UNIVERSITÄT OSNABRÜCK

8.3390: Study Project: Computation at the Edge of Chaos

A Study on *Computation at the Edge of Chaos in Echo State Networks*

Submitted By:
Taher Habib, Matrikel Nr. 975761
Johannes Nowack, Matrikel Nr. 976130

Supervisor:
Prof. Dr. Gordon Pipa

Submitted as a Study Project carried out during WS2018/19-WS2019/20.

Osnabrück, Germany
02-March-2020

Acknowledgements

We would like to express our sincere gratitude to Professor Dr. Gordon Pipa for initiating this study project on *Computation At The Edge Of Chaos* and giving us an opportunity to work on a challenging topic like this which has roots in Dynamical Systems, Chaos Theory, Information Theory and Neuro-evolution. We have learnt an immense lot by working on this project. We are also greatly indebted to him for his guidance and support throughout the project.

We would also like to thank our team members Turan Orujlu and Mohammed Chaposhloo for the interesting discussions that we had with them around this broad topic.

Taher Habib, Johannes Nowack

Contents

(**Ω**: Taher's, **π**: Johannes', **χ**: Both)

Abstract

In the following report, we develop our work performed as part of the study project from November 2018 in WS2018/19 until February 2020 in WS2019/20. There are two main objectives of this work. Firstly, it is to give an overview of the idea of *Computation at the Edge of Chaos* (EOC) from the perspective of Reservoir Computing. We use the framework of Echo State Network (ESN) in order to study the main hypothesis. We replicate the original results of [2] and [8] where they find support to the EOC hypothesis that computational power of randomly connected ESNs is maximized on the edge of chaos. Secondly, many theoretical biologists believe – after the advent of this hypothesis – that biological neural networks, like our brains, show such remarkable and unbeaten performance by staying at the edge of chaos even if they grow more or less randomly in nature (although this is not completely true [25]). We, therefore, study in this work the question of whether or not natural evolution exhibits preference towards selecting individuals whose biological neural networks perform at the edge of chaos. We find evidence in support of this hypothesis too. We, again, use the framework of ESNs to test this hypothesis alongwith the NEAT algorithm to perform neuroevolution of ESNs, similar to [9].

Structure of Github Repository

An implementation of an Echo State Network along with other conducted experiments of this study can be found at: https://github.com/ljscfo/esn-neuroevolution. The programming language used is Python 3.6.8. In addition to Python's standard modules, we used the following ones (version number in brackets):

- numpy (1.17.0)
- scipy (1.3.0)
- pandas (0.25.0)
- matplotlib (3.1.1)
- ddeint (0.1.2)
- dill (0.3.0)
- sklearn (0.21.3)
- idtxl (1.0)

For the neuroevolution, the module *peas* is used with a few modifications (see chapter IV.B).

A python notebook containing tests and results from computation of Lyapunov Exponent of the ESN is located in *Lyapunov_Expo_Experiment*, along with the data files. The code belonging to the random reservoirs experiment as well as the result files are located in *Random_Experiment*, the neuroevolution experiment files are inside the folder *Neuroevolution_Experiment*. The root directory holds the files needed for both experiments, which are: first, *esn_cell.py*, holding the implementation of an echo state network along with functions for computing its lyapunov exponent, transfer entropy and active information storage; second, *benchmark_esn.py,* which instantiates an ESN and conducts the benchmark tasks introduced in chapter IV; third, *visualizer.py*, which creates diagrams of the experiments' results. Inside *peas/* is the Python-module *peas*[1], which implements the neuroevolution algorithm NEAT. We didn't create it ourselves but use it in the neuroevolution experiment.

[1] see https://github.com/noio/peas

I. Introduction

In our daily lives, we distinguish between physical systems that perform computation, like laptop computers and scientific calculators, from physical systems that *generally* do not perform any computation, like rocks, etc. Amongst the ones that do compute something *meaningful* for us, we also tend to differentiate physical systems into the ones that are more powerful/capable than the ones which are less so. For instance, the difference in the money we need to buy a laptop or a scientific calculator is, fundamentally, due to the difference in the computational capacity of the two systems. It is worth noting that the dynamics of a computing system – here, we would limit ourselves to artificial neural networks, since ESN is also a neural network model – is intricately linked with its computational capabilities. This fact is undoubtedly recognized in the community. Hence, it is particularly important to analyze the relationship between the dynamical behaviour of a system and its computational capabilities so that we may be able to design better and more capable computers and facilitate their learning to maximize their performance on the task at hand. The hypothesis of *computation at the edge of chaos* provides a promising direction of research in this domain. It states that *interesting*/*meaningful* complexity and *computational capacity*, like the ones exhibited in biological living systems, manifests spontaneously within dynamical systems that operate at the edge of chaos, i.e., neither orderly nor chaotic but somewhere in between.

However, there is a problem in the domain of such research regarding the clarity on the definition of the term computational capacity. As it stands, there exists no widely agreed-upon notion of computational capacity/power of a given system – there is no agreement on the definition of abstract concepts like *computation* and *computational power* [26, 2]. But while there is generally no agreement on the definition of computational capacity/power of a computer, one of the obvious ways in which these abstract notions can be assessed is by assessing the *complexity* and diversity of associations (functions) of inputs to outputs that can be implemented on the system at hand. It should be noted that we define a *computation* here as a routine or algorithm that assigns outputs to inputs. We will define *computation* more concretely in the context of echo state networks when discussing the particulars of the task that we assign the ESN to compute (refer to chapter IV). There are also the widely-accepted primitive component operations of *Turing universal computation*, which are clearly defined in terms of *informational dynamics*. These are:

storage, *transmission* and *modification* of information [17]. We will come back to some of these notions again in chapter IV, where we would define them quantitatively for Echo State Networks. An increase in any of these primitives components of a Turing Machine is also indicative of higher computational capacity of the system.

Outline of Chapters

Chapter II begins with an overview of Dynamical Systems and Chaos and introduces the echo state network, along with the architecture used in the study, and discusses it in the light of a dynamical system. Furthermore, it describes an algorithm to quantify the dynamics of a system (here, ESN) using Lyapunov Exponents. Finally, we discuss the variations encountered in the Lyapunov exponents of ESNs based on changes in their various structural attributes and parameters. In chapter III, we introduce the main idea of Computation at the edge of chaos, where we only give a high-level overview of the hypothesis with regards to its history and generality. Chapter IV describes in detail the various tests/proxies for computational capacity used in this study. The results of the experiments using random ESNs, which support the main hypothesis, are presented in chapter V. Chapter VI is dedicated to an experiment that uses neuroevolution to evolve the reservoirs of ESNs towards highest computational capacity. Again, the idea of computation at the edge of chaos gets supported by the results. We state our conclusions from the study in chapter VII.

II. Dynamical Systems & Chaos

II.A. Defining and Characterizing Dynamical Systems

A **dynamical system** is simply a deterministic mathematical rule for time evolution on a *state space*, where a **state space** can be defined as a set of all possible states of the dynamical system [18]. Each possible state of the system corresponds to a unique point in the set of state space. When considering physical systems, the deterministic mathematical rule refers to the equation which governs the evolution of the state of the physical system through time. By the term *deterministic*, we mean that the dynamical system exhibits unique evolution, i.e., any given state of the system is always followed by the same history of state transitions [19]. It should be noted that as time evolves and new states of the system are generated from an initial state at the beginning, the collection of all these states ordered in time is referred to as a **trajectory** of the dynamical system.

The dynamics of a physical dynamical system can vary from orderly to chaotic. The orderly behaviour of a dynamical system is defined as behaviour which is unchaotic. In other words, order is defined in terms of chaos. Hence, we first define chaotic behaviour of a system. Loosely stating, chaos in a system is linked with the trajectory of the system being *aperiodic* and the system being *unstable*. Stating it formally, a system is said to be chaotic if it exhibits what is known as **Sensitive Dependence on Initial Conditions** (SDIC, for short). SDIC is what leads to the long-term unpredictability of the system since a slight deviation from the trajectory in the state space can lead to dramatic changes in future behavior.

Stated briefly, a dynamical system has SDIC *if arbitrarily small differences in initial conditions eventually lead to arbitrarily large differences in the orbits*. More formally, let $\mathbf{x}(0)$ and $\mathbf{y}(0)$ be any two initial conditions in the state space of the dynamical system such that $|\mathbf{x}(0) - \mathbf{y}(0)| < \delta, \ \forall \ \delta > 0$. If there exists $\varepsilon > 0$ and some time $t > 0$, such that $\mathbf{x}(t)$ and $\mathbf{y}(t)$ (the states of the system at time t, starting from $\mathbf{x}(0)$ and $\mathbf{y}(0)$ respectively) satisfy $|\mathbf{x}(t) - \mathbf{y}(t)| > \varepsilon$, then the system is said to have sensitive dependence on initial conditions. The essential idea is that the dynamical system acts such that regardless of how close together $\mathbf{x}(0)$ and $\mathbf{y}(0)$ are, the trajectory initiating from

y(0) will eventually diverge by ε from the trajectory initiating from **x**(0). This way of defining SDIC is generally referred to as the *weak form of SDIC* as it does not specify the rate of divergence (it is compatible with linear rates of divergence) nor does it specify how many points surrounding **x**(0) will give rise to diverging trajectories (it could be a set of any measure, e.g., zero) [19].

The strong form of SDIC is defined using what is called the **Lyapunov Exponent λ**. Lyapunov exponents are mathematical tools that provide a measure of the sensitivity of a dynamical system to its initial conditions. Generally speaking, an *N*-dimensional dynamical system has *N* Lyapunov exponents, one each for measuring the rate of distortion along one of the standard normal directions in the state space in which the system's trajectory is evolving. The *strong form of SDIC* states that: there exists λ such that for *almost all* points **x**(0) and for *almost all* points **y**(0) in a small neighborhood δ around **x**(0), i.e. $|\mathbf{x}(0) - \mathbf{y}(0)| < \delta$, $\forall\ \delta > 0$, there exists $t > 0$, such that $|\mathbf{x}(t) - \mathbf{y}(t))| \approx |\mathbf{x}(0) - \mathbf{y}(0)|e^{\lambda t}$, where using the "*almost all*" caveat entails that we take the set of neighbouring points in state space as a set of non-zero measure [19]. Here, λ denotes the maximum **global** Lyapunov Exponent and represents the *average rate of exponential divergence/convergence* of neighboring trajectories issuing forth from some small neighborhood centered around **x**(0). The maximum Lyapunov exponent plays a major role as it dominates the rate of divergence or convergence in the state space and thereby represents a useful indicator of the stability of the whole system during its evolution. Clearly, if $\lambda > 0$, then the trajectories are taken to be exponentially diverging, whereas convergence is implied if $\lambda < 0$ [20]. Of course, there are chaotic dynamical systems that exhibit larger than exponential divergence. Such systems are then characterized as exhibiting chaos using the weak form of SDIC, instead of the strong form. Note that we will use the strong form of SDIC to characterize chaos in our echo state network.

Thus, in conclusion, a dynamical system is said to exhibit *deterministic chaos* when:

- The system's trajectories are aperiodic and bounded.
- The system has sensitive dependence on initial conditions.

It should be stated that instead of evaluating global lyapunov exponents of the ESNs as defined above in the strong form of SDIC, we will use **Local** Lyapunov Exponents (LLEs) for our analysis since in common practice it is impossibly difficult to numerically compute global lyapunov exponent of such high-dimensional system (see section II.B.) as an echo state network. What we therefore use instead is a *local* measure of divergence of nearby trajectories, i.e. *local finite-time* approximations of divergence/convergence over a trajectory followed by the dynamical system while it is driven by an input signal [21, 22]. The way in which global and local lyapunov exponents of a dynamical system differ is that while the global parameter gives a measure for the *total* (un-)predictability of the system, the local exponent gives the (un-)predictability of the system locally around a point **x** in the state space. As a result, global lyapunov exponents are more robust parameters for detecting the presence of *deterministic chaos*. Local lyapunov exponents' estimations are more susceptible to error since one may wrongly conclude that a system has deterministic chaos simply by observing a positive maximum LLE on a small, finite time scale since this can be achieved in the case of evolution of the system near an unstable fixed point[2] in a high dimensional manifold of the state space which may not be a true indicator of chaotic dynamics as the underlying dynamics may occur unconfined in an infinite state space. We overcome this particular challenge in our numerical estimation of maximum LLE by approximating it over a large number of samples (≈100,000). We describe in detail this numerical computation and estimation of local Lyapunov Exponent of the ESN in section II.C.

It is also worth noting that although chaotic dynamics of a system is defined using divergence of nearby orbits, it doesn't imply that the trajectories are bound to diverge to infinity eventually. Chaotic dynamics especially in time-discretized systems like ESNs is characterized as being confined to some attractor or at least to a bounded region of the state space, which guarantees the fact that trajectories in state space are dense [19]. In our analysis, since the ESN is a very high-dimensional system which makes it difficult to analyze its state space and attraction basins or bifurcations, we ensure this restriction – on the state space of the ESN being *bounded* – by evaluating (for detecting chaos) the dynamics of the network on every single iteration (one

[2] A **fixed point** of a dynamical system is one where the system's state does not change with time; an **unstable** fixed point is one which is not stable, where a **stable** fixed point is roughly defined as the point in the state space to which the (sufficiently) neighbouring points (initial conditions) converge as time goes on.

time-step) of the network instead of any longer timeline. Again, see section II.C. for details on the process.

II.B. Echo State Networks

In the machine learning community, it is of special importance to design models that can be used to develop feature representations of temporal information in order to have a natural approach to tasks based on time-series data featured by multiple time-scales. A Recurrent Neural Network (RNN) is one such standard model. Extending RNN approach, Reservoir Computing (RC) is a framework that was first introduced in the early 2000s in the form of two models (developed independently): Liquid State Machines and Echo State Networks [3, 4]. This framework provides one of the ways in which recurrent neural networks can be trained in a supervised-learning paradigm without the cost of traditionally long training times and instability issues. The basic idea is to feed an input signal into a fixed, random dynamical system called a **reservoir** which maps the input to a higher dimension. A readout mechanism, which is what gets trained during learning, is employed to read the state of the reservoir and map it to the desired output. The RC approach therefore consists of two conceptually different parts: a dynamical system with "rich/high dimensional" dynamics and a memoryless readout function. The basic design principle that motivates the approach of RC is the assumption that a large random network can map/transform the input into so many variations, due to its nonlinearity, that extraction of useful information becomes simple [4].

Note that, at a fundamental level, **echo state networks** are dynamical systems whose trajectories in the state space are determined by the initial conditions and the external input. As a result, they are a natural choice if one wants to examine the link between system dynamics and any particular computational task.

The Architecture

The standard implemented ESN model with N neurons in the reservoir is depicted below schematically:

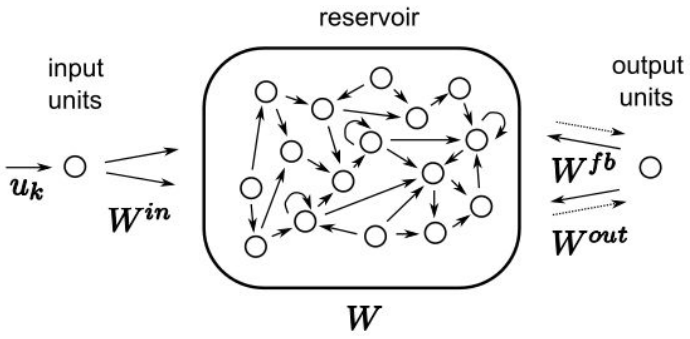

Fig. II.1 (Source: [14])

It is a discrete-time dynamical system with state $\mathbf{x}_{k+1}$ at time-step k+1 defined through a function F of the current state $\mathbf{x}_k$ ($\in \mathbf{R}^{N\text{x}1}$), the next input $\mathbf{u}_{k+1}$($\in \mathbf{R}^{K\text{x}1}$) and the output x_k^{out} at time step k:

$$\mathbf{x}_{k+1} = \boldsymbol{F}(\mathbf{x}_k, \mathbf{u}_{k+1}, x_k^{out}),$$

where $\boldsymbol{F}$ is defined by the equations below:

$$\mathbf{x}_{k+1} = \boldsymbol{f}(W\mathbf{x}_k + W^{in}\mathbf{u}_{k+1} + W^{fb}x_k^{out}),$$
$$x_k^{out} = \boldsymbol{g}(\boldsymbol{W^{out}}[\mathbf{x}_k; \mathbf{u}_k]). \qquad \text{(Eq. 1)}$$

Here, $W \in \mathbf{R}^{N\times N}$ is the internal weight matrix of the reservoir that is generally quite sparse with every neuron connected only to 10% of the reservoir's neuron population, $W^{in} \in \mathbf{R}^{N\times K}$ is the input matrix, $W^{fb} \in \mathbf{R}^{N\times L}$ is the feedback matrix, $W^{out} \in \mathbf{R}^{L\times(N+K)}$ is the output matrix. The state activation function $\boldsymbol{f} = (f_1, \ldots, f_N)^T$ is a sigmoid function applied component-wise with $\boldsymbol{f}(\mathbf{0}) = \mathbf{0}$ and the output activation function is $\boldsymbol{g} = (g_1, \ldots, g_L)^T$ where each g_i is usually the identity (here) or a sigmoid function. [;] denotes vector concatenation and $\mathbf{x}^T$ denotes the transpose of a vector $\mathbf{x}$. We have used the *tanh* function as a sigmoid activation in our experiments.

Considering ESN from the perspective of a dynamical system, it should be noted that the size of the reservoir N determines the dimension of the system. Hence, for N=100 (say), the ESN is said to be a 100-dimensional system. Similarly, the state space is defined in terms of the reservoir size of the network. For instance, the state space of an ESN with 100 reservoir units would be $\mathbf{R}^{100}$.

Here, we consider only ESNs without feedback, i.e. $W^{fb} = 0$. The echo state network F with no feedback connection becomes:

$$\mathbf{x}_{k+1} = \boldsymbol{F}(\mathbf{x}_k, \mathbf{u}_{k+1}) = \boldsymbol{f}(W\mathbf{x}_k + W^{in}\mathbf{u}_{k+1}) \qquad \text{(Eq. 2)}$$

Sometimes, as in the case of leaky integrator discrete-time ESN, the state update equation is a combination of the new state $\mathbf{x}_{k+1}$ and the current state $\mathbf{x}_k$, signifying – based on the parameter α – whether the network has a *memory* or not:

$$\mathbf{x}_{k+1} = (1 - \alpha).\mathbf{x}_k + \alpha.\mathbf{x}_{k+1}. \quad \text{(Eq. 3)}$$

α here is the **leak rate** of the reservoir. Clearly, if α=1, then the ESN's state update happens purely due to the new state at time k+1, i.e. $\mathbf{x}_{k+1}$. Otherwise, for $\alpha \in (0, 1)$, the network's state is updated as a linear combination of its current state at time k and the new state.

Training Echo State Networks

One of the major attractions in using the ESN approach for machine learning is the requirement to learn W^{out} only. The reservoir W remains untrained. Since the readout function of an ESN is linear and feed-forward, the training procedure typically involves solving a least square problem in an offline manner and using a closed-form equation, with respect to the teacher outputs y_k. In this study, we use the popular *ridge regression* technique to carry out the training using Tikhonov regularization:

$$W^{out} = \mathrm{Y}^{target}\mathrm{X}^{\mathrm{T}}(\mathrm{X}\mathrm{X}^{\mathrm{T}} + \beta\mathrm{I})^{-1}, \quad \text{(Eq. 4)}$$

where, the term βI acts as a regularization term for preventing extremely large values of W^{out}, X contains $\mathbf{x}_k$ for all the training time-steps combined and Y^{target} contains target values y_k for all the training time-steps combined [5]. During our experiments, we found the regularization strength β to have a considerable effect on the reservoirs' performances. We performed a grid search (see chapter V.A) and achieved the best results using the small value of 3×10^{-9}.

ESN Dynamics & Computation

As stated above, during training the ESN's nonlinear high-dimensional reservoir remains unaltered and it is only the output weights W^{out} that are trained. As a result, part of the dynamics of the network are governed by reservoir parameters that are randomly initialized at the beginning. Echo state networks can therefore exhibit very different types of dynamics depending on their connectivity structure. For instance, when the reservoir weights are too large, the reservoir output mostly looks like a white noise with no inherent structure. This we identify as chaotic dynamics. In contrast, when the reservoir weights are too small, the activity of the reservoir tends to die out and this we identify as ordered dynamics of the ESN [9]. Informally

(and as defined generally in section II.A.), we say a network behaves chaotically if arbitrary small differences in any two network states are highly amplified and do not vanish as time goes on. On the other hand, if the network behaves orderly then it "forgets" all such small differences and the state of the network at any given point of time is largely determined by the input signal that it receives.

Echo State Property

In most machine learning tasks, employing ESNs necessitates their use only in the ordered dynamic regime, instead of chaotic. One of the properties that the reservoir should possess in order for the ESN learning algorithms to have any utility is the Echo State Property (ESP). The ESP is a condition that, if it exists in the reservoir of an ESN, would lead to the asymptotic convergence of the network's state to the dynamics of the input signal. In other words, the state of the network would asymptotically be driven by the external input signal rather than its initial value. Put simply, if the ESP holds for a given ESN, all dependencies on the initial conditions of the network would vanish with time such that the state of the network would come to represent an "echo" of the input signal. As a consequence, the echo state property should be prominent in ESNs that are ordered in their dynamics while absent from ESNs with chaotic dynamics.

ESP is a subtle mathematical concept related to the algebraic properties of the resevoir's weight matrix W and the driving input signal $\boldsymbol{u}$. Some research [14, 16] has been recently done on ESN dynamics in order to find out the sufficient condition for the ESN to have echo state property. Specifically, it is largely accepted – though it is not always true (*see* [14]) – that ESNs which exhibit the echo state property are ones such that the maximum absolute eigenvalue of their reservoir weight matrix W, also called as the **spectral radius**, is *smaller than unity*. Thus, relating this result with the statements before, it can be concluded that, in general, the spectral radius less than unity would be exhibited by ESNs with ordered dynamics, while greater than unity values would be seen mostly with ESNs which are operating in the chaotic regime. This statement is depicted by the figure II.2:

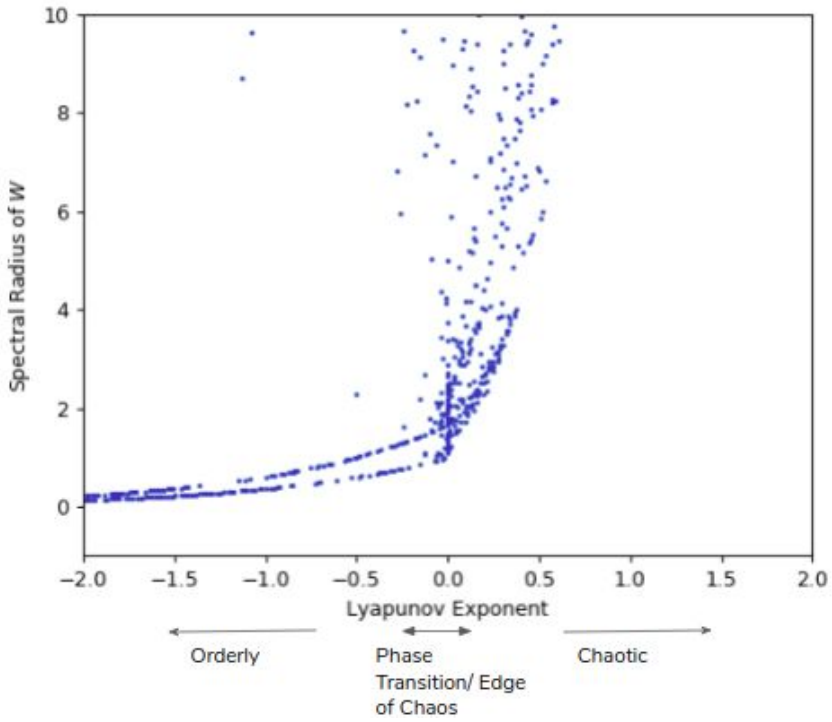

Fig. II.2. Depiction of the relation between ESP (measured by spectral radius) and dynamics of the ESN (measured by lyapunov exponent).

It should be noted that larger values of spectral radius are associated with larger values of the standard deviation of the reservoir weight matrix, as seen in figure II.3:

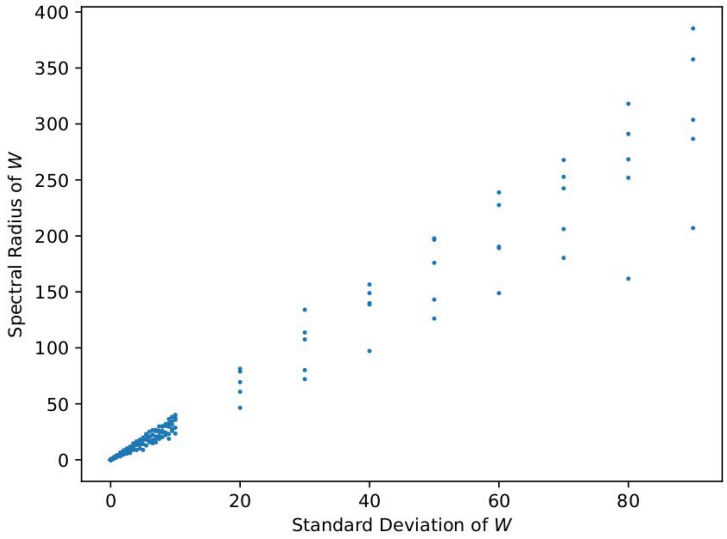

Fig. II.3. As the standard deviation of the reservoir matrix increases so does its spectral radius. Note that for each value of the standard deviation, spectral radius of five ESN reservoirs of arbitrary sizes 50, 75, 100, 125, 150 have been shown.

II.C. Computing the Largest Local Lyapunov Exponent of the ESN

The range of dynamics of the network from order to chaos is measured by computing what is called as the largest Local Lyapunov Exponents (LLEs, λ). LLEs measure the rate of growth of small *generic tiny* perturbations around a given point in the state space. As explained before in section II.A., we measure how much the network is sensitive to perturbations in its initial conditions in order to determine whether it has ordered or chaotic dynamics. Concretely, if differences in initial conditions enlarge as the network's trajectory moves in the state space, then we say the network is chaotic; otherwise, if those initial differences tend to vanish over time, then we label the network's dynamics as orderly. The core idea is, therefore, to let the ESN run on a random input from two *slightly* different initial states and measure the distance between the states of the network in the two cases at consecutive steps. If the average distance between the two network states tends to converge, then the network is in the ordered regime and $\lambda < 0$. If the distance diverges, then the network is chaotic and $\lambda > 0$. Consequently, the dynamics at the edge of chaos lie right in the middle, where the two networks tend to keep almost the same distance from each other and $\lambda \approx 0$.

Intuitively, given some initial condition $\mathbf{x_0}$ in the state space of the ESN, consider a nearby point $\mathbf{x_0} + \gamma_0$, where γ_0 represents a very slight perturbation in the initial state. If we let the ESN run for n timesteps starting from both the original initial state and the perturbed state and if the initial γ_0 becomes γ_n, then under the assumption of exponential divergence of trajectories, the relationship between γ_0 and γ_n is given by: $|\gamma_n| \approx |\gamma_0| e^{\lambda n}$, where λ is the largest lyapunov exponent, since the separation between the two trajectories after n iterations (given by γ_n) is *dominated* by the largest positive λ_k for all the k-dimensions of the system [12]. From this, we can derive a more computationally favourable formula for computing the largest lyapunov exponent by taking natural logarithms.

$$\lambda = \frac{1}{n} \ln \left| \frac{\gamma_n}{\gamma_0} \right| \qquad \text{(Eq. 5)}$$

In the limit when $n \to \infty$, the expression above will give us the largest global Lyapunov exponent (an asymptotic quantity). However, since in our numerical estimation algorithm we use a finite value of n, we are essentially computing the *largest local* Lyapunov exponent.

The Algorithm for Estimating Maximum LLE

The algorithm that we use for ESN is based on the numerical estimation technique of the largest Lyapunov exponent of a dynamical system from its time series data by J.C. Sprott [13]. The algorithm has been employed in various other sources for an identical purpose (see [8], [9]). It is explained below for the case of an ESN:

1. The input for the ESN is generated randomly as a sequence of 2000 values, where a uniform distribution is used with range between -0.1 and 0.1 (of the order ~ 10^{-1}).
2. The ESN is then simulated on the first 1000 input values/time steps and its outputs are discarded in order to stabilize the network against the effects of transient random initialization (longer durations are found to make no significant difference [8]). This step is also performed in order to ensure that the network's trajectory is in the attraction basin inside the network's state space, as explained in [13].
3. After this initial run, a copy of the ESN is generated. In the copy, the activation value of one of its neurons (from the reservoir) is perturbed by the value of γ_0. This leads to the separation in the initial states of the perturbed network $\mathbf{x}^2(0)$ and the unperturbed network $\mathbf{x}^1(0)$ by an amount γ_0. As stated in [13], this value of γ_0 should be chosen carefully. It

should be as small as the limited numerical precision of the computer allows in order to measure its influence. The precision of *np.float64* in python is of the order of 10^{-16} (52-bit mantissa gives $2^{-52} \approx 10^{-16}$ distinct representations). We use $\gamma_0 = 10^{-12}$ in our experiments, instead of 10^{-16} in order to be safe from numerical overflows that may occur by exploiting the full representational capacity of the data type *np.float64*. This value is also the one recommended in [8] and [9].

4. Both the original and the copy networks are advanced one time step and the distance (measured by the Euclidean norm ||.||) between their states $\mathbf{x}^2(1)$ and $\mathbf{x}^1(1)$ is calculated and recorded as $\gamma_1 = \|\mathbf{x}^2(1) - \mathbf{x}^1(1)\|$.
5. The state of the perturbed network is then normalized to the initial distance of γ_0, using: $\mathbf{x}^2(k) = \mathbf{x}^1(k) + (\gamma_0/\gamma_1)(\mathbf{x}^2(k) - \mathbf{x}^1(k))$. There are two reasons behind carrying out this step. Firstly, this step helps in keeping a check on numerical overflows that might occur due to the fact that the activation function *tanh* of the ESN neurons is capped between (-1, 1) and if the input to *tanh* becomes too large then we get values which are almost equal to +1 or -1. Thus, in the case when the states $\mathbf{x}^2(n)$ and $\mathbf{x}^1(n)$ diverge too far from each other it might lead to a numerical precision error. Secondly and more importantly, the manner in which this step is performed (see below) ensures that while we have the separation between the two states normalized to the initial value γ_0, the *direction* of the separated trajectory from the original trajectory is preserved. Thus, although, for the next iteration over the next time-step, we start with the same separation γ_0 as the previous iteration, the direction in which the perturbed orbit evolves for future iteration remains preserved which additionally ensures that the orbit stays on the same manifold in the attraction basin. A visualization of this step is shown below in figure II.4. Note that this figure is adopted from [8], and though it depicts the normalization step above mostly correctly, there is an error in the depiction with regards to setting the direction of the next iteration (i.e., the direction of the γ_0-separated normalized $\mathbf{x}^2(k)$) of the trajectory should be same as the direction of the previous step (i.e. the direction of $\mathbf{x}^2(k)$).

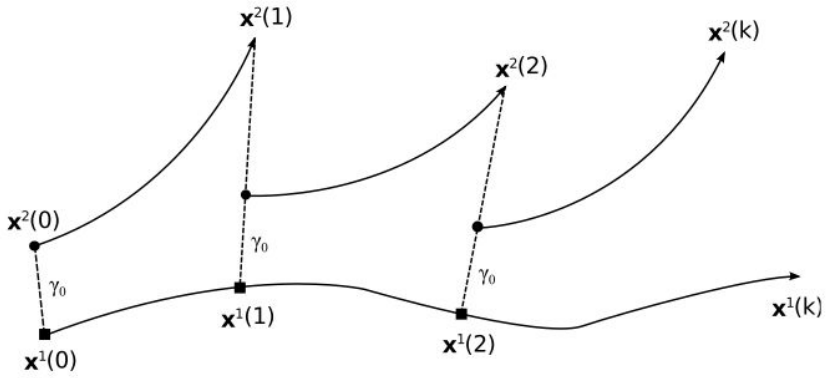

Fig. II.4. Visualization of the normalization step performed after each iteration.

6. Steps 4 and 5 above are repeated till the end of the input sequence and all the collected γ_k are added to a running average.
7. A different neuron is chosen (return to step 3) and steps 4 and 5 are again repeated.

The final lyapunov exponent is computed as the average of all the collected γ_k values in repetitions of step 3, 4 and 5 as shown below:

$$\lambda = \left\langle \ln\left(\frac{\gamma_k^n}{\gamma_0}\right) \right\rangle_{k,n} \qquad \text{(Eq. 6)}$$

where, the average **<.>** is taken over all time steps k and all neurons n. γ^n signifies the perturbation induced in the n^{th} neuron.

For an insight and discussion on how different structural parameters of the network affect the value of λ, see the appendix towards the end.

II.D. Some Observations on the Relationship between Local Lyapunov Exponent and the Structure of the ESN

Some of the observed results regarding the relationship between the dynamical complexity of the ESN and its structural properties – reservoir size, standard deviation of reservoir's weights and leak rate – are discussed in this section.

Dependence of λ on Reservoir Size

It is observed from experiments that the Lyapunov exponent λ generally increases with increase in the size of the reservoir, i.e., the number of neurons in the ESN's reservoir. This can be seen from the figure II.5 below:

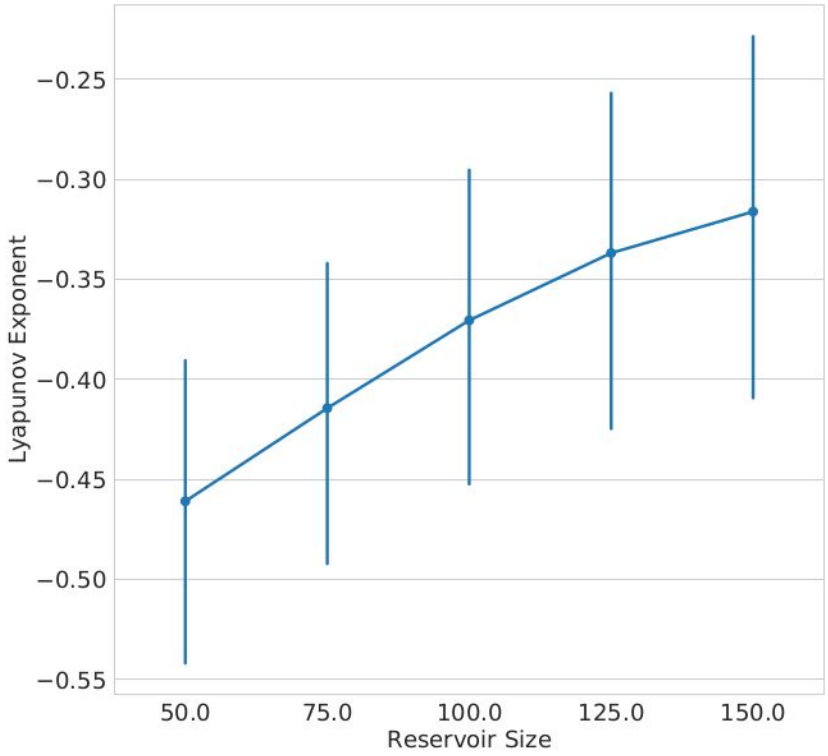

Fig II.5. Mean of the Lyapunov exponent increases with increase in the number of neurons in the reservoir. The variance in the values is due to different values of leak rate and different values of reservoir's weight matrix' standard deviations.

It can be concluded from the figure above that, in general, with higher number of neurons in the reservoir and everything *else fixed*, the network *tends to be less orderly*. Note that most of the network's with sizes 50, 75, 100, 125 and 150 are having a λ less than zero. However, what is significant is that the number of networks with λ greater than zero generally increases as the size of the reservoir increases. This is visualized in figure II.6 below:

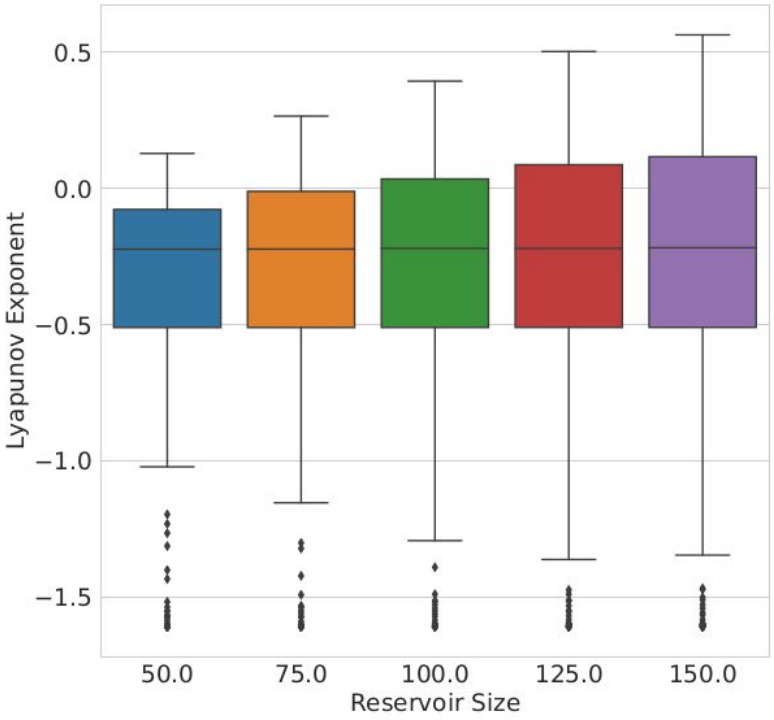

Fig II.6. As the size of the reservoir increases, there is a greater number of ESNs with λ greater than zero.

Thus, it can be concluded that there is a relatively higher probability of generating networks that exhibit sensitive dependence on initial conditions with larger reservoir size.

Dependence of λ on Leak Rate

The value of the largest LLE λ generally decreases with increase in the leak rate of the reservoir, keeping everything *else fixed*, as can be seen from figure II.7 below. Tests for other (different from the ones shown in the plot) values of leak rate between 0 and 1 were also conducted and trends similar to the one discussed below were observed.

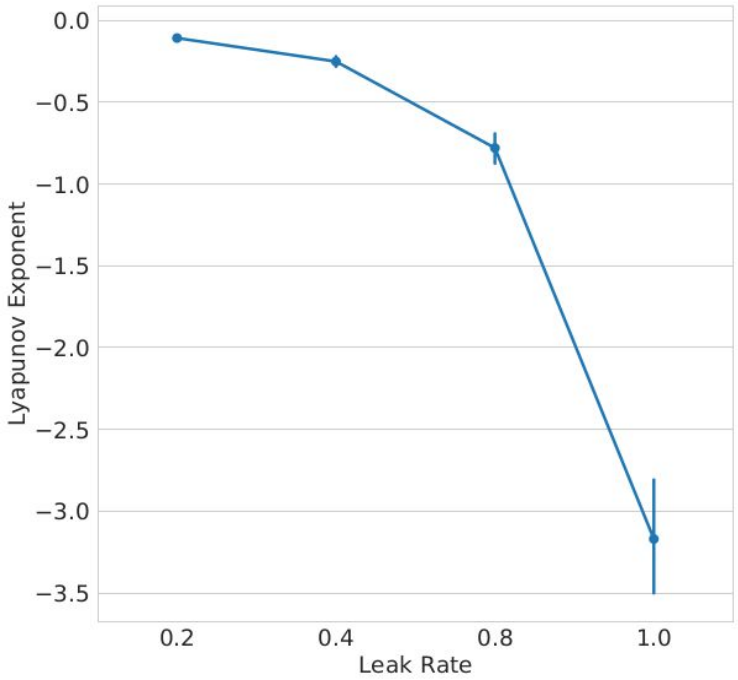

Fig II.7. As leak rate of the reservoir increases, a general decrease in the mean of Lyapunov exponent λ is observed. Note that there is a sharp decline in the value of λ when the leak rate is 1.0. See text for an explanation.

Recall that leak rate (α) is associated conceptually with the "memory" of the reservoir. If $\alpha = 1$, then the network exhibits no memory at all since all of the state updates that happen are due to the newly computed state of the network $\mathbf{x}_{k+1}$, whereas if α is closer to 0, it means that the network's evolution happens with the current state containing a track of its previous states (the so called *echo*). Thus, we can expect that if there is a small perturbation in the state of the ESN, then this perturbation is likely to remain as the trajectory evolves in time through the state space, if the network "possesses more memory" or lower value of α. Hence, networks with lower values of α are on average going to be *less orderly* than the ones with larger values of α. This is visualized in figure II.8 below. Note that when $\alpha = 1$, the sharp decline in the value of λ on average can be attributed to the fact that ESNs with "no memory" are mostly driven by input signals only and hence any minute perturbation in their state trajectories vanish *almost instantaneously* (since we compute the values of λ here only locally using LLEs). In our experiments, we use values of α much less than 1 and closer to 0.

In figure II.9, we show box plots depicting the relationship between α and λ combined with the variation in reservoir sizes indicated in different color schemes.

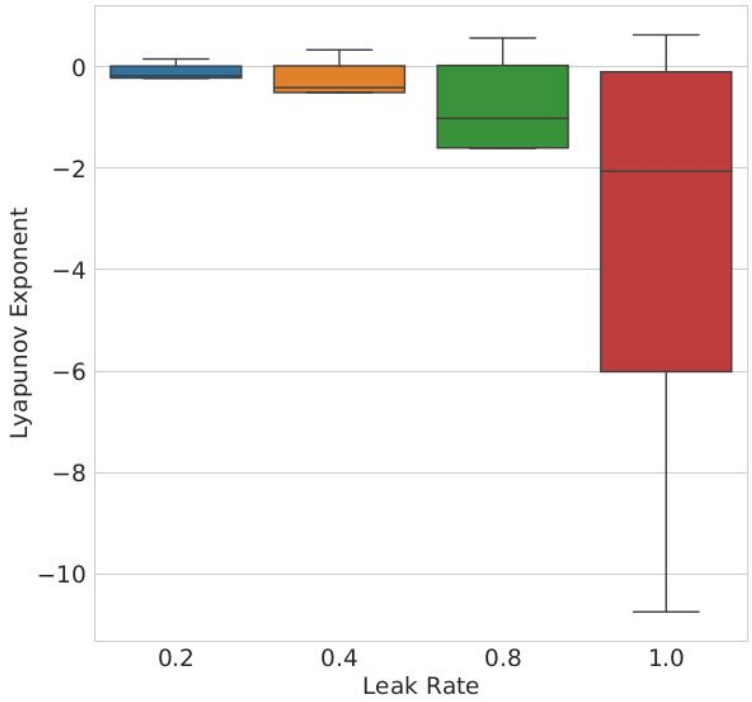

Fig II.8. Depicting the variability in Lyapunov exponent λ with leak rate of the reservoir.

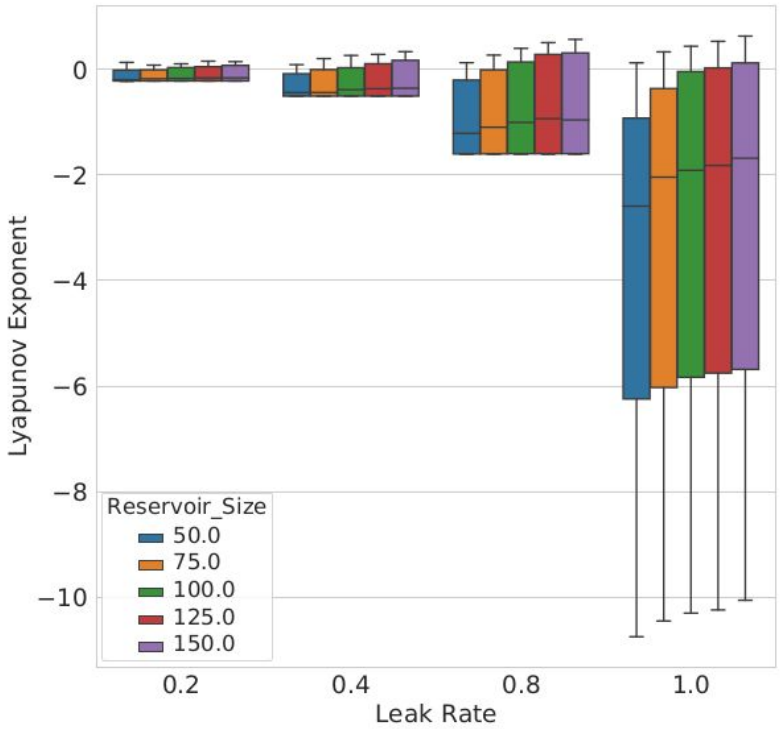

Fig II.9. Combined visualization of the effect of leak rate and reservoir size on Lyapunov exponent λ.

Dependence of λ on Variance of Reservoir Weights

It is observed that the Lyapunov exponent λ of the ESN increases with increase in the variance (standard deviation σ) of the weights of the reservoir as shown below in fig II.10. This means that with larger values of the reservoir weights, the network tends to show greater sensitive dependence on initial conditions. It was found from experiments that after $\sigma \approx 0.6$, the ESNs would mostly exhibit a $\lambda > 0$ for all sizes of reservoir starting from 50 neurons and above.

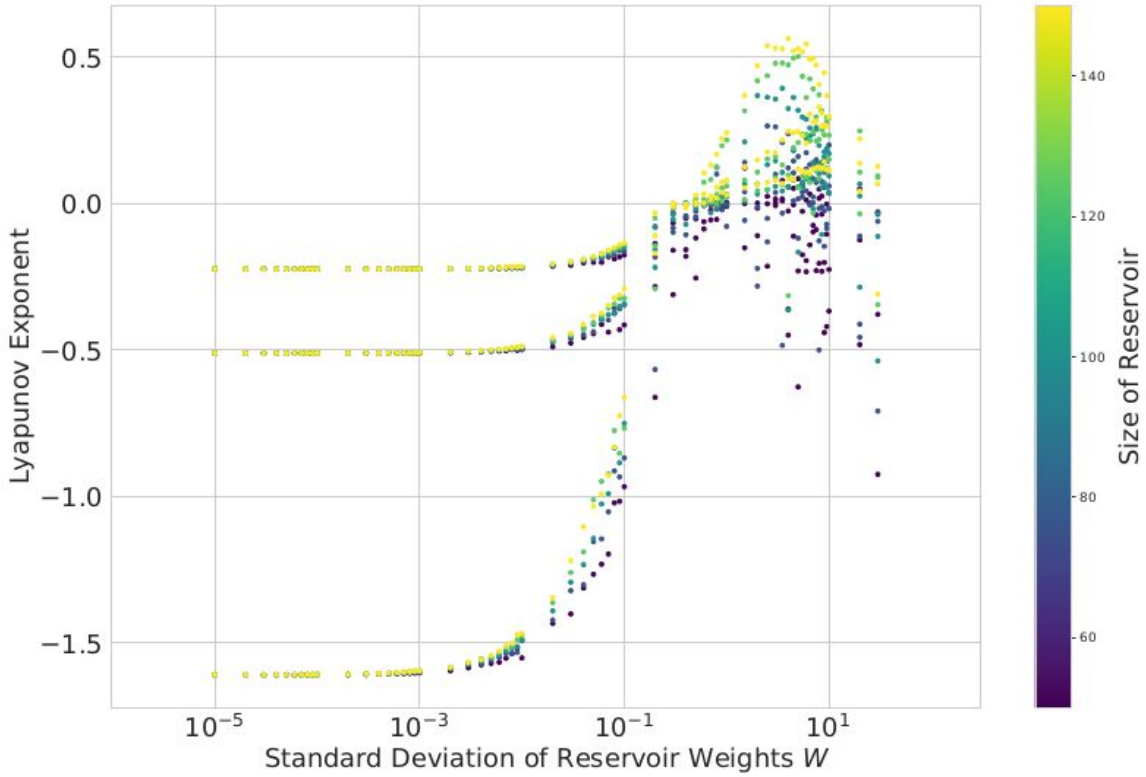

Fig II.10. Lyapunov exponent λ increases with rise in standard deviation σ of reservoir weights. Each dot represents a single candidate network. Reservoir size is depicted using the colormap and the three clusters of points formed on the left hand side of the plot are due to the three leak rates: 0.2, 0.4, 0.8 in order from topmost to bottommost cluster. See text for the reason behind the decline of λ values beyond a certain point as σ continues to increase.

However, the increase in λ holds only until a certain value of σ – below $\sigma \approx 5.0$ – after which λ begins to drop again. The drop in the value of lambda can be explained as a result of the numerical overflow due to the asymptotic behaviour of the S-shaped *tanh* activation function as one goes away from the origin on either side. With $\sigma > 5.0$, the input of the *tanh* function $\boldsymbol{f}$ (i.e., $W\mathbf{x}_k + W^{in}\mathbf{u}_{k+1}$, see Eq. 2) is, easily, enough further away from 0 on either side so that its output

(i.e. the new state of the reservoir) is an *N*x*N* matrix filled mostly with values very close to either +1 or -1. This quickly leads to all initial perturbation in the state of the reservoir to wash away leading to a decrease in the reservoir's sensitivity towards initial perturbations.

III. Computation at the Edge of Chaos

The phrase *Edge Of Chaos* (EOC) was coined by Doyne Farmer, an American complex system scientist, to describe the transition space between order and disorder that is hypothesized to exist within a wide variety of dynamical systems. It describes the phenomenon that *interesting/ meaningful* complexity, such as the one displayed by biological entities and their dynamics, lies in between the two extremes of boring order/regularity and noisy disorder/chaos. Early indications towards this hypothesis were reported by computer scientist Christopher Langton in 1990 in his studies on Cellular Automata (CA) where he established an experimental fact – in line with the ideas of Stephen Wolfram [23] – that "*the optimal conditions for the support of information transmission, storage, and modification, are achieved in the vicinity of a phase transition*" [1] or near EOC.

> *"Perhaps the most exciting implication [of CA representation of biological phenomena] is the possibility that life had its origin in the vicinity of a phase transition and that evolution reflects the process by which life has gained local control over a successively greater number of environmental parameters affecting its ability to maintain itself at a critical balance point between order and chaos."* [1]

Around the same time as Langton, physicist James Crutchfield described more or less the same concept using the phrase *onset of chaos*. Although the generality and applicability of the idea (as explained and interpreted in these works) had been called into question several times, the interest in the hypothesis on the connection between computational capabilities and edge of chaos has been growing ever since its inception. Doubts have been cast over the way in which one defines the parameters of the dynamical system over the state space that leads to the transition from order to chaos and the relationship of these parameters with the rise in complexity of the system in general (see [24] for one such study).

In this study, of course, we examine this hypothesis in the context of Echo State Networks. It has been shown [7, 8, 2, 27] that the ability of a reservoir computing network (echo state and otherwise [27]) to achieve a desired computation is maximized when the network is in a state near the edge of chaos. Or, in other words, there is a sudden peak in its computational capacity when it

undergoes the transition from order to chaos. However, it should be noted that the reasons behind increase in the network's performance in the vicinity of the edge of chaos remain elusive.

We use the largest local lyapunov exponent (as explained before in II.C.) as a measure of the dynamical regime of the ESN and several other indicators or proxies for computational capabilities of the ESN. We describe these measures in the following chapter.

IV. Measures/Tests/Proxies for Computational Capacity

As discussed in the introduction chapter, it is unclear how one can directly and quantitatively determine the computational capacity of a computer/dynamical system. However, one can indirectly conclude that *if a system is capable of performing a complex computation, then it can be labelled as a system with high computational capacity*. A **complex computation**, using ideas from Computational Complexity theory, can be informally defined in terms of the amount of resources – time and memory – required for doing it. The higher the resources needed, the more complex the computation is said to be. This ties in directly with the primitive components of computation: *storage* (memory/space complexity), *transmission* (time complexity) and *modification* (time and space complexity) of information, stated earlier in chapter I. Thus, in the absence of any direct measures, we take the help of indirect proxies for assessing computational capacity of our network. In this study, we have used the following indicators to quantitatively judge/determine the computational capacity of the echo state network. Note that transfer entropy is determined without the necessity of the ESN to produce output, whereas memory capacity, memory mean squared error and nonlinear autoregressive moving average require the network's output to be trained for each benchmark.

IV.A. Transfer Entropy (TE)

In order to capture information transmission at a local (neuron/node) level, we make use of information theoretic measures to describe distributed computational properties of the echo state network. **Transfer Entropy** is an information-theoretic measure for quantifying information transfer between each neuron pair in the reservoir. The information transferred between a source and a destination neuron is defined as the amount of information (in bits) provided by the source (source's past) about the destination's next state that was not already contained in the destination's history.

Concretely, TE (first introduced by Schreiber [28]) from a source neuron Y to a target neuron X is the *mutual information* between the previous state of the source y_n and the next state of the target

x_{n+1}, conditioned on the semi-infinite history of the target process $x_n^{(k)}$ (as $k \to \infty$, Lizier et al. [30]):

$$T_{Y \to X} = \lim_{k \to \infty} \sum_{\mathbf{u}_n} p(\mathbf{u}_n) log_2 \frac{p(x_{n+1} | x_n^{(k)}, y_n)}{p(x_{n+1} | x_n^{(k)})}, \quad \text{(Eq. 7)}$$

where $\mathbf{u}_n$ is the state transition tuple $(x_{n+1}, x_n^{(k)}, y_n)$.

Of course, since we cannot measure the TE over an infinite past of the target neuron, we compute $T_{Y \to X}(k)$ which denotes a finite-k approximation [29]. We measure the TE of the whole ESN (as a single value) by averaging over all the nonzero connections in the reservoir.

Note that higher values of TE signify the fact that there is a higher mutual information between a source neuron's current state and a target neuron's next state, which intuitively means that with higher values of TE we have more intraneuronal transmission of information which consequently signifies an increase in the ability of the network to perform better in one of the three primitive components of Turing universal computation, as stated earlier. *Thus, if a rise in the average value of the TE of ESNs is observed at the edge of chaos, it will count as indirect evidence for the increase in computational capacity of the ESNs and hence will lead to support towards the main hypothesis of this study.*

Multivariate Transfer Entropy

Transfer entropy, as defined in Schreiber [28] and eq. 7 above, is a *bivariate* measure of information transfer between two neurons (a target X and a source Y). While this is useful in a pairwise analysis of information transfer between neurons in the reservoir, it is inherently incapable of capturing the "actual" information transfer between pairs of neurons in a multivariate setting where any given target neuron's future state may be affected by multiple sources, like in our ESN's reservoir. While on the one hand, interaction of a target neuron with multiple sources (directly or indirectly via other neurons) may lead to inference of redundant amounts of transfer entropy due to cascading effects [32], on the other hand, bivariate TE may also miss out on capturing synergistic interactions between *multiple relevant sources* (defined as the set of all source nodes in the network that collectively contribute to the computation of the target node's next state) and the target in cases where multiple sources *jointly* transfer more information into the target than what could be detected from examining each source contributions individually

[32]. Thus, using the standard bivariate TE may lead to both false positive and false negative results and therefore, we resort to the computation of multivariate transfer entropy between two neurons in the reservoir.

Concretely stated, if $\boldsymbol{Y} = \{Y_i, i=1,2,...,K.\}$ denotes the given set of all the sources of a target neuron X in the network, then multivariate TE from a source neuron Y_i to a target neuron X is the information that the past of Y_i provides about the current value of X, in the context of *both* – X's own past (like in the case of bivariate TE) and all other sources of X, i.e. $\boldsymbol{Y} \backslash Y_i$. The challenge is then to find out the minimal set of *relevant* sources of X from the given set $\boldsymbol{Y}$ of all sources of X. One of the main reasons for finding only a minimal optimal set of relevant sources instead of a brute force search for all possible source-target combinations is to make the algorithm data-efficient. Since we only use a finite history of the states of both target neurons and corresponding source neurons, we only have a finite small amount of data to approximate the true multivariate TE. Note that this is clearly an intractable problem even for a small number of sources of X. As stated in Wollstadt et al. [29], since the exhaustive computation of multivariate TE is intractable (even for a small number of potential sources for a given target node), a suitable approximation algorithm is implemented in IDTxl [29]. For more detailed information on how multivariate TE is computed in IDTxl, refer to Wollstadt et al. [29]. A quick introduction to the multivariate TE algorithm implemented in IDTxl is at: https://github.com/pwollstadt/IDTxl/wiki/Theoretical-Introduction.

Some Implementation Details and Practical Considerations in approximating multivariate Transfer Entropy

We make use of the python package Information Dynamics Toolkit xl (IDTxl [29]) in order to efficiently infer the multivariate transfer entropy of the ESN. Some of the used parameter settings and their significance are discussed below:

a. *Conditional Mutual Information (CMI) Estimator*: As seen in eq. 7 above, transfer entropy requires computation of conditional mutual information between random variables (or functionals thereof), which involves reconstructing – essentially, estimating – probability distributions for the processes of interest from *finite* data samples. This estimation of a probability distribution is obviously a complex problem with several partially conflicting goals. Any estimator can be characterized in terms of its bias and

variance. The bias and variance of an estimate are its systematic deviance from the true value and its variability across different realizations of the sampling, respectively. The estimate converges to the expected value as the number of samples grows. Also, one estimator is more efficient than another if it needs less samples to achieve a given level of variance. Generally, when computing an estimator, one is interested in controlling the balance between bias and variance. For example, if one decides to contrast the estimate from one (empirical) data set with that of surrogate data (see c. below, refer to [32]), one might choose to reduce the variance (statistical error) of its estimate at the expense of increasing the bias as the increase in bias of the estimate would cancel out if the surrogate data is suspected to have a bias similar to the observed empirical data [32]. In a contrasting situation, if one decides to interpret the estimated value as expected of the true distribution, one might also choose to have an estimator with a low bias. Thus, selecting the appropriate numerical estimator for a given application is crucial and largely depends upon the the number of available samples, the levels of quantization of the samples, the optimal bias-variance balance, and the requisite computational resources [32].

Since our ESN states data is continuous (real numbers), we use a continuous variable's estimator in our experiment called *Kraskov-Stögbauer-Grassberger (KSG) estimator* [34] for estimating the conditional mutual information. While the most intuitive way to approximate a probability distribution is the histogram approach of binning the state space, rather arbitrarily, and counting the number of samples falling in each bin, the KSG estimator follows a nearest-neighbour technique which exploits the statistics of distances between neighbouring data points in a given high dimensional space, in a data efficient way. Intuitively, the larger the distance between one point to its nearest neighbor the lower the local density around that point. KSG estimator is a nonlinear and model-free—only makes a mild assumption about the continuity and smoothness of the estimated probability distribution, and no assumption on its family [32], which seems appropriate in the case of our ESN's state space—estimation technique with bias correction, high data efficiency, accuracy, robustness to noise and is effectively parameter-free (the number of neighbours to consider in the estimation process is indeed a parameter to incorporate; however, results are usually relatively stable to a reasonable

choice of this parameter. The chosen value of the number of nearest neighbours is 4, for the JIDT implementation of KSG estimator used in IDTxl [33]). Thus, it is a widely used estimator for all practical purposes. For a detailed description of the KSG and other estimators, refer to [32].

b. *Time-lags*: Time-lags for target and source time series are set separately. We set values for maximum and minimum time lags for each of the target and source time series. Maximum time-lag value specifies how far back in time should we look for a potential, predictive information transfer – for both target and source time series. These maximum time-lag values are set to 3 for both target and source time series in our computation. Minimum time-lags are set to 1 for both target and source time series. For a target time series, time-lag of 1 ensures self-prediction optimality [35]. For a source time series, time-lag of 1 means that there is an immediate (unit time-step) information transfer from source neuron to target neuron. Any value greater than 1 for minimum time-lag in the source time series would not make sense in our discretized-ESN's case since data is generated over discretized time points rather than sampled from a continuous data-stream with a certain sampling frequency.

c. *Statistical Testing*: As stated earlier in a., the probability distribution estimate of the conditional mutual information is never free from bias due to finite data samples. For instance, two (actually) uncorrelated time series with zero transfer entropy would still exhibit some non-zero value of transfer entropy simply because they are finite in the data samples—a potential Type I error. To check if an estimated TE value is non-zero because of bias in the estimator due to finite data or because of "actual" information transfer, one has to estimate the expected values of TE for finite (same length as original) data that are as close as possible to the original data but have no information transfer. This is what is called as surrogate data [32]. Surrogate data should have (at least) the same autocorrelation properties as the original data, while at the same time it should be guaranteed to have no predictive information transfer. In IDTxl, this is achieved by destroying the temporal precedence between the source and the target time series by permuting the samples in time, that would otherwise have a potential predictive information transfer. Note that this method of creating surrogate data is only applied when there are not enough trials (or replications) of the time series data for the same

process. In our case, we only have 1 replication of the states of all the neurons in the reservoir per unit time-step.

Once a (unbiased) distribution of TE values from surrogate data is computed, the TE value from the original data is compared for statistical significance. The statistical tests employed in IDTxl are mainly controlling the probability of a Type I error (i.e., Family-wise Error Rate) at a node-node level by implementing the false discovery rate correction. A quick introduction to the statistical tests used in IDTxl is at: https://github.com/pwollstadt/IDTxl/wiki/Theoretical-Introduction#statistical-tests-used-in-the-mTE-algorithm.

IV.B. Memory Capacity (MC)

Memory Capacity (MC) denotes a measure for how long an ESN can recall its past input, the evaluation is the correctness of remembered inputs. Said input is a time series of random draws from a uniform distribution with range [-1, 1]. The ESN gives theoretically an infinite amount of time series as output with the corresponding target sequences being the same as the input sequence, but each delayed by an incrementing k that goes from 1 to infinity. [9]

Mathematically, MC is defined as:

$$MC = \sum_{k=1}^{\infty} \frac{cov^2(y_k, o_k)}{var(y_k)var(o_k)} \qquad \text{(Eq. 8)}$$

with y_k being the k-th target sequence and o_k being the k-th output sequence of the ESN. cov^2 gives the squared covariance, var the variance. [9]

In order to compute MC, a delay limit for k has to be set. We set it to 300, which is the limit used for MC by Matzner [9] and Boedecker et al. [8]. Jaeger [15] investigates the delay limit used for MC in the context of reservoir computing and puts 300 well above the time that our kind of ESN is able to remember a past input. Evaluating MC for delays after which the ESN is not able to remember its input just adds zeros or values close to zero to the sum in eq. 8, so it does not change MC significantly to cut the part after a delay of 300 (or even earlier).

To acquire an intuition for the measure, figure IV.1 shows input/output-sequences of an ESN while running the MC benchmark.

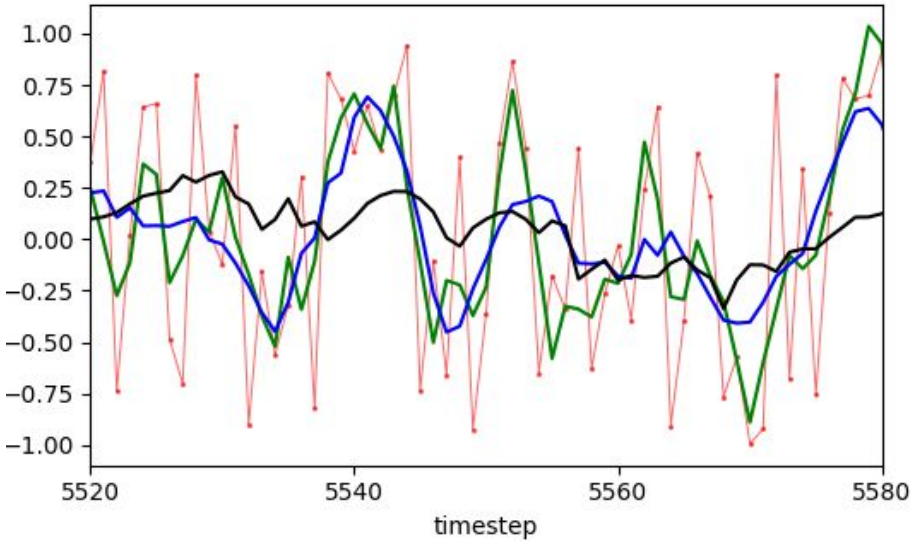

Fig. IV.1. Target sequence with delay 0 (red, equals input sequence) and predicted time series (extracts) for MC/MMSE with delay 2 (green), delay 10 (blue) and delay 100 (black). Note that the predicted time series are horizontally shifted by the negative value of their delay to match the target series' position.

IV.C. Memory Mean Squared Error (MMSE)

Memory Mean Squared Error (MMSE) has a comparable function as MC, but states an error in remembrance and is reported to be numerically more stable [9]. Additionally, MMSE is designed to have a limited delay k, otherwise input and target sequences are similar to MC. [9]

$$MMSE = \sqrt{\langle (o_k(t) - y_k(t))^2 \rangle_{t,k} \,/\, var(y_0)} \qquad \text{(Eq. 9)}$$

with $o_k(t)$ being the value of k-th output sequence at time t, $y_k(t)$ the value of the k-th target sequence at time t (y_o is therefore the input sequence). $\langle . \rangle_{t,k}$ gives the arithmetic mean over all output/target sequences and all time steps t. var calculates the variance. [9]

Because of the similarity between MC and MMSE and the finite nature of MMSE in contrast to the theoretically infinite sum of MC, MMSE's delay limit does not have to be higher than the one used for MC, thus we set it to 300 as well.

Figure IV.1 is also applicable to gain an intuition for MMSE.

IV.D. Nonlinear AutoRegressive Moving Average (NARMA)

Nonlinear AutoRegressive Moving Average (NARMA) is a quantity of error in prognosticating a nonlinear combination of earlier random inputs $x(t)$ of an ESN. Each value of the sequence $x(t)$ is randomly drawn from a uniform distribution with the range [0, 0.5]. Additionally to remembering the inputs, the neural network's ability to work with them is tested by NARMA. The combination which the network is supposed to prognosticate is defined as:

$$y(t+1) = 0.2\, y(t) + 0.004\, y(t) \sum_{i=0}^{29} y(t-i) + 1.5\, x(t-29)\, x(t) + 0.001 \qquad \text{(Eq. 10)}$$

with $y(t) = 0$ for $t < 0$. [9, 31]

NARMA is evaluated using the *normalized root mean squared error (NRMSE)* of the prediction. [9]

$$NARMA = \sqrt{\langle (o(t) - y(t)) \rangle_t / var(y)} \qquad \text{(Eq. 11)}$$

with $o(t)$ being the value of the output sequence at time t and $y(t)$ the value of the introduced target sequence at time t. $\langle . \rangle_t$ gives the arithmetic mean over time, var the variance. [9]

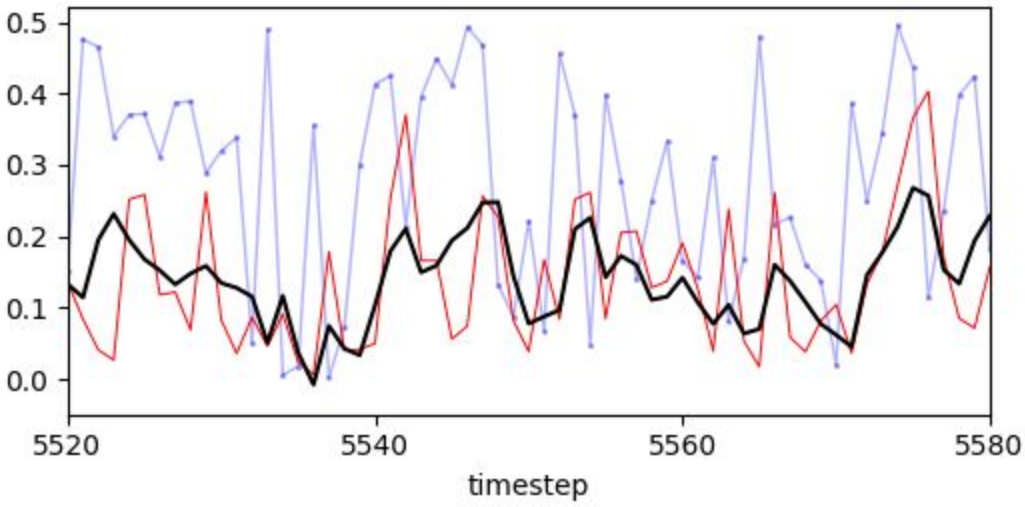

Fig. IV.2. NARMA computation by an ESN: target sequence (red), predicted sequence (black), input sequence (blue). (extracts, not shifted)

V. Results from Random Echo State Networks

This chapter shows how ESNs with randomly generated reservoirs perform on each of the aforementioned computational capacity measures in the light of the hypothesis of computation at the edge of chaos. Firstly, we describe the experimental settings used and then we state the results. As discussed below, we see strong evidence in support of the hypothesis. The experiments follow the ideas from Bertschinger et al. [2] and Boedecker et al. [8] and are largely a repetition of the experiments carried out in the latter work.

V.A. Experimental Settings

ESN Reservoir Weights' Standard Deviation

In Boedecker et al. [8], the reservoir weights of the random ESNs are drawn randomly from a normal distribution with a standard deviation σ. σ is used as one of the control parameters in the experiment to set the dynamic regime of the ESN as depicted above in fig II.2 and fig II.3 where it is shown how the dynamics of the ESN depends on its spectral radius, which in turn is a function of the σ. In our experiment, σ takes its value from within an approximate range of $3x10^{-2}$ to $495x10^{-2}$. The choice was made so that the Lyapunov exponent (λ) varies in a suitable range around 0 in order to capture a significant part of the spectrum from order to chaos. λ varies between -0.4 to 0.4 approximately for this range of σ (when the leak rate is set to 0.4) and between -0.15 to 0.10 (when the leak rate is set to 0.15). Within the stated range, we vary σ more finely near the edge of chaos ($\lambda \approx 0$) so as to get more data in the region of interest.

ESN Architecture

The reservoir consists of 100 neurons for the computation of transfer entropy and 150 neurons for other computational measures as proposed by Boedecker et al. [8]. There is no particularly technical reason for keeping the reservoir size at 100 for the transfer entropy task. The choice of 100 neurons was made only to finish up the computation in a shorter time span. The neurons have a negligible bias (bias has been set to 10^{-40}) and we use the *tanh* activation function.

Besides the reservoir neurons, the network has one input neuron with connections to each reservoir neuron. As suggested in [9], the input weights for the connections between the input neuron and the reservoir neurons are drawn uniformly from [-0.1, 0.1].

In addition to the input neurons, we also have 301 output neurons in the ESN. NARMA is represented using one output neuron, another 300 are used for MC and MMSE (when using 300 as the delay limit for MC and MMSE). Each of these 300 neurons have the same basic target sequence but with different delays, so there is one output neuron for each delay. MC and MMSE are sharing the same neurons as their target sequences are the same.

Besides the standard deviation of the reservoir weights' distribution, another parameter that controls a reservoir's behaviour is the leak rate (see chapter II.A.). For the computation of transfer entropy, a leak rate of 0.4 was chosen to ensure a moderate variability in the values of λ (see figure III.5) to allow for a better capturing of the regime at and around the edge of chaos. To find a suitable value of the leak rate for the other computational measures, a short grid search was conducted using σ and β parameter of the ridge regression algorithm (see chapter II.B). The value of leak rate was chosen based on the best performing candidates on the computational tasks. A leak rate of 0.15 was found to be the most suitable (networks with leak rate of 0.2 having almost equal performance, and the ones with leak rate of 0.1 and 0.25 slightly worse).

Input and time steps

The input of each time step is randomly and uniformly chosen from a range [-1, 1] for MC and MMSE; while a range of [0, 0.5] is used for NARMA and TE calculations (it didn't affect which of the two ranges for input drive was chosen for TE). The ranges were so chosen to keep the input drive (for all tasks) comparable in value to the state of the reservoir (after discarding the initial transients), so that the network remains input-driven to an appropriate extent. The first 1000 time steps are used for stabilization of the ESN and hence discarded. The output produced during the next 1000 steps is the basis for training the network for the MC, MMSE and NARMA tasks. Another 1000 time steps are used to compare the now trained network's output to its target values in order to compute the benchmark scores [9]. The TE calculations are done using only the first 1000 time steps after discarding the transients.

Since the input ranges differ for MC/MMSE and NARMA, the random reservoirs' experiment is conducted two times. Yet, the ESN's architecture is designed to evaluate all benchmarks in the

same step, since that is how it is required for the the neuroevolution experiment (see chapter VI.D).

Multiple evaluations

The repeated computation of the same benchmark score for a given reservoir results in considerably varying values, therefore each score is computed three times as proposed in [9]. The final score is the mean over those three values, the standard deviation is stored alongside. Every calculation includes an independent training of the output weights.

V.B. Results

Here we show the results of our experiments with random reservoirs for each of the computational measures of the echo state networks.

Transfer Entropy

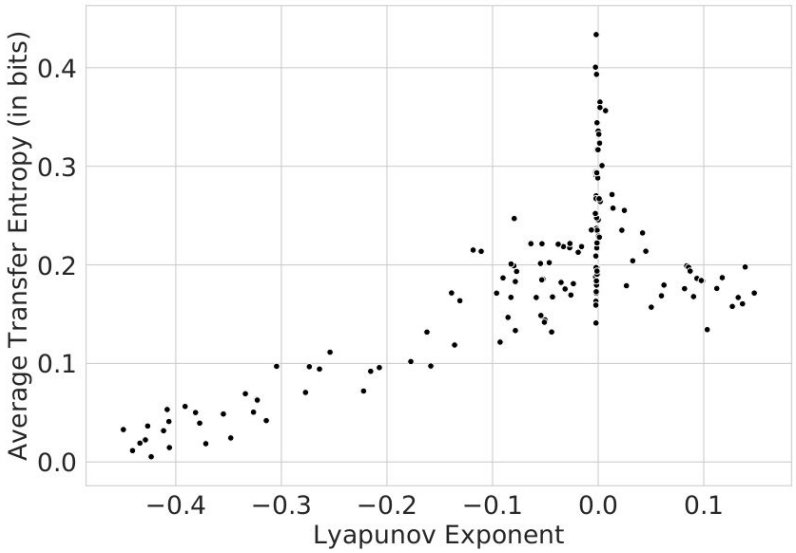

Fig. V.1. Depicting the change in average Transfer Entropy (in bits) of the ESN with respect to the spectrum of dynamics (measured by λ) of the ESN from orderly to chaotic. The transfer of information within the network sharply increases at $\lambda \approx 0$.

As the figure V.1. below depicts, the average transfer entropy of the network increases from almost close to zero to a considerable positive value as it approaches the edge of chaos. Then,

there is a sharp rise in the transfer entropy of the network, on average, near the edge of chaos ($\lambda \approx 0$), on the orderly side of the dynamic regime as expected (See [8] and [9]). This observation strongly aligns with the main hypothesis of the study that computational capacity of a system peaks near the edge of chaos. The reason for such a rise in the transfer entropy of the ESN was earlier elaborated in section IV.A.. As the network enters into the chaotic regime, one expects only little or almost no information transfer, which means the value of transfer entropy of the ESN in $\lambda > 0$ region should be (almost) zero. However, this expectation is clearly not met by our plot above. A possible reason for this is explained later in this section where we compare our findings with the works of Boedecker et al. [8] and Matzner [9].

Memory Capacity

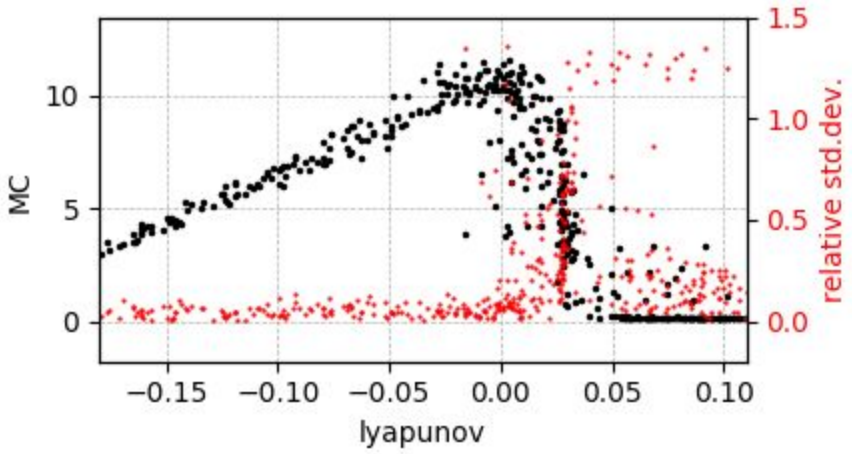

Fig. V.2. MC score (black) and its relative standard deviation (red) against lyapunov exponent (x-axis). Each pair of black and red dots represents a single reservoir. Higher is better for MC, lower for the relative standard deviation.

Figure V.2 shows the memory capacity (MC) score and its relative standard deviation (coefficient of variation) plotted against the lyapunov exponent of the echo state networks' reservoirs. As explained above, the benchmark scores for each reservoir are averaged over three calculations. The aforesaid standard deviation is taken between those three values. The maximum MC score is reached around the edge of chaos, between $\lambda = -0.035$ and $\lambda = 0.01$. Shortly afterwards, the score drops rapidly and goes in most parts to a value near 0 at $\lambda \geq 0.05$. While the reservoir increasingly roams in the chaotic regime, the time series that are part of MC evidently cannot be predicted

reliably anymore.

The relative standard deviation takes values mostly between 0 and 14% in the ordered regime but reaches higher values once the edge of chaos is passed.

Memory Mean Squared Error

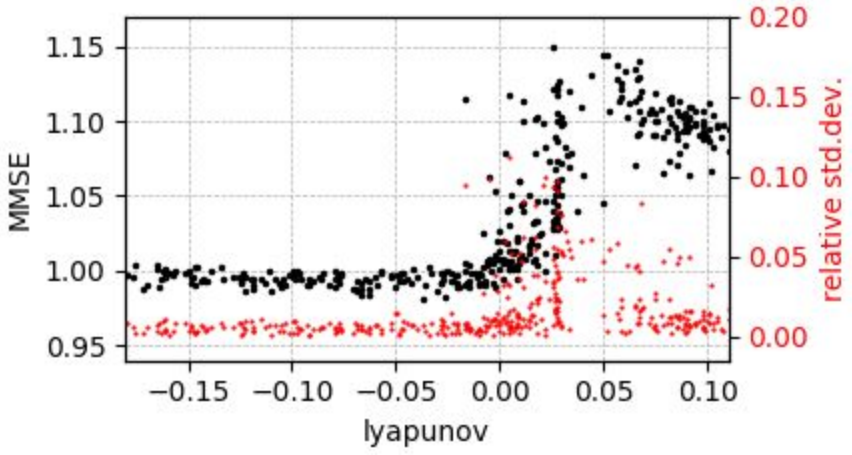

Fig. V.3. MMSE score (black) and its relative standard deviation (red) against lyapunov exponent (x-axis). Lower is better for both measures. Note the different scale of the relative standard deviation axis in comparison to the other benchmarks' diagrams.

The memory mean squared error (MMSE) benchmark's best score (lower is better) is reached on the ordered side of the edge of chaos with the maximum being quite broad. Just as with MC, the score rapidly gets worse shortly after getting into the regime of chaotic dynamics. The relative standard deviation is much smaller than the one of the MC score (note the different scaling) and gets highest values at $\lambda \approx 0.25$.

Nonlinear AutoRegressive Moving Average

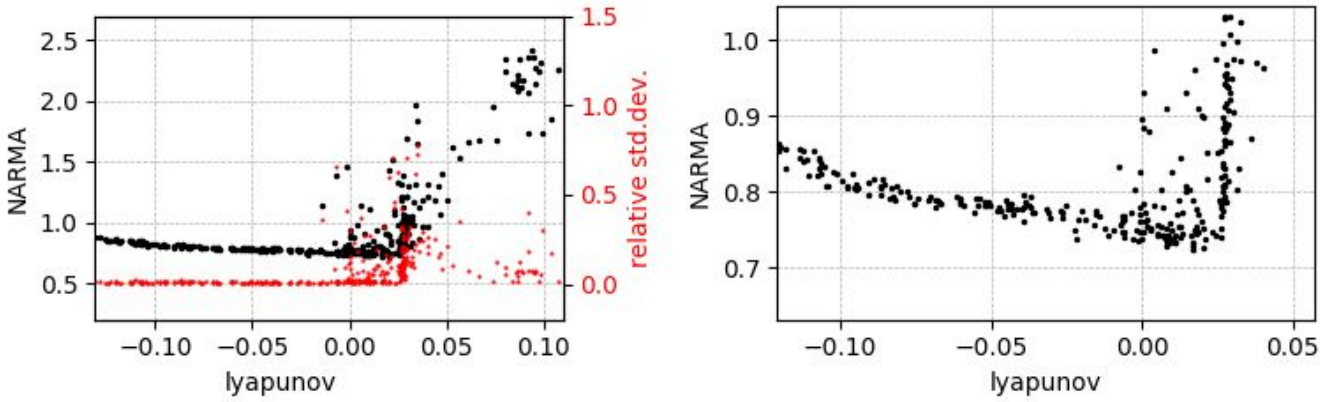

Fig. V.4. NARMA score (black) and its relative standard deviation (red) against lyapunov exponent (x-axis). Lower is better for both measures. The right figure is a zoomed-in version of the NARMA score.

Figure V.4 shows the nonlinear autoregressive moving average (NARMA) score (lower is better) in relation to the lyapunov exponent.

Just as with the other two benchmarks, the score gets best near the edge of chaos. The optimum here is located shortly after λ passed 0, so it's on the chaotic side of the edge of chaos. Directly afterwards, the score worsens rapidly. The same holds for the relative standard deviation which is rather insignificant for $\lambda < 0$, but gets high though scattered for $\lambda \geq 0$.

Comparison to other works

Since the resulting scores and information transfer, in general, in the experiments above have quite dispersed values, an exact point of maximized performance of reservoirs is hard to define. Yet, it is safe to say that the best performances in the utilized benchmarks are reached in the region close to the edge of chaos. This finding is in accordance with the cited experiments by Bertschinger et al. [2], Boedecker et al. [8] and Matzner [9]. By using MC and MMSE on the one hand and NARMA on the other, two foci on different qualities of a reservoir are applied (see chapter IV). The maximized performance near the edge of chaos holds for each benchmark, which of course cannot be seen as adequate representatives for every task an echo state network may be set on. Nevertheless, when seen jointly with the fact that information transfer (TE) also peaks at near the edge of chaos, it suggests a more general validity of the edge of chaos as the computationally efficient region on the dynamic spectrum from order to chaos.

A comparison with Boedecker et al. [8] and Matzner [9] on the quantitative side reveals differences. Regarding computation of transfer entropy, one thing that seems peculiar about Fig V.1—when compared with corresponding figures in [8] and [9]—is that we have considerable non-zero values of TE before and (especially) after the edge of chaos; while in the other previous works, we see only a negligible amount of information transfer in the chaotic as well as orderly regime. The only possible reason for this seems to be that IDTxl's algorithm for transfer entropy is reporting many false positives (Type I errors) during statistical testing. This can be directly linked to the fact that our data is not abundant enough for a better approximation of the TE values. While authors in [8] and [9] used 2000 time-steps long time series of reservoir states to compute their respective measures of transfer entropy, we have used only 1000 time-steps for the same computation (in order to speed up computation). Hence, using a longer time-series should definitely improve the measure of transfer entropy. However, this maneuver to decrease computing time clearly did not have any effect on the result of the study as far as the hypothesis of computation at the edge of chaos is concerned.

The benchmark scores that we reached for the NARMA task are significantly worse than the scores of the named papers. Both achieve results of about 0.41 whereas our best NARMA score amounts to 0.72.

As for the memory capacity (MC), the two papers are diverse in their results. Boedecker et al. [8] reach scores up to 39 with single reservoirs, but the vast majority of scores at the edge of chaos remain under 20, even 10. In contrast to Matzner [9] and our study, Boedecker et al. [8] didn't take the mean out of multiple evaluations. This could explain the scattering of the values and would make their result close to ours. Matzner reaches Boedecker et al.'s maximal score of 39 consistently at $\lambda = 0$, even going slightly above 40. They report the difference to the results of Boedecker et al. without giving an explanation, but state their congruency to the scores of Barančok and Farkaš [7] [9].

The MMSE benchmark was not applied to the randomly generated reservoirs by either Matzner or Boedecker et al., but Matzner used MMSE in their neuroevolution experiment. There, the values go to almost zero, which is a distinctly better result than our 0.98. It has to be kept in mind that the two compared values have a different context.

VI. Neuroevolution of Echo State Networks

In the following, we describe the conduction and results of the *neuroevolution experiment*. It investigates whether evolutionary methods that develop an ESN's reservoir weights push it towards the edge of chaos. The experiment follows a similar one described by Matzner [9]. Once more, the results support the idea of favorable computation at the edge of chaos.

VI.A. Neuroevolution

Neuroevolution is an algorithmic technique that can change a neural network's weights and topology as to improve the network by standards of an indicated measure. The algorithm is based on evolutionary algorithms in general and was adapted to be able to work with a neural network as an individual. The fitness value can - and will in our context – be the network's performance on a certain task. For an in-depth introduction to neuroevolution, see for example [11].

NEAT and HyperNEAT

Matzner [9] uses HyperNEAT [10] as the neuroevolution technique, we rely on NEAT [6] however. We see it as the far simpler technique – trying to work with HyperNEAT made us realize that our intended application would go beyond this study project's scope. Nevertheless, we do not want to skip shortly discussing possible advantages the use of HyperNEAT could have:

Whereas NEAT directly encodes the neural network's topology (genes represent connections), HyperNEAT encodes indirectly. It uses the NEAT algorithm to evolve a neural network called *connective CPPN* that, in turn, is used to construct the desired neural network, which can be extracted with arbitrary neuron density. The CPPN can be smaller than the desired network by magnitudes thus less expensive to evolve. The target network's topology is encoded geometrically in the CPPN – neurons have a spatial position there. That gives the possibility to include a problem's geometric representation in its solving by the evolving network. [10]

VI.B. Experiment description

We use neuroevolution to evolve ESN's reservoirs that were initially generated randomly. In the experiment, the task is the evolution of existing reservoir connections without touching the topology. The scope here lies on the reservoirs' position in regard to the regime of ordered or chaotic dynamics. A topology for all reservoirs is initially generated randomly and remains unchanged during the neuroevolution as to be able to recombine connection weights.

Each reservoir is used to compute a specific benchmark, for which output neurons are added and output weights are trained. The weights of the connections between the input neuron and the reservoir neurons are considered as part of the reservoir in this context. The performance on the benchmark is measured and the equivalent score is taken as the fitness value of the corresponding reservoir for the following steps of the neuroevolution algorithm.

Apart from speciation, which gets introduced in speciation chapter IV.C, the workings of the neuroevolution algorithm NEAT are not discussed further here, but can be reviewed for example in [6]. A basic understanding of evolutionary algorithms in general should be sufficient to be able to follow this report.

VI.C. Implementation

The echo state network components as well as the benchmark implementations (see page 4) that were used in the random reservoirs experiment are also suitable to serve the neuroevolution experiment. From NEAT's point of view, those components just return a fitness value for a given evolved reservoir. The neuroevolution experiment's main file can be found in the introduced GitHub repository (see page 4) in *Neuroevolution_Experiment/neuroevolution_experiment.py*. It builds the framework between the NEAT-implementation *peas* (see next paragraph) and the named ESN- and benchmark components (*esn_cell.py* and *benchmark_esn.py*).

Modifications of neuroevolution module peas

For performing neuroevolution, we used the Python module *peas*[3], which implements the NEAT algorithm (as well as the HyperNEAT algorithm). Since our experiments are based on Python 3, we changed the peas implementation from the original Python 2 basis to make it compatible to the newer version.

Another change that we applied can be found in the initializer of the class *NEATGenotype* implemented in *peas/methods/neat.py* (lines 152-153). Now the layer of output neurons is set to *max_layer*, which normally is the maximal value representable by *max_layer*'s data type. Originally, this layer index was just one ahead of the last reservoir neuron's, which resulted in problems inserting neurons in between. This behaviour did not affect the described experiment though, but unfinished attempts of altering topology.

VI.D. Settings and parameters

Fitness function and input range

As proposed by Matzner [9], we define the fitness of a reservoir as *2-MMSE-NARMA* (see chapter IV for explanations of MMSE and NARMA). Using the MMSE and NARMA benchmarks separately as in the random reservoirs experiment offers the opportunity to apply different input ranges. Now, however, the fitness function combines both benchmarks, thus, a single input range has to be used. As [-1; 1] holds values outside the input range advised for NARMA, we use the range [0; 0.5].

Initial positions in regard to the edge of chaos

Crucial for the experiment's course is the initial position of the reservoirs in regard to the edge of chaos because the random reservoirs experiment suggests highest fitness values there. We conducted the experiment two times with differing standard deviations of the normal range from which the initial reservoir weights are drawn. The first run has a standard deviation of 0.01, resulting in ordered dynamics, the second conduction gets initialized using 1 as standard

[3] see https://github.com/noio/peas

deviation. We supposed producing chaotic dynamics with the latter value, but the assumption gets relativized by the results.

NEATGenotype parameters

Some standard parameters of NEAT have to be adjusted, mostly set to smaller values to fit size and weight range of the reservoirs used in the experiments. The adjustments were set by repeated try-out and include the following *NEATGenotype* (found in *peas/methods/neat.py* (line 29ff)) parameters (new values in brackets):

- *prob_mutate_weight* (0.2): probability of mutation for each reservoir weight
- *stdev_mutate_weight* (0.01): standard deviation of the normal-distributed random draw for a weight's value change while mutating
- *prob_reset_weight* (0.008): probability of resetting a connection weight to an initial random value (with the discussed initial standard deviation)
- *prob_mutate_bias* (0.1): probability of mutation for each bias weight
- *stdev_mutate_bias* (0.01): standard deviation of the random bias weight's value change while mutating

Speciation

The NEAT algorithm includes a speciation mechanism which allows individuals in a formerly unexplored part of the problem's search space to evolve there, competing firstly only with similar/near individuals before facing a larger part or the whole population. With that, innovations have the time to develop properly and individuals are given the chance to search for local optima. In order to do so, a distance consisting of the differences in two individual's topologies as well as the differences of the individual weights is calculated. As the topology is fixed in this experiment, only the weight differences account for the total distance between two individuals, factorized by the parameter *distance_weight* of the *NEATGenotype* class. The threshold that decides whether two individuals belong to the same species is the parameter *compatibility_threshold* of the class *NEATPopulation* (found in *peas/methods/neat.py* (line 419ff)). It's adjusted automatically to reach a target amount of species *target_species* by adding or subtracting *compatibility_threshold_delta.*

As the standard values of those parameters doesn't match the ones required for our problem, we found it to be crucial to adjust them accordingly. We set *target_species* to a sixth of the population size, the initial *compatibility_threshold* to 0.01, its delta value to 0.001 and *distance_weight* to 1.

Architecture settings

The neuroevolution's population consists of 75 reservoirs, each being made of 50 neurons (plus one input neuron). Differently to the random reservoirs experiment, the reservoir neurons have biases, which are set to zero in the beginning. During the neuroevolution, those values change, controlled by the NEAT parameters introduced below.

Parameter choices to reduce computational cost

The amount of neurons used in a reservoir is less than the 150 neurons (plus one input neuron) utilized by Matzner [9] as well as the 150 used in the random reservoirs experiment. Also, the population size of the neuroevolution has the value of 75 in contrast to 150 in Matzner's experiment. Both values are relatively small due to the high computational cost of this experiment, especially with using NEAT instead of HyperNEAT. Reducing the reservoir- and population sizes results in fewer and faster computations of the fitness values as well as the ESNs lyapunov exponents. Additionally, the time needed for the NEAT algorithm is reduced.

VI.E. Results

Initial low weights' standard deviation

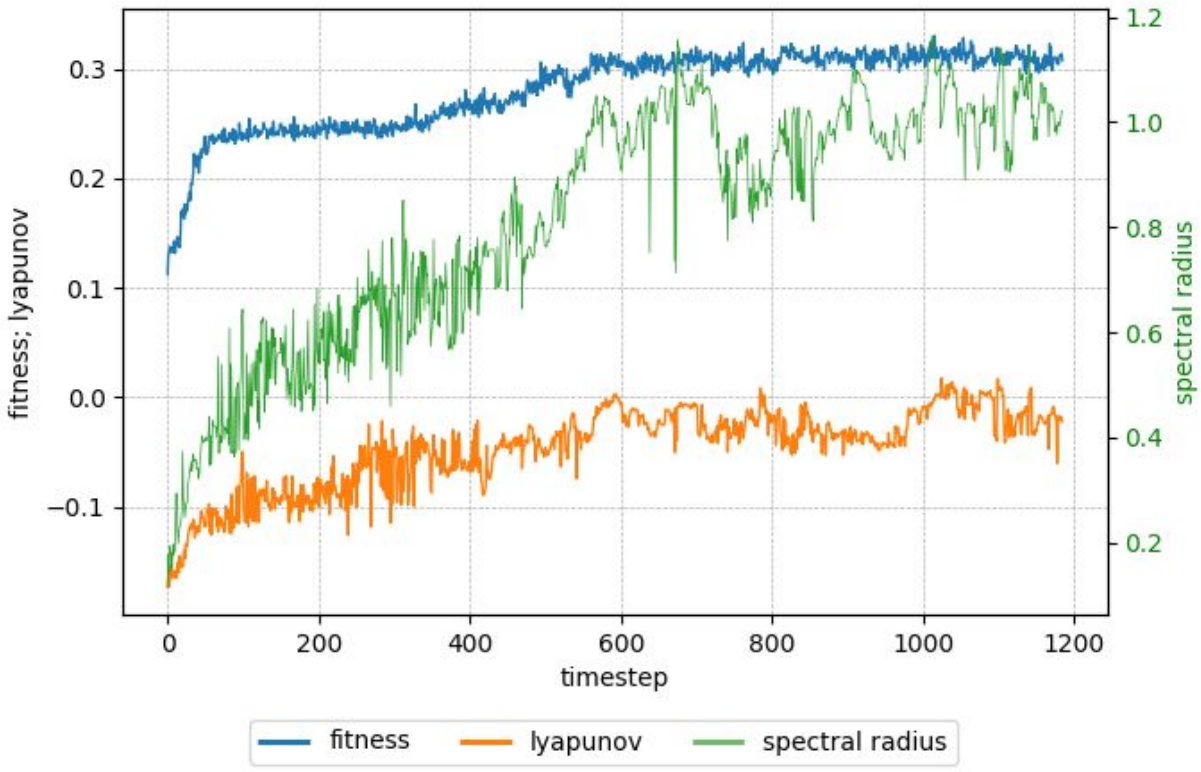

Fig. VI.1. Fitness (blue), lyapunov exponent (orange) and spectral radius (green, note the different scale) of the best reservoir of each generation while neuroevolution runs. Initial weight's standard deviation of 0.01.

Figure VI.1 shows the development of a population of reservoirs which gets evolved by the NEAT algorithm. The values depicted are the fitness, lyapunov exponent and spectral radius of the best reservoir of each generation. The initial standard deviation of the reservoir's weights is set to 0.01, which results in reservoirs with lyapunov exponents of about -0.17 to -0.18. The fitness is supposed to be maximized (see figure VI.2) near the edge of chaos, so at a lyapunov exponent of around 0. The expectation that the reservoirs develop into the direction of maximized fitness is met. After a steep rise of the fitness in the first 35 generations, it takes another 525 generations to get to the area where fitness does not improve anymore apart from oscillations.

These cover a range of fitness values between 0.29 and 0.33.

Meanwhile, the lyapunov exponent generally rises to a value of about -0.05 and then oscillates between -0.05 and 0.02. That range requires notice, being comparably wide, without having a huge effect on the fitness.

This is rather remarkable, since at least the NARMA benchmark, which is besides MMSE a component of the fitness function, has a distinct best score which is not reached before the edge of chaos (see chapter V, figure V.3). This finding however is based on experiments on reservoirs consisting of 150 neurons instead of 50.

To further pursue the matter, the random reservoirs experiment is repeated for reservoirs consisting of 50 neurons with the scope lying directly on the fitness function used in the neuroevolution experiment.

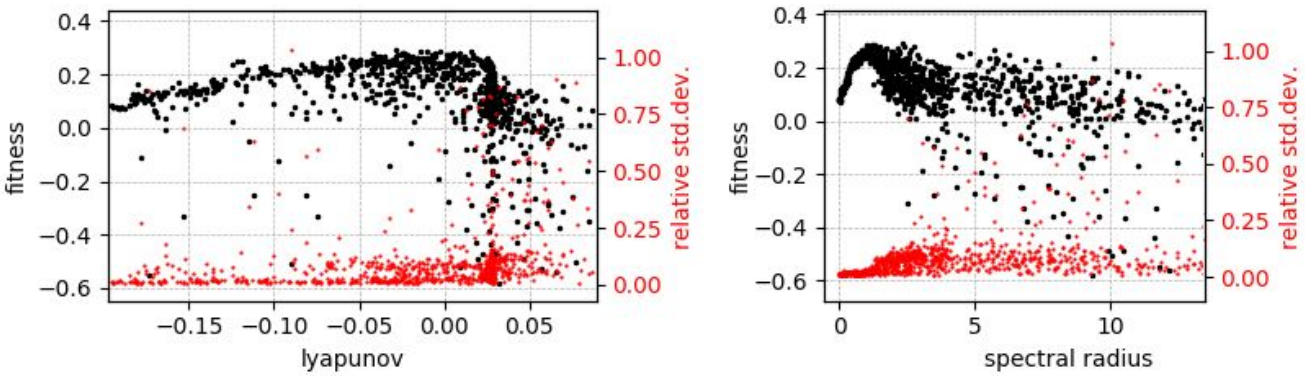

Fig. VI.2. Fitness (2-MMSE-NARMA) scores (black) with their relative standard deviations computed for random reservoirs consisting of 50 neurons.

Figure VI.2 shows the results of the random reservoirs experiment with 50 reservoir neurons. The fitness peak (left diagram) is quite broad, presumably due to the MMSE component and the small reservoir size. Additionally, the standard deviation of the fitness assumedly plays a part in making the fitness peak less distinct. All together can explain the fluctuation of the best reservoirs' lyapunov exponents in the neuroevolution experiment.

Another observation in figure VI.1 requires reflection: the fluctuation of the fitness value and the occurring drops that go along with it. NEAT preserves the best individuals - reservoirs in this

case. The reason for the fitness losses lies in the variation of the fitness values for one and the same reservoir, represented by the fitness' standard deviation. Figure VI.3 includes that measure in the fitness' development during neuroevolution.

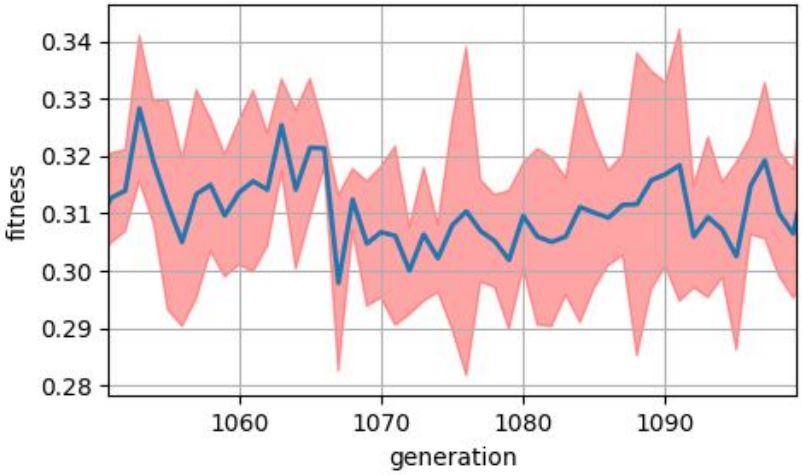

Fig. VI.3. Fitness (blue) and its standard deviation (light red area); values for the best reservoir of each generation. Only an extract of the whole neuroevolution is shown.

Initial high weights' standard deviation

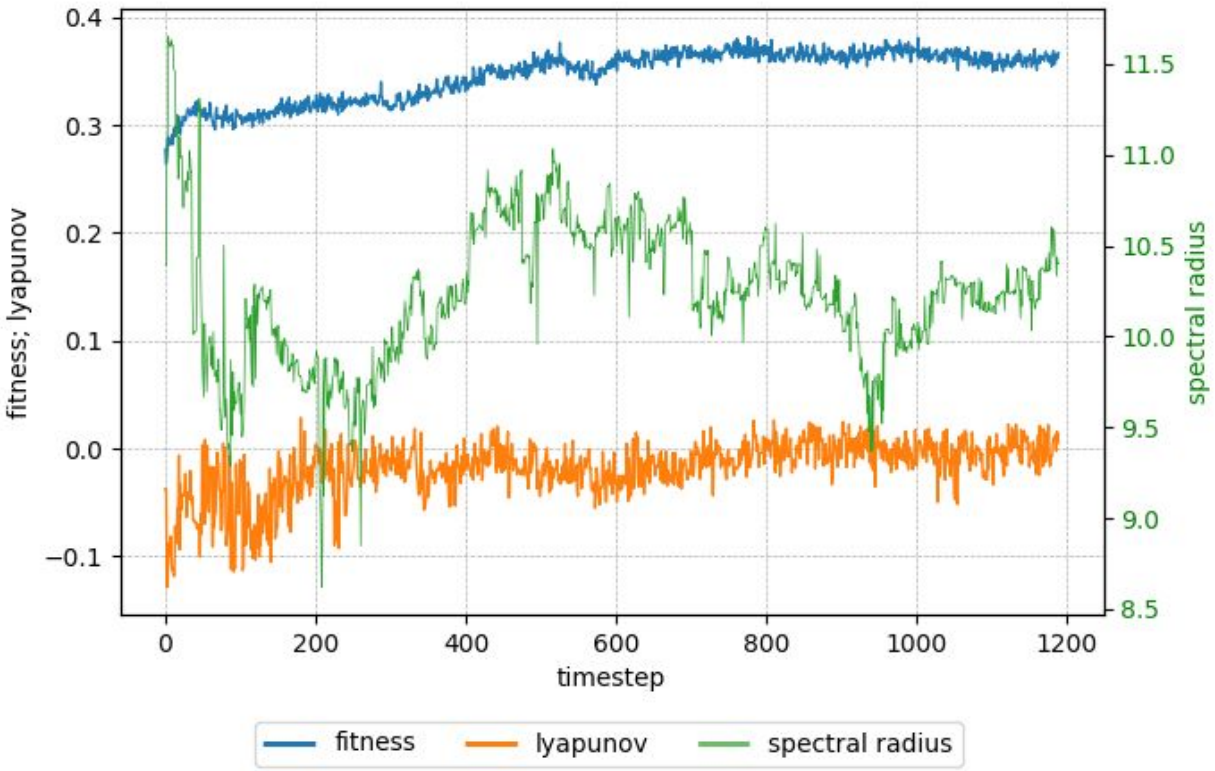

Fig. VI.4. Fitness (blue), lyapunov exponent (orange) and spectral radius (green, note the different scale) of the best reservoir of each generation while neuroevolution runs. Initial weight's standard deviation of 1.

Figure VI.4 shows the neuroevolution experiment's course with the initial weights' standard deviation of 1. The maximal fitness value reached is 0.382, which is higher than the one reached in the previous experiment run. The values for the single benchmarks (MC, MMSE, NARMA (not shown)) are also better than the ones reached during the random reservoirs experiment in spite of the lower number of reservoir neurons.

Further notable are the lyapunov exponents in correspondence with the spectral radii. The lyapunov exponents shown in the diagram belong to each generation's best reservoir and do not reach far into the regime of chaotic dynamics, especially at the very beginning of the experiment. This finding seems to not match the high spectral radii and the high initial standard deviation of

the reservoirs' weights. As shown in figure II.2, a reservoirs' dynamics are likely to be but not necessarily chaotic when encountering a spectral radius > 1. The figure also shows a much less distinct relationship between reservoirs' spectral radii and their lyapunov exponents in the area of chaotic dynamics. Considering moreover figure II.3, both observations can be assumed to also hold for the correlation of reservoir weights' standard deviations and their lyapunov exponents.

Our assumption is that for the high spectral radii in the experiment's results positive lyapunov exponents and with that chaotic dynamics are expected, but ordered dynamics are also in some cases encountered in spite of high spectral radii. The neuroevolution algorithm rejects the reservoirs of stronger chaotic dynamics, as they produce lower fitness values, presumably because of their undesirable position in relation to the edge of chaos.

Our assumption of reservoirs with high spectral radii and ordered dynamics gets additional support from a tiny feature of figure VI.2, namely that in the left part of the diagram some fitness values of reservoirs with lyapunov exponents smaller than zero are noticeably lower than most others and the course of a possible regression curve. A similar feature is not present in the right part of the figure for spectral radii smaller than 1. Thus, we suppose that those reservoirs have high spectral radii but ordered dynamics.

VII. Discussion & Outlook

In this study, we examined the hypothesis of Computation at the Edge of Chaos from the perspective of reservoir computing. We had set out to study two broad questions – first, the (extent of the) validity of the hypothesis with regards to Echo State Network (ESN); and second, if the "natural" process of evolution exhibits any preference towards selecting computing dynamical systems that operate near the edge of chaos (criticality). We have observed strong evidence for a positive answer to both of these questions.

Our experiments with randomly generated ESNs consistently gave evidence for a maximized computational capacity—measured in terms of information transfer (transfer entropy), information storage (memory capacity and memory mean squared error) and information processing (nonlinear autoregressive moving average task)—when the network's dynamics are close to criticality. This is a confirmation of the idea of computation at the edge of chaos. Similarly, our experiments with neuroevolution show the evolutionary development of ESNs towards the edge of chaos. The defined fitness function as a combination of the MMSE and NARMA benchmarks is maximized by the evolved networks on the ordered as well as the chaotic side of the edge of chaos.

An important conclusion to draw from the last chapter of the neuroevolution experiment is a considerable unreliability of using spectral radius to measure chaotic dynamics of an ESN, as also pointed out in Yildiz et. al. [14]. We have found (as in fig VI.4) that a spectral radius of more than unity does not necessarily imply chaos and destroy the echo state property in an ESN. The echo state property remains rather more strongly dependent on the actual *variance* in the strength of the connections among reservoir neurons than the eigenvalues of the weight's matrix. However, our results do not give evidence against the criticised requirement (in [14]) of a spectral radius smaller than unity to always satisfy the echo state property.

Neuroevolution offers many more opportunities to evolve reservoirs of echo state networks and get further insights into the idea of computation at the edge of chaos. One approach that comes to mind is to evolve a reservoir's topology, either by starting with a minimal set of connections and

then progressively growing the reservoir connectivity or by letting neuroevolution modify/simplify an existing reservoir structure.

References

[1] Langton, C. G. (1990). Computation at the edge of chaos. *Physica* D, 42.

[2] N. Bertschinger and T. Natschläger (2004). Real-time computation at the edge of chaos in recurrent neural networks. *Neural Computation*, 16(7):1413–1436.

[3] Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560.

[4] Jaeger, H. (2002). Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the ”echo state network” approach. Technical report, German National Research Center for Information Technology. GMD Report 159.

[5] Lukoševičius M. (2012) A Practical Guide to Applying Echo State Networks. In: Montavon G., Orr G.B., Müller KR. (eds) *Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science, vol 7700.* Springer, Berlin, Heidelberg

[6] Stanley, K. O., Miikkulainen, R. (2002). Evolving Neural Networks through Augmenting Topologies. In *Evolutionary Computation,* 10(2): 100.

[7] Barančok P., Farkaš. I (2014). Memory Capacity of Input-Driven Echo State Networks at the Edge of Chaos. In *International Conference on Artificial Neural Networks.* Springer International Publishing, 41–48.

[8] Boedecker J, Obst O, Lizier JT, Mayer NM, Asada M. (2012). Information Processing in Echo State Networks at the Edge of Chaos. *Theory in Biosciences* 131, 205–213.

[9] Matzner, F. (2017). Neuroevolution on the Edge of Chaos. In *Proceedings of the Genetic and Evolutionary Computation Conference 2017*

[10] Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks. In *Artificial Life*, 15(2): 185–212.

[11] Stanley, K. O. (2017). Neuroevolution: A different kind of deep learning. *O'Reilly Media*.

[12] Sprott, J. C. (2003). Chaos and Time-Series Analysis. *Oxford University Press.*

[13] Sprott, J. C. (2015). Numerical Calculation of Largest Lyapunov Exponent. http://sprott.physics.wisc.edu/chaos/lyapexp.htm Accessed online: 2016-05-25.

[14] Yildiz, I. B., Jaeger, H., & Kiebel, S. J. (2012). Re-visiting the echo state property. *Neural Networks*.

[15] Jaeger, H. (2001). Short term memory in echo state networks. *GMD report 152*. German National Research Center for Information Technology.

[16] Manjunath G., Jaeger H. (2013). Echo state property linked to an input: Exploring a fundamental characteristic of recurrent neural networks, *Neural computation* 25 (3) 671–696.

[17] Lizier J. T., Prokopenko M., Zomaya A. Y.. (2014). A Framework for the Local Information Dynamics of Distributed Computation in Complex Systems. In *Guided Self-Organization: Inception. Springer Berlin Heidelberg*, 115–158.

[18] Meiss J. (2007). Dynamical systems. *Scholarpedia*, 2(2):1629.

[19] Bishop R. Chaos. *The Stanford Encyclopedia of Philosophy* (Spring 2017 Edition)

[20] Gallicchio C., Micheli A., Silvestri L. (2018). Local Lyapunov exponents of deep echo state networks, Neurocomputing, https://doi.org/10.1016/j.neucom.2017.11.073

[21] Bailey B. (1996). Local lyapunov exponents: predictability depends on where you are. In *Nonlinear Dynamics and Economics, Cambridge University Press*, 345–359.

[22] Abarbanel H. D. I., Brown R., Kennel M. (1992). Local lyapunov exponents computed from observed data, *Journal of Nonlinear Science* 2 (3), 343–365. doi:10.1007/BF01208929.

[23] Wolfram S. (1983). Statistical Mechanics of Cellular Automata, *Reviews of Modern Physics*, 55(3): 601–644. doi:10.1103/RevModPhys.55.601

[24] Mitchell M., Hraber P., Crutchfield J. P. (1994). Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations, *Complex Systems*, 7(2): 89–130.

[25] Squire L., Berg D., Bloom F., Du Lac S., Ghosh A., Spitzer N. C. (2013). Fundamental Neuroscience (4th ed.). *Academic Press.*

[26] Miłkowski M. (2014). Computational mechanisms and models of computation. *Philosophia Scientiæ*, *18*(3), 215–228.

[27] Legenstein R., Maass W. (2007). Edge of chaos and prediction of computational performance for neural circuit models. *Neural Networks* 20(3):323–334.

[28] Schreiber T. (2000). Measuring information transfer. *Physical Review Letters* 85(2):461–464.

[29] Wollstadt P., Lizier J. T., Vicente R., Finn C., Martinez-Zarzuela M., Mediano P., Novelli L., Wibral M. (2018). IDTxl: The Information Dynamics Toolkit xl: a Python package for the efficient analysis of multivariate information dynamics in networks. *ArXiv preprint*: https://arxiv.org/abs/1807.10459.

[30] Lizier J. T., Prokopenko M., Zomaya A. Y. (2008). Local information transfer as a spatiotemporal filter for complex systems. *Physical Review E* 77(2):026,110.

[31] Maat, J. R., Gianniotis, N., Protopapas, P. (2018). Efficient Optimization of Echo State Networks for Time Series Datasets. *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018: 1-7.

[32] M. Wibral et al. (eds.) (2014). Directed Information Measures in Neuroscience, Understanding Complex Systems, *Springer-Verlag Berlin Heidelberg 2014*.

[33] Lizier J. T. (2014). JIDT: An Information-Theoretic Toolkit for Studying the Dynamics of Complex Systems. *Frontiers in Robotics and AI 1*, 11.

[34] Kraskov A., Stoegbauer H., Grassberger P. (2004). Estimating mutual information. *Phys. Rev. E Stat. Nonlin. Soft Matter Phys. 69(6 Pt. 2), 066138*.

[35] Wibral M., Pampu N., Priesemann V., Siebenhühner F., Seiwert H., Lindner M., Lizier J.T. & Vicente R. (2013). Measuring information-transfer delays. *PloS one, 8(2)*, *e55809.*