Reservoir Computing Stuff

Lukas Manuel Rösel

February 25, 2020

Hallo

1 todo

1.1 things on the todo list.

1.1.1 programming work

1.1.2 research work

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

1.1.3 graphics/plots

- stuart landau oscillator: complex plain, 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].

1.1.4 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

1.1.5 results

- increasing network size
- increasing virtualization
- increasing input strength

1.1.6 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 one

2.1 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.

2.1.1 blabla

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/7164783). Last, but not least: the most apparent machine learning application to date: voice recognition algorithms are predicting the meaning of audio information everytime we utter the magical words "Hi, Siri", "Ok, Google" or "Alexa...". It can also be argued that classical prediction tasks that do not involve time-conscious datasets like images fed into one directional feed forward networks

2.1.2 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 Theory

3.0.1 Reservoir computing

Formalities

$\mathbf{U}$	uniformly distribution
$u^{-h} = (u_{-h}, u_{-h+1}, u_0) \in \mathbf{U}^h$	sequence of numbers drawn from a uniform distribution
$\mathbf{O}$	set of Targets or *true* values
$\hat{\mathbf{O}}$	set of Predictions that the system makes

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 previously fed into it and its ability to make nonlinear transformation of the data.

The used benchmarks have an input sequence $\mathbf{U}(t)$ and a *target* sequence $\mathbf{O}(t)$ with $t \in \mathbb{Z}$ being the time counted in clockcycles (cc). For a specific time t the reservoir offers a number of outputs $\mathbf{x}_k(t)$ which can be linearily combined in order to reach some *predicted* value $\hat{\mathbf{O}}(t)$. Ideally $\hat{\mathbf{O}}(t) = \mathbf{O}(t)$ which would represent a perfect reconstruction of the target value through the system. Usually machine learning systems give imperfect predictions $\hat{\mathbf{O}}(t)$ that vary from the *true* value $\mathbf{O}(t)$.

Linear Memory Recall

The simplest task a reservoir can perform is to simply repeat the the information that was fed into it at a certain point in time.

$$\begin{aligned} O_1(t) &= u_{t-1} \\ O_2(t) &= u_{t-2} \\ O_3(t) &= u_{t-3} \\ &\vdots \\ O_n(t) &= u_{t-n} \end{aligned} \tag{3.1}$$

3.0.2 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 means the estimation of a value. A weather model which has been fed past temperature data can be tasked to "predict" the past 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

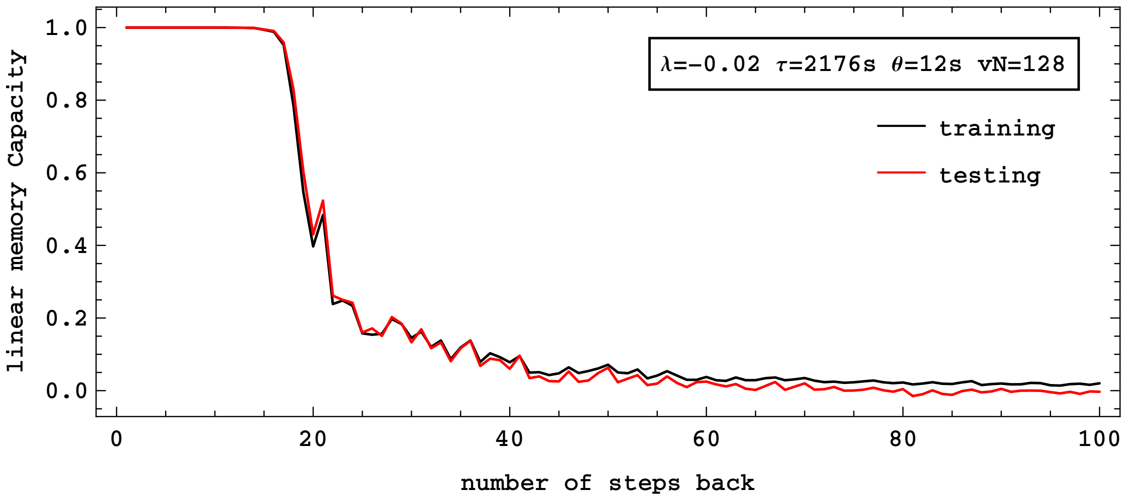

Figure 3.1: The linear memory capacities for varying steps into the past. The system is able to perfectly reproduce inputs up until 12 steps into the past. $N = 1, vN = 128, \lambda = -0.02, \omega = 1, \gamma = -0.1, \theta = 12, \tau = 2176$.

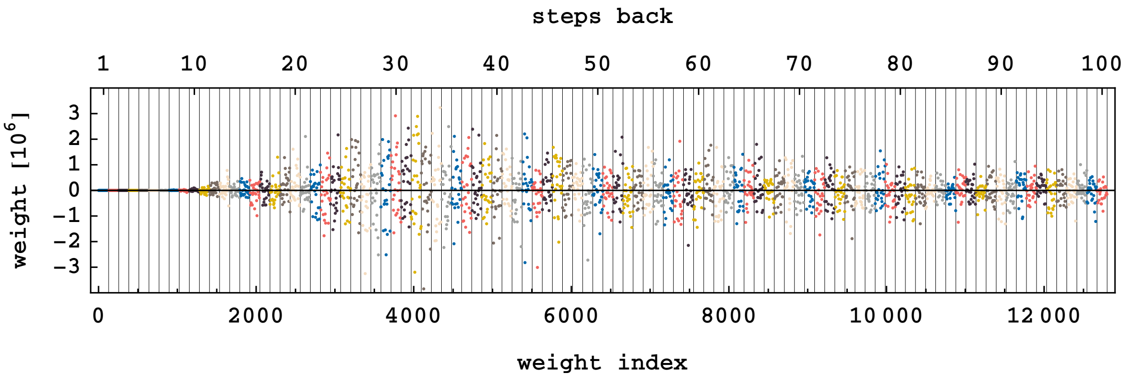

Figure 3.2: All weights $W_{i,s}$ attained through linear regression of the linear memory recalls $s \in [1, 100]$ steps back. The system was a unidirectional ring with of $N_{real} = 8$ and $N_{virtual} = 16$ nodes. The total read-out dimension is 128. For reconstruction of more recent inputs the inputs 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 $[-1, 1]$.

truth quantifies the error. The closer the prediction to the ground truth, the better the system performs a given task. It is important to note that usually these predictions are not of singular values, but give a vector of probabilities. A neural network used for image classification will output a vector with values e.g. it is quantifying the 'dog-ness', the 'tree-ness' or the 'car-ness' of an input image. The actual decision is made by choosing the entry with the highest probability. 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 discrete nature e.g. "yes" or "no" the system will usually output a real number.

In general a sequence of inputs u is drawn from a 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.

3.0.3 Legendre polynomials as Nonlinear Transformations

In order to investigate the nonlinear transformation capabilities one can use Legendre polynomials (Fig. 3.3) in order to transform inputs $\mathbf{U}(t)$. Legendre Polynomials have the useful property of being orthonormal to every other 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.3.3. 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.

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).

$$f : U^h \to \mathbb{R} : u^{-h} \to f\left[u^{-h}\right] \tag{3.2}$$

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

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

They are used in Neural networks as well blabla citecite findfind. [**?**]

NARMA10 - the Nonlinear Autoregressive Moving Average Task

$$A_{n+1} = 0.3A_n + 0.05A_k \left(\sum_{i=0}^{9} A_{k-9} \right) + 1.5u_{k-9}u_k + 0.1 \tag{3.3}$$

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

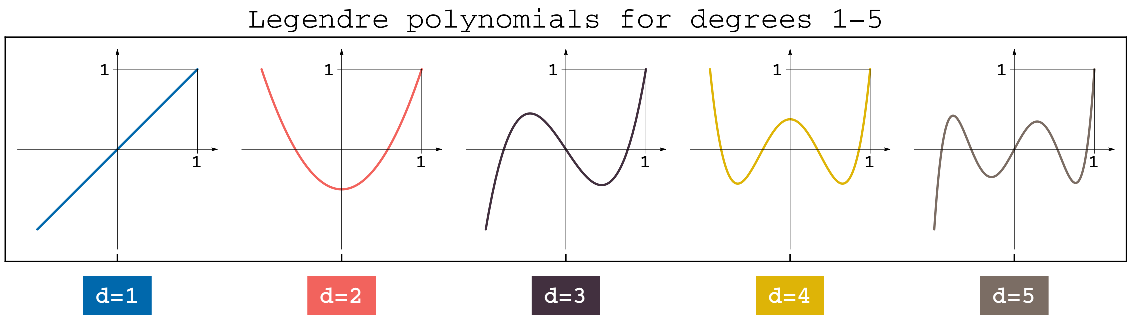

Figure 3.3: Legendre polynomials $L_d(x)$ for degrees $d \in \{1-5\}$ each shown for $x \in [-1,1]$.

last 10 steps while also being fed a product of a random sequence taken at two different positions. In order to perform well, systems need memory up to 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 [**?**].
The NARMA10 sequence is created by the iterative formula given by 3.3. It is fed 2 inputs from a sequence u which is drawn from a uniform distribution $U(0,0.5)$ on interval $[0,0.5]$.

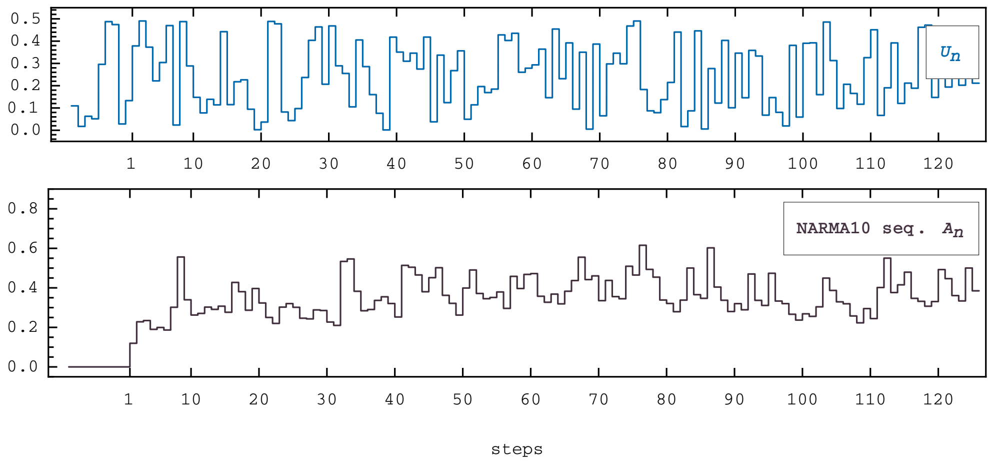

Figure 3.4: Example for a uniformly drawn random sequence $U_n \in [0,0.5]$ that is used to create a Narma10 sequence A_n with it. Here $A = 0$ for $n < 1$.

3.0.4 Stuart-Landau-Oscillator

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 (3.4) or a set of two equations written in polar coordinates (3.5). 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 accordance to its practical meaning.

$$\dot{z} = \left(\lambda + i\omega + \gamma |z|^2\right) z \tag{3.4}$$

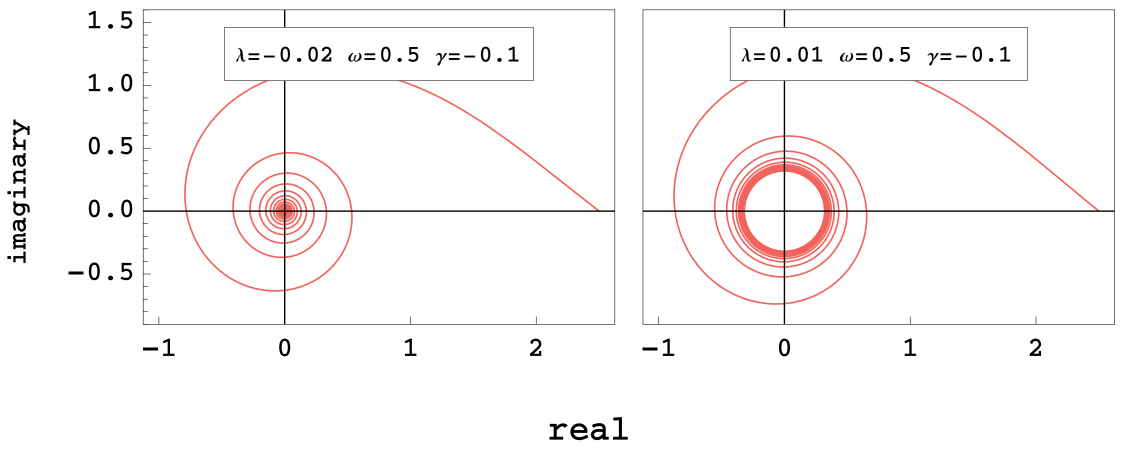

Figure 3.5: 2 very basic scenarios of the Stuart-Landau oscillator: Decay towards a single fixed point (off-state, left) or towards a stable oscillating state (limit cycle, right).

$$\begin{aligned}\dot{r} &= \lambda r + \mathrm{Re}(\gamma)\; r^3 \\ \dot{\phi} &= i\omega + \mathrm{Im}(\gamma)\; r^2\end{aligned} \tag{3.5}$$

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).

The limit cycle (LC) which is shown in fig 3.5 is depending on the ratio or λ and $Re\,[\gamma]$.

As can be seen in (eq. 3.4), the equation has a linear and a nonlinear term regarding the absolute value of z.

$$\dot{z} = \left(\lambda + GJ(t) + i\omega + \gamma|z|^2\right) z + \kappa e^{i\phi} z(t-\tau) \tag{3.6}$$

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 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. 3.9).

In Fig. 3.6 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 circle 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 the given readouts

3.0.5 Networks

Vertices blabla Edges blabla.

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.

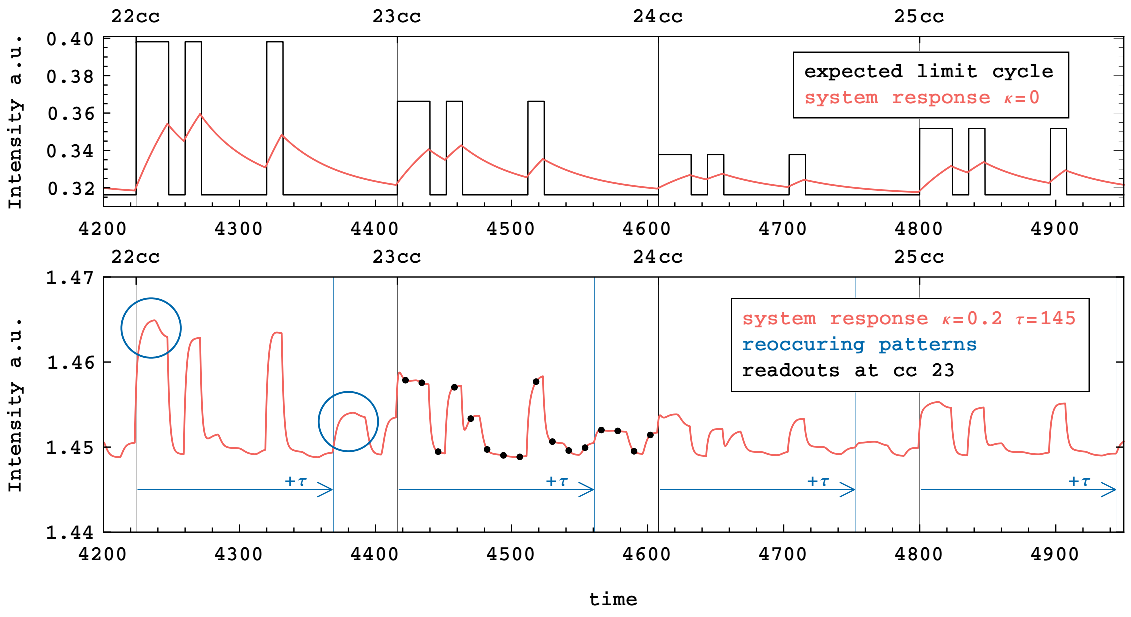

Figure 3.6: $|z(t)|$ of a driven Stuart-Landau oscillator with and without delayed feedback. (top): the expected limit cycle $r_{LC} = \sqrt{(\lambda + GJ(t))\,/Re(\gamma)}$ according to the pump current $J(t) \in [0, 1]$ with $G = 0.01$ and $\lambda = 0.01$. (bottom): With delayed feedback patterns repeat after delay time τ. The bump within the blue circle is not caused by input $J(t)$ at the time. For each clockcycle the system state is read out once per virtual node (black dots).

3.0.6 virtual Nodes and multiplexing

here: papers for explanation! [KUR18]

[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.

3.0.7 Dynamics of rings of stuart landau oscillators

pony-states (von André)

3.1 Results

3.1.1 Highly symmetrical network topologies

Ring networks

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

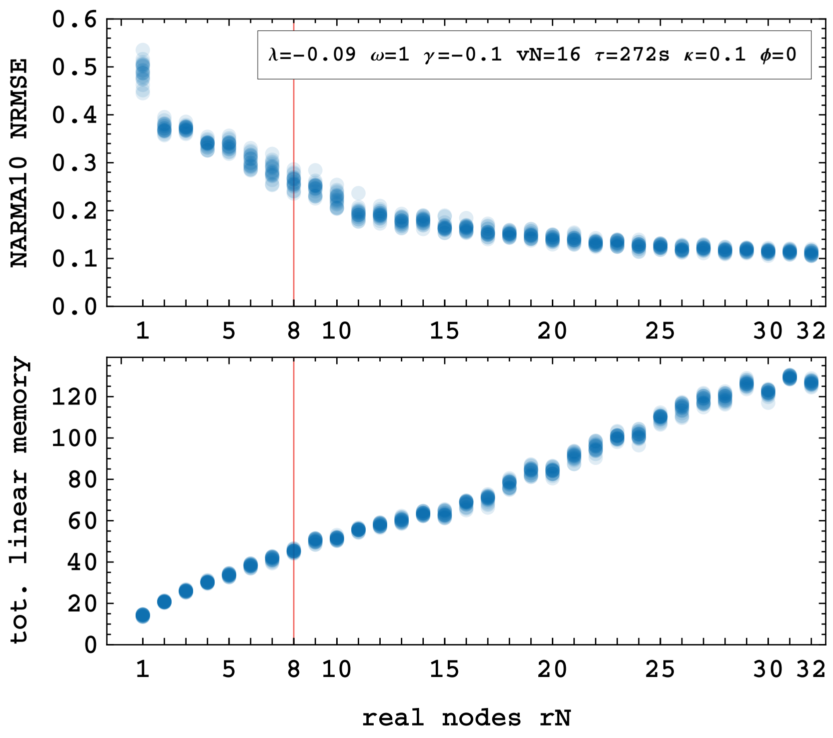

Figure 3.7: changing rc performance for increasing number or real nodes rN in unidirectianally coupled ring networks. (see some plot).

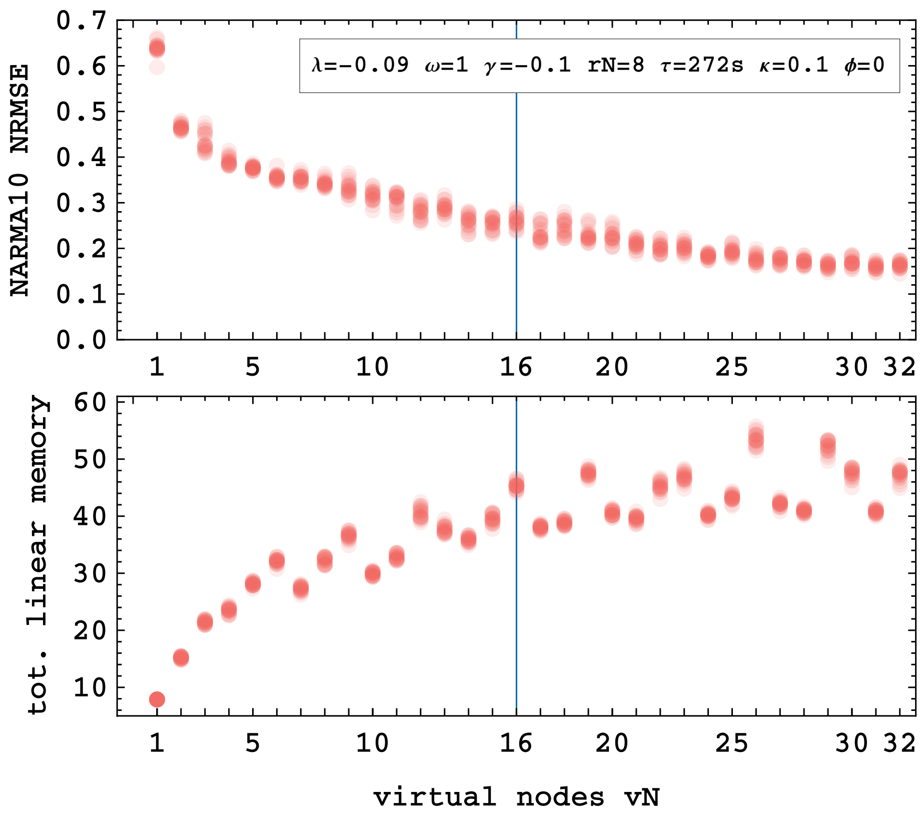

Figure 3.8: changing rc performance for increasing number or virtual nodes rN in unidirectionally coupled ring networks. (see some plot).

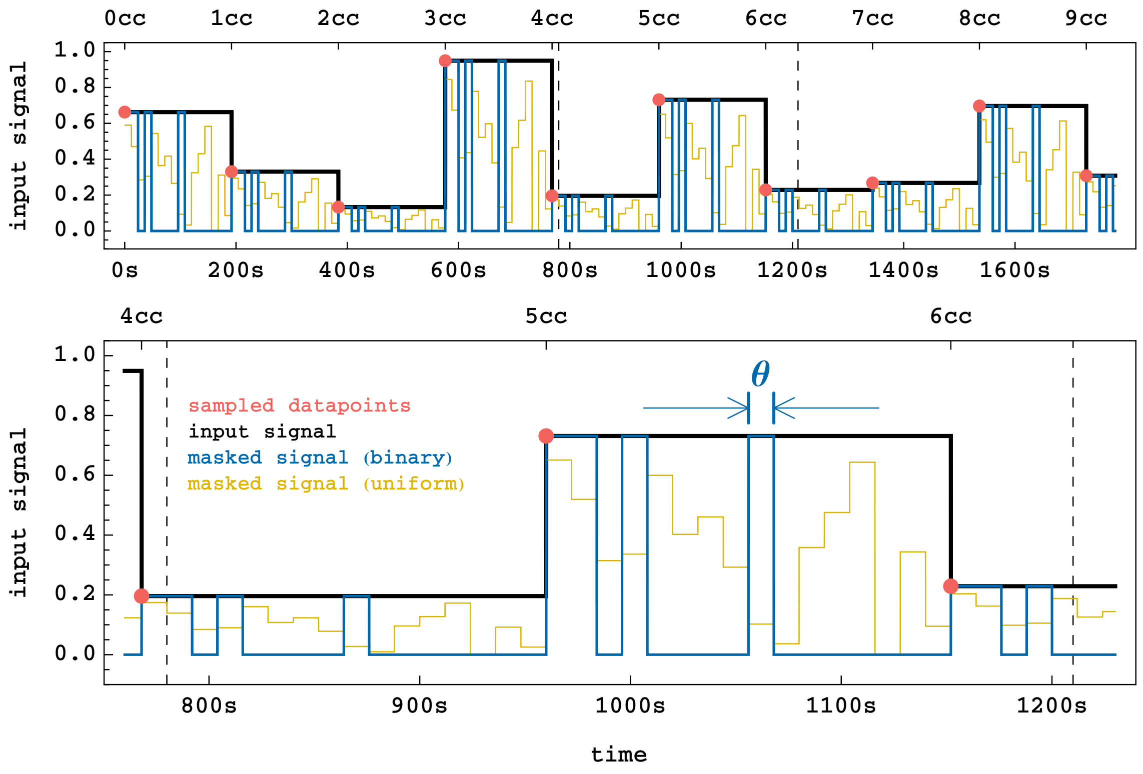

Figure 3.9: A timeseries of sampled data points ● with its constant-interpolated signal between samples ▬ and the corresponding masked signals ▬ and ▬. The mask times (or lengths) are usually defined as 1 clockcycle (cc, upper tick labels) and the time per virtual node called θ. Here $\theta = 12s$ and $1cc = 16\theta = 272s$

3.1.2 All to all coupled networks

N=3 - N=16 row normalization

3.1.3 Less symmetrical network topologies

unidirectional rings with jumps

Bibliography

[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).

[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).

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

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

[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.

[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).

[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).

[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 Networks **a**ccepted (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).

Bibliography

[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).

[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).

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

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

[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.

[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).

[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).

[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 Networks **a**ccepted (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).