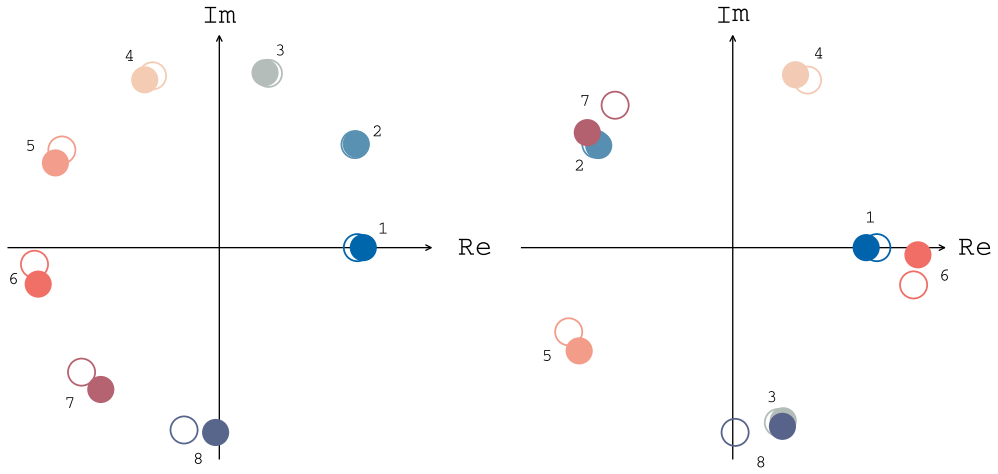

Figure 16: In the symmetry broken network (**1c**/unidirectional ring with edge $1 \leftarrow 2$) from Fig. 8 with $N_r = 8$ the exact configuration of splay-like states varies depending on the global coupling phase ϕ. Shown are 2 families/continua of splay-like states $\mathbf{z}^A$ and $\mathbf{z}^B$ with 2 examples each (○ and ●). Each color represents a single oscillator, also indicated by the index. (left / A) Circles show a state with $\phi^A_{\circ} = 0.8 \times 2\pi$ and disks show a state with $\phi^A_{\bullet} = 0.1 \times 2\pi$. A similarity to a normal splay state with $m = 1$ is apparent. (right / B) Circles show a state with $\phi^B_{\circ} = 0.85 \times 2\pi$ and disks show $\phi^B_{\bullet} = 0.27 \times 2\pi$. Both states on the left and right respectively can easily be categorized into the same continuum. New states in the same categories can be created through values $\phi_{\circ} < \phi < \phi_{\bullet}$. For values $\phi \in [\phi^A_{\circ}, \phi^A_{\bullet}] \wedge [\phi^B_{\circ}, \phi^B_{\bullet}]$ multistability exists for (at least) states A and B. All states were rotated so that $\arg(z_1) = 0$. The remaining parameters were as in Fig. 14.

memory capacities of different degrees on the input scaling factor G was not the subject of this work as it was fixed on $G = 0.01$. The strong influence of G shall be noted though. It is also relevant for networks of coupled oscillators as the input $G\,J(t)$ is not the only way oscillators are perturbed. Perturbations travel from oscillator to oscillator by the means of delayed feedback coupling. The coupling strength κ can therefor be regarded as a scaling factor on its own that influences linear or nonlinear memory capacities as well. In previous works the result of electro-optically coupled oscillators was also investigated with regards to reservoir performance and the behavior turned out to be similar to only optically coupled oscillators [EHL18].

Fig. 19 shows a solution for $|z|(t)$ with and without delayed feedback. Without feedback the intensity is exponentially decaying towards the new **LC** (in black). With it (bottom) previous inputs reappear after delay τ: The little bump encased in the second blue box is information that is fed back into the system. Usually the information isn't as visible or locally distinct, but persists for a while as a linear combination of all given readouts.

3.8 Virtual nodes and multiplexing

The traditional way of understanding reservoir computing is that of recurrent neural networks **RNN**s (also **ESN**s or **LSM**s) that consist of N_r coupled nodes. These nodes are sometimes called *artificial neurons* and can have various degrees of dynamical resemblance to real (biological) neurons [HOD52, FIT55, HIN82]. Such reservoirs can be visualized in the form of Fig.1 i.e. as a network consisting of individual nodes distributed in space. The state of such a reservoir $\mathbf{s}(t)$ is defined through the states of each individual nodes

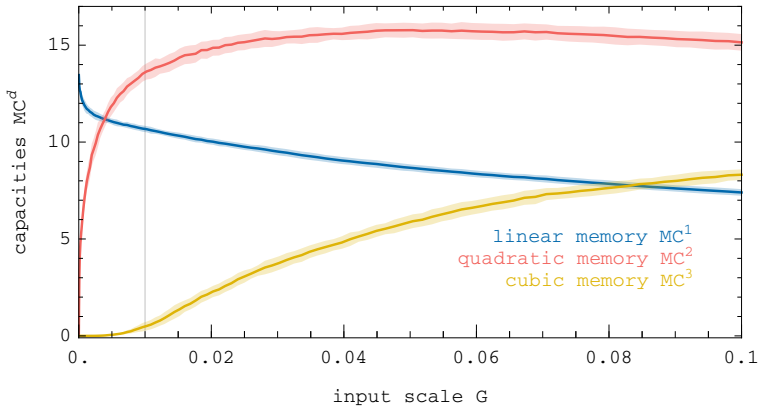

Figure 17: Capacity means and standard deviations from 20 random realizations per input scale G respectively (varying masks/initial state). G strongly determines the reservoir performance of different tasks. Parameters: $N_r = 1, \lambda = -0.02, \omega = 1, \gamma = -0.1, \tau = 68, N_v = 64, \theta = 3/4, T = N_v\theta = 48$. In this work an input scaling of $G = 0.01$ (indicated as vertical line) was used as the standard value.

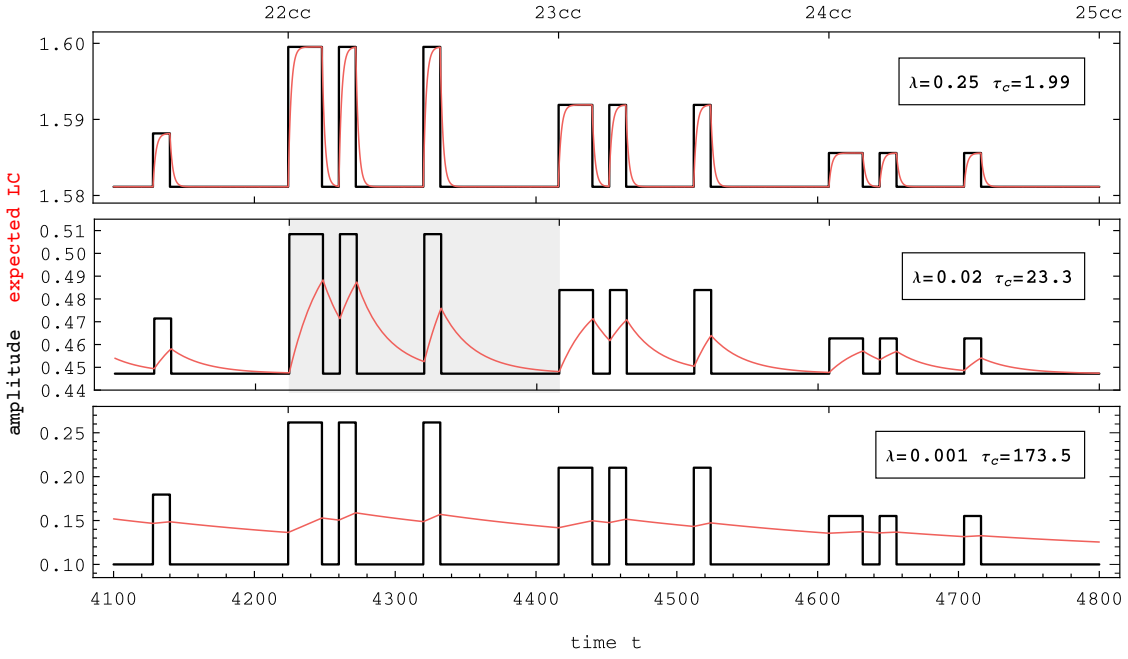

Figure 18: Different values for λ influence how fast the system's intensity ▬ is responding to a given input signal. For a large value of $\lambda = 0.25$ (top) the system's intensity almost instantaneously decays towards the expected limit cycle LC ▬. The system is therefore determined almost exclusively by the current pump current. For intermediate values e.g. $\lambda = 0.02$ (mid) the system's response is slower and can keep information longer. Very small values e.g. $\lambda = 0.001$ (bottom) make the system respond too slow to react to a given input signal. For each solution the characteristic timescales τ_c were gained by fitting a function $ae^{-t/\tau_c} + b$ to $|z(t)|^2$ with $t \in [4332, 4400]$. "cc" stands for clockcycle (T).

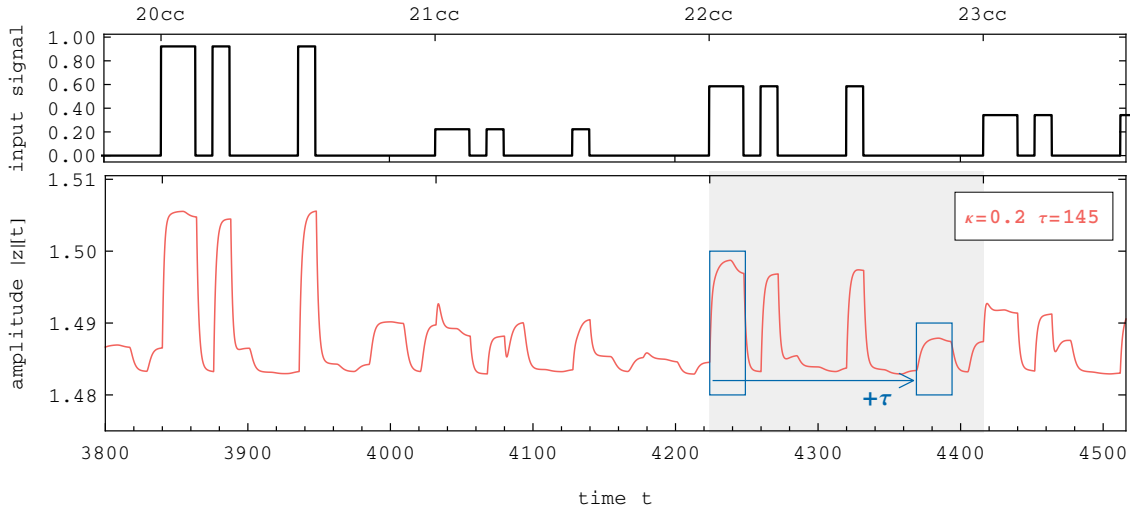

Figure 19: $|z|(t)$ of a driven Stuart-Landau oscillator with delayed feedback. With delayed feedback patterns reappear after delay time τ: the "bump" within the right blue rectangle is not caused by the input signal (above), but is an echo of the bump within the left blue rectangle. Without the delayed feedback ($\kappa = 0$) the system response can be seen in the grey area in Fig.18, which was has the same input u and parameters (besides κ).

$\mathbf{s}(t) = (s_1(t), s_2(t), \ldots, s_{N_r}(t))^T$. The state is updated through a differential equation

$$\dot{\mathbf{s}}(t) = \mathbf{f}(\mathbf{s}(\mathbf{t})) + \mathbf{W}_{in} I(t) \tag{26}$$

with $I(t)$ being an input fed to a subset of the nodes of the reservoir. The weights $\mathbf{W}_{in}$ define how much of input $I(t)$ each node is receiving.

Appeltant et al. later introduced the idea of *delay based* reservoir computing [APP11]. Simply put the idea is instead of building a physical network that has nodes distributed in space it is possible instead to use the time dimension to lay out the network. The actual network structure i.e. the thing that determines which node is connected to which is created by multiplying a mask signal $M(t)$ with the input signal $J(t)$. The mask $M(t)$ is a random but fixed periodic function that is repeating with input period T. It usually consists of N_v discrete segments of different height. N_v is the number of "virtual nodes" and is often called the virtualization factor. The length in time of each individual virtual node is called the virtual node time θ. The length in time of these segments is $\theta = T/N_v$. Early publications always used binary masks which consist of N_v segments being either 0 or 1. with binary masks the virtual nodes can be understood as to either receiving input (1) or no input (0). The current state $s(t)$ of a system is determining all states that come later. This is similar as a sort of coupling wherein earlier states influence later states. If these states are seen as discrete time intervals they can be seen as nodes. The longer the distance in time, the smaller this influence becomes which can be translated into weights that become smaller (unless the dynamical system is chaotic, which we don't want in RC). This means nodes that are far apart in time have a small coupling weight. Obviously later states cannot influence earlier states.

With the introduction of a delayed feedback we add new links to this "network". As we can choose the delay τ freely. The current state $s(t)$ is then determined not only by states in its immediate past, but also by states $s(t-\tau)$ that came before the delay τ. In oscillating dynamical systems the length of the delay τ can strongly influence how earlier

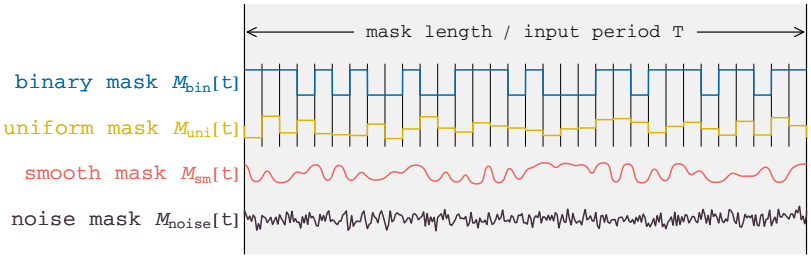

Figure 20: Different repeating functions $M(t)$ that can function as masks. Here binary and uniform masks consist of $N_v = 32$ segments each with length $\theta = T/N_v$ and constant levels. In binary masks virtual nodes have two possible levels (0 and 1) and for uniform masks they have random ones drawn from a uniform distribution. Other masks are possible, but the concept of discrete *virtual* nodes quickly breaks down here. Noise-masks are not noise in the strict sense of the work. They are only created by sampling from noise. All masks are repeating with input period T.

states are influencing later states. Depending on the exact value of τ the delayed state $s(t-\tau)$ and the current state $s(t)$ can be either in-phase, anti-phase or anywhere in between. The phase relationship of $s(t-\tau)$ and $s(t)$ is influencing how earlier perturbations $J(t-\tau)$ of the rotating wave solution are interacting with the current state $s(t)$ and/or newer perturbations $J(t)$. Where phase relationships are either in-phase or anti-phase the coupling is only perturbing the oscillator in the radial direction i.e. adding or diminishing its amplitude $|z(t)|$. Through the lens of a classical network this can be interpreted as excitatory or inhibitory coupling weights. But if the phase relationship is anywhere between the delayed perturbation is either advancing or delaying (retarding) the oscillator's phase. This perturbed phase is then later - again - changing the way delayed perturbations are influencing current states.

The fact that earlier states influence later states is (of course) an obvious statement in the context of differential equations evolving in time. It is only mentioned here, because of the analogy of a "virtual network".

The benefit of delay based RC is its simplicity. A single semiconductor laser with a delay line can act as a reservoir computer. The data and the "virtual network" is introduced into the system through the masked signal $J(t)$ that is controlling e.g. the pump current and the read-out of the data is performed through sampling at a rate that is a multiple of the input rate $1/T$. There is no need of many physical (and potentially very expensive laser-) nodes and connections between them.

In a more general description of delay based RC this "virtual network" metaphor breaks down once we use arbitrary masks. Whereas binary and uniform masks have discrete segments with distinct levels that can be called virtual nodes, continuous functions or repeating noise patterns (see Fig. 20 red and black) do not comply with this categorization. Noise masks can surpass reservoir performance of more traditional binary or random level masks [KUR18]. It is therefore a good idea to not insist on the idea of virtual nodes too much and replace the number of virtual nodes with the more general "virtualization factor" N_v. The ultimate goal of the masking process is to increase the amount of input introduced by one data point $u(t)$ and thereby increasing the phase space dimension of the current state of the reservoir. This way the introduced perturbation has more ways

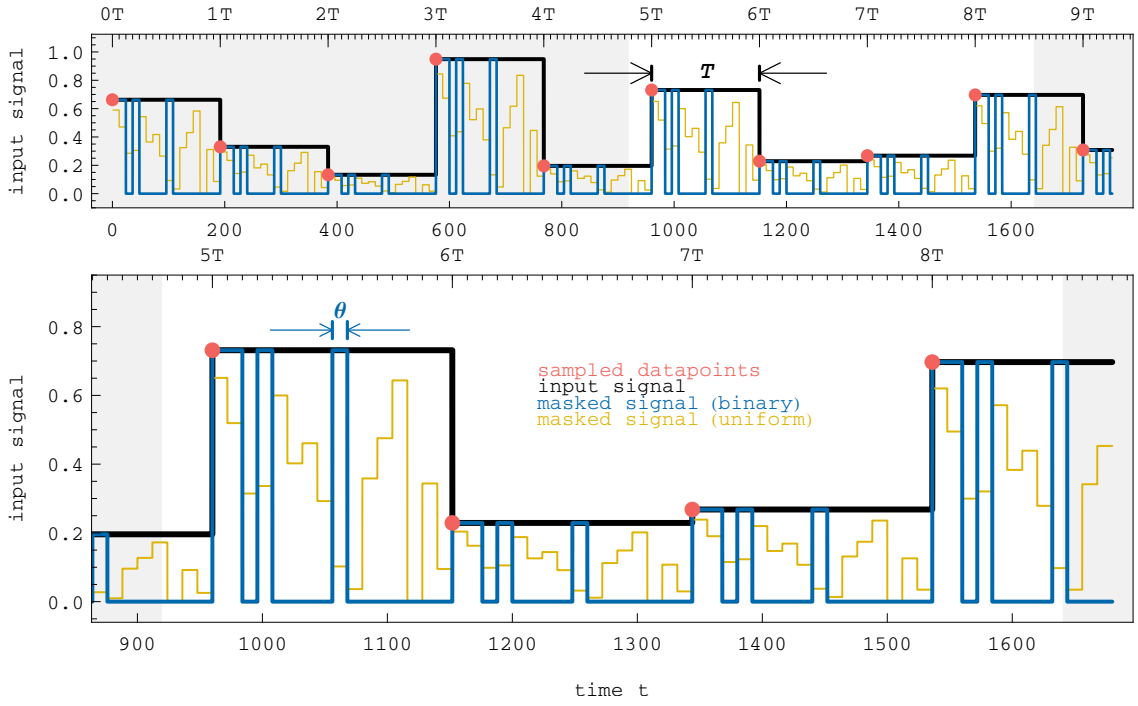

Figure 21: A series of data points ● with its constant-interpolated input signal $I(t)$ between samples ━ and the corresponding binary ━ and uniformly ━ masked signals $J_{bin}(t)$ and $J_{uni}(t)$. The mask times (or lengths) are usually defined as input periods T or clockcycle (indicated T, upper major ticks) and the time per virtual node is called θ (upper minor ticks). Here $\theta = 12$ and $T = 16\theta = 272$

of interacting with previously introduced perturbations which results in a richer dynamic response and (possibly) more useful nonlinear responses.

In this work uniform masks (random level masks) were used at all times. The masking process is shown with more detail in Fig. 21: There we start with a sequence of data points $\mathbf{u}$ which are symbolized by one red dot ● for every new time T. Via constant interpolation (i.e. "sample-and-hold") we gain a continuous input function $I(t)$. By multiplying the input $I(t)$ with the T-periodic mask $M(t)$ we obtain the masked signal $J(t)$ that is used in 25.

3.9 Combining real networks with virtual networks

In this work the older concept of networks that consist of N_r real nodes has been combined with the delay based approach as was done in [ROE18b, ALB19]. The instantaneous links between nodes are now delayed and the mask $M(t)$ is replaced by a set of masks $\mathbf{M}(t) = (M_1(t), M_2(t), \dots), M_{N_r}(t))$ that are different at every node. The (full) read-out dimension is thus $N_r \times N_v$ because the states $s_i(t)$ are being read out N_v times at each of the N_r nodes. This way each node is given the same input, but in a different way which results in a different response. With the delay lines the individual oscillators are coupled to each other. In most calculations the virtualization factor was set to a relatively small value of $N_v = 16$. As this work deals with networks of oscillators the overall number of read-outs is larger. In Fig. 23 a typical timeseries of a network of delay-coupled oscillators is shown with the read-out mechanism visualized. On the left is are the 4 timeseries over a few input periods T (here marked as "cc" / clockcycles) and on the right one input period

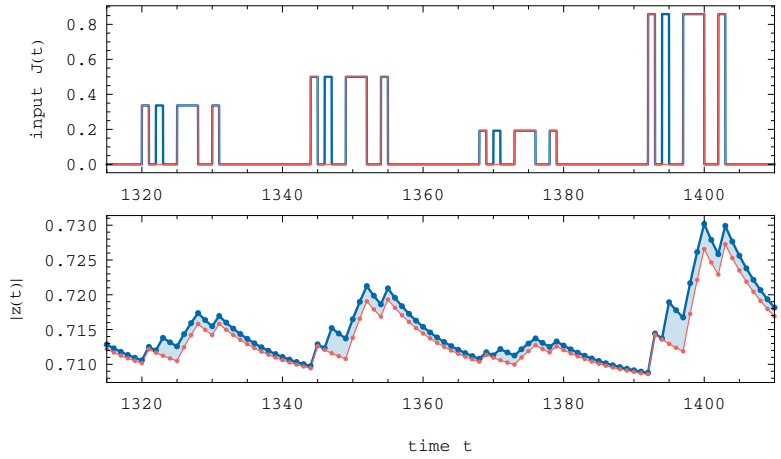

Figure 22: Two different masks with their respective response functions and . The masks differ in only one single step (i.e. at one virtual node). But the resulting difference in response (light blue area) shows us, that this small difference persists much longer than the node itself. All states after this virtual node are influenced by this node. When regarding the system's states through discrete read-outs (and) it is easier to understand the virtual network analogy.

is exploded with all the read-outs shown as dots. The virtualization factor is $N_v = 32$. Thus there are 32 dots for every one of the 4 timeseries at which the state is read-out and saved in the statematrix $\mathbf{S}$. In (28) the process of saving inputs in the statematrix is shown with colors according to the example in Fig. 23. The individual read-outs that are taken at one oscillator and one virtual node position (27) are symbolized as dots.

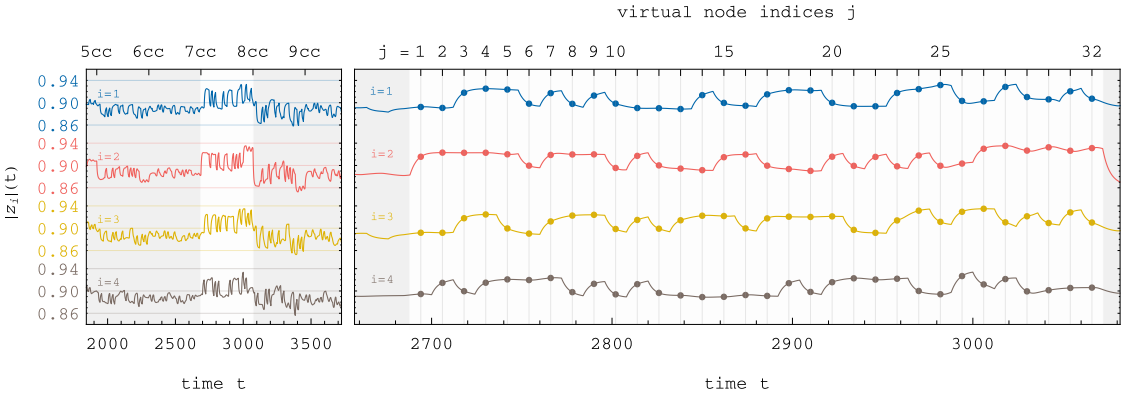

Figure 23: The timeseries $|z_i|(t)$ of 4 nodes of a system with $N_r = 4$ nodes and virtualization factor $N_v = 32$. Input period of the 7th input sample is indicated by the white background and the corresponding readouts $\mathbf{x}(7T)$ are indicated by the small dots on the right. All dots together make one row in the state matrix $\mathbf{S}$ (see Fig. 29).

$$x_{ij}^k = |z_i(kT + j\theta)| \tag{27}$$

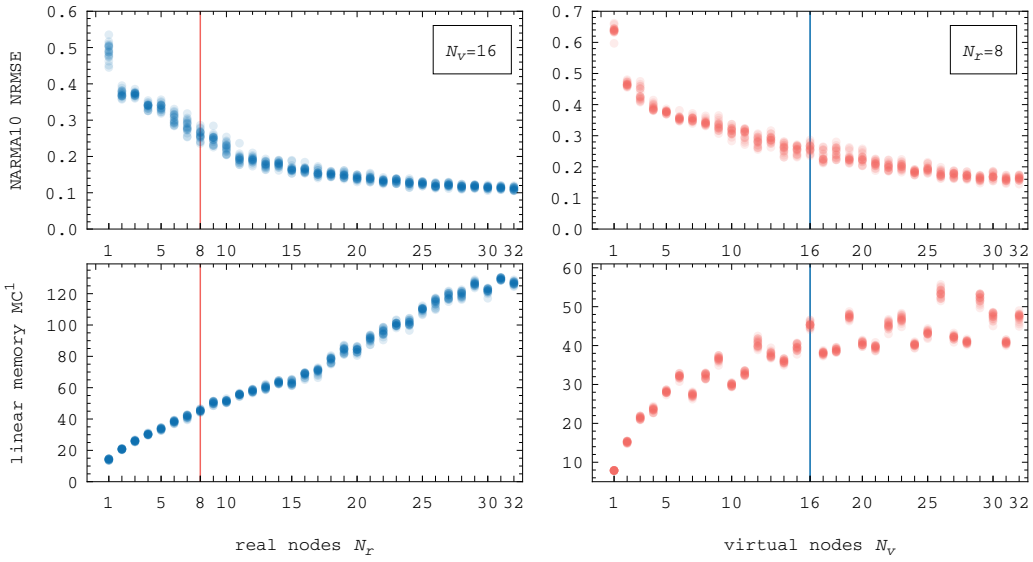

Figure 24: The changing RC performances in unidirectional rings with varying number of nodes N_r and virtualization factor N_v and parameters $\lambda = -0.09, \omega = 1, \gamma = -0.1, \kappa = -0.1, \phi = 0, \tau = 272$. (left) $N_r = \{1 - 32\}$ is plotted with $N_v = 16$ constant. (right) $N_r = 8$ is constant while $N_v = \{1 - 32\}$. Shown are results for 20 different masks each.

$$\mathbf{S} = \begin{pmatrix} \bullet\,{}^{1}_{1} & \bullet\,{}^{1}_{1} & \bullet\,{}^{1}_{1} & \bullet\,{}^{1}_{1} & \bullet\,{}^{1}_{2} & \bullet\,{}^{1}_{2} & \bullet\,{}^{1}_{2} & \bullet\,{}^{1}_{2} & \cdots & \bullet\,{}^{1}_{N_v} & \bullet\,{}^{1}_{N_v} & \bullet\,{}^{1}_{N_v} & \bullet\,{}^{1}_{N_v} \\ \bullet\,{}^{2}_{1} & \bullet\,{}^{2}_{1} & \bullet\,{}^{2}_{1} & \bullet\,{}^{2}_{1} & \bullet\,{}^{2}_{2} & \bullet\,{}^{2}_{2} & \bullet\,{}^{2}_{2} & \bullet\,{}^{2}_{2} & \cdots & \bullet\,{}^{2}_{N_v} & \bullet\,{}^{2}_{N_v} & \bullet\,{}^{2}_{N_v} & \bullet\,{}^{2}_{N_v} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots \\ \bullet\,{}^{k}_{1} & \bullet\,{}^{k}_{1} & \bullet\,{}^{k}_{1} & \bullet\,{}^{k}_{1} & \bullet\,{}^{k}_{2} & \bullet\,{}^{k}_{2} & \bullet\,{}^{k}_{2} & \bullet\,{}^{k}_{2} & \cdots & \bullet\,{}^{k}_{N_v} & \bullet\,{}^{k}_{N_v} & \bullet\,{}^{k}_{N_v} & \bullet\,{}^{k}_{N_v} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots \\ \bullet\,{}^{K}_{1} & \bullet\,{}^{K}_{1} & \bullet\,{}^{K}_{1} & \bullet\,{}^{K}_{1} & \bullet\,{}^{K}_{2} & \bullet\,{}^{K}_{2} & \bullet\,{}^{K}_{2} & \bullet\,{}^{K}_{2} & \cdots & \bullet\,{}^{K}_{N_v} & \bullet\,{}^{K}_{N_v} & \bullet\,{}^{K}_{N_v} & \bullet\,{}^{K}_{N_v} \end{pmatrix} \tag{28}$$

The read-outs x^k_{ij} are performed during the numerical integration of $\mathbf{z}(\mathbf{t})$ and saved in the statematrix $\mathbf{S}$. In a real experimental experiments they could be performed by sampling 4 lasers 32 times pare input period. The nodes $i = 1, 2, 3, 4$ (●●●●) are colored according to the example in Fig.23. Row k of $\mathbf{S}$ contains all readouts x for the input period k with $k \in [1, K]$. Each column corresponds to one virtual node of one real node.

Increasing the number of nodes N_r and/or virtualization factor N_v increases the reservoir performance. In the case of N_v the performance seems to saturate more quickly though (see Fig. 24 bottom right/red.) A more thorough investigation (of not only this aspect) was also conducted by Röhm et al. in [ROE19]. Increasing N_r yields improvements of the linear memory capacity MD^1 for a longer period of time. Therefor it seems appropriate to combine the benefit of both approaches.

3.10 Measuring capacities at individual nodes

During integration the read-out works by writing for each training step k the amplitude $|z|$ once per real and virtual node into the k-th row of the statematrix $\mathbf{S}$. Using the example 23 and keeping the colors for each real node, the read-outs are saved in the matrix as can be seen in 28. Each column of $\mathbf{S}$ represents the reservoirs dynamic response to input sample k with $k \in [1 - K]$. By only taking certain columns we can simulate a different

read-out procedure without having to integrate the whole system again. By using the "sub-statematrix" $\mathbf{S}_{\bullet}$ we can calculate the capacities with only one node $\bullet$. By using the complement $\mathbf{S}_{\neg\bullet}$ we can also test capacities *without* read-outs from node $\bullet$. We gain the new sub-statematrix $\mathbf{S}_{\bullet}$ simply through selection of only readouts from node $\bullet$. By omitting readouts from $\bullet$ (here $i = 2$) we get the complement $\mathbf{S}_{\neg\bullet}$.

$$\mathbf{S}_{\bullet} = \begin{pmatrix} \bullet^{1}_{1} & \bullet^{1}_{2} & \dots & \bullet^{1}_{N_v} \\ \bullet^{2}_{1} & \bullet^{2}_{2} & \dots & \bullet^{2}_{N_v} \\ \vdots & \vdots & & \vdots \\ \bullet^{k}_{1} & \bullet^{k}_{2} & \dots & \bullet^{k}_{N_v} \\ \vdots & \vdots & & \vdots \\ \bullet^{K}_{1} & \bullet^{K}_{2} & \dots & \bullet^{K}_{N_v} \end{pmatrix} \quad \mathbf{S}_{\neg\bullet} = \begin{pmatrix} \bullet^{1}_{1} & \bullet^{1}_{1} & \bullet^{1}_{1} & \bullet^{1}_{2} & \bullet^{1}_{2} & \bullet^{1}_{2} & \dots & \bullet^{1}_{N_v} & \bullet^{1}_{N_v} & \bullet^{1}_{N_v} \\ \bullet^{2}_{1} & \bullet^{2}_{1} & \bullet^{2}_{1} & \bullet^{2}_{2} & \bullet^{2}_{2} & \bullet^{2}_{2} & \dots & \bullet^{2}_{N_v} & \bullet^{2}_{N_v} & \bullet^{2}_{N_v} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ \bullet^{k}_{1} & \bullet^{k}_{1} & \bullet^{k}_{1} & \bullet^{k}_{2} & \bullet^{k}_{2} & \bullet^{k}_{2} & \dots & \bullet^{k}_{N_v} & \bullet^{k}_{N_v} & \bullet^{k}_{N_v} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ \bullet^{K}_{1} & \bullet^{K}_{1} & \bullet^{K}_{1} & \bullet^{K}_{2} & \bullet^{K}_{2} & \bullet^{K}_{2} & \dots & \bullet^{K}_{N_v} & \bullet^{K}_{N_v} & \bullet^{K}_{N_v} \end{pmatrix} \tag{29}$$

In order to understand reservoir computing properties of networks of delay-coupled oscillators this work also attempted to measure capacities on a per-node basis. The read-out mechanism for delay-based reservoir computing works by measuring $|z|$ once per virtual node at each real node. The full read-out state $\mathbf{x}$ has a dimension of $D = N_r \times N_v$. In an experimental setup with laser-systems this would mean to measure and sample $|z|$ at each node of the network once per virtual node time θ. But it is possible to only measure at a single node or a subset of all nodes, determine $\mathbf{w}$ for the linear transformation and compare reservoir performances. By omitting certain nodes in the sub-statematrix $\mathbf{S}_{\neg\bullet}$ and calculating certain capacities $C_{\neg\bullet}$ we can through comparison with C (the capacity calculated by using the full read-out) calculate the relative importance of a node $\bullet$ by comparing these two capacities (not shown in this work). Similarly we can calculate individual memory capacities $MC_1^d, MC_2^d, \dots$ that only use read-outs from individual nodes and visualize the amount of abundant information and the amount of gained capacity by using all read-outs.

In Fig. 25 a simple example i shown: We take a uni-directional ring with $N_r = 8$ and virtualization factor $N_v = 16$ and by using each sub-statematrix $\mathbf{S}_1, \mathbf{S}_2, \dots, \mathbf{S}_8$ we calculate the linear memory capacities $C^1[\mathbf{S}_1], C^1[\mathbf{S}_2], \dots, C^1[\mathbf{S}_8]$ (the linear memory capacities gained by only using readouts from node $1, 2, \dots 8$ hinted in black). Expectedly they vary a little from node to node due to the varying masks $M_i(t)$, but are overall fairly similar in scale and area of drop-off. Taking the minimum value for each recall step n we can calculate the "mutual information" (yellow) which is present in every node. The red curve is the best result possible from all individual capacities. In blue we can see the result of taking the read-outs of all 8 nodes into account. The difference between the best individual results and the result of the full read-out is visualized in light blue. In this example the additional information gained by combining all read-outs i.e. the light blue area compared to the area under the red curve is significant. A different way of describing this result would be that "the sum is greater than the parts".

3.11 Implementation details

In order to save time some adjustments were made where possible. The NARMA10 task uses inputs u_n drawn from a uniform distribution $U \in [0, 0.5]$ to calculate sequence members A_n. The (non-)linear memory capacities gained through the Legendre polynomials rely on inputs u drawn from a symmetric unform distribution $U \in [-1, 1]$. To avoid processing two sets of calculations - one with NARMA10 and one with the Legendre task

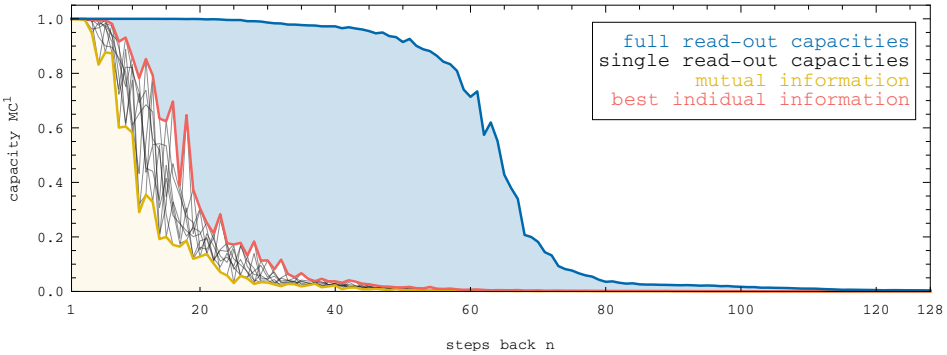

Figure 25: A unidirectional ring with $N_r = 8$, $N_v = 16$, $\theta = 12$, $G = 0.001$, $\kappa = 0.1$, $\phi = 0.2\pi$, $\tau = 272$, $\lambda = -0.0975$, $\omega = 1$, $\gamma = -0.1$. Using the full read-out we get the best linear memory capacities in blue. The yellow area is the minimum of all per-node capacities (gained through sub-statematrices $\mathbf{S}_1, \mathbf{S}_2, \dots$) at recall step n. It is the mutual information present in all 8 nodes. The faint black traces are the individual per-node capacities.

both tasks were made using the same sequence $u \in [-1, 1]$. For NARMA10 the sequence u was mapped with the function m (30) onto $m(u)$.

$$m : [-1, 1] \to [0, 0.5] : u \to (u + 1)/4 \tag{30}$$

3.11.1 Numerical Integration

$$\dot{z}(t) = f(t, z(t), z(t - \tau)) \tag{31}$$

$$\begin{aligned} k_1 &= f(t, z(t), z(t - \tau))) \\ k_2 &= f(t + \tfrac{h}{2}, z(t) + h\tfrac{k_1}{2}, z(t - \tau + \tfrac{h}{2})) \qquad (32) \\ k_3 &= f(t + \tfrac{h}{2}, z(t) + h\tfrac{k_2}{2}, z(t - \tau + \tfrac{h}{2})) \qquad (33) \\ k_4 &= f(t + h, z(t) + k_3, z(t - \tau + h)) \end{aligned}$$

$$z(t + h) = \tfrac{1}{6}(k_1 + 2k_2 + 2k_3 + 2k_4)$$

Integration was done through the Runge-Kutta method of fourth degree RK4 with a fixed stepsize h of usually $1/64$ (sometimes smaller: $h = 1/100, h = 1/128, h = 1/200, h = 1/256, \dots$ in order to confirm validity of results). Since we are dealing with delay differential equations, we run into a problem as the delayed state $z(t - \tau)$ is only available as discrete values that are intervals h apart. The Runge-Kutta steps k_2 and k_3 are evaluated at times of $\frac{h}{2}$ (see Eq. (32),(33)). Thus an interpolated value for $z(t - \tau + \frac{h}{2}))$ has to be approximated from the existing values $z(t - \tau)$ and $z(t - \tau + h)$. Linear interpolation should be avoided as the error resulting from the poor interpolation method would be of higher order than the error of the fourth order Runge-Kutta method RK4. Instead Hermite-interpolation was used which takes into account also the derivates $\dot{z}(t - \tau)$ and $\dot{z}(t - \tau + h)$. This means we have to save not only past states z but also derivatives $\dot{z}$. The implementation was done in **C++** where double-precision was used throughout.

3.11.2 Capacities

In order to calculate the capacities the linear algebra library **Armadillo** was used which provides the ability to use vectors and matrices together with vector and matrix operations that were needed (multiplication, inversion for e.g. the Moore-Penrose-Pseudoinverse). The NARMA10 NRMSEs are fast to calculate. For the capacities of Legendre-tasks with higher degrees it was necessary to implement a break-condition based on the observation that memory capacities $MC^d_{\{n_1,n_2,\dots\}} \to 0$ for any combination of sufficiently large n_i.

For the NARMA10 task the (sub-)statematrices were augmented with an additional column filled with 1s (34). This additional column is representing the bias term which is utilized in other implementations of this benchmark. It is not explicitly needed in most cases, but it was still used in order to follow convention.

$$S_{bias} = \begin{pmatrix} x_1^1 & \dots & x_1^D & 1 \\ \vdots & & \vdots & \vdots \\ x_K^1 & \dots & x_K^D & 1 \end{pmatrix} \tag{34}$$

4 Results

4.1 Multiple self-linked nodes

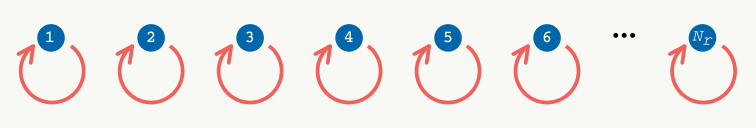

Figure 26: Combining readouts from individual nodes $i = \{1, 2, \ldots, N_r\}$ that have only self-loops, but receive input through different masks $M_i(t)$ results in better reservoir performance than only one node. The system is able to combine the responses that are different at each node

For the simplest case (as a reminder) the impact of input period T and delay τ is shown in Fig.27. There are lines of bad performance whenever τ and T form a simple relationship e.g. $\tau = \{T, 2T, \frac{3}{2}T, \frac{2}{3}T, \frac{4}{3}T, ...\}$ – a rational number that can be expressed as a fraction of (small) natural numbers. This phenomenon has appeared in other investigations. Recently this phenomenon has been investigated more thoroughly by Stelzer et. al. [STE20]. It is therefore prudent to choose "off-resonant" values for τ and T in order to not avoid these creases of bad performances. In this work $\tau = \frac{17}{12}T$ has been used often as a compromise for being fairly off-resonant but easy to remember. It was also already used in [ROE18b]. For $\theta = 3, N_v = 16$ this results in $T = 48$ and $\tau = 68$, for $\theta = 12, N_v = 16$ it results in $T = 192$ and $\tau = 272$. Linear memory capacities mostly inhabit areas where $\tau > T$ whereas memory capacities of higher degree MD^2 and MD^3 inhabit areas of shorter delay relative to the input period. The higher the degree d of the total capacity MD^d the shorter the delay usually is [KOE20a]. In this work only degrees up to $d = 3$ have been calculated. The reason is that with higher degrees the number of combined Legendre-tasks $L_{\{d\}}(u_{\{n\}})$ that need to be considered becomes *very* large. This makes MD^d the sum over a very large number of very tiny capacities. In order to distinguish real capacities from noise the input sequence $\mathbf{u}$ has to be very long i.e. K has to be very high. For individual nodes i.e. $N_r = 1$ this can be done easily, but for larger networks it becomes very time-consuming. So this work has stopped at degree $d = 3$.

In order to understand dynamical systems of interconnected nodes we look first at the opposite kind consisting only of self-coupled nodes (Fig. 26). As every node i receives input through a different mask $M_i(t)$ the dynamical response is different for each node. Increasing N_r enables us to exploit a greater number of readouts x and thus a greater variety of nonlinear transformations in order to reconstruct target values. The results were gained by calculating each task from statematrix $\mathbf{S}_{1-N_r}$ (the sub-statematrix that contains all columns associated with nodes 1 through N_r). This way only one simulation of the dynamical system had to be performed with the maximum number $N_{r,max}$. Expectedly the memory capacities MD^1, MD^2, MD^3 increase with the number of real nodes N_r, although the capacities quickly saturate against a threshold (Fig. 28, left). If we also randomize the delay $\tau_{i \leftarrow i}$ at each node i the capacities are increasing much longer. A likely explanation is a greater variety of the nodes due to different masks *and* different delays. This result is also shown in [SUG20], where the reservoir performance of a growing number of individually self-coupled Lang-Kobayashi-Lasers [ALS96] was investigated (among others). A greater variety among these individual nodes increased RC performance similarly as it did here.
Another interesting observation: the total cubic memory MD^3 stays very low for the first

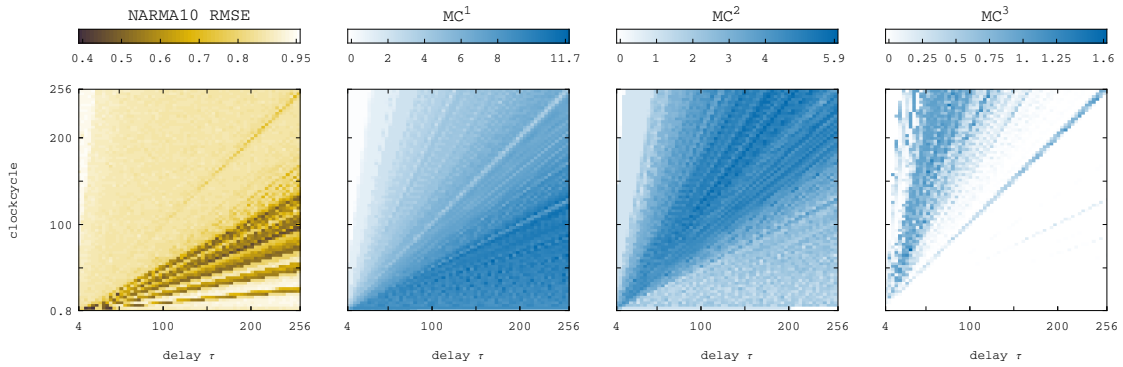

Figure 27: Single node $N_r = 1, \lambda = 0.02, \kappa = 0.1, \phi = 0$. NARMA10 errors and memory capacities MC^1, MC^2, MC^3 for varying values for τ and T. If τ/T is a fraction of small natural numbers the memory capacity decreases. This decrease is visible in the lines from the origin of the plot.

few nodes before it is starting to slowly increase with N_r in an upward bend (before slowing down again). This seems to suggest a minimum of nodes (or readout dimension D) necessary in order to reconstruct target functions of high combined degree d.

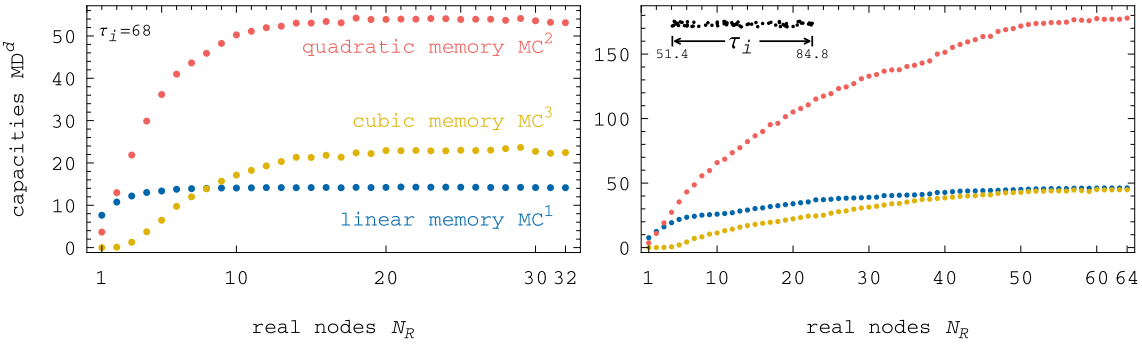

Figure 28: Capacities MD^1 ●, MD^2 ●, MD^3 ● with a growing number $N_r = \{1 - 64\}$ of individual self-coupled nodes (see Fig.26). (left) with all nodes having the same delay $\tau_{i \leftarrow i} = 68$ the total capacities are quickly saturated. (right) when each node has a different delay $\tau_{i \leftarrow i} = 68(1 + u)$ with $u \in [-1/4, 1/4]$ the capacities are increasing longer towards a higher total value.

This simple example suggests that variety in the dynamics of individual nodes in a network increases reservoir performance, as the whole system can exploit a greater number of responses in order to reconstruct a desired target function.

4.2 Unidirectional non-recurrent sequentially coupled nodes / chains

In order to understand the propagation and processing of an initial input perturbation it is a good idea to first investigate networks without or only few recurrent links and restrict input $J(t)$ to a few nodes. Such a unidirectional chain network is similar to a uni-directional ring, but lacks an edge from last to first node, thereby removing the possibility of inputs propagating in closed circles. The general input scaling was set to $G = 0$ except at the first node where it was set to $G_1 > 0$. Therefor only the first node of the chain is perturbed with the masked input signal $J(t)$. This perturbation propagates from

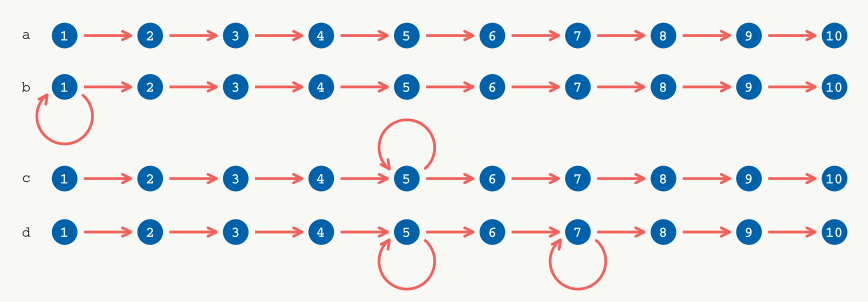

Figure 29: A few simple examples of small unidirectional chain networks with $N_r = 10$. a) has has no additional recurrent links, b) has an additional self-link $1 \to 1$, c) $5 \to 5$ and d) has two self-links $5 \to 5$ and $7 \to 7$.

node to node until it arrives at the last node or becomes too small to be useful (see Fig. 30). (In an experimental setting the signal might be considered too small if its signal to noise ratio becomes too small. In a numerical context the signal becomes too small when it approaches the limit of numerical precision). Of course these non-recurrent types of networks are less practical as reservoirs than recurrent networks as information quickly evaporates if it cannot be fed back into it. We are also more restricted in the set of parameters: we must set $\lambda > 0$ otherwise the first node with no feedback will decay towards the off-state (see Fig. 10 left with $\lambda < 0$) which will in turn result in all subsequent nodes being in the off-state as well.

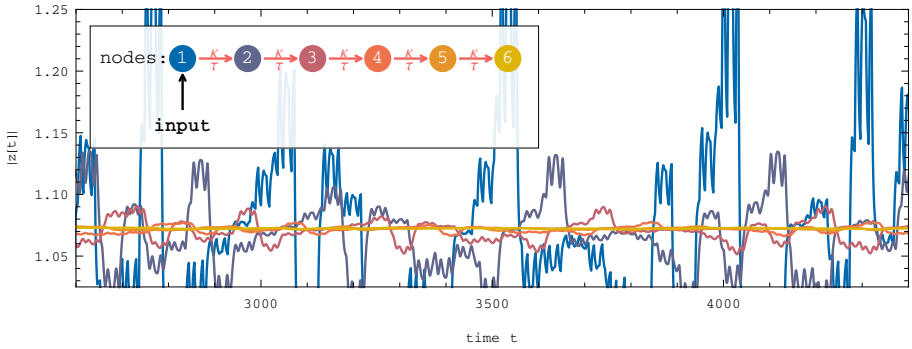

Figure 30: a small unidirectional chain network with $N_r = 6$ nodes that gets input at only the first node ●. As the perturbation propagates from the first node ● to the last node ● it exponentially decays in size and at the same time smoothes out. Here the coupling strength is $\kappa = 0.1$ but even for greater κ where the perturbation decays slower the smoothing remains.

In Fig. 29**a** we can see a simple unidirectional chain. Each edge has coupling parameters κ, ϕ and τ. For links that only branch off the coupling phase ϕ is usually not important. Only for nodes that receive input from more than one node these inputs can interact with each other and therefor the coupling phases of those two links can be important.

A longer unidirectional chain fed input only at the first node can show how information is propagating from node to node in Fig. 31. In order to increase travel-length the coupling strength has been increased to $\kappa = 1.0$. Otherwise the perturbation quickly decays. The delay from node to node was chosen to be equal to the input period $\tau = T$. This means that the perturbation propagates as fast as new inputs are fed into the network,

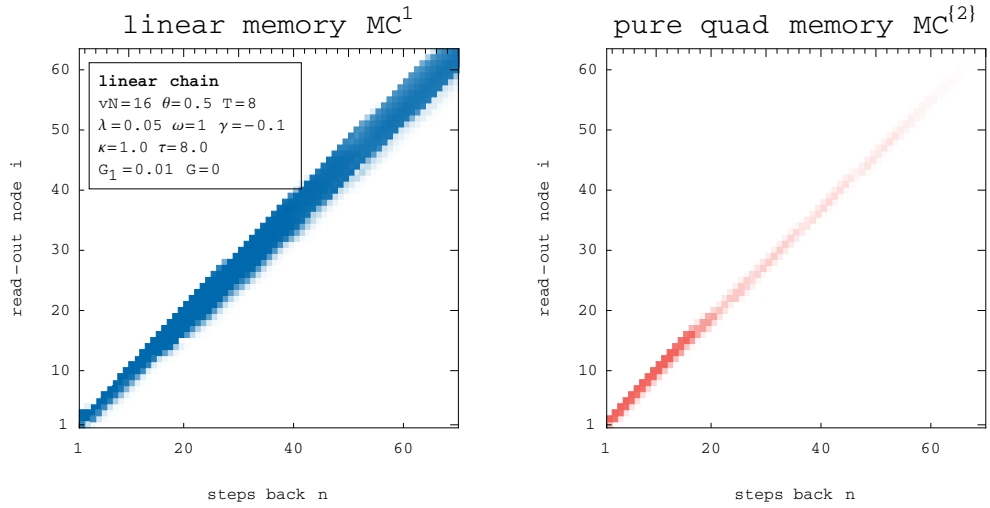

Figure 31: Linear (left, blue) and pure quadratic (right, red) per-node memory capacities in a simple unidirectional chain network with $N_r = 64$. The darkest shade of blue and red denotes a capacity of 1, white denotes 0. As the perturbation created in the first node is propagating from node to node so does the capacity to reconstruct past input. It is also slightly lacking behind in later nodes although the input period T and delay τ are identical. This is likely due to the effect shown in Fig. 13 (top). The information imprint also diffuses which makes it possible for later nodes to reconstruct more than one previous input sample. This can be seen in the widening of the blue band of high linear memory capacity. Interestingly the pure quadratic memories $MC^{\{2\}}$ are not widening and are quickly fading towards 0 for.

because $\tau = T$. Interestingly the purely quadratic memory capacities per node (red on the right) seem to only be present at the moment the perturbation is introduced by the coupling (pure quadratic memory is constrained into a narrow band). This means that the perturbation can only be used in order to predict the values of $L_2(u_{-n})$) with n depending on when the perturbation is reaching a node. This seems to hint that the parts of the perturbation that carry information about $L_2(u_{-n})$) are created at the moment the perturbation is fed into the next node and then quickly decay. At least the parts that can be used from readouts at only one node.

If we add an additional self-loop (recurrent loop) similar to Fig:29 (**c**) the perturbation is reintroduced periodically at this node. Consequently each reappearance of this distinctive perturbation is propagating through all following nodes as well. This addition of linear memory capacities can be seen as additional bands of memory capacities in 32 compared to the single band in 31. An interesting effect can be seen when the imaginary part of $\gamma \neq 0$. In Fig:33 the bands are longer visible. A possible explanation is that the amplitude-phase coupling which lets the input signal $J(t)$ not only effect amplitude, but phase velocity as well. Phase-differences and phase-synchronization are phenomenon emergent between oscillators in the network, whereas radial perturbations relative to a limit cycle are local dynamics. As a result it is expected that any information encoded in the phases of the oscillators will likely persist longer. Additionally information encoded in the phase is not recognized directly in the read-out state, because it only takes into account the amplitudes of each oscillator. Instead the phase position is determining how a node is reacting to new input.

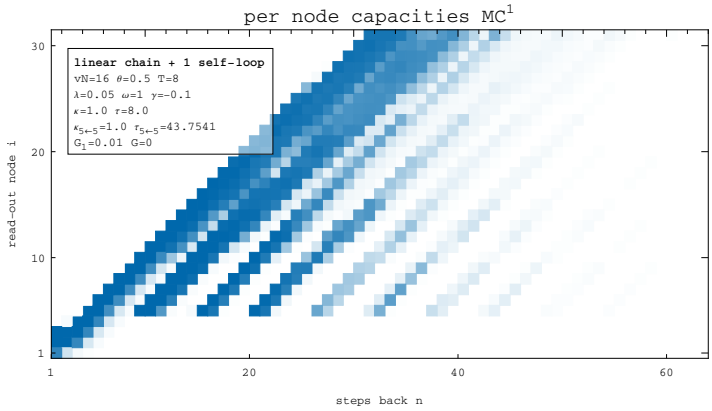

Figure 32: A unidirectional chain network with $N_r = 32$ nodes that has a single recurrent link at node 5. The additional delay $\tau_{5\leftarrow 5} = 43.7541$ was chosen not to be an integer multiple of the input period $T = 8$.

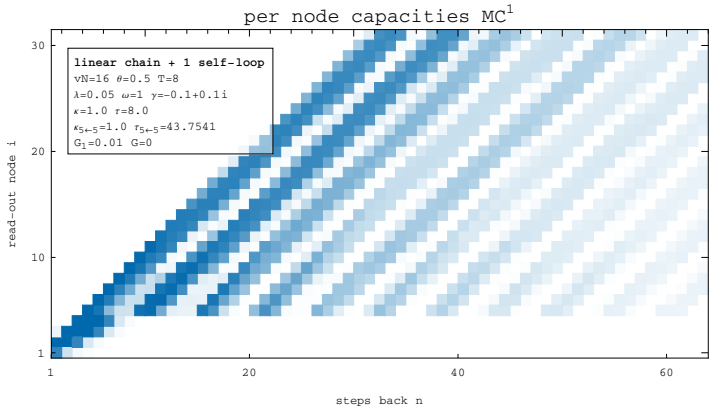

Figure 33: A unidirectional chain network with $N_r = 32$ nodes that has a single recurrent link at node 5. The same network as in 32 but with a complex γ. The added amplitude-phase interaction seems to enhance linear memory as the bands are more visible.

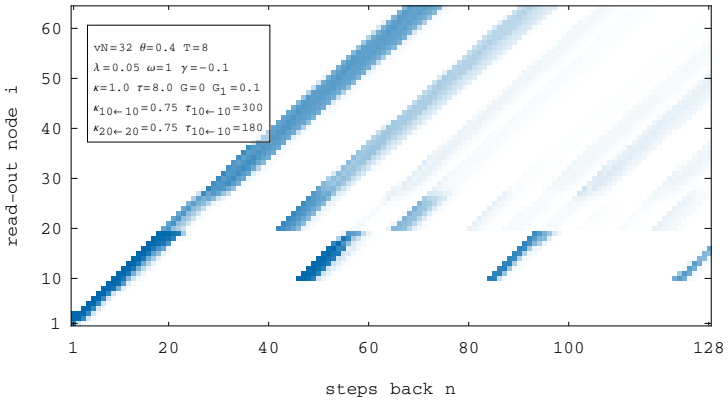

Figure 34: A long unidirectional chain network with $N_r = 64$ nodes and two recurrent links at nodes 10 and 20 with long respective delays $\tau_{10\leftarrow 10} = 300$ and $\tau_{20\leftarrow 20} = 180$. The recurrent delays are much longer than the delays from subsequent nodes $\tau = 8$. The result is that information can be reintroduced into the system that otherwise would already have disappeared completely from the particular node (or the whole system altogether). The information can reappear at node 10 with delay $\tau_{10\leftarrow 10}$ and at 20 with $\tau_{20\leftarrow 20}$. The additional "recallability" introduced at node 10 is also recurring at node 20 with delay $\tau_{20\leftarrow 20}$. This means that information (e.g. the respective perturbations) is in a way mixing and interacting with each other.

In Fig. 34 the addition of 2 recurrent links to a linear chain demonstrates the multiplicative aspect of recurrent edges. Instead of simply propagating as seen in Fig. 31 the perturbation is reappearing at 2 different nodes (10 and 20). At node 10 it reappears after (about) integer multiples of $\tau_{10\leftarrow 10}$. But at node 20 the previous inputs are not only recurring with (about) integer multiples of $\tau_{20\leftarrow 20}$, but also the with combinations of both delays (e.g. $a\,\tau_{20\leftarrow 20} + b\,\tau_{20\leftarrow 20}$ with $a, b \in \mathbb{N}$). These bands are of course very faint in this calculation. They can be enhanced with a higher virtualization factor N_v at the cost of higher calculation time(s). As an analogy for this phenomenon, we can think of the first recurrent link making (delayed) copies of the perturbations. Afterwards second recurrent link also copies the copies. Every copy is an additional perturbation that interacts with a newer perturbation. The multiplicative nature of these recurrent links means that with every new recurrent link the amount of possible interactions increases dramatically. At the same time though these additional perturbations which result in subsequent nodes gaining (some) additional recall capacities also seem to suppress certain capacities. In Fig. 34 it is easy to see that additional recallability-bands introduced at node 10 suddenly disappear again almost entirely at node 20 (where the second recurrent link is positioned). Also worth mentioning is the slight drop of memory capacity in the primary band (the band which we would expect in network without any recurrent links e.g. Fig. 31), immediately after node 20. These linear chains already show that even when we restrict the points at which inputs (perturbations) are introduced and can interact with each other in a network the results are highly already complex and should probably be investigated further.

4.2.1 Simple rings and all-to-all connected topologies

In unidirectional ring networks the dynamical state of the system can have an impact on the reservoir capacities. In a network of $N_r = 8$ nodes the synchronized state and splay-state move linear memory capacities toward areas of shorter delays relative to the input period T. In Fig. 35 (bottom) the splay state (prepared with adjacent oscillators' phases $2\pi/N_r$ apart plus a tiny amount of randomness) results in more area of high the

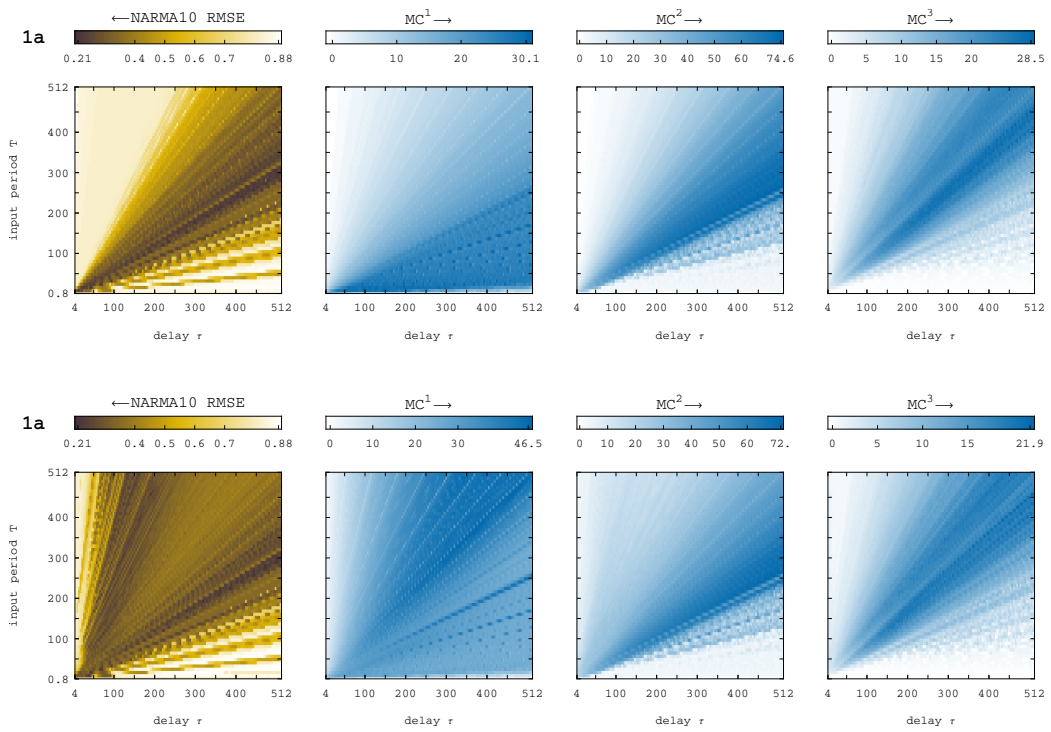

Figure 35: Several capacities in a unidirectional ring network $N_r = 8$. Parameters $\lambda = -0.05$, $\omega = 0$. Capacities over delay τ and input period T. (top) synchronized state, (bottom) splay state. Depending on the state the capacities are different. Capacities of linear or nonlinear memories usually decrease if τ/T is a natural number or a fraction with a low denominator.

linear memory capacity as well as a higher maximum compared with the synchronized state. (top) all oscillators are in a synchronous state. The nonlinear capacity MC^2 is only slightly worse, but MC^3 loses almost 25% of its best value compared to the synchronized state. In general the plot for the synchronized state (top) looks similar to the single node (in Fig. 27 besides the higher absolute capacities due to the smaller readout dimension D). The splay-state is only possible for the ring network with $N_r > 1$. It shall be noted though, that for the calculation in Fig. 35 the intrinsic phase velocity was set to $\omega = 0$ in order to avoid artifacts resulting from interplay of ω and delay time τ. As the linear memory capacity was higher in a greater area with the splay states and the also low NARMA10 NRMSEs occurred in a larger area the following calculations were initialized to prefer a splay state (with $m = 1$ and amplitude(s) of 1 and slight random offset of magnitude 0.01). The delayed state was set to zero.

A bigger focus of this work though was the dependency of the capacities on the parameter λ and the coupling phase ϕ. In Fig. 36 the dependencies of the different capacities in a simple unidirectional ring with $N_r = 8$ are shown. The difference between normal and generalized order parameter shows where a bifurcation between synchronized and splay-state is occurring. This bifurcation is also traceable in the reservoir performances. Here the linear and quadratic memory capacity increase when they approach the bifurcation (from the areas of synchrony) while capacities MD^3 seem to decrease. Phase-wise MD^1 is dropping for a rather narrow band ($\phi \approx 0.8125$), But increasing for higher values *lambda*.

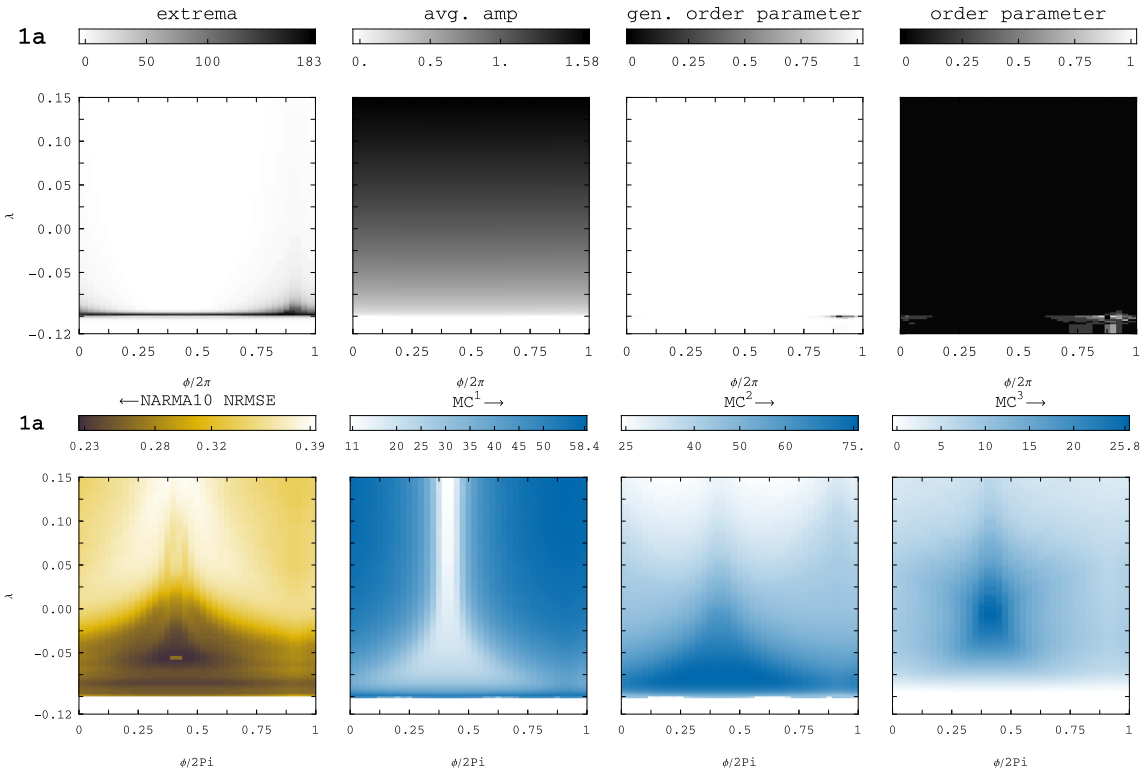

Figure 36: $N_r = 8$ unidirectional ring (**1a**). NARMA10 error, memory capacities MC^d and dynamical indicators over parameters λ and coupling phase ϕ with $\tau = 272$ $\theta = 12$ $\kappa = 0.1, \omega = 1, \gamma = -0.1$. Arrows indicate greater performance. Areas of high memory capacities MD^d are different depending on degree d. Linear Memory is high near the Hopf-bifurcation at $\lambda \approx -0.095$ where the coupling phase seems to have little impact. For higher values λ another area of high capacity exists. Here the coupling phase ϕ is important as for certain values the capacity drops significantly. The quadratic memory seems to be inversely correlated with the linear memory. Being high in places where linear memory is low and vice versa (although slightly shifted toward lower λ). The highest values for the cubic memory are forming a more isolated round patch. It appear that the higher the combined degree d, the further away we have to be from bifurcations. The NARMA10 NRMSE shows a lot more fine structure due to the fact that it is a more narrow task performance opposed to the more general combined quantities $MD^1, MD^2 and MD^3$ which are the accumulations of many task performances.

The nonlinear memories (especially MD^3) seems to be higher in areas where MD^1 is low. This can be regarded as another manifestation of the trade-off between linear memory and nonlinear transformations that was also investigated in [DAM12]. The cubic memory seems to be most concentrated in a single area. At least the shape is simplest to describe as resembling a drop (or an isolated blob).

Dynamics and RC performance of the bidirectional ring (**2a**) in Fig. 37. Though more complicated in structure overall RC performance goes down for MD^1 and MD^3, stays approximately the same for MD^2. Varying coupling phases ϕ seem to not affect RC performance considerably. Values $\lambda > -0.04$ result in a splay state which is visible in the difference in genaralized and normal order parameters. Values $-0.095 < \lambda < -0.04$ result in a state strongly determined by the initial condition (which here was a splay state j=1 with a random but fixed offset for each oscillator of $\delta = 0.01$).

The complete graph in Fig. 38 had the poorest performance of all tested networks

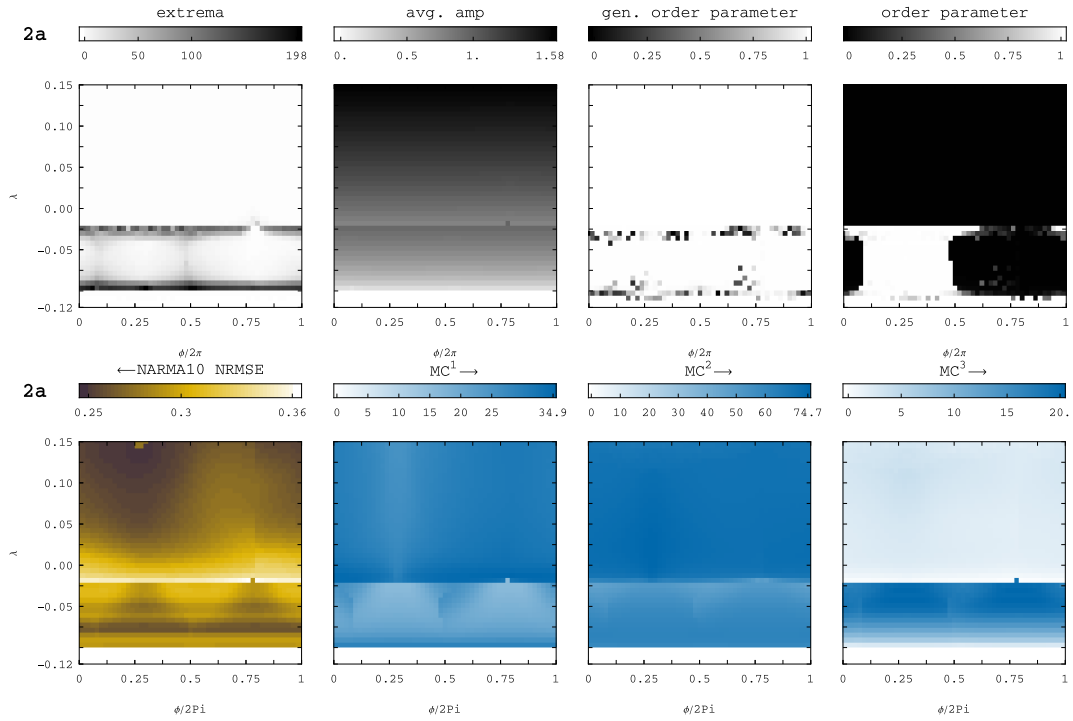

Figure 37: Bidirectional ring with $N_r = 8$ (**2a**)

with $N_r = 8$. As every node is linked to every other node (includineach self links) inputs are maximally mixed in this network. As mentioned before in the section with the linear chain networks each recurrent link copies perturbation and reintroduces it at a later time. The effect of "copies of copies" is likely strongest in a (large) all-to-all network. A way to describe this is that every node tries to accomplish everything all the time. This might lead to many very low capacities that stay below a detectable threshold.

4.3 Rings and symmetry-broken Rings

Investigating networks without symmetry is difficult: Every time we increase the number of nodes N_r and edges $\mathbf{e}$ the number of possible topologies increases dramatically. It is impossible to investigate all possible networks. Picking any random network from the vast amount of possible topologies makes it difficult to generalize gained insights. The approach was therefor to start at a known area - namely unidirectional ring topologies - and break symmetries by adding individual edges. In Fig. 8 a unidirectional ring with $N_r = 8$ (**1a**) is visualized with all possible symmetry-broken topologies we yield by inserting 1 additional edge (**1b-h**).

In the following calculations the incoming connections have been normalized so that $\sum_{j=1} \kappa_{i \leftarrow j} = \kappa$ for any node i with κ being the coupling strength between nodes in a (non-symmetry-broken) unidirectional ring (**1a**). The reason is better comparability, as it leads to a similar value λ at which the system's Hopf-bifurcation occurs - the change from the off-state to a LC. This means that for e.g. the network in (**1b**) with $\kappa = 0.1$ we change the coupling strengths $\kappa_{1 \leftarrow 8} = 0.05$ and $\kappa_{1 \leftarrow 1} = 0.05$ so that node 1 receives a total input strength of $\kappa_{1 \leftarrow 8} + \kappa_{1 \leftarrow 1} = \kappa = 0.1$. In cases where the coupling phase $\phi = 0$ this usually results in all oscillators having the same amplitude z.

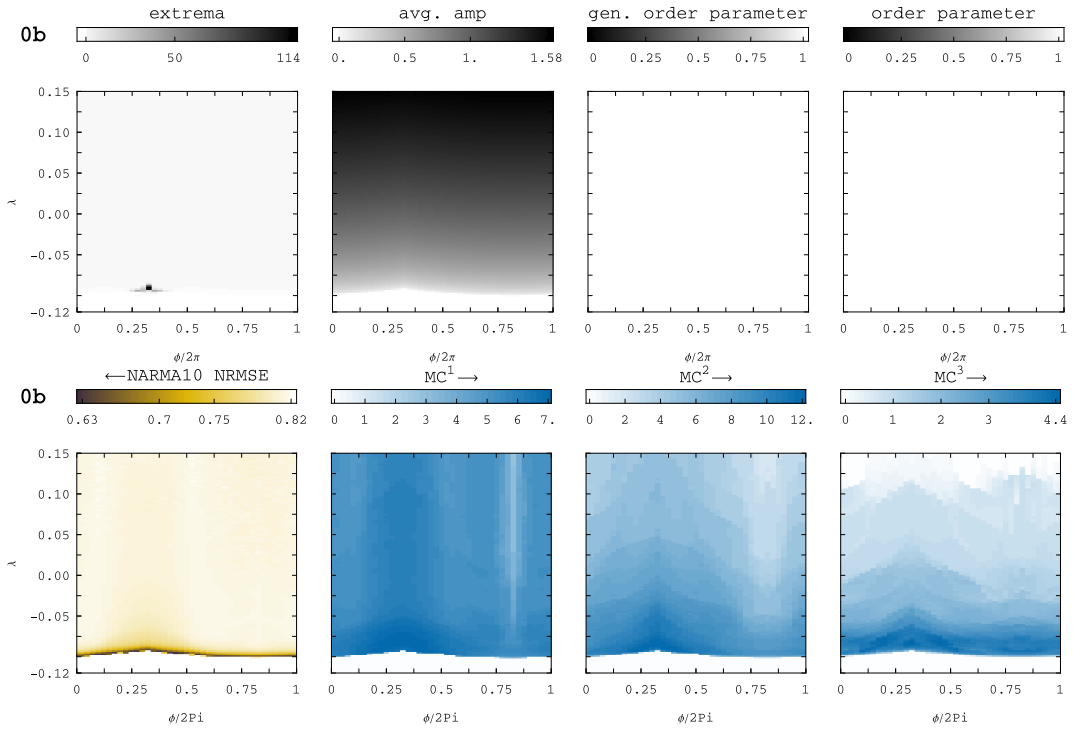

Figure 38: All-to-all connected network with $N_r = 8$.

4.3.1 3-rings with and without symmetry-breaking links

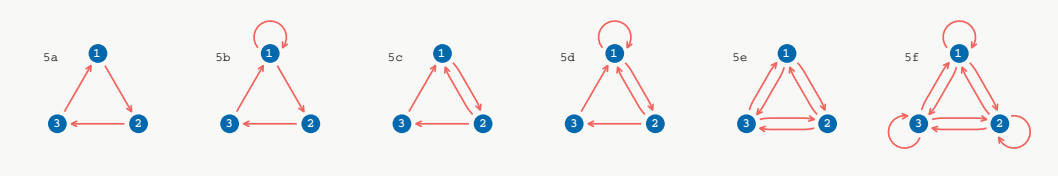

Figure 39: Some possible networks with $N_r = 3$. Unidirectional ring (**5a**) with one (**5b-5c**) or two additional link(s) (**5d**). The bidirectional ring (**5e**) is equal to the complete graph without self-links and the bidirectional ring with self feedback (**5f**) can also serve as a complete graph or diffuse coupling if the self-links are compensating the incoming links from adjacent nodes i.e. $\kappa_{i\leftarrow i-1} + \kappa_{i\leftarrow i+1} = -\kappa_{i\leftarrow i}$ (negative values κ can also be interpreted as $\phi = \pi$).

As a baseline we use the unidirectional ring (**5a**). The simplest network with $N_r = 3$, the unidirectional ring is shown in Fig. 40. The Hopf bifurcation occurs again like other networks at $\lambda \approx -0.095$ where the system changes from off-state to a stable LC. Optimal values for $N_r = 3$ and $N_v = 16$ can be seen in Fig. 40 as the largest values in the color bar for the three memory capacities or the lowest for the NARMA10 NRMSE. The 3 memory capacities in the unidirectional rings $N_r = 8$ and $N_r = 3$ look very similar, besides being scaled higher in the case of more nodes. Nonlinear memory capacities are highest in areas where linear memory is low. The further away we are from the Hopf-bifurcations and the vertical line at $\phi/2\pi \approx 0.125$ (slightly visible in the extrema plot) the better the nonlinear capacities get. The higher the degree d of MD^d the further the distance has to be. It is also apparent that the plots have mirror symmetry regarding a vertical axis at $\phi/2\pi = 0.125$ or $\phi/2\pi = 0.625$. It shall be mentioned that the plot for NARMA10

performance is surprisingly noisy compared to the memory capacities. Every data point in the plot was calculated using different uniformly drawn masks $M_i(t)$ for the three nodes $i = \{1, 2, 3\}$. This is a strong indication that the NARMA10 NRMSE is highly dependent on the exact shape of the masks (with only $N_r N_v = 3 \times 16 = 48$ virtual nodes of randomly chosen values it is more likely to have a particularly "bad" set of masks than it is for e.g. a set of 16 masks of 32 virtual nodes each). At the same time the memory capacities are fairly smooth – although they were calculated with the same (random) masks –, because they are the sum over a number of individual capacities. Individual capacities e.g. the capacity to recall input u_{-13} fluctuate more (see Fig. 24) depending on a particular mask, the compounded sum MD^1 does not. For higher degrees d the number of individual capacities MD^d is the sum over a *vastly* larger number of individual capacities.

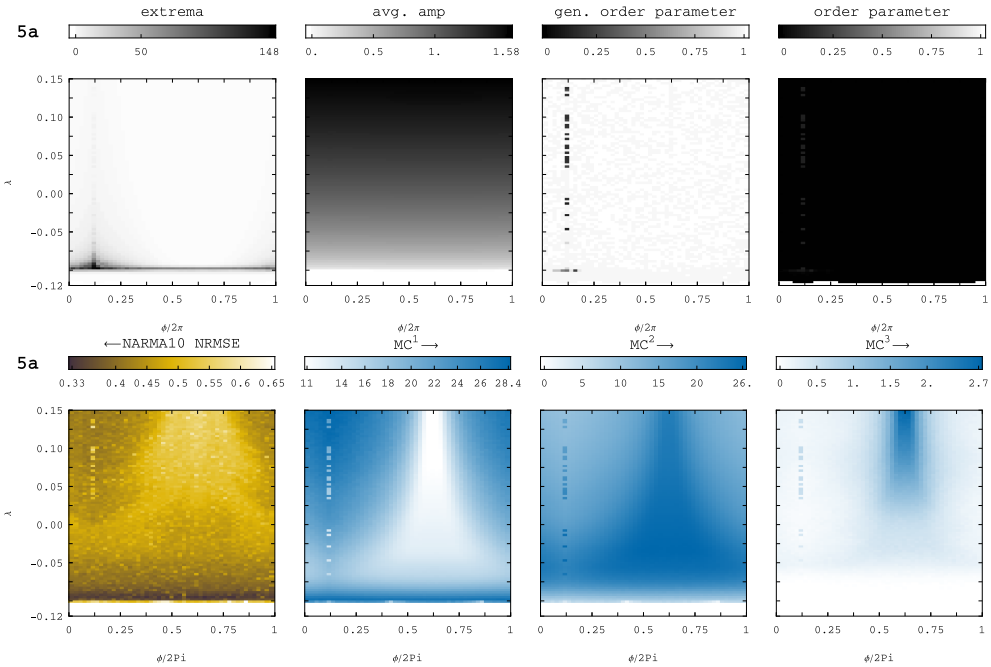

Figure 40: Network (**5a**) unidirectional ring with $N_r = 3$, $\omega = 1$, $\gamma = -0.1$, $\kappa = 0.1$, $\tau = 272$. The system has been initialized in the splay state (with small amount of randomness). The different memory capacities MD^1, MD^2 and MD^3 clearly inhabit different areas in the ϕ/λ-parameter space. Linear and quadratic memory are almost anti-correlated: Where linear memory is high, quadratic is is low and vice versa. The cubic memory MD^3 seems to occupy an even smaller subset of the area of high quadratic memory. It shall be noted that the linear memory capacity MC^1 remains above 11 at every place.

Once we break the symmetry of the unidirectional ring by adding an edge $1 \leftarrow 1$ – visualized as (**5b**) – the plots lose their mirror symmetry (mentioned above) and exhibit some bifurcation lines of slight, but sudden drops of performance. These drops are affecting different memory capacities differently. E.g. a parabola-shaped line in the upper left part of the MD^3 plot where performance slightly dips for areas above it is not visible in the plots for MD^1 and MD^2. Another bifurcation is a skewed line on the right part of the plots. It is unclear what is causing this bifurcation, as no indication in the extrema, amplitude or various order parameters is visible. An analysis of the lyapunov exponents might be reveal the reason here.

A second calculation using the same network topology (**5b**), but with a weaker cou-

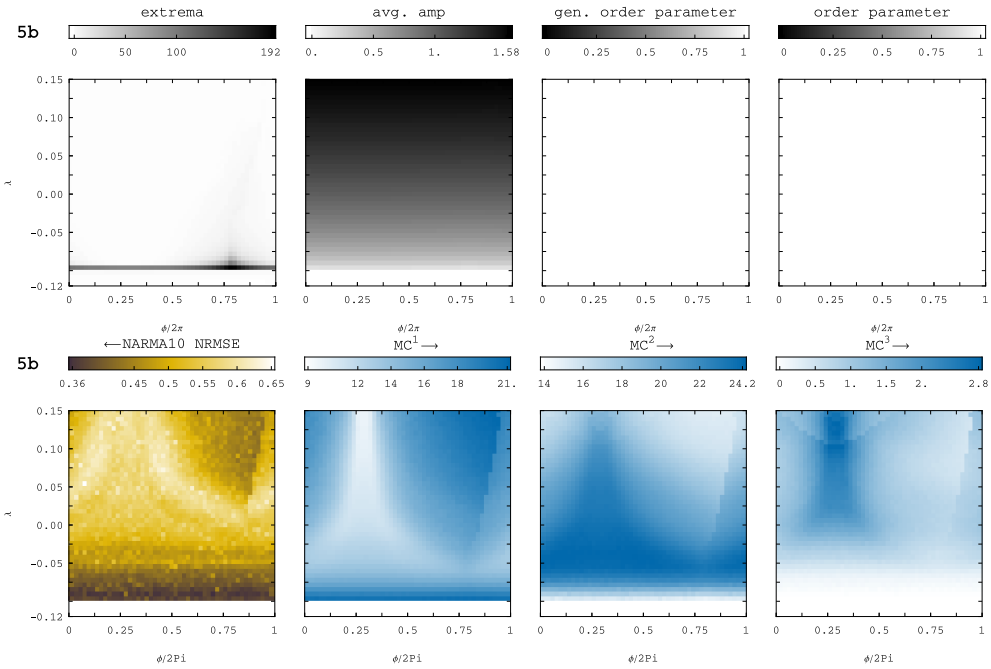

Figure 41: Network (**5b**). Unidirectional ring with an additional link $1 \leftarrow 1$. Same parameters as in 40, but with coupling strengths adjusted so that $\kappa_{1\leftarrow 1} = \kappa_{1\leftarrow 3} = 0.05$.

pling strengths of the additional edge $\kappa_{1\leftarrow 1}$ shows a completely different plot Fig. 42. Above a skewed bifurcation line reservoir performance immediately drops (close) to zero. This area shows that various synchronization states exist. The initial conditions were set so that the 3 oscillators were set to a splay state with a random offset of magnitude 0.01 (the delayed states were set to zero). The slight variation of initial conditions lead to different synchronization states visible in the great fluctuation in both order parameter plots. These plots were created using the unperturbed system. The randomization in the initial conditions is of the same magnitude as the scaling factor $G = 0.01$ which introduces a constant perturbation into the system. Consequently we can expect that the oscillators change their synchronization state constantly in this area (or that they never stay really synchronized as they are constantly perturbed with relatively large inputs). The system looses a fundamental condition for reservoir condition which maps similar inputs on similar predictions. Depending on the current synchronization state the predictions for similar inputs can now differ dramatically.

Another network with $N_r = 3$ was (**5c**) which has an additional link $1 \leftarrow 2$. We see a distinct area with a different synchronization state in the top right corner which is neither a synchronized nor a splay state as it appears dark in both plots of the generalized and normal order parameter. The new types of "splay-similar" states that exist in symmetry-broken networks are sometimes difficult to classify in a generalized manner, as the relative phase differences between oscillators varies. The top right corner where there is no synchronized state has the highest linear memory MC^1, but low quadratic memory MD^2 and lowest cubic memory MD^3. Comparing it to the topology (**5a**) the optimal cubic memory is slightly higher (3.2 combared to 2.7).

The plots for the network (**5d**) with additional edges $1 \leftarrow 1, 1 \leftarrow 2$ are shown Fig. 44. Performance is very similar to the network (**5b**) which has one connection less. The

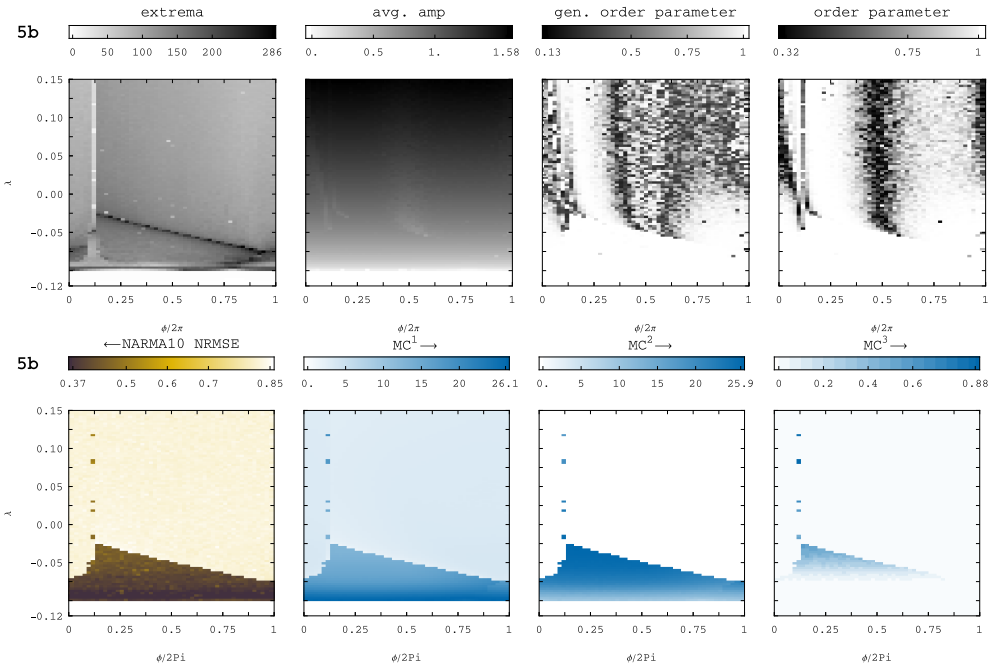

Figure 42: Network (**5b**). Unidirectional ring with an additional link $1 \leftarrow 1$. Same parameters as in 40, but with a weaker coupling strengths of the additional link. Coupling strengths adjusted so that $\kappa_{1\leftarrow 1} = 0.0091$ and $\kappa_{1\leftarrow 3} = 0.091$. Above a skewed line where some kind of bifurcation occurs almost all reservoir capacity disappears. In the same area the 2 order parameters show the existence for various synchronization states, that intersect each other and depend heavily on the slight variations of the initial conditions.

difference is only a slight optimal performance drop for NARMA10 NRMSE, linear and quadratic memory and a significant drop of 25% from 3.2 to 2.4. The only remarkable difference between the two is in the cubic memory: In the plot for MD^3 of (**5d**) a higher number of (visible) bifurcation lines exist in the top left area which performance drops.

The small bidirectional network with $N_r = 3$ (**5e**) is shown in Fig. 45. The plots for the order parameters show that the oscillators are in a splay-state for values $\lambda = 0.05$ (with slight variation depending on the coupling phase ϕ). Again, the better linear memory MD^1 is found the case of the splay state and quadratic memory MD^2 seems to be anti-correlated i.e. preferring the synchronized state. Cubic memory MD^3 is low and also prefers areas of synchronized solutions. Although it completely drops to zero for the splay-state.

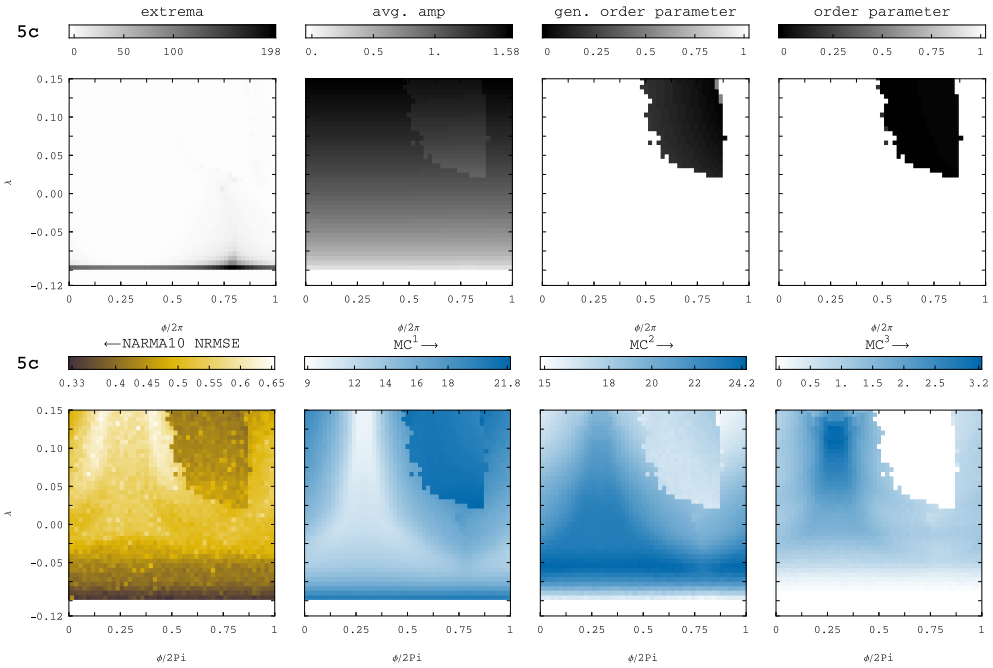

Figure 43: $N_r = 3$ with topology (**5c**) that has an additional link $1 \leftarrow 2$. Oscillators were mostly synchronized which the order parameters show. In the black areas in the top right both order parameters drop as the oscillators are neither synchronized nor in a "real" splay-state of constant relative phase difference. In the same area the average amplitudes drop slightly below the values of the synchronized state. The additional link seems to increase the optimal cubic memory slightly compared to the unaltered unidirectional ring (**5a**).

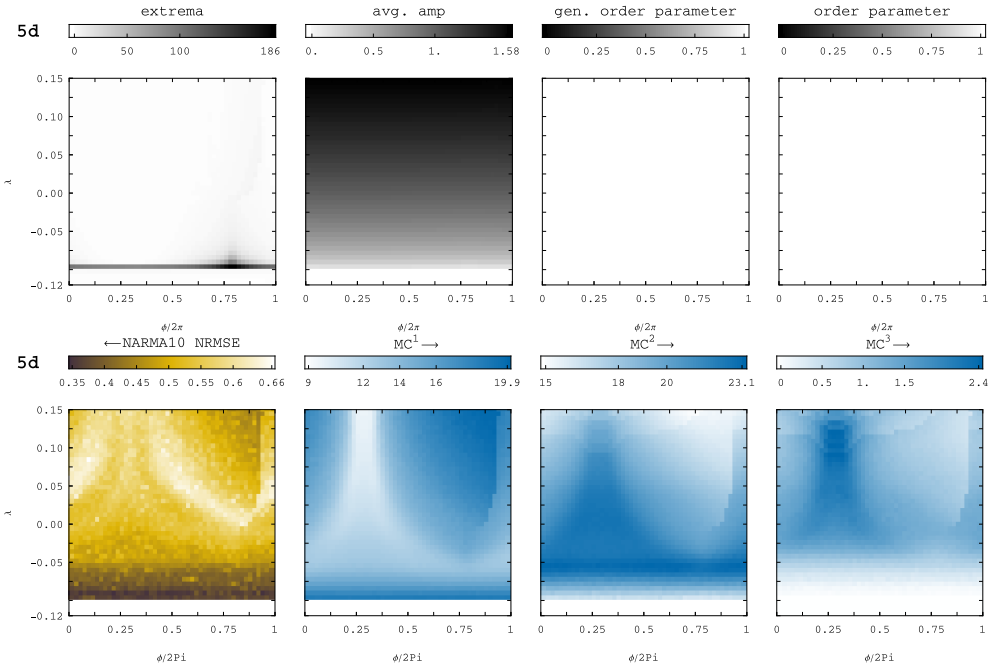

Figure 44: $N_r = 3$ with topology (**5d**) which has 2 additional links $1 \leftarrow 1$ and $1 \leftarrow 2$. Coupling strengths are $\kappa_{1\leftarrow 1} = \kappa_{1\leftarrow 2} = \kappa_{1\leftarrow 3} = 0.01/3$. The plots looks almost identical those in Fig. 41 (**5b**). The difference is a slight drop in best linear and quadratic memory (plus a tiny increase of the smallest NARMA10 NRMSEs) and a significant drop of 25% from 3.2 to 2.4 of the best cubic memories.

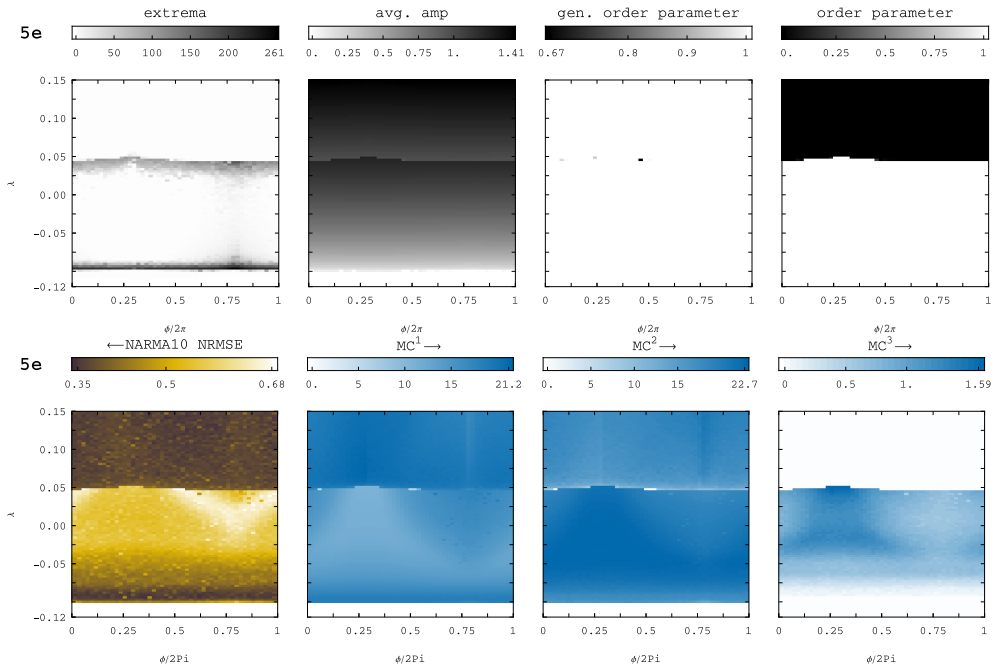

Figure 45: $N_r = 3$ with bidirectional ring topology (**5e**). All coupling strengths $\kappa = 0.05$ after normalization. Oscillators are in 2 different synchronization states. For $\lambda > 0.05$ oscillators are in a splay-state (the line of division slightly wanders depending on ϕ). The splay-state enables higher linear memories MD^1 and NARMA10 performances and lower quadratic memories MD^2. Cubic memories MD^3 directly drop to zero with the splay state.

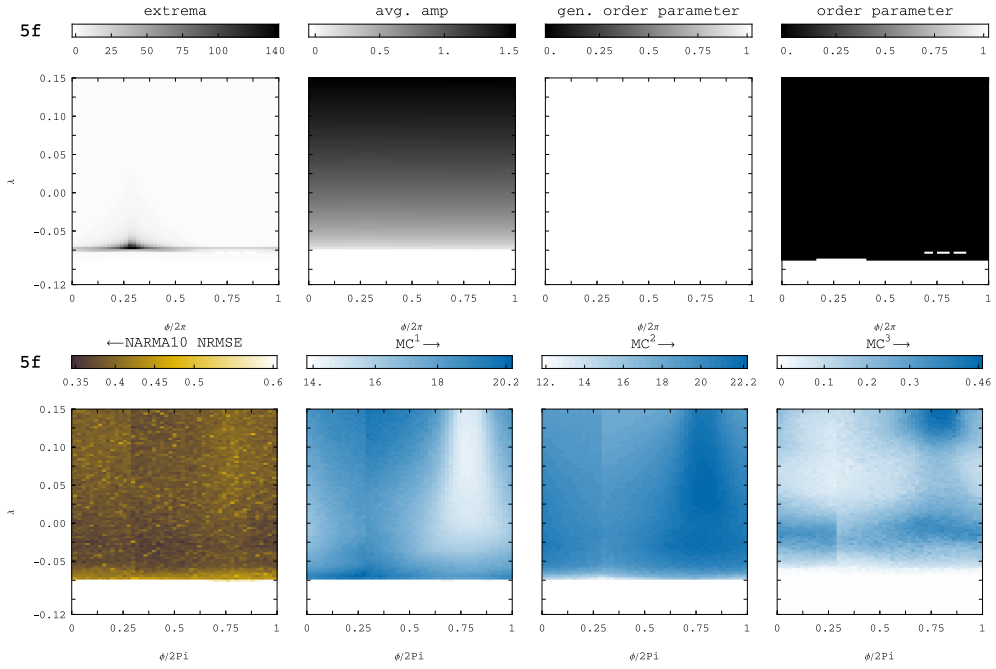

Figure 46: $N_r = 3$ with links as in (**5e**) and diffuse coupling so that $\kappa_{i\leftarrow i} = 2\kappa_{i\leftarrow i\pm1}$ and $\phi_{i\leftarrow i} = \phi_{i\leftarrow i\pm1} + \pi$. Here coupling strengths were $\kappa_{i\leftarrow i\pm1} = 0.025$ and $\kappa_{i\leftarrow i} = 0.05$ after normalization.

4.3.2 Networks with $N_r = 4$ nodes

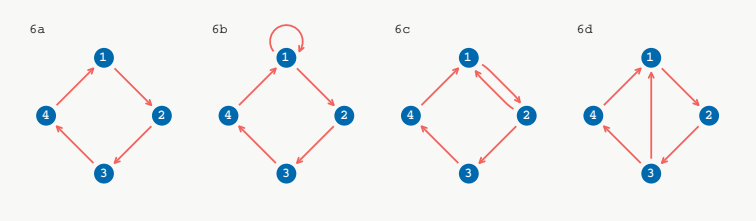

Figure 47: A few networks with $N_r = 4$. Unidirectional ring (**6a**) with one additional link (**6b-6d**). Inputs were normalized to received magnitude of input was the same for every node. Results for (**6a**) and (**6d**) are shown below.

In Fig. 47 the unidirectional ring (**6a**) with all 3 ways of placing an additional edge (**6b-d**) are shown. Here only the plots for (**6a**) and (**6d**) are shown. The unidirectional ring in Fig. 48 shows the same familiar behavior as the $N_r = 3$ variant shows (see Fig. 40) albeit with higher memory capacities and lower NARMA10 NRMSE's. Interestingly the capacities MD^1 and MD^2 of the two rings $N_r = 3$ and $N_r = 4$ scale about linearly compared with the increase in read-outs (33% more) as shown in Fig. 49. However the cubic memory MD^3 increases much more. This is likely the same phenomenon seen in the the increasing number of individually self-linked nodes shown in Fig. 28. The cubic memory seems to rely on a certain minimum number of readouts D it can exploit.

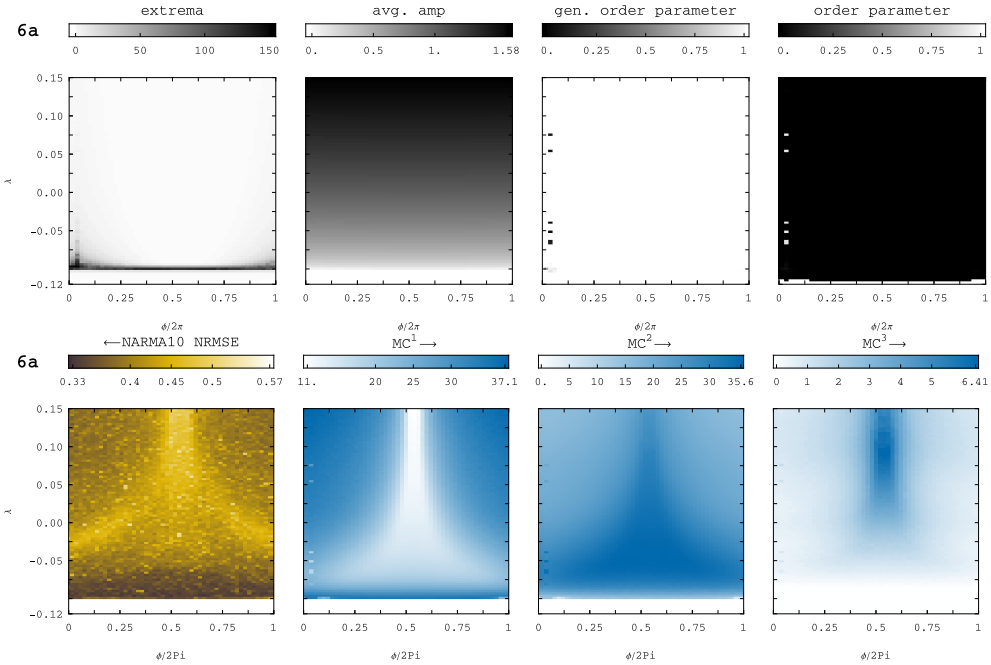

Figure 48: Capacities for $N_r = 4$ simple unidirectional ring (**6a**). Initialized in a splay state with $m = 1$. The plots look familiar to the unidirectional ring with $N_r = 3$, altough with higher capacities due to the higher amount of readouts $D = N_r \times N_v = 4 \times 16 = 64$.

The addition of an edge $1 \leftarrow 3$ results in synchronized oscillators for certain areas in the λ/ϕ parameter space (white areas in Fig. 50). The 2 different synchronization states (white and black areas for synchronized and splay-like states) exhibit the same split

	N_r (D)	MD^1_{max}	MD^2_{max}	MD^3_{max}
	3 (48)	28.4	26	2.7
	4 (64)	37.1	35.6	6.41
ratio	1.3333	1.31	1.37	2.37

Figure 49: Comparison of maximum memory capacities MD^1, MD^2, MD^3 in 2 unidirectional rings with $N_r = 3$ and $N_r = 4$. Values from Fig. 40 and 48.

again. Linear memories MD^1 preferring the splay(-like) states and nonlinear memories MD^2 and MD^3 preferring synchronized states. Compared to the unidirectional ring the maximum values of the respective memory capacities decrease with the additional edge. The additional edge brings no benefit compared to the unidirectional ring. Although it is not clear if the reason is the lack of symmetry or simply the higher connectivity. As the all-to-all example shows (Fig. 38 and 45), higher connectivity generally results in worse reservoir performance (compared to their less connected counterparts). Changing the topology from (**6a**) to (**6d**) means going from 4 edges to 5 edges i.e. an increase of 20%. It is possible that any advantage gained by the increased variety of individual nodes is outweighed by the disadvantage that denser networks bring.

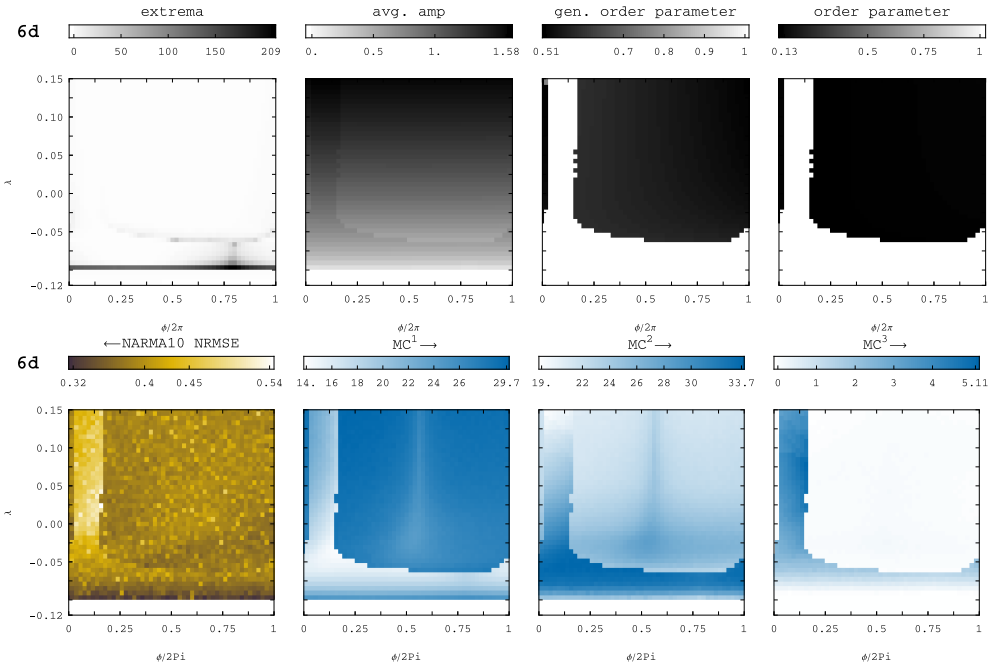

Figure 50: Capacities and some dynamical parameters for $N_r = 4$ unidirectional ring with additional edge $1 \leftarrow 3$ (**6d**). Initialized in a splay state with $m = 1$ which disappears again in the white areas visualized for the order parameters where oscillators are completely synchronized. The black areas in the order-parameter plots are not 0. The oscillators are in a splay-like state that is difficult to capture with the order parameter. Only capacities are plotted that were measured i.e. found linear memory was within an intervall $MD^1 \in [14, 29.7]$. Again nonlinear transformation capacities are best highest in areas where the oscillators are synchronized. The overall capacities are worse than the symmetric unidirectional ring (Fig. 48).

4.3.3 $N_r = 8$ with symmetry breaking links

For a ring with $N_r = 8$ nodes different variants of additional edges have been investigated. Unfortunately only a small fraction of all possible calculations was done. As 8 different splay(-like) states are possible each network topology would have required to be made with (at least) 8 different synchronization states (splay-like states) as initialization. Instead oscillators were synchronized at the start of the calculation. In general the results do not differ greatly. Thus only some are shown here. The delay was set to $\tau = 68$, the virtual node time to $\theta = 3$ and $\theta = 3$ (resulting in an input period of $T = 48$) and the oscillators were initialized in a synchronous state. This was done to decrease calculation time as the calculations would otherwise take too long. This time all masks were the same for each calculation (but still different from node to node, of course). For some topologies (**1b-1h**) calculations were made also with a smaller coupling strength of the additional (symmetry-breaking) edge. In general all plots look almost identical for (**1b-1h**) with very small changes of capacities. Thus only plots for some topologies are shown. In Fig. 52 only the fully synchronized state is present, whereas in Fig. 53 (weaker symmetry breaking edge) a cone-shaped area appears separated by a bifurcation line, in which the oscillators change into a splay-state (or one similar to a splay-state).

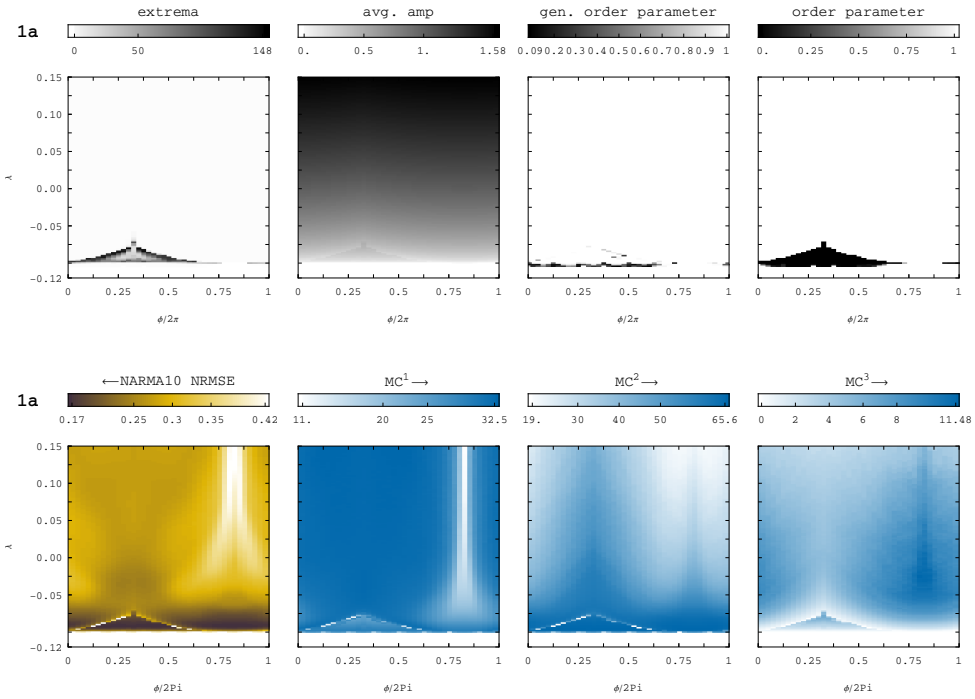

Figure 51: $N = 8$ ring with $\kappa = 0.1$ (**1a**). Other parameters were $\tau = 68, N_v = 16, \theta = 3, T = 48$. Initial condition were synchronous with small offset. Oscillators are mainly in a synchronous state besides a cone-shaped area near the bottom left.

Placing the edge differently gives a very similar result. Topology (**1e**) with an edge $1 \leftarrow 4$ was used in Fig. 54) with both edges towards node 1 having the same coupling strength $\kappa_{1\leftarrow 7} = \kappa_{1\leftarrow 3} = 0.05$. Best memory capacities and lowest NARMA10 NRMSEs are almost identical. The only small difference worth mentioned is the appearance of a second small cone-shaped area within the bigger cone-shaped are in the lower left that is separated by some sort of bifurcation line. It is only visible in the linear and cubic memory

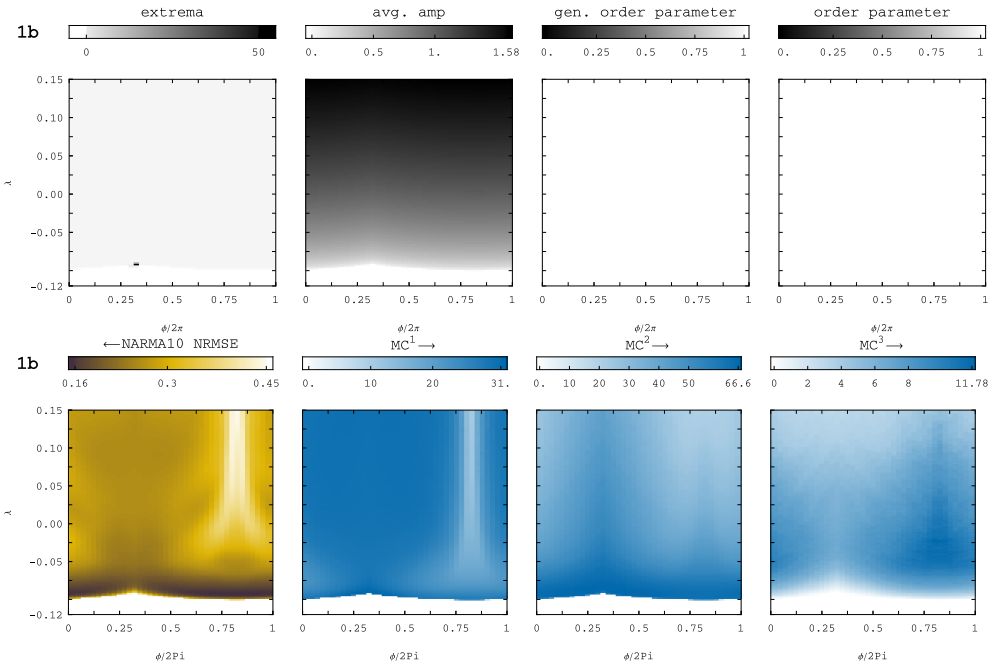

Figure 52: $N = 8$ ring with $\kappa = 0.1$ and an extra self-edge $1 \leftarrow 1$ (**1b**) with $\kappa_{1\leftarrow 1} = \kappa_{1\leftarrow 8} = 0.05$. Other parameters were $\tau = 68, N_v = 16, \theta = 3, T = 48$. Initial condition were synchronous with small offset. Compared to the symmetric uni-directional ring in Fig. 51 the additional edge seems to stabilize the synchronous state in areas where where it would otherwise change into a splay-state.

capacities MD^1 and MD^3.

One possible explanation for the small amount of change in between networks (**1a-1f**) is that there is only one symmetry-breaking edge compared to 8 edges that make the unidirectional ring. So the influence of this one edge is too small in order to significantly change the behavior.

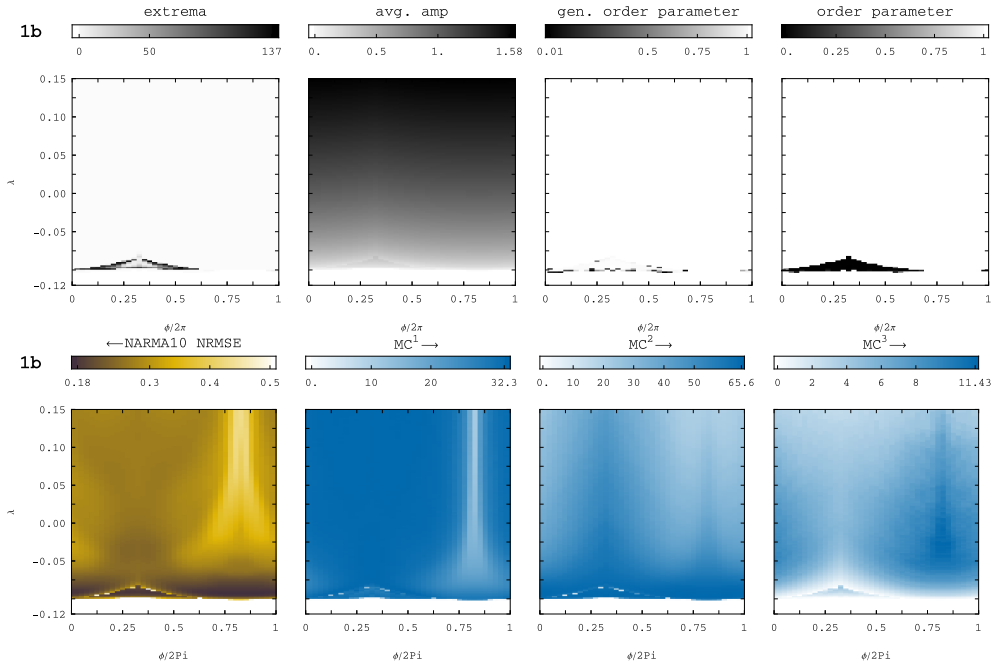

Figure 53: Same topology as in Fig. 52 (**1b**), but with the additional edge havin a weaker coupling strength: $\kappa_{1\leftarrow 1} = 0.1 \times 1/11$, $\kappa_{1\leftarrow 8} = 0.1 \times 10/11$. Initial condition = synchronous with small offset. As a result the another cone-shaped area (visible in black in the plot of the order parameter) appears where the oscillators inhabit a splay state. Due to the smaller weight of the additional edge the same area os almost white in the generalized order plot i.e. it is very close to a splay state.

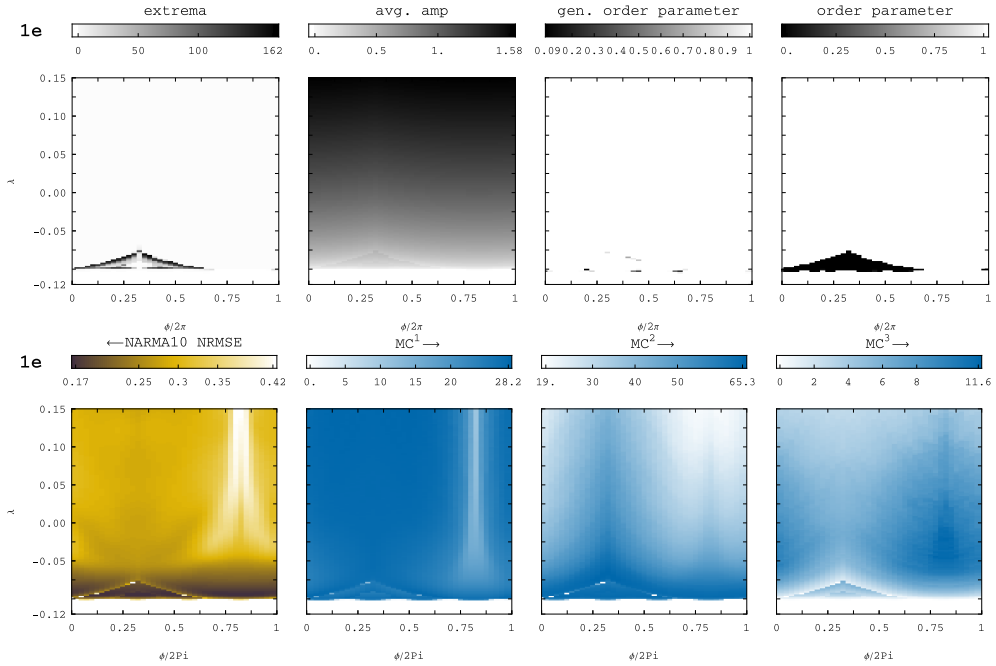

Figure 54: Topology (**1e**) uni-directional ring with an additional edge $1 \leftarrow 4$. Oscillators initialized in a synchronous state with small random offsets. The plots are very similar to the ones above with the difference of a second bifurcation line, that separates a secondary smaller cone-shaped area within the bigger cone-shaped area. It is only (easily) visible in the plots for memory capacities MD^1 and MD^3, but does not affect overall capacities.

4.3.4 Impact of selective input to single nodes

A small investigation into individual linear and quadratic memories was done for 2 unidirectional ring networks with $N_r = 8$ nodes. The first one was simply the ring (**1a**) and the second was (**1g**), a unidirectional ring with additional edge $1 \leftarrow 6$. Two sets of calculations were done. In the first case both networks were tested when all nodes receive input i.e. the global scaling factor was $G = 0.01$. In the second case both networks only received input at the first node i.e. $G = 0$ with $G_1 = 0.01$. Linear and pure quadratic memory capacities for all 4 resulting scenarios are shown in Fig. 55. Mixed quadratic memory capacities with the full readout and per-node read-outs are shown below in Fig. 56 and 57. The virtualization factor was increased to $N_v = 64$ while the virtual node time was decreased to $\theta = 3$. This way the input period was the same as in a network with $\theta = 12$ and $N_v = 16$, but the number of read-outs was increased to $D = 512$. This was necessary in order to make capacities visible (increase to near 1) that were otherwise very faint.

Fig. 55 top-left (**1a**): The linear memory capacities gained by the sub-statematrices $\mathbf{S}_i$ is very similar for the network (**1a**) with equally distributed inputs. The individual capacities slightly fluctuate due to every node having a different mask M_i. Some diagonal structure can be seen in the area where per-node capacities drop from 1 to ≈ 0 (between recall steps $n = 10$ and $n = 20$). The reason is that any fluctuations introduced through different masks propagate through the network in the direction of the coupling (from lower to higher index). The purely quadratic per-node capacities drop to 0 much quicker. The benefit of taking the full read-out of all nodes is bigger for the linear memory where relatively high capacities can be gained at recall steps, where the individual capacities have disappeared completely (e.g. at recall step $n \approx 15$). (2) top-right: The additional symmetry breaking of this network (**1g**) seems to increase the full linear memory capacities in a way that they decay much slower towards 0 in areas of larger recall steps n. At the same time the purely quadratic capacities seem the decrease slightly. The per-node linear memory capacities gain a little bit for larger recall steps. The increase of linear memory is not really useful though, as the area of (almost) perfect reconstruction of previous inputs ($C_n^1 \approx 1$) stays about the same. Only the decline is slower. When we compare the mixed quadratic memory capacities $C^{\{1,1\}_{\{n_1,n_2\}}}$ (all read-outs) of both networks from Fig. 56 we can see a slight change (visualized on the right).

(3) bottom-left: If we perturb only one singular node the reservoir capacities drastically change. Then the individual nodes have much more unique capacities. There is little to no mutual information present in multiple nodes (see Fig. 25). Each node can only recall a selection of previous inputs. Similarly the pure quadratic memories propagate around the ring, with the difference of decreasing slower. The result is that the linear and pure quadratic memory capacities from the full read-outs are both increased (bottom plot) compared with the all-node input calculation above. The linear memory is still very high (> 0.9) at recall step $n = 34$ and the pure quadratic memory is high at recall step $n = 14$. Compared to the regular network this improves memory capacity 10 steps further into the past.

(4) bottom-right: The effect of one symmetry-breaking edge is much more pronounced in the case of only one input-driven node. The additional link re-introduces the input a little earlier. This increases the number of places these perturbations can interact with each other. As a result the linear and pure quadratic memories decrease (decay quicker) compared to the symmetric network (left).

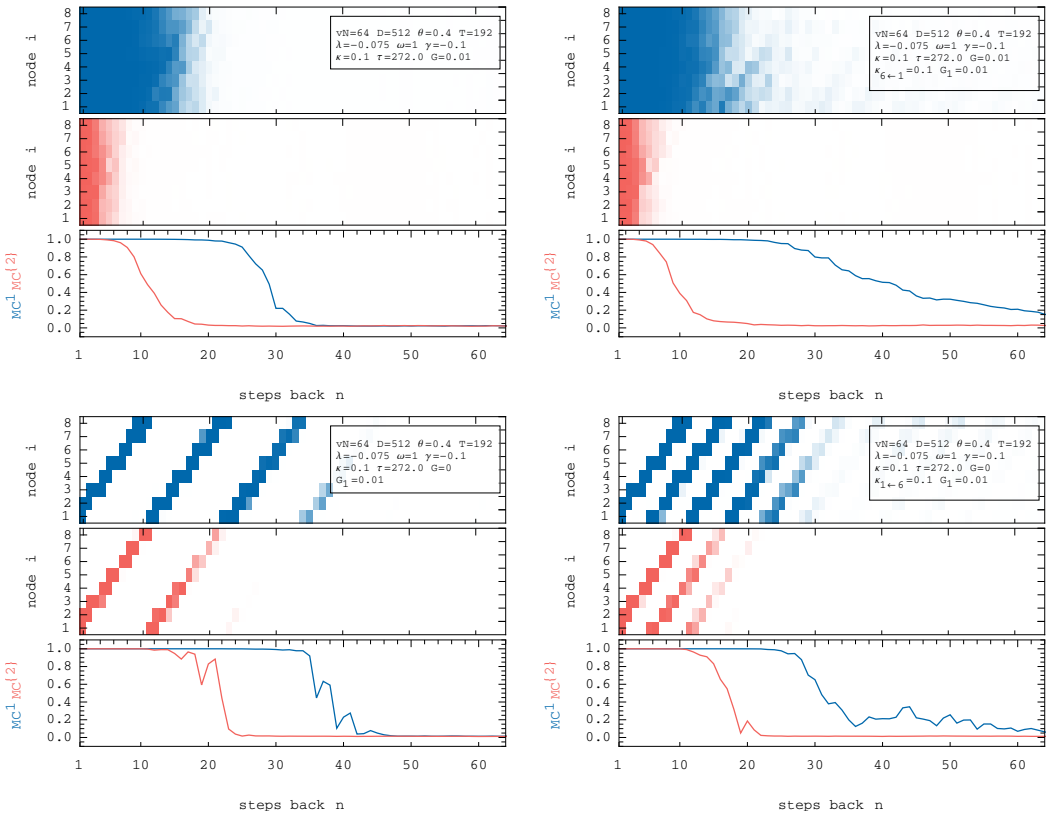

Figure 55: 4 different scenarios, each with a set of 3 plots. On the left side a simple uni-directional ring was used (**1a**), on the right the uni-directional ring had an additional symmetry-breaking edge $1 \leftarrow 6$ (**1g**). In the top row of plots all of the $N_r = 8$ oscillators received input with scaling factor $G = 0.01$, but in the bottom row only the first node was driven with a selective scaling factor of $G_1 = 0.01$ ($G = 0$). Shown for each scenario are the linear per-node memory capacities $C_n^1(\mathbf{S}_i)$ (top, blue) and the pure quadratic per-node memory capacities $C_n^{\{2\}}(\mathbf{S}_i)$ (mid, red). Each plot in the 4 bottoms shows both capacities $C_n^1(\mathbf{S})$ ——— and $C_n^{\{2\}}(\mathbf{S})$ ——— with the full statematrix $\mathbf{S}$ taken into account.

The tradeoff between linear memory and nonlinear tranformations is visible in the comparison of the linear memories from Fig. 55 and the mixed quadratic memories in Fig. 56 (**1g**) bottom-mid. But also transitions between different kinds of nonlinear transformations can be seen. Following the transition from (**1a**) to (**1g**) (single node input) we can see pure quadratic memories disappear and mixed quadratic memories appear instead. In the symmetry broken network much more specialized mixed quadratic memory capacities exist than in the symmtric one (bottom left). This makes the reservoir much more specialized in order to solve special tasks. In Fig. 56 the mixed quadratic memories only exist for certain combinations of inputs and completely miss for others where *some* capacity existed in the case of all nodes receiving input. The in- and decrease of capacities is shown on the right.

For the mixed quadratic capacities it is also interesting to visualize the per-node capacities again. In Fig. 57 the 8 per-node capacities for all 4 described scenarios are shown. Each row shows nodes $i = \{1 - 8\}$. The two topmost rows are the all-node input cases, which do not vary much from node to node. The two lower rows are each enabling an even

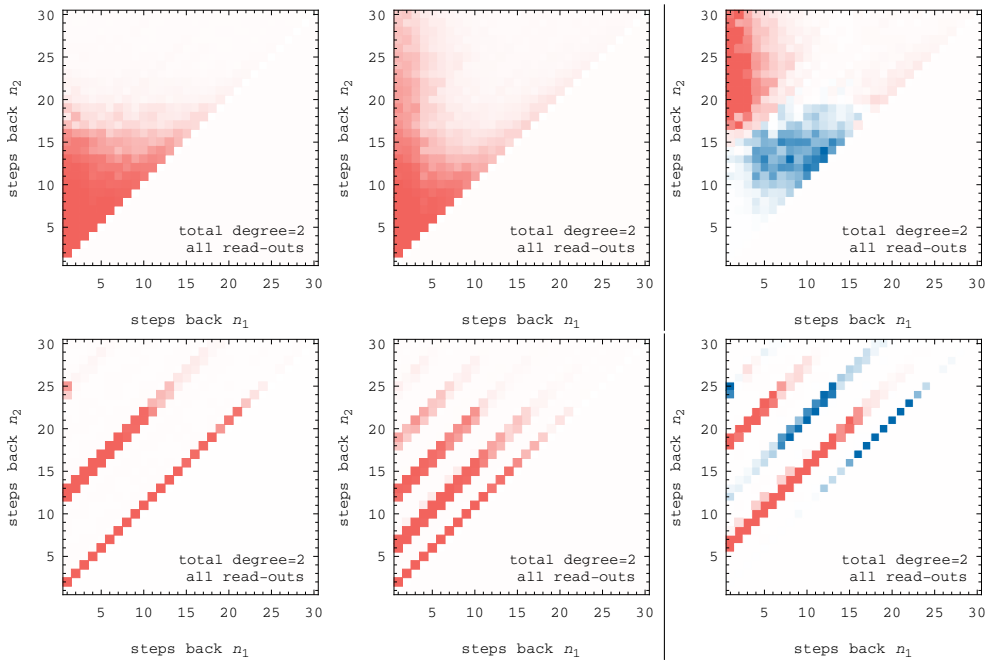

Figure 56: On the left of the vertical line: the same 4 scenarios positioned similar to Fig. 55 with plots for the mixed quadratic memory capacities $C^{\{1,1\}}_{\{n_1,n_2\}}$ gained by using all read-outs (the full statematrix $\mathbf{S}$). Each block represents the system's capacity to reconstruct a function $L_1(u_{-n_1})L_1(u_{-n_2})$. Only the upper triangle is calculated calculated as the lower triangle would only mirror certain capacities. Therefor $n_2 > n_1$. The 2 plots on the right show the difference between the plots on the (left) and (mid). On the right side of the vertical line: in blue areas capacities decrease, in red areas they increase when we change the network topology by adding an extra connection $1 \leftarrow 6$ (transition from **1a** to **1g**). The change in topology effectively re-distributes reservoir capacities from one area to another.

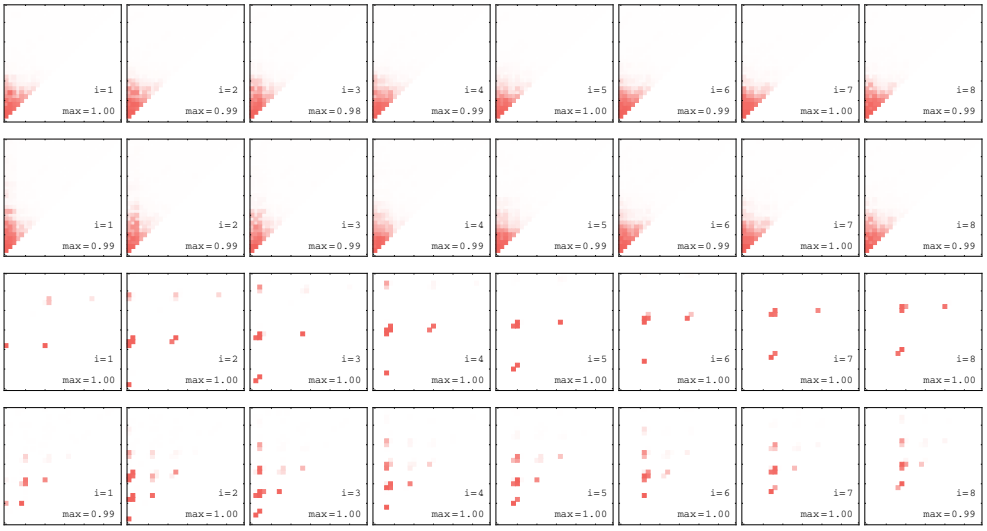

Figure 57: The same 4 scenarios from Fig. 55 with per-node plots for nodes $i = \{1 - 8\}$ (left to right). Each row shows per-node capacities of one scenario: row 1 is network (**1a**) with input at all nodes, row 2 network (**1g**) with input at all nodes, row 3 network (**1a**) with input only at node 1 and row 4 network (**1g**) with input only at node 1. The ticks were omitted in order to save space, but each plot covers the same area with $n_1, n_2 \in [1, 30]$ as the full-readout plots in Fig. 56. All capacities were gained by only using the respective sub-statematrices $\mathbf{S}_i$.

narrower selection capacities than the combined readout (the ticks have been omitted in order to increase visibility, the axes are the same as in the case of the full read-out.) It might come as a benefit if in a hybrid reservoir like (**1a**) or more specialized (**1a-g**) with multiple real nodes only the readout from one node is needed in order to perform a certain task.

5 Conclusions

5.1 Symmetry breaks in networks

The investigation of symmetry breaks in networks is difficult as a very high number of potential new networks must be investigated in order to make any general statements. Investigation of reservoirs that consisted of individual self-linked oscillators (see Fig. 26) shows that a greater variety among nodes beyond just different masks increases the combined RC performance. Consequently a symmetry broken network should have increased variety, as the dynamics of individual nodes also varies. But testing this was difficult, as combined memory capacities varied only slightly between symmetric (ring) and symmetry-broken networks. The symmetry-breaking nodes change the underlying dynamics in a way that was difficult to investigate. Known methods like the (generalized) order parameters appeared ill-suited to identify and distinguish these symmetry broken states. But identifying and "labeling" them is required in order to further investigate their properties. Some of these states seemed to be utterly unstable, which almost completely inhibited RC performance (see Fig. 42). This new challenge was not initially expected and made it very difficult to investigate these networks. Completely unsymmetrical network (random networks) would only increase this difficulty. As a result only some symmetry breaks were investigated that could be seen as derived from symmetrical networks like rings. The relative change seemed to be larger in smaller networks where one additional edge seems to have a larger impact. In larger rings ($N_r = 8$) the overall performance of symmetry-broken networks seemed to only slightly change reservoir performance. Only late in this thesis were the very general memory capacities MD^1 and MD^2 dissected and its individual parts investigated separately. As a result it was shown that symmetry breaks can shift capacities from certain areas to other (desired) areas (see Fig. 56, right side). Although another effect obfuscated the investigation, as some performance changes were also potentially dependent on simply the number of edges in a network i.e. the amount of connectivity. Complete networks (all-to-all coupled) had the worst RC performances of given a certain number of nodes N_r. Which also makes sense, because with more parts in the network where different perturbations interact with each other more secondary responses are resulting. This means that information more quickly dissolves. We can also in a way interpret a reduced input layer as a sparser network: Going back to the virtual network analogy of the masks we see a restricted input to few nodes as setting the virtual node values in most masks to 0. This means that we reduce the number of connections in this virtual network. We can conclude that sparser networks (real or virtual) can aide more narrow tasks, although it might still decrease the total memory capacity of a system.

5.2 Phase information and amplitude information

Oscillators in (some) splay-states make possible that one oscillator's (current) amplitude is affecting the next oscillator's phase and vice versa. The exact condition of when a splay-state can aide the (mainly linear) memory capacities depends on the phase-difference $\arg(z_i(t)) - \arg(z_j(t-\tau))$ between 2 oscillators coupled through an edge $i \leftarrow j$. If the coupling phases in a unidirectional ring are chosen $\phi = \Omega_m \tau - 2\pi m/N_r$ the system becomes monostable [SCH12]. The author suspects that this results in $\arg(z_i(t) - z_j(t-\tau)e^{i\phi}) = 0$ which also removes amplitude-phase-coupling (in the case or real values γ). The reason is that the current states of the oscillators $z_i(t)$ are in phase with the delayed ones when turned around an angle ϕ. In this case a splay-state behaves in the same way as the fully synchronized solution with regard to the relationship of $z_i(t)$ and $z_j(t-\tau)$. The initial

perturbation, which is in a radial direction is affecting the next coupled oscillator in the radial direction as well. Information is restricted to the amplitudes of the oscillators.
In general it takes longer for the phase-differences of a splay-state to reach equality. Amplitudes equalize faster, as one oscillator's equilibrium amplitude is largely defined by the LC i.e. the oscillator itself (with small values κ it is mostly determined by λ and $GJ(t)$). Equalization of the relative phase differences of coupled oscillators only happens through interactions between coupled oscillators - after all the Stuart-Landau equation for the phase-velocity $\dot{\phi}$ (18) is not depending on ϕ itself. Synchronization/equalization of oscillators i and j (coupled through $i \leftarrow j$ and indirectly through $j \leftarrow ... \leftarrow i$) with regards to the phases can only happen through interactions that are delayed by $\tau_{i \leftarrow j}$ and/or the total indirect coupling delay from node i to node j (the other way around). This means that a splay-state that is perturbed in the angular direction of its oscillators takes (much) longer to reach equilibrium again. When we imagine the process of equalization of adjacent oscillators as a negotiation trying to reach a compromise, with the equalization of amplitudes the oscillators sit in a room, whereas equalization of phase-differences is discussed via letters through mail i.e. every message with its subsequent response is delayed significantly.

As amplitudes and phases of coupled oscillators feed into each other for certain splay states (without amplitude-phase coupling – $\mathrm{Im}(\gamma) = 0$) this phase information can be extracted again through the amplitude. With $\mathrm{Im}(\gamma) > 0$ amplitude-phase coupling can occur without splay states or within a single oscillator. The relationship between $\mathrm{Im}(\gamma) > 0$ and memory capacities remains ambiguous though. More research with a narrower focus in systems with fewer degrees of freedom is needed here.

5.2.1 Phases and Spikes

Encoding information in the phase seems to enhance linear memory. For more complicated laser models the phase-relationship of coupled oscillators can be translated into the relative positions (in time) between pulses of either an oscillator or oscillators within a network. Investigating further the relationship between relative phase-distances might yield insights in how to encode information into the timings of pulses in order for a system to memorize it or perform computation on it. Considerable research is done in the construction of optical neurons [DIL19, COO11, KOE18a, MOS00] i.e optical systems that behave similar to an excitable neuron that creates optical pulses. These optical neurons aim at mimicking the behavior of biological neurons with which very powerful information processing is possible, but on much shorter timescales and with much less energy consumption. Understanding further the relationship of information encoding into perturbations of oscillator-phases might help to bridge the gap between reservoir computation with information encoded into amplitudes and the potentially more advantageous spike-time encoding as such systems can potentially encode and sustain a multitude of such spatio-temporal spike patterns [POP11] (which might aid the linear memory capacity of a system).

The author suspects that spike encoding adds another mechanism to information processing which is the ability to model decision making. As mentioned in 3.1.1, more traditional architectures of artificial neural networks return probabilities rather than definite answers. A feed-forward artificial neural network returns a vector with entries that represent the probability that a certain pattern or concept exists in an input i.e. the picture of a dog, a cat, a bicycle or any other item the net was trained on. The decision is made

by choosing the the entry with the highest likelihood (typically above a certain threshold) and discarding all other entries. The process of picking one item and disregarding the rest can be seen as a decision which moves a statement from the realm of fuzzy logic into the realm of discrete (digital) values. It can also be seen as a process of reducing the amount of information present within a system. In artificial neural networks that more closely model the behavior of biological neurons (Hodgkin-Huxley, FitzHugh-Nagumo, Hindmarsh-Rose [HOD52, FIT55, HIN82]) information encoded in the spike-amplitudes of (weighted) *inputs* of a neuron primarily decides if a neuron produces a spike. The heights of output-spikes do not depend much on the incoming spike-heights. This represents a basic decision process: It removes (or drastically reduced) information about the height of spikes from each connected dendrite and results in a basic "spike" or "no spike" result and only leaves information encoded in the spike-time. In the no spike scenario it removes input information entirely.
In the case of hybrid networks with large connectivity (all-to-all networks or to a lesser degree small bi-directional rings) resulted in less linear reservoir performance compared to the sparser connected uni-directional ring. The reason is probably that any previous input u_{-n_1} acts as uncorrelated noise upon an input u_{n_2} (this might not be the case for time-series prediction tasks in which two seperate previous inputs can very well be correlated). More connectivity of a network therefor means more dilution of various inputs and the resulting nonlinear transformations of it until it becomes basically unusable. There might still be some correlations that can be exploited in order to predict values of a certain target function, but prediction will suffer from large errors. In other words such a reservoir might be capable of performing a large number of tasks badly (below the threshold), but almost none well. Similar to a tiled bathroom full of talking people - after a certain amount the verbal noise it becomes impossible to decipher and derive meaningful information. Beyond a certain amount of simultaneous speech we tend to split into separate groups similar to large organizations that compartmentalize in order to reduce flow of unnecessary information. This might be a good method for larger reservoirs - hybrid reservoir networks seem to profit from a sparser connectivity - at least beyond a certain point. But with mechanisms that actively reduce the amount of information or simply the transition from Stuart-Landau oscillators towards excitable laser systems might make higher connectivity have less of an impact.

5.2.2 Detection of splay-like states

A new problem with categorization arises in symmetry broken networks. A splay-state of oscillators is together with the synchronized solution a natural or obvious result of the network topology. But if the network topology changes the "natural" solutions are not as easily detectable if we search for them with the more orthodox methods like the order parameter P_o of the generalized order parameter P_g. A simple way to produce new splay-like states is to randomize edge parameters. Another way of creating these states can be seen in [ROE18b], where "symmetry broken" states come as a result of of a complex parameter γ. As shown before in section 3.7.1 the exact shape of these new states can also change when global coupling parameters are changed. This means that in order to identify distinct states we have to group together whole continua of states that vary slightly into their respective categories. We have to invent a new definitions of "similarity" in order to categorize these states. This is different to identifying the different splay-states in a unidirectional ring network with global values for κ, ϕ and τ: The splay-states do not change their relative phase-positions once we change e.g. ϕ. The only thing that changes is the stability of certain (splay-)states.

For a normal splay-state (plus synchronized state) we have a very simple model that is given by (22) with parameters $A \in \mathbb{R}^+$ and $m \in [1, N_r]$ (amplitude and relative phase-differences). In order to categorize splay-states we disregard the amplitudes A and split categories by m: We can easily visualize splay-states as dots in a 2-dimensional A over m plot. With $N_r = 8$ nodes and a large enough number of calculations with different initial conditions and values λ and ϕ each splay-state solution would appear as a dot within 8 lines parallel to the A axis. Each line would be at one value m. It is very easy to categorize these splay states, as we can simply project the dots onto the m axis and thereby disregard the amplitudes. Each category is now uniquely associated with a single m value. But with more complicated states e.g. these described splay-like states, categorization is more complex. Now individual amplitudes A_i and relative phase-differences m_i among oscillators can vary. Additionally they can now depend non-trivially on oscillator parameters (λ etc.) and on (global) coupling parameters κ, ϕ and τ. The dots of such splay-like states are no longer simple straight lines that position themselves nicely along certain parameter axes. Instead they will sit on curved manifolds in high-dimensional space and have therefore be described individually. But it can be expected, similar to splay states, that can be separated into different categories (by m), that these splay-like states sit in separable patches that we can identify. In order to do that we will need to accurately model these irregular states. Here methods adapted from ML might help us to fit models to these states and thereby reduce the high dimensionality.

5.2.3 Restricting input to a selection of real nodes

Restricting the amount of inputs a reservoir receives might in some cases give a benefit, as single capacities (narrow task performances) can be controlled more precisely. Only late into this thesis was it realized, that instead of perturbing all nodes in a hybrid network individual reservoir capacities can be increased or decreased by only perturbing one node. This comes as a surprise as the usual approach is that the more complex the input is (the masks) more complex the trajectory of the reservoir in phase space is. This results in more nonlinear transformations that can be exploited. As a result the noise-masks can outperform constant-level masks like the binary or uniform masks [KUR18]. It is possible that restricting perturbation in some parts of the (hybrid) reservoir could increase individual narrow capacities, while at the same time reducing the total (combined) capacity of a system. Ultimately a desired goal in RC is that we want a reservoir to excell at few tasks with instead of performing many with mediocre capacities. This phenomenon would be interesting also in single-node ($N_r = 1$) reservoirs, where a larger ratio of virtual node values could be set to 0. An advantage of reducing the number of input-nodes (shrinking the input layer) is that it enables us to better understand where and how inputs interact with each other and how this enables certain reservoir performances.
Another potential benefit of restricting input to few or only a single node could be that we can reduce the number of nodes altogether. Klinshov et. al. showed that the dynamics of a uni-directionally delay-coupled ring can be used to embed in the the dynamics of a single self-coupled oscillator (with delay) [KLI17]. With some adjustments this could be used to replace a unidirectional ring network that is only perturbed by input at one node with a single node. The adjustments would probably be an adapted read-out mechanism that adds readouts from previous input periods to the current readout. For networks with symmetry broken topologies this approach could be combined with [LUE13] in order rearrange a network and also reduce the number of physical nodes which would simplify experimental setups.

5.3 Failures and Challenges

The approach of taking medium or large network architectures of high symmetry and introduce symmetry-breaking links seems too broad of an approach. Too many possibilities of symmetry-breaking exist in order go get a clear answer to the very general question of "does reservoir performance increase in symmetry-broken topologies?". The reason is that the benchmarks for the (non-)linear memory capacities MD^d are the accumulations of a great number of individual capacities. This is visible in the sometimes noisy NARMA10 NRMSE values when randomized sets of masks $\mathbf{M}(t)$ were used in each calculation. As NARMA10 is only one singular (narrow) task that can greatly depend on a particular set of masks a larger fluctuation of performance is the result. In some symmetry-broken networks these sums $(MD^1, MD^2, MD^3, ...)$ are in- or decreasing slightly depending on the network topology. It will be necessary instead to investigate how individual capacities are depending on network topologies. A good idea would be to also select a small set of capacities and investigate their dependency on the topology (and additionally also on the masks). For this a better understanding of *how* and *where* nonlinear responses originate that enable a reservoir to predict values of a specific target function e.g. $L_2(u_{-3})L_1(u_{-5})$. The author of this work initially thought that testing reservoir performance in a generalized way through the Legendre-tasks was an ideal catch-all method. As it theoretically captures all possible combinations of tasks one can perform, that it could simply replace other narrower tasks like NARMA10 or chaotic series predictions which are used in other works [PAT17, PAT18, ROE18a, CUN19, EST19, ORT17a] (to only name a few). But as mentioned the combined quantities like MD^d are very broad and therefore lack specificity. Timeseries prediction tasks are more valuable as singular narrow tasks than previously anticipated and should investigated as well.

It was not possible to answer the question concisely of when exactly certain nonlinear capacities are created. In order to describe the problem better we can consider a person as a reservoir that is being given a new random number $u(t)$ every minute t. This person is tasked to return the result of a nonlinear transformation L_2 of this random number, but with a delay of 10 minutes. In terms of Legendre-tasks this is a the benchmark $L_2(u_{-10})$ which is the Legendre polynomial of degree 2 of an input u given 10 time steps (here: minutes) earlier. We can consider different ways of performing this task: (1) calculate + remember result: the person performs the calculation the very moment it receives a new value u, remembers the result for 10 minutes and then gives the value of the computation. (2) remember input + calculate: The person remembers the input values for 10 minutes, then performs the calculation and directly gives the result. In the context of RC this either means the initial nonlinear dynamic response that creates a specific capacity is remembered as if it was its own input in the system (1) or the input is remembered by the system and the nonlinear response is created at the last possible moment. In reality it will probably not be either (1) or (2), but a combination of both. This question remains unanswered. It might be answered through using different sets of input sequences that are fed into the reservoir at different nodes.

Another big challenge are the new kinds of phase-locked states which were here described as "splay-like" states. Normally separate splay-states exist and can be easily differentiable through their relative phase-difference. But with symmetry-broken networks, these phase-differences are not the same anymore and can change depending on which two nodes are compared and also by other (global) parameters. It is unhelpful to initialize the oscillators randomly and plot all results in the same plot as the results can vary

a lot depending on the state of synchronization. A suited method of different identifying symmetry-broken states and distinguish them from each other was not available or could not be identified. The result was a rather limited approach of using the normal order parameter and generalized order parameters which were at least able to identify the synchronized state and "not synchronized" state.

5.4 Ideas for further investigations

5.4.1 More fine grained benchmarks

As mentioned above, a more precise investigation of RC performance is needed. Investigating total memory capacities like MD^d for high degrees d is not very precise of a measurement. As MD^d is only to sum over a very large number of individual capacities. Investigating individual capacities $\{C\}$ of a system and compare or correlate them with a system's set of lyapunov exponents might make a better and more fine-grained description of a reservoirs capacities. Standard ML practices like dimension-reduction and/or manifold learning methods like [MAA08, MCI18] and more traditional neural network architectures might come into play here. E.g. small feed-forward-connected artificial neural networks might be able to accurately predict RC performances through the set of lyapunov exponents alone. Instead of investigating only total memory capacities MD^d the specificity of individual tasks is equally important. E.g. a value MD^2 does not tell us if the reservoir performs a large number of Legendre-tasks with mediocre quality, or a select few with (almost) perfect accuracy. In practice we will likely prefer the latter.

5.4.2 Investigating systems through lyapunov exponents

Unfortunately this work did not involve the investigation of the dynamical systems through its lyapunov sprectra. But the existence of sometimes numerous sharp lines of sudden increase of decrease in RC performance in plots over λ and ϕ show that there exist more subtle processes (possibly bifurcations) either affecting RC-performance as a whole or affecting capacities of individual Legendre-tasks specifically or classes thereof. It remains unclear nonetheless.

5.4.3 Uncoupling readout dimension D from virtualization factor N_v

As mentioned it is unclear how much information a certain mask introduces into a reservoir. What would be the "virtualization factor" of a continuous, smooth mask or masks derived from white noise (see "smooth mask" and "noise mask" in Fig. 21)? Does a hybrid network that is driven only at one of the real node still have a combined number of "virtual nodes" of $N_r \times N_v$? Does the total virtualization decrease if many virtual nodes have mask values of 0? The investigation of capacities through a subsets of read-outs $S_{\neg 1}, S_{\neg 2}, \ldots$ has led to a deeper question: How much of a system's internal information can we extract through the readouts x. In each input period T we have a continuous function $|z(t)|$ that we sample at θ-intervals. Can we increase capacities if we sample $|z(t)|$ at $\theta/2$-intervals? It might be interesting to investigate how much information a reservoir can contain and if it is more than the virtualization factor N_v.. The equivalence of D with the product of real nodes and virualization factor $N_r \times N_v$ might turn out to be a limitation. In Fig. 58 the traditional paradigm is shown (left) next to a potential alternative (right) where readout positions are chosen at random while at the same time the number of read-outs D has been increased independently of $N_r \times N_v$.

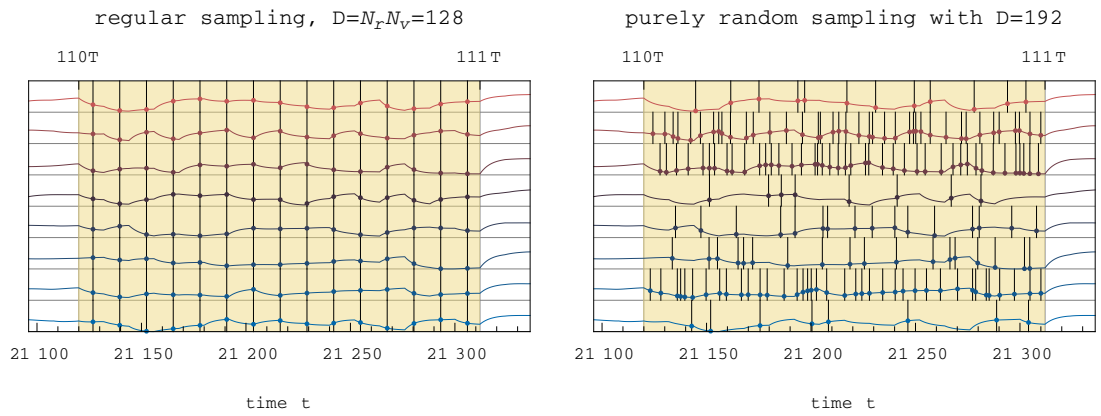

Figure 58: The dynamic state of a delay-based reservoir with $N_r = 8$ and $N_v = 16$ over input interval 110 is the set of 8 (continuous) timeseries (one per node) within an interval $t \in [110T, 111T]$ (light yellow). (left) the traditional understanding demands that the state is read out once every virtual node i.e. at each node N_v times at precisely θ-intervals. (right) a more general approach uncouples the readout dimension D from the virtual nodes and also randomizes the readout positions (indicated through vertical bars). The readouts are random but fixed, meaning they repeat with each input period T.

5.4.4 How do certain capacities manifest themselves in the dynamical evolution?

In ML training deep artificial neural nets involves backpropagation of the system's errors. Predictions and consequently its errors are created through weighted sums of previous layers in such a feed-forward network. Backpropagation traces back the largest contributors of the error in order to correct the network. It is in a way looking where the error comes from. In delay-based RC it might be possible to backtrace through the specific weights of a capacity the exact shape of a perturbation which creates the capacity in question. Instead of backtracing the error we might backtrace the origins of capacities. Once a certain capacity has been connected to a certain perturbation it should be possible to calculate the lyapunov exponent of this specific perturbation. And find if they are connected.

5.4.5 Multiple input sequences that drive specific nodes of a reservoir

Using a set of uncorrelated input sequences $\{\mathbf{u}^1, \mathbf{u}^2, \dots\}$ that are driving individual nodes respectively it might be possible to answer the question of *where* and *when* certain nonlinear capacities more precisely i.e. the dynamical responses that are responsible for specific capacities that are created. Using several distinct input sequences makes it possible to separate the network into nodes that only receive input $\mathbf{u}^1, \mathbf{u}^2, \dots$ and nodes that can receive both. This way it is possible to easily identify specific nodes at which certain capacities are created. Additionally it might be interesting how well reservoirs can do several completely independent tasks (based on separate inputs $\{\mathbf{u}^1, \mathbf{u}^2, \dots\}$) simultaneously and if/how they interfere with each other. It makes easy to divide the network into parts that are only capable of accessing a certain subset of different input sequences. In Fig. 59 a simple example is shown. Here two chains that receive different (uncorrelated) inputs at their respective first nodes and merge at a certain point. Nonlinear transformations that depend on both inputs can only be produced through nodes *after* the merging i.e. nodes $5-9$ on the left and nodes $11-12$ on the right. Comparing the per-node capacities before and after merging would give insight how information is transformed. Investigating the

capacity for a simple mixed task e.g. $\mathbf{u}^A \times \mathbf{u}^B$ at node $5-9$ (left) and comparing it to the capacity measured at node 11 could show how they are created. Are mixed task capacities originating at the point of confluence and afterward simple propagate as linear memory, or are they created at every point after merging?

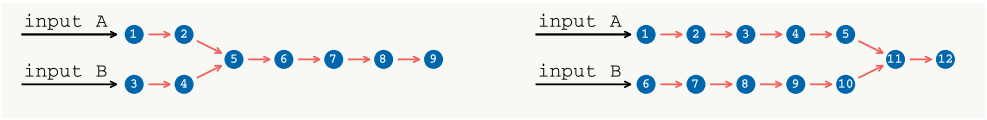

Figure 59: 2 uncorrelated input sequences $\mathbf{u}^A, \mathbf{u}^B$ are fed to the beginnings of 2 chains that later merge into one. The parts that only receive perturbations originating from either $\mathbf{u}^A$ or $\mathbf{u}^B$ are easy to distinguish. Only *after* the confluence of both chains is it possible for the system to predict targets of nonlinear functions $f(u^A, u^B)$ that depend on both input streams e.g. the values of a function $f = L_1(u^A) \times L_2(u^B)$. By shifting the point of confluence of both information streams it is possible to compare per-node capacities at nodes that have the same travel-distance from the point beginning.

5.4.6 Use of dimensionality-reduction methods to tackle high degrees of freedom in networks of DDEs

A network of several nodes with many potential edges that each have its own parameters κ, ϕ, τ together with different parameters λ, ω, γ and many possible masks $M(t)$ is a very large number of parameters that effect reservoir performance. This makes it difficult to visualize the dependence of capacity on these parameters. In order to understand these it is probably a good idea to use nonlinear dimensionality-reduction techniques like t-SNE, UMAP or other manifold learning approaches [MAA08, MCI18]. The manifold hypothesis i.e. "most real-world-data lies on low-dimensional manifolds embedded in high dimensional space" probably holds true here as well. This possibility should be utilized.

5.4.7 A more nuanced approach to masks

As masks can greatly determine performances of individual (narrow) tasks it might be a good idea to better categorize them. As an example: Two different binary masks M_1 and M_2 might have different average values which basically result in different λ-values for that calculation. So masks should either be adjusted in order to all have the same average value (difficult if only values 1 and 0 are possible) or masks could be created by shuffling the order of virtual nodes, therefor keeping the averages. For uniform masks with virtualization factor N_v each mask can be seen as a point within an N_v-dimensional hypercube. Each side of the cube represents the values of one virtual node in the mask. Investigating how certain regions within this mask-space are correlated with higher or lower reservoir capacities in certain tasks might lead to a better understanding of how masks work and which masks enable certain kinds of capacities. Dimension-reduction methods from ML might be suited for the task.

References

[ALS96] P. M. Alsing, V. Kovanis, A. Gavrielides and T. Erneux. Lang and Kobayashi phase equation. Phys. Rev. A **53**, 4429–4434 (1996).

[APP11] L. Appeltant, M. C. Soriano, G. Van der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. R. Mirasso and I. Fischer. Information processing using a single dynamical node as complex system. Nat. Commun. **2**, 468 (2011).

[ARG18] A. Argyris, J. Bueno and I. Fischer. Photonic machine learning implementation for signal recovery in optical communications. Sci. Rep. **8**, 1–13 (2018).

[BRU13a] D. Brunner, M. C. Soriano, C. R. Mirasso and I. Fischer. Parallel photonic information processing at gigabyte per second data rates using transient states. Nat. Commun. **4**, 1364 (2013).

[BUT13b] J. B. Butcher, D. Verstraeten, B. Schrauwen, C. R. Day and P. W. Haycock. Reservoir computing and extreme learning machines for non-linear time-series data analysis. Neural Netw. **38**, 76–89 (2013).

[CHO09] C. U. Choe, T. Dahms, P. Hövel and E. Schöll. Controlling synchrony by delay coupling in networks: from in-phase to splay and cluster states. Phys. Rev. E **81**, 025205(R) (2010).

[COO11] W. Coomans, L. Gelens, S. Beri, J. Danckaert and G. Van der Sande. Solitary and Coupled Semiconductor Ring Lasers as Optical Spiking Neurons. Phys. Rev. E **84**, 036209 (2011).

[CUN19] A. Cunillera, M. C. Soriano and I. Fischer. Cross-predicting the dynamics of an optically injected single-mode semiconductor laser using reservoir computing. Chaos **29**, 113113 (2019).

[DAM12] J. Dambre, D. Verstraeten, B. Schrauwen and S. Massar. Information processing capacity of dynamical systems. Sci. Rep. **2**, 514 (2012).

[DIL19] M. Dillane, I. Dubinkin, N. Fedorov, T. Erneux, D. Goulding, B. Kelleher and E. A. Viktorov. Excitable interplay between lasing quantum dot states. Phys. Rev. E **100**, 012202 (2019).

[EHL18] D. Ehlert. Data injection scheme influence on optical reservoir computing, 2018.

[EST19] I. Estébanez, I. Fischer and M. C. Soriano. Constructive Role of Noise for High-Quality Replication of Chaotic Attractor Dynamics Using a Hardware-Based Reservoir Computer. Phys. Rev. Appl. **12**, 034058 (2019).

[FER03] C. Fernando and S. Sojakka. Pattern Recognition in a Bucket. In *Advances in Artificial Life*, Seiten 588–597, 2003.

[FIT55] R. FitzHugh. Mathematical models of threshold phenomena in the nerve membrane. Bull. Math. Biol. **17**, 257–278 (1955).

[HER12] M. Hermans. PhD thesis, Universiteit Gent, 2012.

[HIN82] J. L. Hindmarsh and R. M. Rose. A model of the nerve impulse using two first-order differential equations. Nature **2**96, 162 (1982).

[HOC98] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. nternational Journal of Uncertainty, Fuzziness and Knowledge based Systems **6**, 107–115 (1998).

[HOD52] A. L. Hodgkin and A. F. Huxley. A Quantitative Description of Membrane Current and its Application to Conduction and Excitation in Nerve. J. Physiol. **1**17, 500–544 (1952).

[INU17] M. Inubushi and K. Yoshimura. Reservoir Computing Beyond Memory-Nonlinearity Trade-off. Sci. Rep. **7**, 10199 (2017).

[ALB19] Albrecht J. *Optimizing reservoir computing performance of laser networks with delay.* Master's thesis, TU Berlin, 2019.

[JAE01] 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.

[JAE02] H. Jaeger. Short term memory in echo state networks. GMD Report 152, GMD - Forschungszentrum Informationstechnik GmbH, March 2002.

[KLI17] V. Klinshov, D. Shchapin, S. Yanchuk, M. Wolfrum, O. D'Huys and V. I. Nekorkin. Embedding the dynamics of a single delay system into a feed-forward ring. Phys. Rev. E **9**6, 042217 (2017).

[KOE20a] F. Köster, D. Ehlert and K. Lüdge. Limitations of the recall capabilities in delay based reservoir computing systems. Cogn. Comput. **i**n press (2020).

[KUB19] T. Kubota, K. Nakajima and H. Takahashi. Dynamical Anatomy of NARMA10 Benchmark Task. arXiv preprint arXiv:1906.04608 (2019).

[KUR18] Yoma Kuriki, J. Nakayama, K. Takano and A. Uchida. Impact of input mask signals on delay-based photonic reservoir computing with semiconductor lasers. Opt. Express **2**6, 5777–5788 (2018).

[KOE18a] F. Köster. *Synchronization in models for optical neurons.* Master's thesis, TU Berlin, July 2018.

[LUE13] K. Lüdge, B. Lingnau, C. Otto and E. Schöll. Understanding electrical and optical modulation properties of semiconductor quantum-dot lasers in terms of their turn-on dynamics. Nonlinear Phenom. Complex Syst. **1**5, 350–359 (2012).

[MAA02] W. Maass, T. Natschläger and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural Comp. **1**4, 2531–2560 (2002).

[MCI18] L. McInnes, J. Healy and J. Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. arXiv (2018).

[MOS00] E. C. Mos, J. J. L. Hoppenbrouwers, M. T. Hill, M. W. Blum, J. J. H. B. Schleipen and H. de Waardt. Optical neuron by use of a laser diode with injection seeding and external optical feedback. IEEE Trans. Neural Netw. **1**1, 988–996 (2000).

[WIF12] Randall Munroe. What-If no.1: Relativistic Baseball (https://what-if.xkcd.com/1/), 2012.

[ORT17a] S. Ortín and L. Pesquera. Reservoir Computing with an Ensemble of Time-Delay Reservoirs. Cognitive Computation **9**, 327–336 (2017).

[PAT17] J. Pathak, Z. Lu, B. Hunt, M. Girvan and E. Ott. Using machine learning to replicate chaotic attractors and calculate Lyapunov exponents from data. Chaos **27**, 121102 (2017).

[PAT18] J. Pathak, B. Hunt, M. Girvan, Z. Lu and E. Ott. Model-Free Prediction of Large Spatiotemporally Chaotic Systems from Data: A Reservoir Computing Approach. Phys. Rev. Lett. **1**20, 024102 (2018).

[POP11] O. V. Popovych, S. Yanchuk and P. Tass. Delay- and Coupling-Induced Firing Patterns in Oscillatory Neural Loops. Phys. Rev. Lett. **1**07, 228102 (2011).

[ROE18b] A. Röhm. PhD thesis, TU Berlin, 2018.

[ROE18a] A. Röhm and K. Lüdge. Multiplexed networks: reservoir computing with virtual and real nodes. J. Phys. Commun. **2**, 085007 (2018).

[ROE19] A. Röhm, L. C. Jaurigue and K. Lüdge. Reservoir Computing Using Laser Networks. IEEE J. Sel. Top. Quantum Electron. **2**6, 7700108 (2019).

[SCH12] E. Schöll, A. A. Selivanov, J. Lehnert, T. Dahms, P. Hövel and A. L. Fradkov. Control of synchronization in delay-coupled networks. Int. J. Mod. Phys. B **2**6, 1246007 (2012).

[STE20] F. Stelzer, A. Röhm, K. Lüdge and S. Yanchuk. Performance boost of time-delay reservoir computing by non-resonant clock cycle. Neural Netw. **1**24, 158–169 (2020).

[SUG20] C. Sugano, K. Kanno and A. Uchida. Reservoir Computing Using Multiple Lasers With Feedback on a Photonic Integrated Circuit. IEEE J. Sel. Top. Quantum Electron. **2**6, 1500409 (2020).

[VAN08a] K. Vandoorne, W. Dierckx, B. Schrauwen, D. Verstraeten, R. Baets, P. Bienstman and J. V. Campenhout. Toward optical signal processing using photonic reservoir computing. Opt. Express **1**6, 11182 (2008).

[VAN14] K. Vandoorne, P. Mechet, T. Van Vaerenbergh, M. Fiers, G. Morthier, D. Verstraeten, B. Schrauwen, J. Dambre and P. Bienstman. Experimental demonstration of reservoir computing on a silicon photonics chip. Nat. Commun. **5**, 3541 (2014).

[SAN17a] G. Van der Sande, D. Brunner and M. C. Soriano. Advances in photonic reservoir computing. Nanophotonics **6**, 561 (2017).

[MAA08] L. van der Maaten and G. E. Hinton. Visualizing Data using t-SNE. JMLR **9** (2008).

[VIN15] Q. Vinckier, F. Duport, A. Smerieri, K. Vandoorne, P. Bienstman, M. Haelterman and S. Massar. High-performance photonic reservoir computer based on a coherently driven passive cavity. Optica **2** (2015).

[YAN09] S. Yanchuk and P. Perlikowski. Delay and periodicity. Phys. Rev. E **7**9, 046221 (2009).