Reservoir Computing Stuff

Lukas Manuel Rösel

June 10, 2020

Contents

1 todo things on the todo list.

1.1 reading list

todo:

- [ZOU09b] splay states in a ring of coupled oscillators: from local to global coupling.
- [YAN11, POP11, KLI17, PAD13, LUE13b] - von Yanchuk empfohlen. delay-reduzierung. usw.
- [YAN09] periodische lösungen

done:

- [EAR03, CHO09, YEU99] synchrony in delay coupled networks

1.2 programming work

1.3 research work

- check different weights
- check symmetry breaking nodes
- check "standard" topologies
- check breaking of standard topologies (extra links)

1.4 graphics/plots

- stuart landau oscillator: complex plane, driven by input, with delayed feedback
- linear memory recall curve
- feed in plus system response for different ratios of theta and lambda
- rN from 1 to 32, rN from 1 to 32 with plot.
- generalized order parameter -> impact of extra edge to generalized order parameter

here comes the intro and all the important references... small world [WAT98].

1.5 theory

- reservoir computing. definitions RC with delay, cc, masks, virtual nodes (more later)
- virtual networks through multiplexing. breakdown of analogy (random masks)
- tasks, training, covariance
- linear memory, legendre polynomials, narma
- linear regression. result: weights (plot weights for linear memories).
- networks

1.6 results

- increasing network size
- increasing virtualization
- increasing input strength

1.7 further investigations/open question

interesting subjects that were not investigated in this work

- the flow of information within the network by omitting the readout from different nodes
- designing tasks where two seperate data streams are being fed to different nodes in the network.
- can a network perform two calculations on different streams of data? (aka quantify multitasking/crosstalk).

by feeding information from different timeseries to different nodes within the same network the flow of information might be investigated. Also the mixing of those two streams of data. From two distributions u_1 and u_2 the tasks $O(u_1, u_2)$ can be derived. By omitting the readout of singular nodes the impact of them to the computation capacities might be investigated. This can be used in order to "chase" the importance of certain Nodes for different tasks.

For a system trying to recall the values from a distribution u_1 the totally uncorrelated inputs from u_2 somewhere else in the system will act like noise.

2 Introduction

Reservoir Computing encompasses the field of machine learning in which the capacity to store and perform computations on data is investigated in a wide variety of dynamical systems (reservoirs). It can be understood as a generalization of earlier recurrent neural network architectures echo state networks (ESN) and liquid state machines (LSM). The name Reservoir Computing (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) need rely on the existence of a digital space in which everything is represented by a combination of discrete values (0 and 1).

To create such a virtual space the usual unpredictability that inhabits the scales in which modern computer circuits exist in has to be tamed in order to create this virtual computing space. As the scales of modern transistors shrink they rapidly approach the scales in which quantum effects become problematic. 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 are so small adequate light sources needed to imprint the conduction paths are becoming rare and hideously expensive to operate. We are at the waning end of Moore's Law.

Reservoir computing could circumvent several of today's difficulties by exploiting the physical dynamics directly without the need of discrete values of digital computation. A wide variety of systems can be used e.g. a literal bucket of water can act as a reservoir that performs computations [FER03]. Albeit the most interesting applications lie in potential optical computers. RC offers a way of utilizing the highly complex dynamics of optical systems in order to perform computation on them. Optical Computers in the form of lasers appear to be an ideal application of RC, because of the timescales that laser dynamics. Optical reservoir computers already perform classification tasks on very timescales unmatched by modern silicon electronics [BRU13a] Another expected benefit would be the vastly lower power consumption.

In recent years reservoir computing has received a lot of attention as bridge between machine learning and physics. As the end of "Moore's Law" is slowly encroaching we are in the waning years of an age of staggering performance leaps in silicon electronic circuits. Reservoir Computing as a way of utilizing the much smaller timescales at which optical systems operate offer a way of building drastically faster computers. There

[SAN17a] overview-paper

[LAR12] i fischer.

[ROE18a] paper von André. (has normalization?)

[JAE01] let's not train the networks.

[APP11] - original paper introducint the delay-based reservoir approach.

[ANT19] ? lesen!

[STE20] ? lesen

[ATI00] - NARMA10 task introduction. [**?**] - vanishing gradient. difficult to train RNNs -> don't train them at all. [AMI19] - timeseries prediction of laser dynamics

2.1 blabla

2.2 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 it is able to predict the real-world destination of the stone.

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

The model is clearly wrong.
It does not account for the weight, size, shape of the stone. It does not account of

aerial friction, wind, the stone's rotation (magnus effect) and/or that of the earth (coriolis effect). We might extend the model in order to get a more accurate prediction (see "ballistic motion"). 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 nuclear fusion and fission of air and stone molecules colliding with each other. But still: the model is *useful*, as it is able to approximate the stone's destination *close enough* in most cases. If we need more detail, we can usually use a more detailed model.

Scientists are usually trying to building models that represent certain aspects of the real world in order to infer underlying states or predict outcomes. Meteorologists have a models that approximate the weather, Astronomers have models that predict planetary motions and were in the past able to infer the existence of planets years before final confirmation. 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 piece of cheese in a fraction of a second. He can even apply 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 soccer player will have more emphasis on dynamics of soccer balls. the player's model is more accurate by accounting for wind, magnus effect, surface friction and even ball-grass interactions. 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. *"Being funny"* (i.e. having a great sense of humour) can be understood as a well working internal model of other people's thought processes.

Intelligence is the *ability to build models*.

Applied to machine learning, this means a more intelligent algorithm is able to more quickly build a usefull model of some benchmark data. This build process can be seen as learning. As an example: 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.

A great benefit of learned models is often that they need less resources in order to make predictions.

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 - predicting tomorrows weather patterns. The applications of extracting information from timeseries are vast. Predicting the stock market, driving an autonomous car based on a video stream and other vehicle metrics, recognizing types of heart arrhythmia or more general heart diseases and infections through analysis of electrocardiography by machine learning algorhithms (https://ieeexplore.ieee.org/abstract/document/716478
Last, but not least: the most apparent timeseries prediction application to this date: voice recognition algorithms are predicting the meaning of audio information everytime we utter the magical 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 tend to investigate a still image by looking at different parts not at once, but one after another, concentrating on details one after another. For areas where meaning is not immediately obvious we tend to "take another look". So even classical machine learning tasks e.g. image recognition tasks can be adapted as timeseries prediction tasks.

2.3 An analogy for reservoir computing

Let us imagine a pond. If we throw stones into the pond, they will make splashes and create concentric waves that propagate over the surface. If we miss the splash and therefor don't know the position of impact, could we still estimate where the stone landed two seconds prior? Could we estimate it three seconds prior? Or 20 seconds prior? The answers would be yes, probably yes, possibly yes 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' amplitude gets smaller. The pond has some kind of fading memory. Additionally throwing similar stones at comparably similar positions of the pond will create similar wave patterns. The response of the pond to the stone's impact is not random. From the pattern of waves other parameters might be extracted as well provided we have seen enough throws of stones with the corresponding wave patterns. We might estimate size or weight, velocity, spin, angle and even shape of the stone just by looking at the waves.

Another counterintuitive possibility is the extraction of information that was *not* inserted via the stones. For example given we might extract the product of the weights of two consecutive thrown stones by analyzing the interfering waves that originated at each of the impact regions.

To understand the idea of reservoir computing one can imaging a pond of water. Now a stone is thrown into, creating ripples that propagate over the surface, reflect along edges and interfere with one another. The pond has some kind of short memory as ripples take time in order to disappear. It is easy to imagine that from the pattern of ripples a spectator could estimate the position of a stone that was thrown in shortly before.

3 Used symbols

Table of used symbols

$z(t)$ / $z_i(t)$	state of stuart landau oscillator (global state / individual)
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 i.e. number of virtual nodes per real node
θ	virtual node time
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 $\mathbf{x}$. $D = N_r \times N_v$ (sometimes $N_r \times N_v + 1$)
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 - the sequence of all previous inputs
$u(t - nT)$ / u_{-n}	the value of u at n clockcycles before time t
$\mathbf{u}_{-n}$	the sequence/vector $\{u(-nT), u(1 - nT), \dots, u(K - nT)\}$
$o(t)$ / $\mathbf{o}$	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 $\mathbf{u}$
$L_{d_1,d_2,\dots}(u_{-n_1}, u_{-n_2}, \dots)$	a product of several Legendre polynomials with degrees $d_1, d_2, \dots$ and arguments $u_{-n_1}, u_{-n_2}, \dots$
$\mathbf{L}_{d_1,d_2,\dots}(\mathbf{u}_{-n_1}, \mathbf{u}_{-n_2}, \dots)$	sequence given through element wise multiplication of sequences $\{\mathbf{L}_{d_1}(\mathbf{u}_{-n_1}), \mathbf{L}_{d_2}(\mathbf{u}_{-n_2}) \dots\}$
f / $f(\mathbf{u}^{-\infty})$	benchmark transformation. The transformation we try to reconstruct through the dynamics of our reservoir.

4 Theory

4.1 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). Where RNNs are usually still networks in a compute, 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 a digital computer.

A reservoir computer needs to exhibit 3 properties in order to perform computations in a meaningful i.e. useful way:

- the echo state property - the system state is defined completely by all previous inputs fed into the system
- similar inputs create similar outputs
- it has to contain a non-linearity

The state of the reservoir is only defined by the previous input. There are no other unknown variables changing its state. The reservoir's response to similar input is similar. Identical inputs give identical outputs.

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 usually tested on time-structured data. Benchmarks try to quantify the capacity to remember information fed in at earlier times and its ability to make nonlinear transformation of the data.

The used benchmarks $f(\mathbf{u}) = \mathbf{o}$ have an *input sequence* $\mathbf{u}(t)$ and a *target sequence* $\mathbf{o}(t)$ with $t \in \mathbb{Z}$ being the time counted in input periods T (often called "clockcycles" - cc). The input period T is the time during which one data point u (also referred to as one "sample") is fed into the network. For a specific time t the reservoir offers D outputs which are combined into the readout vector $\mathbf{x}(t) \in \mathbb{R}^D$. This readout vector can be linearily transformed in order to reach some *predicted value* $\hat{o}(t)$. Ideally $\hat{o}(t) = o(t)$ which would represent a perfect reconstruction of the target value 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 easy and fast.

4.1.1 Reservoir computing tasks

Typically reservoir computing tasks involve transformations on data sequences.

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 a hidden value. A weather model which has been fed past temperature data can be tasked to "predict" (i.e. approximate) the corresponding humidity values. In machine learning 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 base truth and the difference between prediction and ground truth quantifies the error. The closer the prediction matches the ground truth, the better the system performs a given task.

It is important to note that usually these predictions are not of discrete values. Instead of binary truth values machine learning systems operate in the realm of "fuzzy logic" i.e. *partial truths*. For example an image classification system will usually not return a clear answer e.g. "this picture shows an elephant". Instead answers are given as a vector of probabilities. This vector will have entries quantifying the 'dog-ness', the 'tree-ness' or the 'car-ness' of an input image. 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

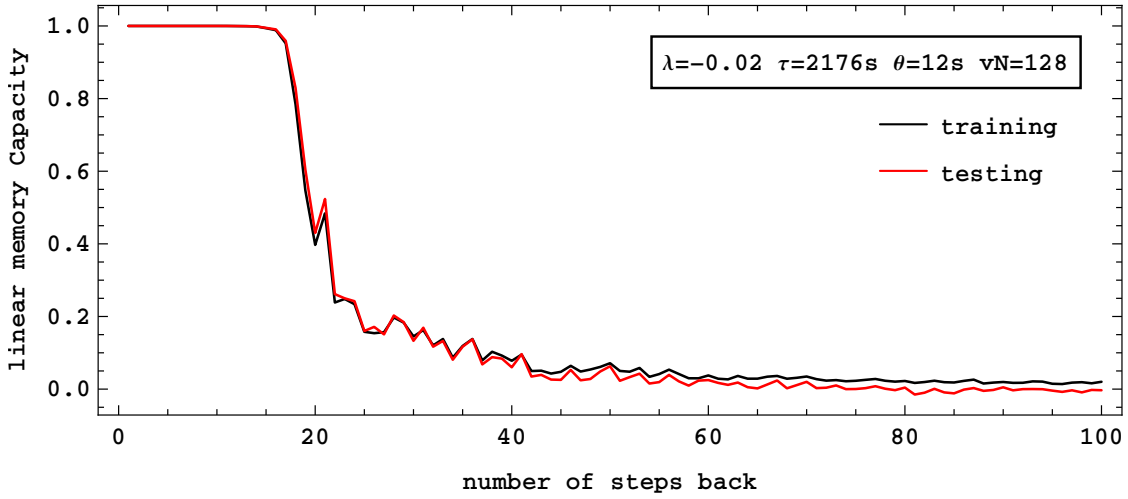

Figure 1: The linear memory capacities for varying steps into the past. The system is able to perfectly reproduce inputs up until 12 steps into the past. $N = 1, vN = 128, \lambda = -0.02, \omega = 1, \gamma = -0.1, \theta = 12, \tau = 2176$.

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 expressing a tendency.

In this work a sequence of inputs u is drawn from a uniform distribution. In this work only uniform distributions have been used. This sequence is used as input of a transformation which maps the sequence u onto its target values o.

4.1.2 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 previously we will write our target sequence $\mathbf{o} = \mathbf{u}_{-2} = \{u(t-2), u(t-2+1), u(t+2), \ldots, u(t-2+K)\}$

$$\begin{aligned} o_1 &= u_{-1} \\ o_2 &= u_{-2} \\ o_3 &= u_{-3} \\ &\vdots \\ o_n &= u_{-n} \end{aligned} \tag{2}$$

4.2 Legendre polynomials as Nonlinear Transformations

In order to investigate the nonlinear transformation capabilities one can use Legendre polynomials (Fig. 3) in order to transform inputs $\mathbf{U}(t)$. Legendre Polynomials have the useful property of being orthonormal to every other Legendre polynomial within an interval $[-1, 1]$. This makes them highly useful for measuring linearily independent nonlinear (but also linear) transformation capacities. Legendre polynomials $L_d(x)$ for degrees $d \in \{1-5\}$ are shown in fig.3. Depending on the definition they are scaled so that $L_d(1) = 1$. The Legendre polynomial $L_1(x)$ is of course simply the identity, which makes it possible to investigate the linear memory capacity also.

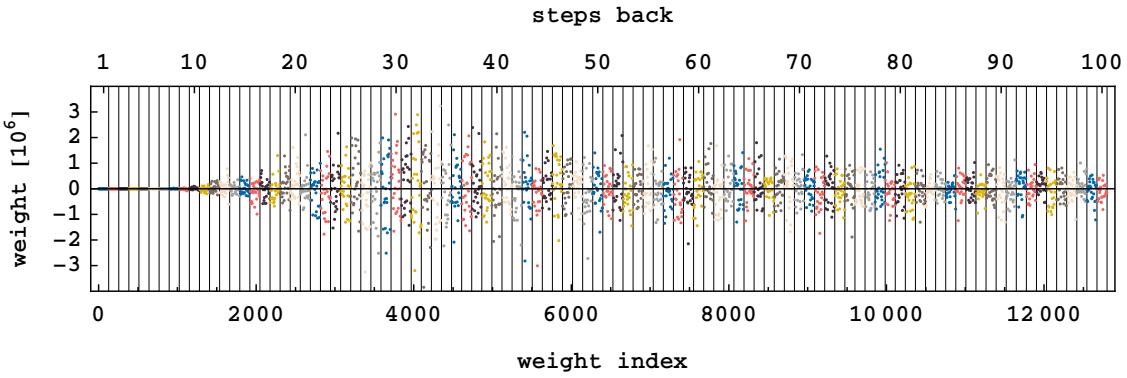

Figure 2: All weights $W_{i,s}$ attained through linear regression of the linear memory recalls $s \in [1, 100]$ steps back. The system was a unidirectional ring with of $N_{real} = 8$ and $N_{virtual} = 16$ nodes. The total read-out dimension is 128. For reconstruction of more recent inputs the inputs 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 $[-1, 1]$.

The idea of measuring nonlinear (as well as linear) transformation capacities through the means of Legendre polynomials is largely inspired from the publication [DAM12]. Hence the terminology used in the latter shall be used here as well (where possible).

We start by defining $u^{-h}(t) = (u(t-h+1), \ldots, u(t))$ as the sequence of h previous inputs. We start with an input $\mathbf{u}$ which more precise will be u^{-h}

$$f : U^h \to \mathbb{R} : u^{-h} \to f\left[u^{-h}\right] \tag{3}$$

A function f transforms inputs

All nonlinear transformations of an input sequence $\mathbf{U}^h$ can be expressed as a linear combination of Legendre polynomials with h being the number of data points already fed into the system. With inputs in $\mathbf{U}^h$ it is necessary to test all combinations. To elaborate: For a singular Legendre polynomial $L_{d_1}(\mathbf{U}(t_1))$ with degree d_1 all values of t_1 have to be tested. For the product of 2 Legendre polynomials $L_{d_1}(\mathbf{U}(t_1))$ and $L_{d_2}(\mathbf{U}(t_2))$ with degrees d_1 and d_2 all combinations of $\mathbf{U}(t_1)$ and $\mathbf{U}(t_2)$ have to be calculated. For the product of 3 Legendre polynomials $L_{d_1}(\mathbf{U}(-t_1))$, $L_{d_2}(\mathbf{U}(-t_2))$ and $L_{d_3}(\mathbf{U}(-t_3))$ all combinations of t_1, t_2, t_3 have to be tested.

So for the Product of n Legendre polynomials there exist permutations $\mathbf{d} = d_1, d_2, ...d_n$ of.

4.2.1 How to sum the right capacities

In order to investigate the different nonlinear transformation capabilities of a system it is necessary to iterate over all possible combinations of Products of Legendre polynomials - an exercise in futility as the number of combinations is infinite. A number of restrictions need be applied to reduce number of calculations. A few (reasonable) assumptions are made:

- capacities converge against zero for long recall distances -see Fig1
- capacities for higher degrees converge against zero.

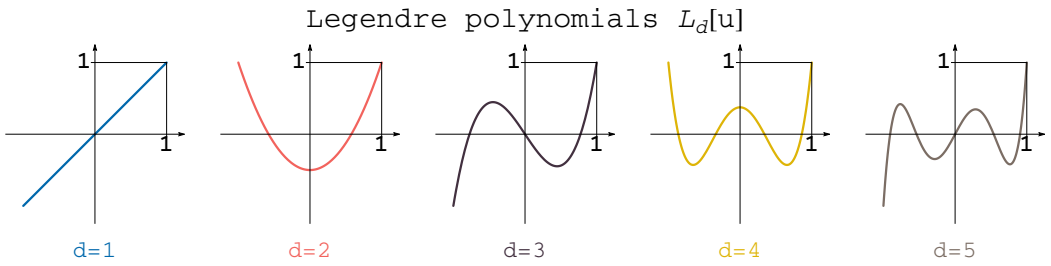

Figure 3: Legendre polynomials $L_d(u)$ for degrees $d \in \{1-5\}$ each shown for $u \in [-1, 1]$.

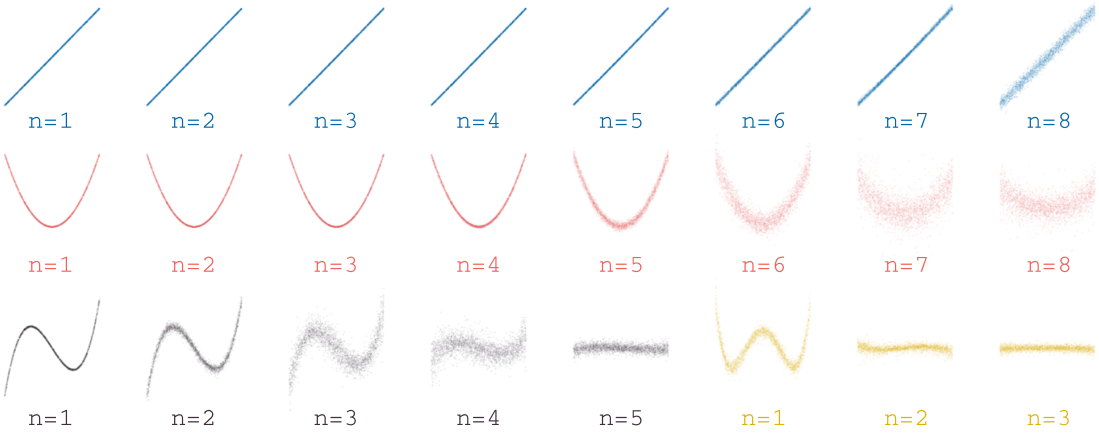

Figure 4: Using only the read-out state of the reservoir $\mathbf{x}(t)$ with some task-specific weight vectors $\mathbf{w}_{task}$ it is to an extent possible to predict the values $L_d(u(t-nT))$ with n being the number of steps into the past. Here for different tasks and with different numbers of steps into the past 2500 predictions based off $\mathbf{x}(t-nT)$ are plottet over values $u(t-nT)$. Reconstructing transformations on inputs further steps n into the past results in poorer reconstructions. For higher degrees d the results' qualities are also decaying faster - for L_4 (yellow) the reconstructions only really works for $n = 1$. The colors of the different tasks are in accordance to Fig 3.

But as we are not working in the limit of $K_{training} \to \infty$ the calculated capacities are subjected to noise. –see fig something. Summing over large amounts of noise results in too high capacities that can largely exceed the number of the theoretically limit which is the readout dimension. We avoid this by choosing the lowest possible threshold above noise level in order disregard all capacities beneath. For different lengths the distributions can be seen in Fig.5. Accumulating capacities for different degrees $CM^1, CM^2, CM^3, ...$ is complicated through the existence of noisy capacity values.

They are used in Neural networks as well blabla citecite findfind.

4.2.2 NARMA10 - the Nonlinear Autoregressive Moving Average Task

$$A_{n+1} = 0.3A_n + 0.05A_k \left(\sum_{i=0}^{9} A_{k-9} \right) + 1.5u_{k-9}u_k + 0.1 \tag{4}$$

Lastly, the performance was 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 is calculated using an average of its last 10 steps while also being fed a product of a sequence $u(t)$ taken at two different positions.

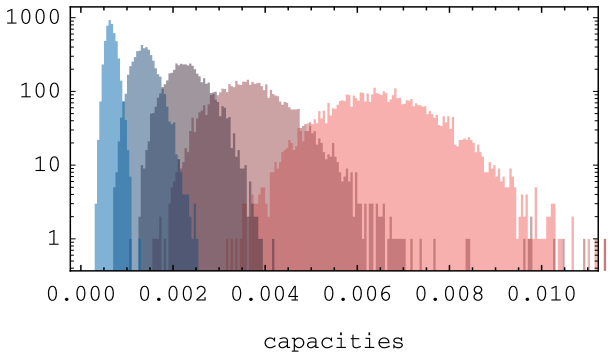

Figure 5: Varying distributions of the last 5000 capacities (noise) are shown. From right to left the distributions resulted from $K_{training} = \{10^4, 2 \cdot 10^4, 3 \cdot 10^4, 5 \cdot 10^4, 10^5\}$.

In order to calculate the NARMA10 sequence one needs memory 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 quickly grow to ∞ when fed a sequence u with a particularly large recent average. 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 by 4. It is fed 2 inputs from a sequence u which is drawn from a uniform distribution $U \in [0, 0.5]$.

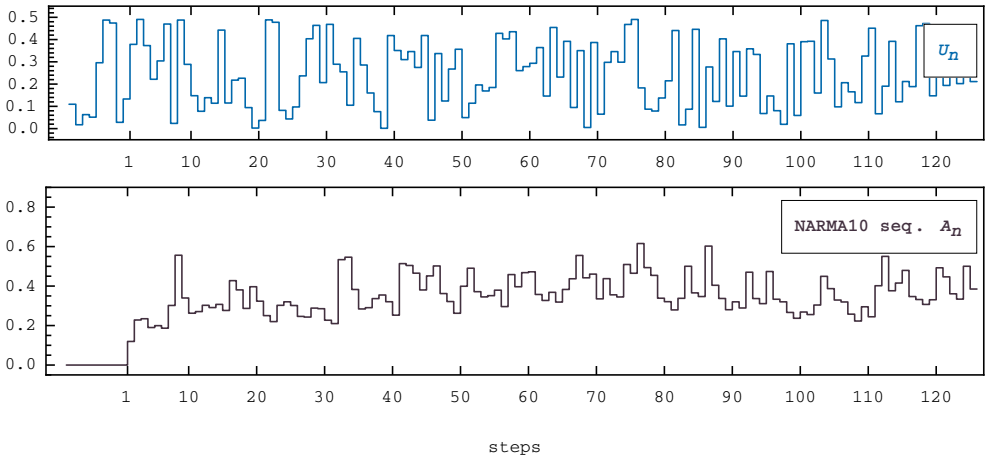

Figure 6: Example for a uniformly drawn random sequence $U_n \in [0, 0.5]$ that is used to create a Narma10 sequence A_n with it. Here $A = 0$ for $n < 1$.

4.3 Stuart-Landau-Oscillator

4.3.1 uncoupled version

The Stuart-Landau oscillator is a dynamical system often used to model basic class 1 lasers i.e. laser systems that do not exhibit pulsing or chaotic behavior (some refs). It can be written either as a single complex differential equation (5) or a set of two equations written in polar coordinates (6). From the equation in polar coordinates easy to see that the equation has rotational symmetry as the radial differential equation does not change with the dynamical variable ϕ. The Parameter λ is often called the *pump current* in

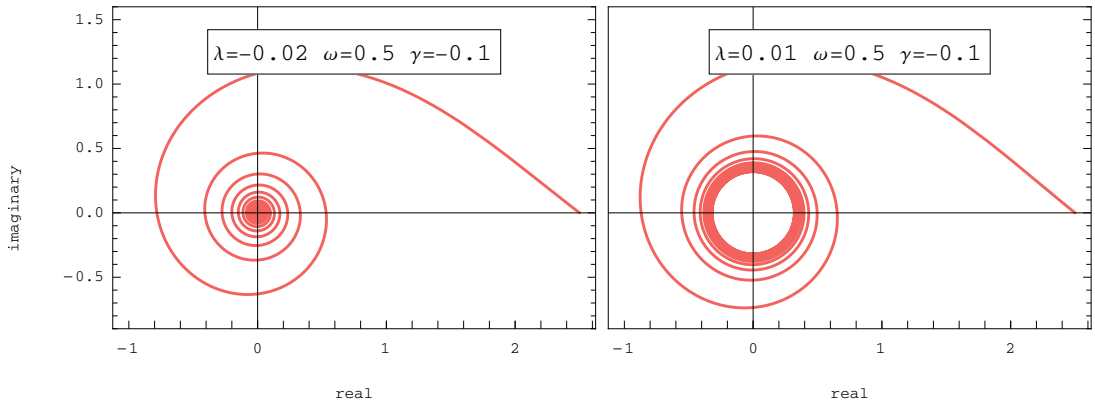

Figure 7: 2 very basic scenarios of the Stuart-Landau oscillator: Decay towards a single fixed point (off-state, left) or towards a stable oscillating state (limit cycle, right).

accordance to its practical meaning.

$$\dot{z} = (\lambda + i\omega + \gamma|z|^2)\, z \tag{5}$$

$$\begin{aligned} \dot{r} &= \lambda r + \mathrm{Re}(\gamma)\, r^3 \\ \dot{\phi} &= \omega + \mathrm{Im}(\gamma)\, r^2 \end{aligned} \tag{6}$$

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 $\mathrm{Re}(\gamma)$. For $\mathrm{Re}(\gamma) < 0$ (supercritical case).

4.3.2 Stuart-Landau with delay coupling

Introducing a coupling delay — a "feedback loop" — drastically increases complexity of the dynamic system. The phase-space dimension for a single delay-coupled Stuart-Landau oscillator (Fig.:9 with $G = 0$) becomes infinite. The reason for this instantaneous growth of complexity is that the initial condition $z_o = z(t_0)$ now becomes a function defined over the the interval $[t_0 - \tau, t_0]$. As the number of possible functions defined over $[t_0 - \tau, t_0]$ is ∞, so becomes the dimension of phase space.

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 through increasing the delay time τ. In [YAN09, CHO09] multistability in delay systems are investigated, specifically for Stuart-Landau oscillators. The method is adapted to fit in this work and to account the value $\gamma < 0$ and for coupling strength κ and coupling phase ϕ. Let us consider the rotating wave solution $z(t) = re^{i\Omega t}$ for 9 (with $G = 0$, $Im(\gamma) = 0$):

$$z(t) = re^{i\Omega t} \tag{7}$$

substituting z in 9 we gain:

$$i\Omega = (\lambda + i\omega + \gamma r^2) + \kappa e^{\phi - i\Omega\tau} \tag{8}$$

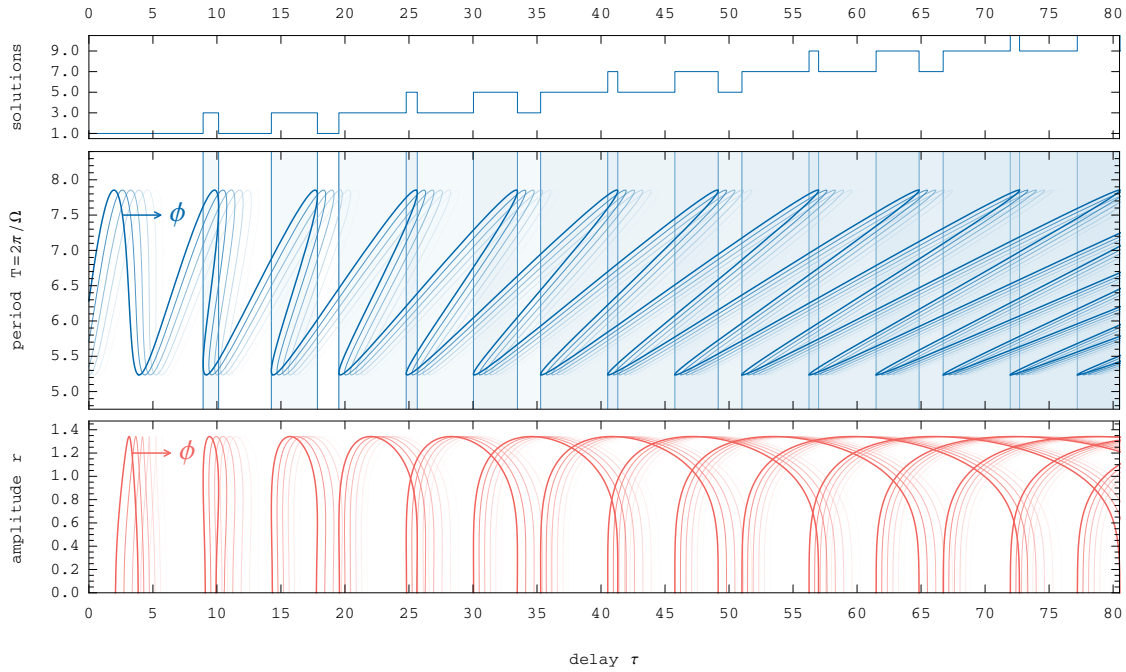

Figure 8: The solutions of (9) with $\lambda = 0.02$, $\omega = 1$, $\gamma = -0.1$,$\kappa = 0.2$, $\phi = 0$ (solutions for $\phi > 0$ are hinted). (top) Larger τ result in greater number of solutions. (middle:) relationship of τ and period T of a rotating wave solutions. Areas of multiple solutions are colored lightly blue. For greater τ these areas increasingly overlap resulting in higher multistability. (bottom) parts where solutions $r \in \mathbb{R}^+$ exist (physical solutions with $r > 0$). For higher τ multiple amplitude levels are possible depending on the system's state.

4.3.3 Variing pump current

The limit cycle (LC) which is shown in fig 7 is depending on the ratio or λ and $Re\,[\gamma]$.

As can be seen in (eq. 5), the equation has a linear and a nonlinear term regarding the absolute value of z.

$$\dot{z} = (\lambda + GJ(t) + i\omega + \gamma|z|^2)\, z + \kappa e^{i\phi} z(t-\tau) \tag{9}$$

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 our case $\hat{\lambda}(t)$ will always be an offset with only slight variations. We will therefore simply write $\hat{\lambda}(t) = \lambda + GJ(t)$ with $G \in \mathbb{R}$ being in the order of or exactly equal to 0.01 (if not stated otherwise). The input signal $J(t)$ will be the actual variation that is used to feed data into the system (see Fig. 15).

In Fig. 10 a solution for $|z(t)|$ with and without delayed feedback is shown. Without it the intensity is exponentially decaying towards the new limit cycle (in black). With delayed feedback (bottom) previous inputs reappear: 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 a linear combination of all given readouts.

4.4 Networks

Vertices blabla Edges blabla.

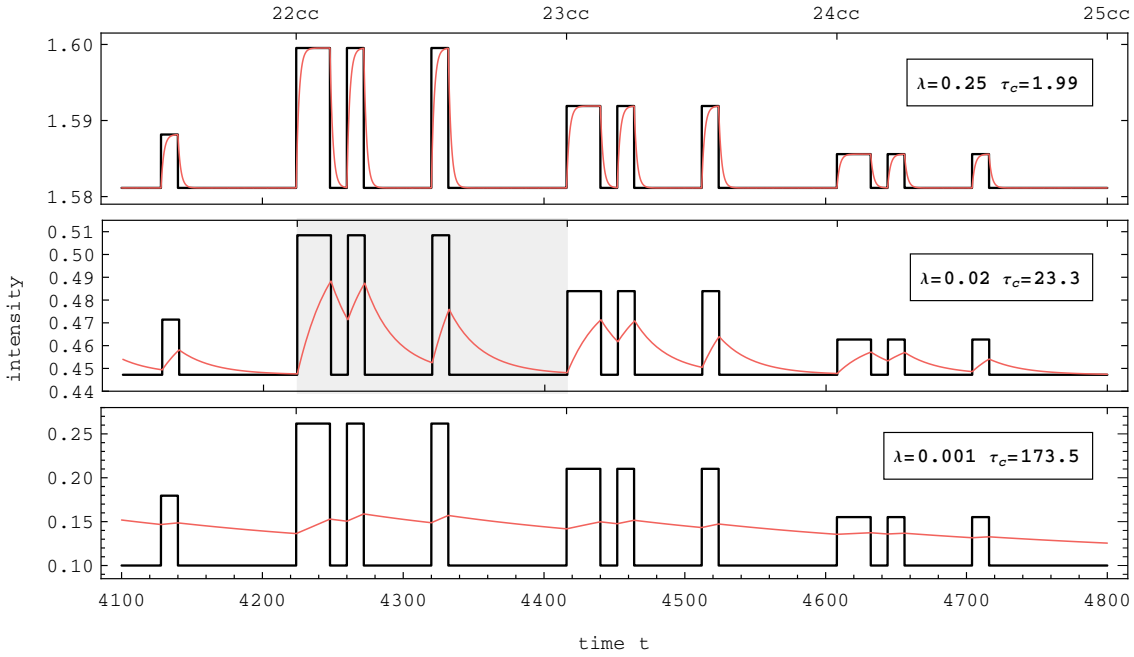

Figure 9: Different values for λ influence how fast the system's intensity ▬ is responding to a given input signal. For a large value of $\lambda = 0.25$ (top) the system's intensity almost instantaneously decays towards the expected limit cycle LC ▬. The system is therefore determined exclusively by 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 $t \in [4332, 4400]$.

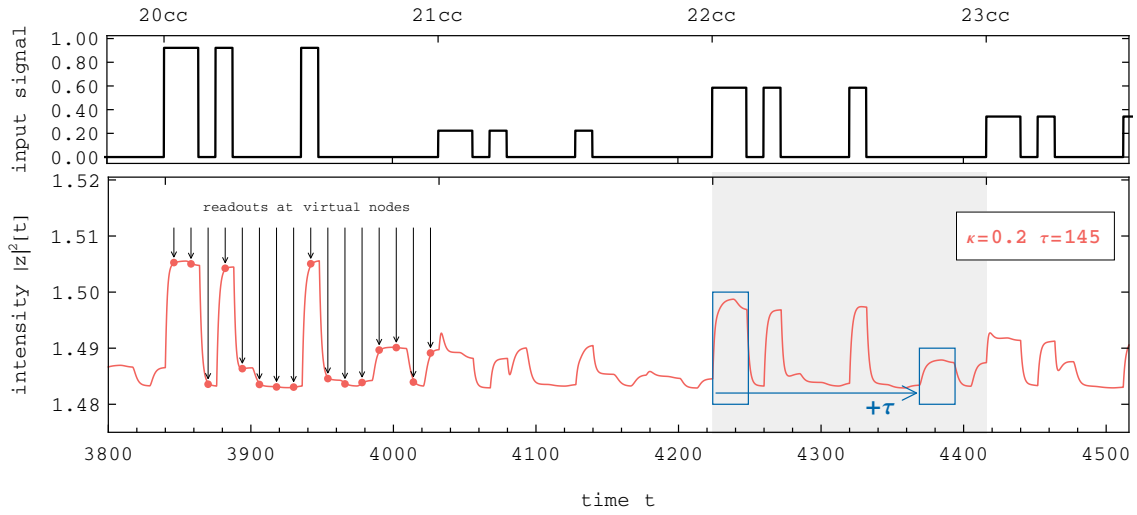

Figure 10: $|z|^2(t)$ of a driven Stuart-Landau oscillator with delayed feedback. Each clockcycle the system state is read out once per virtual node (red dots on the left). With delayed feedback patterns reappear after delay time τ: the bump within the right blue rectangle is not caused by the input 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.9

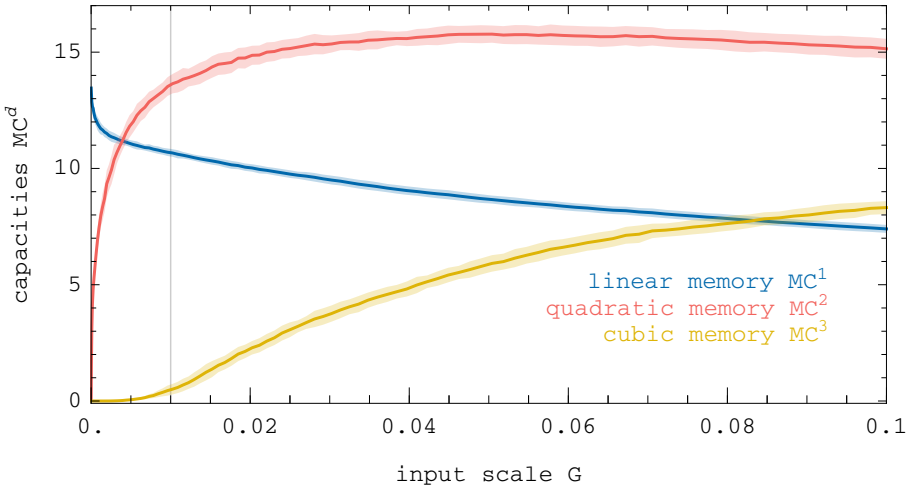

Figure 11: Capacity means and standard deviations from 20 random realizations per input scale G respectively (variing masks/initial state). G strongly determines the reservoir performance of different tasks. Parameters: $N_r = 1, \lambda = -0.02, \omega = 1, \gamma = -0.1, \tau = 68, N_v = 64, \theta = 3/4, T = N_v\theta = 48$. In this work an input scaling of $G = 0.01$ (indicated as vertical line) was used at all times.

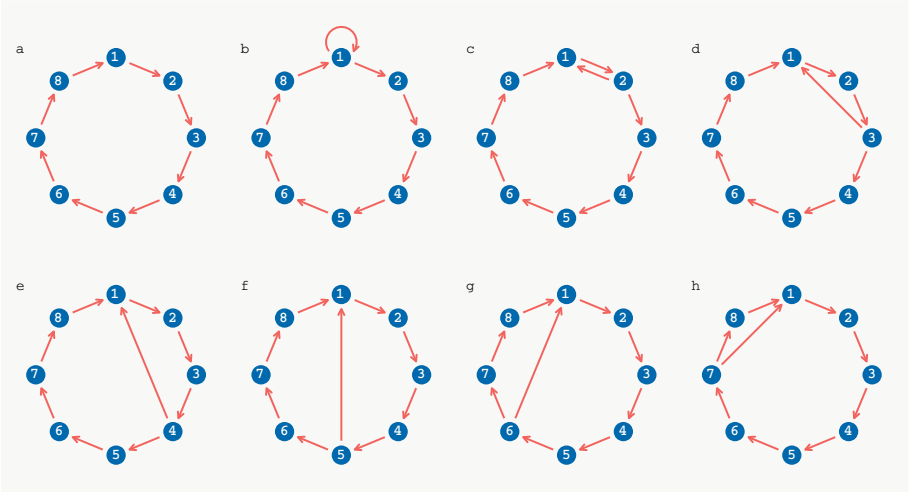

Figure 12: example graph

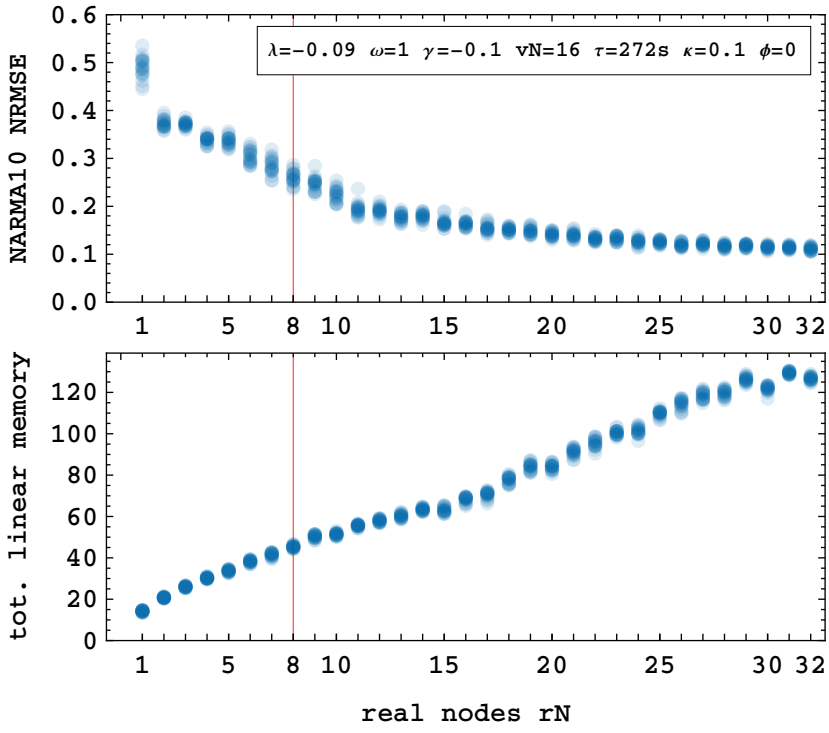

Figure 13: changing rc performance for increasing number or real nodes rN in unidirectianally coupled ring networks. (see some plot).

4.4.1 circulant Matrix

A circulant matrix has the same entries its row vectors, but with its entries rotated one element to the right relative to the previous row.

4.5 virtual Nodes and multiplexing

idea of multiplexing originally introduced by Appeltant et al. in [APP11]. originally with binary mask (on/off), variations are uniform masks or noise masks. here: papers for explanation! [KUR18]? [APP14] comparison of masks (binary<uniform<noise(with high cutoff)) [STE20] // off-resonance = better! –> reason for choosing 17 * 12.

By multiplexing the input signal one can create virtual nodes in a network. The analogy to a real network can be best understood if the input signal is masked with a binary mask containing only values of either 0 or 1.

Different Mask types: discrete values with constant interpolation: binary, uniform - easy to implement) continuous: any function or repeating noise-patterns. (more difficult, but)

here add dependency of total linear memory on number of nodes and virtal nodes.

4.6 Dynamics of rings of stuart landau oscillators

pony-states (von André)

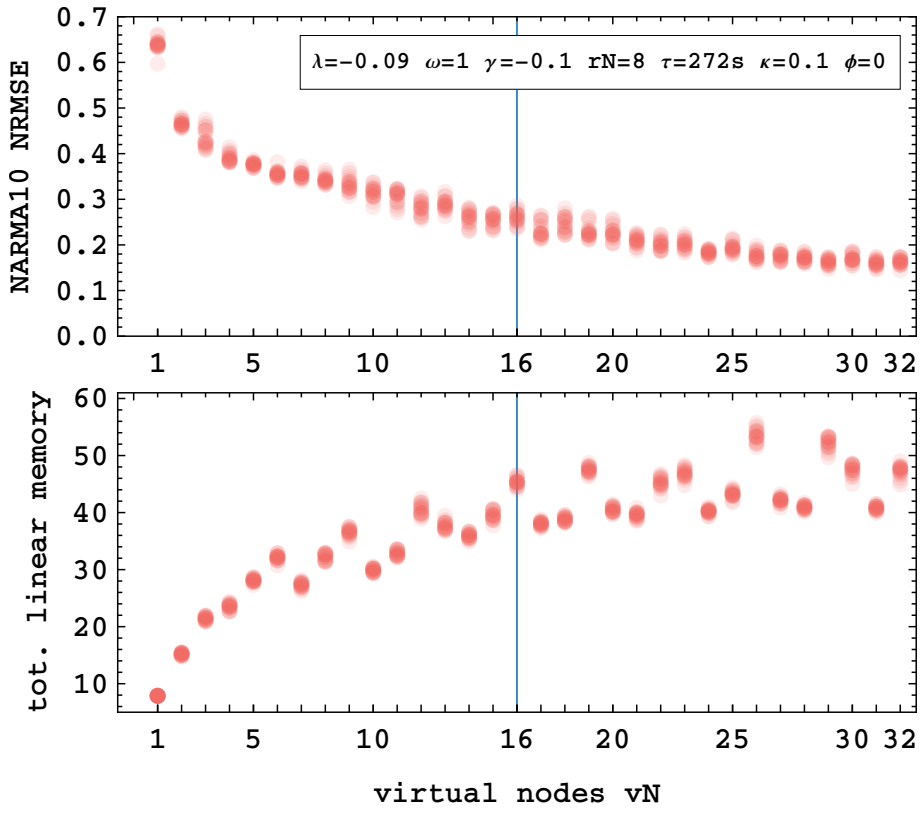

Figure 14: changing rc performance for increasing number or virtual nodes rN in unidirectionally coupled ring networks. (see some plot).

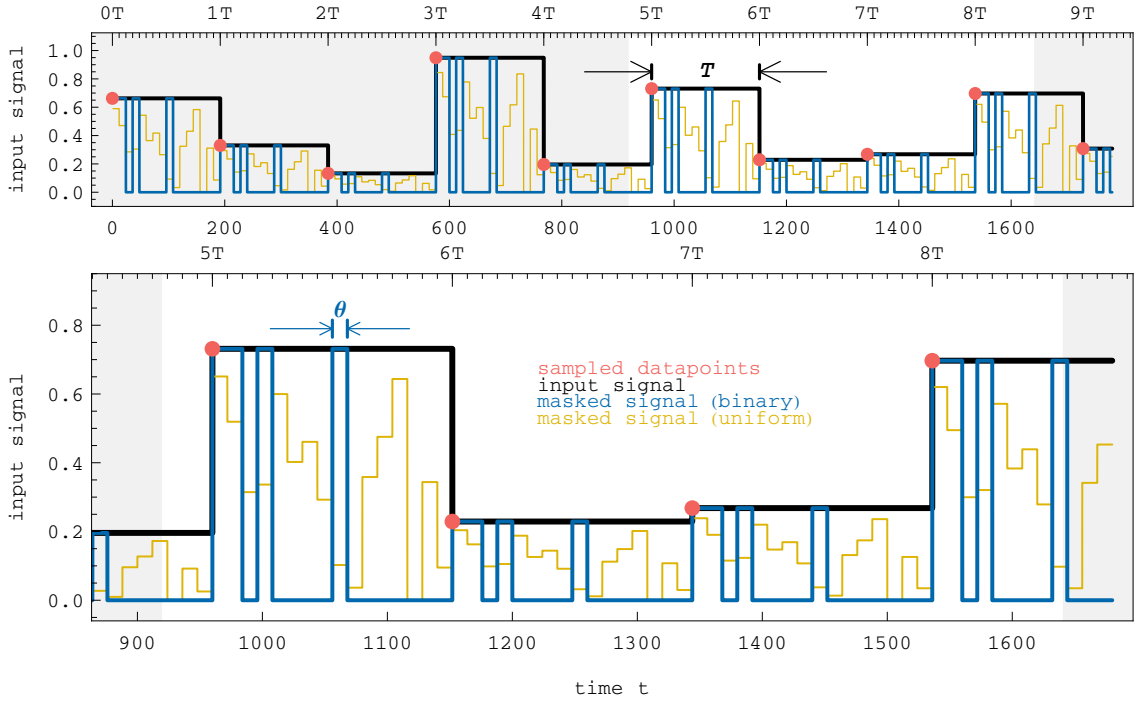

Figure 15: A timeseries of data points ● with its constant-interpolated signal between samples ━ and the corresponding masked signals ━ and ━. The mask times (or lengths) are usually defined as input period T or clockcycle (indicated T, upper major ticks) and the time per virtual node called θ (upper minor ticks). Here $\theta = 12$ and $T = 16\theta = 272$

$$S = \begin{pmatrix} x_1^1 & \dots & x_1^D & 1 \\ \vdots & & \vdots & \vdots \\ x_K^1 & \dots & x_K^D & 1 \end{pmatrix} = \begin{pmatrix} x_1^1 & \dots & x_1^K & 1 \\ \vdots & & \vdots & \vdots \\ x_1^1 & \dots & x_1^K & 1 \end{pmatrix} \tag{11}$$

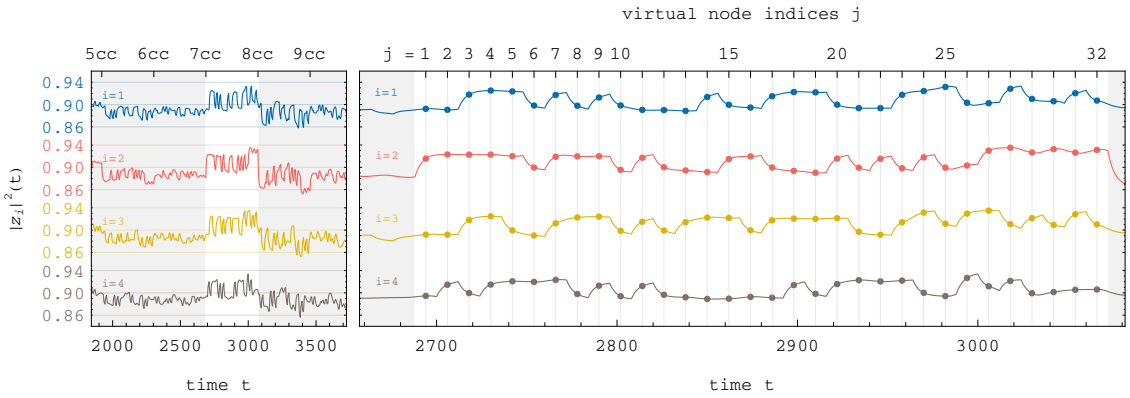

Figure 16: The traces $|z_i|(t)$ with $i = 1, 2, 3, 4$ (●●●●) of a system with $N_r = 4$ nodes and virtualization factor $N_v = 32$. The readouts $\mathbf{x}(7T)$ for input period $T = [7cc, 8cc]$ are indicated by the dots on the right. All dots together make one row in the state matrix $\mathbf{S}$.

4.7 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 both tasks were made using the same sequence $u \in [-1, 1]$. For NARMA10 the sequence u was mapped with the function m (10) onto $m(u)$.

$$m : [-1, 1] \to [0, 0.5] : u \to (u + 1)/4 \tag{10}$$

4.8 mathematical details

Calculating memory capacities can be either done through solving a the system of linear equations or by calculating the covariances of targets and predictions.

4.8.1 integration

Integration was through the Runge Kutta method of fourth degree with a fixed stepsize. Since we are dealing with delay differential equations, we run into a problem when as the delayed state $z(t - \tau)$ is only available as discrete values. In order to calculate k_2 and k_3 an interpolated value for $z(t - \tau + h/2)$ 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. In general [FIL89]

[NEV76] // mabye not

Solving a system of linear equations. Design matrix or State matrix S.

Meaning of values inside the design matrix S.

$$x_{ij}^{k} = |z_i(kT + j\theta)| \tag{12}$$

$$\mathbf{S} = \begin{pmatrix} \bullet_1^1 \bullet_1^1 \bullet_1^1 \bullet_1^1 \bullet_2^1 \bullet_2^1 \bullet_2^1 \bullet_2^1 \cdots \bullet_{N_v}^1 \bullet_{N_v}^1 \bullet_{N_v}^1 \bullet_{N_v}^1 \\ \bullet_1^2 \bullet_1^2 \bullet_1^2 \bullet_1^2 \bullet_2^2 \bullet_2^2 \bullet_2^2 \bullet_2^2 \cdots \bullet_{N_v}^2 \bullet_{N_v}^2 \bullet_{N_v}^2 \bullet_{N_v}^2 \\ \vdots \\ \bullet_1^k \bullet_1^k \bullet_1^k \bullet_1^k \bullet_2^k \bullet_2^k \bullet_2^k \bullet_2^k \cdots \bullet_{N_v}^k \bullet_{N_v}^k \bullet_{N_v}^k \bullet_{N_v}^k \\ \vdots \\ \bullet_1^K \bullet_1^K \bullet_1^K \bullet_1^K \bullet_2^K \bullet_2^K \bullet_2^K \bullet_2^K \cdots \bullet_{N_v}^K \bullet_{N_v}^K \bullet_{N_v}^K \bullet_{N_v}^K \end{pmatrix} \tag{13}$$

$$\mathbf{S}_{\neg\bullet} = \begin{pmatrix} \bullet_1^1 \bullet_1^1 \bullet_1^1 \bullet_2^1 \bullet_2^1 \bullet_2^1 \cdots \bullet_{N_v}^1 \bullet_{N_v}^1 \bullet_{N_v}^1 \\ \bullet_1^2 \bullet_1^2 \bullet_1^2 \bullet_2^2 \bullet_2^2 \bullet_2^2 \cdots \bullet_{N_v}^2 \bullet_{N_v}^2 \bullet_{N_v}^2 \\ \vdots \\ \bullet_1^k \bullet_1^k \bullet_1^k \bullet_2^k \bullet_2^k \bullet_2^k \cdots \bullet_{N_v}^k \bullet_{N_v}^k \bullet_{N_v}^k \\ \vdots \\ \bullet_1^K \bullet_1^K \bullet_1^K \bullet_2^K \bullet_2^K \bullet_2^K \cdots \bullet_{N_v}^K \bullet_{N_v}^K \bullet_{N_v}^K \end{pmatrix} \tag{14}$$

$$\mathbf{S}_{\bullet} = \begin{pmatrix} \bullet_1^1 \bullet_2^1 \cdots \bullet_{N_v}^1 \\ \bullet_1^2 \bullet_2^2 \cdots \bullet_{N_v}^2 \\ \vdots \\ \bullet_1^k \bullet_2^k \cdots \bullet_{N_v}^k \\ \vdots \\ \bullet_1^K \bullet_2^K \cdots \bullet_{N_v}^K \end{pmatrix} \tag{15}$$

14 maybe not as figure ... The readouts x_{ij}^k are performed during the numerical integration of $\mathbf{z}(\mathbf{t})$ and saved in the statematrix $\mathbf{S}$. Here the nodes $i = 1, 2, 3, 4$ (●●●●) are colored according to the example in Fig.16. 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. If we only select readouts from node ● we derive a new statematrix $\mathbf{S}_{\bullet}$. By omitting readouts from ● (here $i = 2$) we get $\mathbf{S}_{\neg\bullet}$.

5 Results

5.1 ring networks

5.2 Highly symmetrical network topologies

- unidirectional ring
- bidirectional ring
- diffuse coupling (bidirectional coupling with equalizing self-links)
- all-to-all coupling including self-links

Rings with from N=1(edge case) to N=16 Bidirectional Rings Bidirectional Rings with self-feedback and diffuse coupling

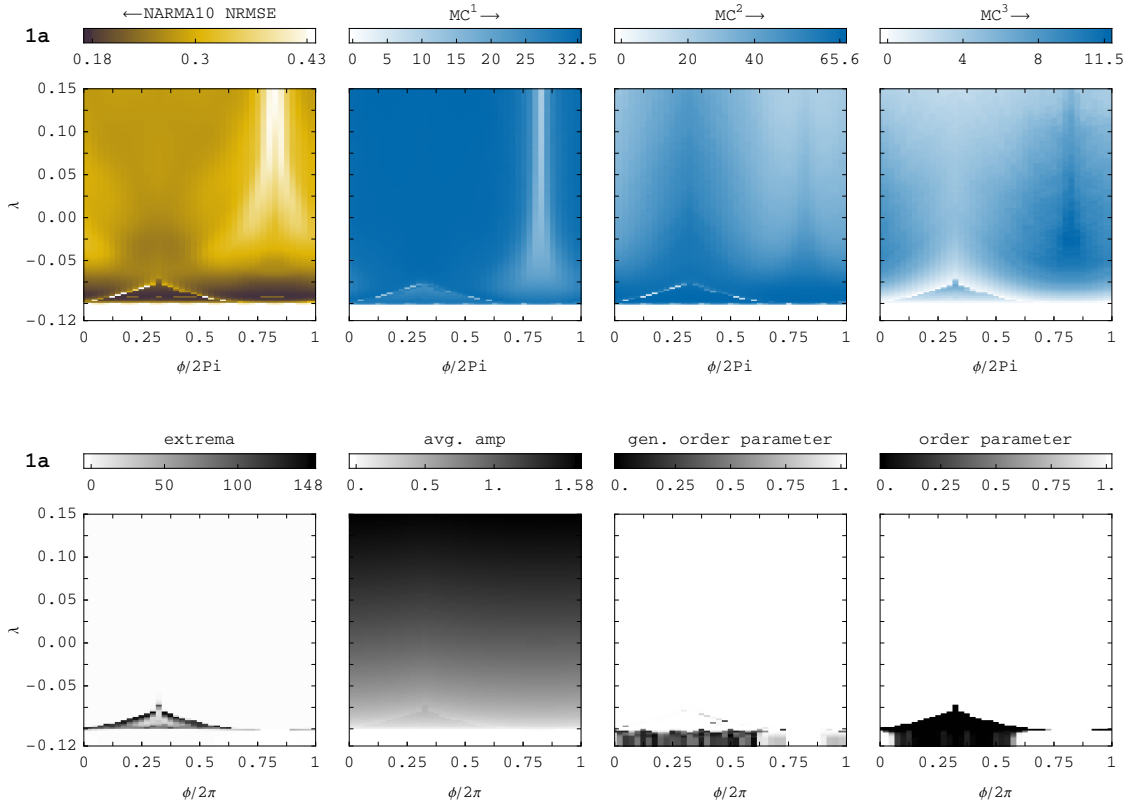

Figure 17: $N_r = 8$: the unidirectional ring (network **1a** in Fig.:**??**). NARMA10 error, memory capacities MC^d and dynamical indicators are shown over λ and coupling phase ϕ with $\tau = 68$ $\theta = 3$ $\kappa = 0.1$. Arrows indicate greater performance.

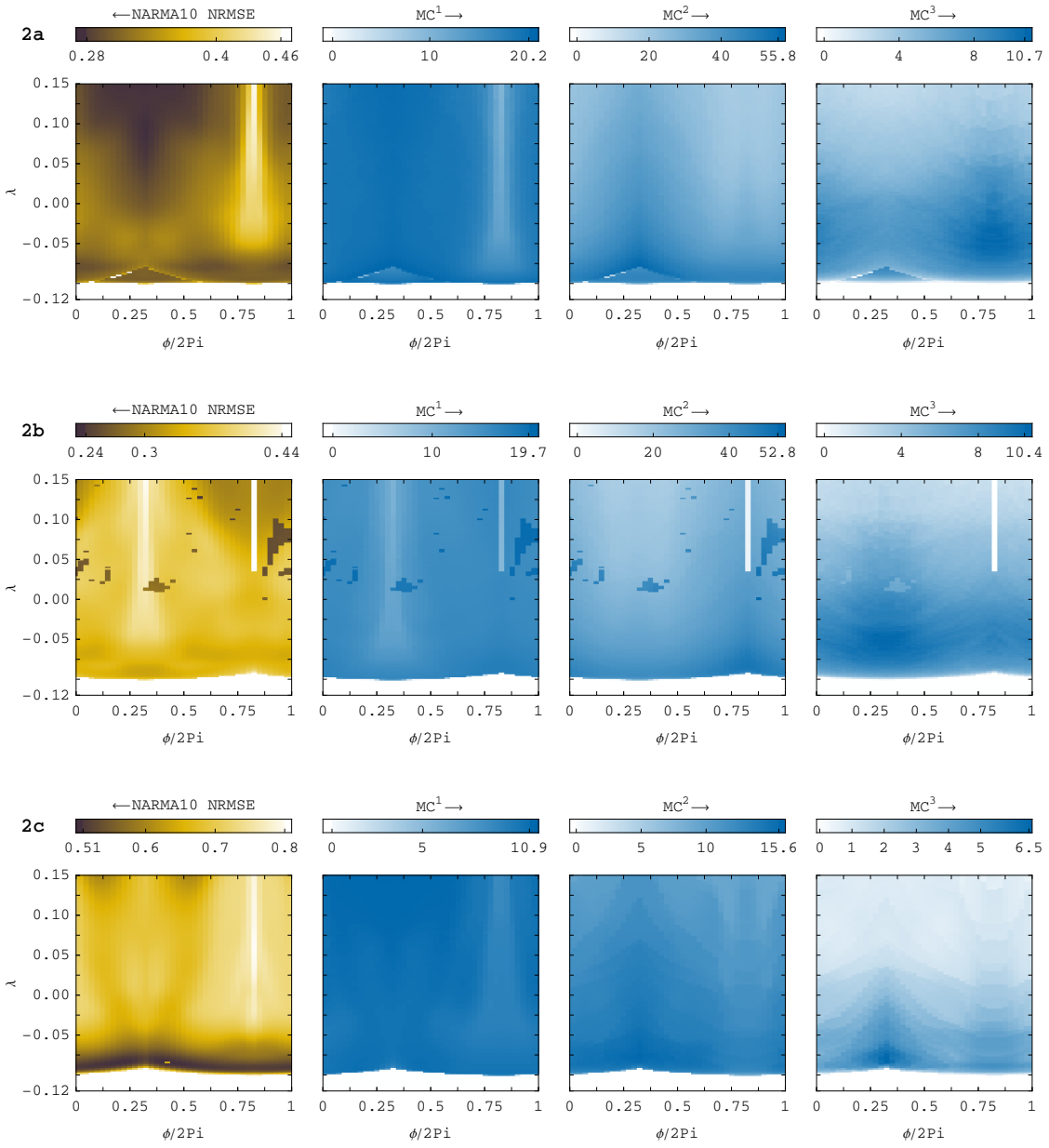

Figure 18: $N = 8$: several highly symmetric (ring-)topologies: bidirectional ring **2a**, diffuse-coupled ring **2b** and the complete graph **2c**. Other parameters: $\tau = 68$ $\theta = 3$ $\kappa_i = 0.1$)

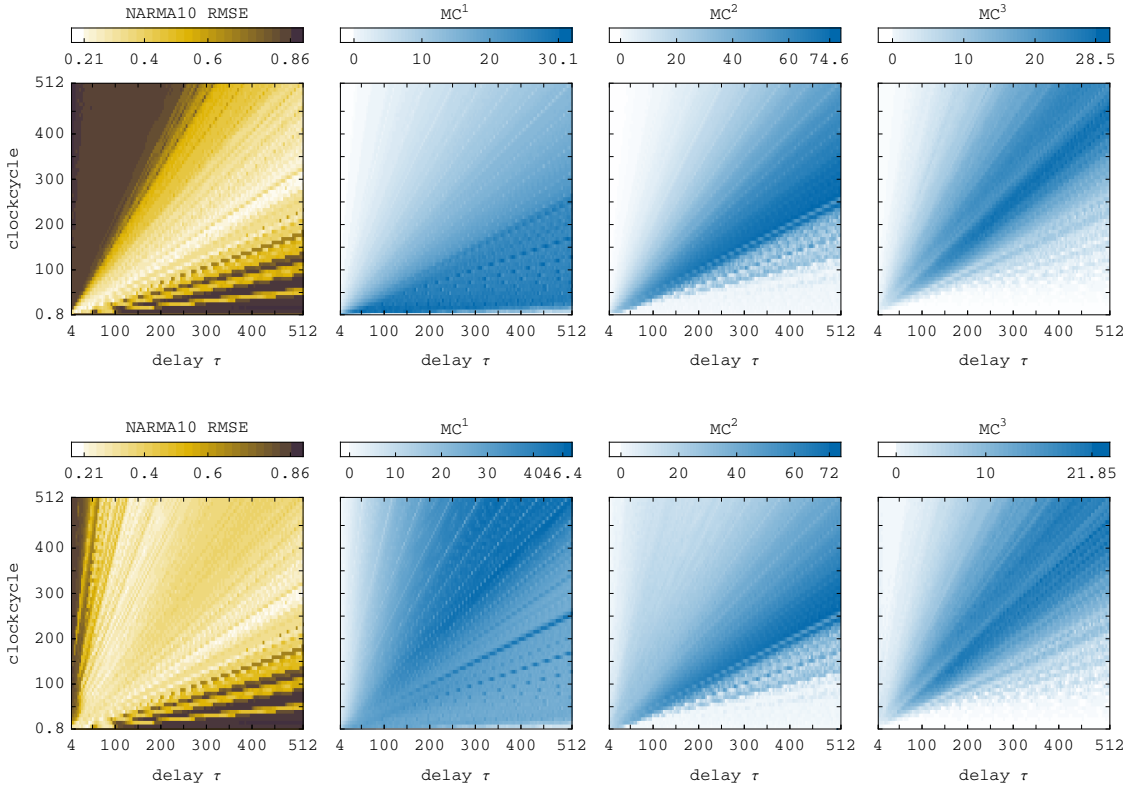

Figure 19: Several capacities in a unidirectional ring network $N = 8$ with $N_v = 16$. Shown are capacities over delay τ and clockcycle T for *synchronized state* (top) and *splay state* (bottom). Depending on the state the capacities are different.

5.3 All to all coupled networks

N=3 - N=16 row normalization

5.4 Less symmetrical network topologies

5.4.1 unidirectional rings with jumps

6 conclusion

needed in the future: individual events (spikes) as binary deciders? - either spike, or no spike.

kicking system into different attractor in order to "switch" task. comparable to classical computation.

References

[AMI19] P. Amil, M. C. Soriano and C. Masoller. Machine learning algorithms for predicting the amplitude of chaotic laser pulses. Chaos **2**9, 113111 (2019).

[ANT19] P. Antonik, N. Marsal, D. Brunner and D. Rontani. Human action recognition with a large-scale brain-inspired photonic computer. Nat. Mach. Intell. **1** (2019).

[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).

[APP14] L. Appeltant, G. Van der Sande, J. Danckaert and I. Fischer. Constructing optimized binary masks for reservoir computing with delay systems. Sci. Rep. **4**, 3629 (2014).

[ATI00] A. F. Atiya and A. G. Parlos. New results on recurrent network training: unifying the algorithms and accelerating convergence. IEEE Trans. Neural Netw. **1**1, 697–709 (2000).

[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).

[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 **8**1, 025205(R) (2010).

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

[EAR03] M. G. Earl and S. H. Strogatz. Synchronization in oscillator networks with delayed coupling: A stability criterion. Phys. Rev. E **6**7, 036204 (2003).

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

[FIL89] S. Filippi and U. Buchacker. Stepsize control for delay differential equations using a pair of formulae. J. Comput. Appl. Math. **2**6, 339–343 (1989).

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

[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.

[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).

[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).

[LUE13b] L. Lücken, J. P. Pade, K. Knauer and S. Yanchuk. Reduction of interaction delays in networks. Europhys. Lett. **1**03, 10006 (2013).

[LAR12] L. Larger, M. C. Soriano, D. Brunner, L. Appeltant, J. M. Gutierrez, L. Pesquera, C. R. Mirasso and I. Fischer. Photonic information processing beyond Turing: an optoelectronic implementation of reservoir computing. Opt. Express **2**0, 3241–3249 (2012).

[NEV76] K. W. Neves and A. Feldstein. Characterization of jump discontinuities for state dependent delay differential equations. J. Math. Anal. Appl. **5**, 689 (1976).

[PAD13] J. P. Pade, L. Lücken and S. Yanchuk. The Dynamical Impact of a Shortcut in Unidirectionally Coupled Rings of Oscillators. Math. Model. Nat. Phenom. **8**, 173–189 (2013).

[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).

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

[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).

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

[WAT98] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. Nature **3**93, 440–442 (1998).

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

[YAN11] S. Yanchuk, P. Perlikowski, O. V. Popovych and P. Tass. Variability of spatio-temporal patterns in non-homogeneous rings of spiking neurons. Chaos **2**1, 047511 (2011).

[YEU99] M. K. S. Yeung and S. H. Strogatz. Time Delay in the Kuramoto Model of Coupled Oscillators. Phys. Rev. Lett. **8**2, 648–651 (1999).

[ZOU09b] W. Zou and M. Zhan. Splay States in a Ring of Coupled Oscillators: From Local to Global Coupling. SIAM J. Appl. Dyn. Syst. **8**, 1324–1340 (2009).

References

[AMI19] P. Amil, M. C. Soriano and C. Masoller. Machine learning algorithms for predicting the amplitude of chaotic laser pulses. Chaos **2**9, 113111 (2019).

[ANT19] P. Antonik, N. Marsal, D. Brunner and D. Rontani. Human action recognition with a large-scale brain-inspired photonic computer. Nat. Mach. Intell. **1** (2019).

[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).

[APP14] L. Appeltant, G. Van der Sande, J. Danckaert and I. Fischer. Constructing optimized binary masks for reservoir computing with delay systems. Sci. Rep. **4**, 3629 (2014).

[ATI00] A. F. Atiya and A. G. Parlos. New results on recurrent network training: unifying the algorithms and accelerating convergence. IEEE Trans. Neural Netw. **1**1, 697–709 (2000).

[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).

[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 **8**1, 025205(R) (2010).

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

[EAR03] M. G. Earl and S. H. Strogatz. Synchronization in oscillator networks with delayed coupling: A stability criterion. Phys. Rev. E **6**7, 036204 (2003).

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

[FIL89] S. Filippi and U. Buchacker. Stepsize control for delay differential equations using a pair of formulae. J. Comput. Appl. Math. **2**6, 339–343 (1989).

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

[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.

[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).

[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).

[LUE13b] L. Lücken, J. P. Pade, K. Knauer and S. Yanchuk. Reduction of interaction delays in networks. Europhys. Lett. **1**03, 10006 (2013).

[LAR12] L. Larger, M. C. Soriano, D. Brunner, L. Appeltant, J. M. Gutierrez, L. Pesquera, C. R. Mirasso and I. Fischer. Photonic information processing beyond Turing: an optoelectronic implementation of reservoir computing. Opt. Express **2**0, 3241–3249 (2012).

[NEV76] K. W. Neves and A. Feldstein. Characterization of jump discontinuities for state dependent delay differential equations. J. Math. Anal. Appl. **5**, 689 (1976).

[PAD13] J. P. Pade, L. Lücken and S. Yanchuk. The Dynamical Impact of a Shortcut in Unidirectionally Coupled Rings of Oscillators. Math. Model. Nat. Phenom. **8**, 173–189 (2013).

[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).

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

[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).

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

[WAT98] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. Nature **3**93, 440–442 (1998).

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

[YAN11] S. Yanchuk, P. Perlikowski, O. V. Popovych and P. Tass. Variability of spatio-temporal patterns in non-homogeneous rings of spiking neurons. Chaos **2**1, 047511 (2011).

[YEU99] M. K. S. Yeung and S. H. Strogatz. Time Delay in the Kuramoto Model of Coupled Oscillators. Phys. Rev. Lett. **8**2, 648–651 (1999).

[ZOU09b] W. Zou and M. Zhan. Splay States in a Ring of Coupled Oscillators: From Local to Global Coupling. SIAM J. Appl. Dyn. Syst. **8**, 1324–1340 (2009).