COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

COMPUTATIONAL ANALYSIS OF MEMORY CAPACITY OF AN ECHO-STATE NETWORK

MASTER THESIS

2015 Radomír BOSÁK

COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

COMPUTATIONAL ANALYSIS OF MEMORY CAPACITY OF AN ECHO-STATE NETWORK

MASTER THESIS

Study programme:	Mathematics
Study field:	Mathematics 1113
Supervisor:	prof. Ing. Igor Farkaš, Dr.
	Department of Applied Informatics

Bratislava 2015 Radomír BOSÁK

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZYKY A INFORMATIKY

COMPUTATIONAL ANALYSIS OF MEMORY CAPACITY OF AN ECHO-STATE NETWORK

DIPLOMOVÁ PRÁCA

Študijný program:	Matematika
Študijný odbor:	Matematika 1113
Vedúci práce:	prof. Ing. Igor Farkaš, Dr.
	Katedra aplikovanej informatiky

Bratislava 2015 **Radomír BOSÁK**

24342441

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Radomír Bosák
Study programme:	Mathematics (Single degree study, master II. deg., full time form)
Field of Study:	9.1.1. Mathematics
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Computational analysis of memory capacity of an echo-state network

Aim: 1. Provide an overview of echo-state neural networks (ESN) and their computational properties related to the regime at the edge of chaos.
2. Implement an ESN, driven by a stochastic scalar input and using computational simulations, systematically analyze its memory capacity, as a function of various parameters.
3. Design and test an iterative method for reservoir orthogonalization, aimed at maximizing the memory capacity.

Annotation: ESNs have become an efficient approach to training recurrent neural networks. However, their computational properties are not yet well understood. Various ways of improving ESN performance based on reservoir adaptation or parameter optimization have been proposed, with focus on different performance criteria, such as the memory capacity.

Keywords: recurrent neural network, reservoir computing, memory capacity

Supervisor:	prof. Ing. Igor Farkaš, PhD.
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	prof. Ing. Igor Farkaš, PhD.
Assigned:	30.11.2013
Approved:	02.12.2013

prof. RNDr. Ján Filo, CSc.
Guarantor of Study Programme

....

Student Supervisor

24342441

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Radomír Bosák
Študijný program:	matematika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	9.1.1. matematika
Typ záverečnej práce:	diplomová
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Computational analysis of memory capacity of an echo-state network
Výpočtová analýza pamäťovej kapacity siete s echo stavmi

Cieľ: 1. Urobte prehľad o sieťach s echo stavmi (ESN) a ich výpočtových vlastností týkajúcich sa režimu na hranici chaosu.
2. Implementujte ESN so skalárnym stochastickým vstupom a systematicky pomocou výpočtových simulácií analyzujte jej pamäťovú kapacitu, v závislosti of rôznych parametrov modelu.
3. Navrhnite a otestujte iteratívnu metódu na ortogonalizáciu rezervoára, s cieľom maximalizovať pamäťovú kapacitu.

Anotácia: Siete s echo stavmi (ESN) sa stali efektívnym prístupom k trénovaniu rekurentných neurónových sietí. Avšak výpočtové vlastností ESN neboli doteraz dostatočne preskúmané. Boli navrhnuté rôzne spôsoby na zlepšenie fungovania ESN, založené na adaptácii rezervoára alebo optimalizácii parametrov, s dôrazom na sledovanie rôznych kritérií na fungovanie modelu, ako napríklad pamäťovej kapacity.

Kľúčové slová: rekurentná neurónová sieť, rezervoárové počítanie, pamäťová kapacita

Vedúci:	prof. Ing. Igor Farkaš, PhD.
Katedra:	FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry:	prof. Ing. Igor Farkaš, PhD.
Dátum zadania:	30.11.2013

Dátum schválenia: 02.12.2013

prof. RNDr. Ján Filo, CSc.
garant študijného programu

....

študent vedúci práce

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím citovaných zdrojov.

....

Poďakovanie Touto cestou sa chcem poďakovat svojmu vedúcemu diplomovej práce prof. Ing. Igorovi Farkašovi, Dr. za ochotu, pomoc, odborné rady a podnetné pripomienky, ktoré mi pomohli pri písaní tejto práce.

Abstract

Echo state networks (ESN) present a novel approach for using recurrent neural networks on prediction tasks. One of the important measures of performance of echo state networks is the short term memory capacity, introduced in Jaeger (2001b). ESNs, as an efficient approach avoids time consuming training of all weight matrices and, instead, requires only tuning of output weights. However, the input and recurrent weight matrices should be suitably initialized. In this thesis we consider a parametrized initialization of these matrices. The tested parameters were the input and hidden weight matrix scaling, reservoir size and matrix sparsity. Through systematic computational experiments, we search the parameter space for values which yield maximal memory capacity. We locate the optima for recurrent matrix scaling parameter and describe which parameters increase (reservoir size) and which decrease the memory capacity (input weight matrix scaling and matrix sparsity). Furthermore, an orthogonalization process for recurrent weights is presented, which considerably increases the memory capacity.

Keywords: Neural networks, Echo state networks, Short term memory capacity

Abstrakt

Siete s echo stavmi predstavujú nový prístup vo využívaní rekurentných neurónových sietí na predikčné úlohy. Jedna z dôležitých mier výkonnosti siete s echo stavmi je jej krátkodobá pamäťová kapacita, zavedená v Jaeger (2001b). Siete s echo stavmi sú efektívnym prístupom, ktorý sa vyhýba časovo náročnému trénovaniu všetkých váhových matíc a namiesto toho vyžaduje iba nastavenie výstupných váh. Avšak vstupné a rekurentné matice musia byť vhodne inicializované. V tejto práci uvažujeme parametrizovanú inicializáciu týchto matíc. Testované parametre boli: škálovanie vstupnej a vnútornej váhovej matice, veľkosť rezervoáru a riedkosť vnútornej matice. Cez experimenty prehľadávame parametrický priestor a hľadáme hodnoty parametrov, ktoré dávajú maximálnu pamäťovú kapacitu. Lokalizujeme optimálne hodnoty parametra pre škálovanie vnútornej matice a popíšeme, ktoré parametre zvyšujú (veľkosť rezervoáru) a ktoré znižujú (škálovanie vstupnej matice a riedkosť vnútornej matice). Navyše prezentujeme ortogonalizačnú procedúru, ktorý značne zvyšuje pamäťovú kapacitu neurónovej siete.

Kľúčové slová: Neurónové siete, Siete s echo stavmi, Krátkodobá pamäťová kapacita

Contents

List of Figures

List of Tables

Introduction

Throughout the history, mankind has always been inspired by nature, be it art or technological inventions and discoveries which fuelled the technological progress. Millions of years of evolution on Earth had enough patience to find useful solutions on various mechanical and other problems, which one encounters on daily basis. The hook-and-loop fastener, for example, is attributed to George de Mestral, a Swiss engineer, who supposedly invented it on a hunting trip in Alps, when examining the burrs of burdock sticking to his clothes.

With the advent of computers, and their increasingly larger use on automation, we need algorithms, which can make complex decisions and consider many factors, often in conditions and situations that human cannot anticipate. Then these algorithms need to be able to extrapolate and generalize. These are the kind of problems which fall to the field of Machine Learning (ML), or more broadly, Artificial Intelligence (AI). The precision of such algorithms can be of great economic importance, if we consider automated quality control of massively produced integrated circuits. The importance may lie in security if talking about monitoring subsystems in nuclear power plants, or autonomous cars, which are already being tested in real traffic.

While in past complex decisions were left entirely to a human, later the so called expert systems were invented. These, however, had to be configured/programmed by a human with expertise in the problem.

The **Artificial neural networks** (ANNs) were invented as an inspiration from human brains. The cooperation of mathematics and biology allowed the creation of a model of a neuron – a fundamental cell the brain is composed of. Later, models of whole neural networks were presented – neurons, together with synapses, which are neural interconnections capable of transmitting information.

When searching for suitable neuron and neural network models, many things had to be taken to account. The given model had to have certain features which a human brain has, such as the ability to compute – to process the input signal and perform (or at least approximate) some function. Or the ability to remember and recall previous input signals or knowledge derived from them and to use them in decision-making.

Then there is the ability to learn - to alter its inner structure (synapses) based on experience and through it - to react better in the future. A well known phenomenon is the brain plasticity: it is capable of rewiring itself and replace the functionality of damaged brain-parts.

One of the of the brain qualities we demand is the ability to perceive the environment and its stimuli in temporal context; to process the input signals as a temporal series, so that the past inputs can be used in computation. One of the attempts to achieve this was the utilization of existing models and projecting the temporal axis on the spatial axis: this means that we add new neurons which repeat the past inputs and provide it to the current computation.

Another possibility (and one more biologically plausible) how to include the temporal aspect is to use **recurrent neural networks** (RNNs). In RNNs the current state of the network depends not only on the current inputs, but also on the previous network state. This way, the neural network gains 'memory'. Many types of RNNs (such as Elman's RNN) were developed over the years. However, a lot of them suffered from computationally expensive learning algorithm and slow convergence. During gradual change of network parameters, bifurcations can occur (Doya, 1992) in network dynamics, which destroy the convergence of commonly used gradient descent methods for synaptic weight adaptation.

In reaction to this a new paradigm called Reservoir Computing (RC) came into light, which used these principles:

1. Instead of training each network synapse, the **synaptic weights are generated randomly** during network initialization. This ensures that the network state will represent enough complex non-linear transformation of the input. The output is then taken as a suitable linear combination of the network state. Only output weights are altered during training.

2. The inner (hidden) layer, also called the **reservoir**, has a forgetting property (later referred as the echo state property), which ensures the stability of the network.

These principles were applied and independently developed under the names Echo state networks (ESNs) (Jaeger, 2001a) and Liquid state machines (LSM) (Maass et al., 2002). In this work we will be working exclusively with echo state networks. From the time of its appearance the ESNs gained much popularity due to its implementation simplicity and non-expensive learning algorithm.

This, however, does not mean there is nothing to improve. How 'randomly' should the network weights be initialized? The reservoir should be able to make complex enough transformations of the input, but at the same time, it should have the forgetting property. How to achieve that the information is not forgotten too soon? How to normalize the input signal so that the unimportant information does not clutter the network and the important information is not lost in numerical errors?

This thesis is focused on the problem of initialization of reservoir in an Echo state network and tries to answer questions such as those just mentioned. From a purely mathematical point of view, it is a study of properties of highly non-linear dynamical systems, particularly recurrent neural networks. Since a strictly mathematical analysis of such a system would be very difficult (there are some that follow that approach) we chose a computational approach. That means that we implement a model on a computer, set up an experiment, and then through repeated simulations we measure the observed model property or feature.

We will end this introduction with an overview of the thesis structure.

- Section 1 will define and describe echo state networks and all notions necessary for understanding. The short-term memory capacity will be defined as well.
- Section 2 provides an overview of results, concerning the memory capacity, already covered by literature.
- Section 3 covers the contribution of this thesis. These are mainly in the form of experiments whose output is depicted as function graphs. Results and observations are commented.
- Section 4 provides a summary of presented results and discusses possible future work.

1 Theoretical introduction

1.1 Artificial Neural networks

Artificial neural networks (ANNs) are a family of algorithms inspired by biological neural networks, which are used in machine learning. By biological neural networks we can imagine an animal's central nervous system, or **brain** in particular. As problem-solving tools, ANNs excel in many areas, where traditionally used tools have been slow and inefficient.

From computation centralization perspective, the ANNs use decentralized approach: Instead of using one central processor / problem solver, performing a specific human-prescribed algorithm, artificial neural networks consist of large number of (often identical) units called (artificial) neurons, each of them performing a simple operation. Though simple individually, in larger numbers these neural networks are capable of producing very complex behaviour.

Another paradigm used by neural networks is massive parallelism. Neurons are performing all their computations in parallel. Technological development in the last decades (multi-core processors, multiple CPUs on one computer, high CPU speed) have made neural computation a viable and affordable option.

1.1.1 Practical ANN representation

In practice, an artificial neural network can be represented as an **oriented graph**. Vertices are the neurons. Edges represent the connections between neurons - the synapses. At each time $t \in T$ (with time being discrete or continuous), each neuron i has its **activation value** $x_i(t) \in \mathbb{R}$. Biologically, the activation value corresponds to membrane potential resident in neuron, or its excitation level, at time t.

The strength of neural synapse from neuron i to neuron j is represented by a real number w_{ji}. A positive number w_{ji} can be understood as an excitatory connection, while $w_{ji} < 0$ acts as an inhibitory connection. In many neural network models, we allow self-connections, that is, neuron is connected to itself, forming a loop in the graph.

1.1.2 Neural network dynamics

When considering a neural network dynamics, a neuron j changes its activation value according to activation values of all neurons connecting to it and their respective synaptic weights. A discrete-time formula may look something like this:

$$x_j(t+1) = \sum_{i \in I} w_{ji} \cdot x_i(t) \tag{1}$$

where I is the index set of all neurons which are connected to neuron j. In addition, the sum is typically passed through **activation function** f, forming:

$$x_j(t+1) = f\left(\sum_{i \in I} w_{ji} \cdot x_i(t)\right) \tag{2}$$

A well chosen activation function f can make all the difference in pursuit of constructing a neural network capable of solving a given task. Activation functions can be used to introduce non-linearity to the model, to bound the neural activations (an unbounded membrane potential is not realistic in biological systems) or the introduce non-symmetry (bias input).

Examples of commonly used activation functions is shown in Table 1.

1.1.3 Types of neural networks

Neural networks can be categorized in many ways; by their size, topology, dynamics (discrete/continuous time), activation function, neuron types. A common distinction is made between **feedforward** and **recurrent** neural networks.

In **feedforward** neural network (FFNN) the synapse graph does not contain cycles, or loops. The network is organized into neuron layers with first layer being the input layer, containing input neurons. The last layer is the output layer and all other are called hidden layers. The only connections which exist are connections from neurons in i-th layer to neurons in $(i+1)$-th layer. No interconnections between neurons in the same layer exist, and no neuron connects to neurons from previous layers.

On the other hand, a **recurrent** neural network (RNN) allows graph cycles and loops. Neurons can be organized into layers, but no strict interconnection conditions

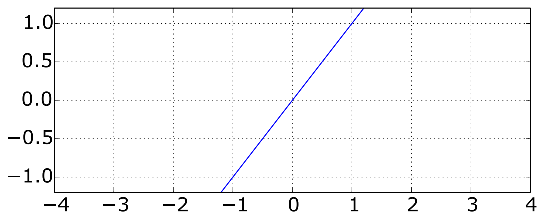
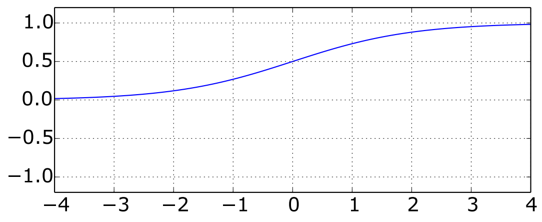
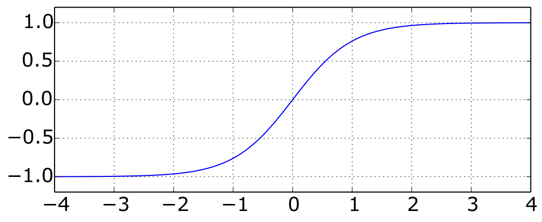
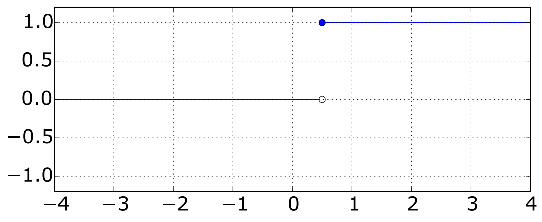
identity function	$f(x) = x$	
sigmoid function	$f(x) = \dfrac{1}{1+e^{-x}}$	
hyperbolic tangent	$f(x) = \tanh(x)$	
threshold function	$f_a(x) = \begin{cases} 0 & x < a \\ 1 & x \geq a \end{cases}$	

Table 1: List of commonly used activation functions.

are enforced. An example of a recurrent network is the simple recurrent network (Elman, 1990). Another example of recurrent network - which will be in the centre of our interest - is the *echo state network*.

1.2 Echo state networks

Echo state network (ESN) is a specific type of recurrent neural network introduced by Jaeger (2001a). It consists of three layers of neurons: input, hidden (recurrent layer, a.k.a reservoir), and output layer.

Input neurons connect to reservoir neurons and they have connections to output neurons. What makes this architecture a recurrent network are interconnections between neurons in reservoir.

The networks we consider are discrete-time, so we have time step $t \in \mathbb{Z}$. Activations

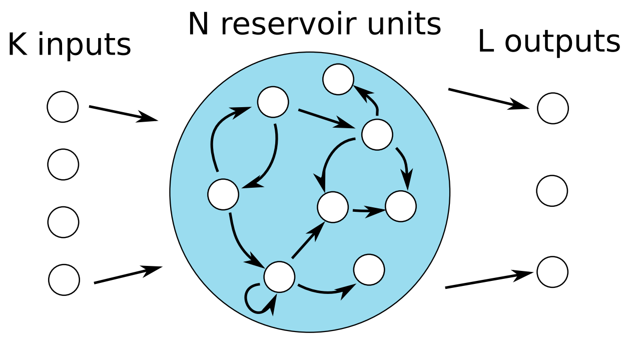


Figure 1: Illustration of ESN architecture

of K input neurons at time t are denoted $\mathbf{u}(t) = (u_1(t), \ldots, u_K(t))$. Similarly we denote activations of N reservoir neurons and L output neurons as $\mathbf{x}(t) = (x_1(t), \ldots, x_N(t))$ and $\mathbf{o}(t) = (o_1(t), \ldots, o_L(t))$, respectively.

Activation of reservoir and output neurons are updated according to formulas:

$$\mathbf{x}(t+1) = f(W^{\text{in}} \cdot \mathbf{u}(t+1) + W \cdot \mathbf{x}(t)) \tag{3}$$

$$\mathbf{o}(t+1) = f^{out}(W^{\text{out}} \cdot \mathbf{x}(t+1)) \tag{4}$$

where $f : \mathbb{R}^N \to \mathbb{R}^N$ and $f^{\text{out}} : \mathbb{R}^L \to \mathbb{R}^L$ are suitable activations functions, typically $f = \tanh$ and $f^{\text{out}} = id$ (tanh is taken element-wise). W^{in}, W and W^{out} are input, recurrent and output matrices, respectively, which represent strength of particular synapses between neurons.

Now we can get to defining property of an echo state network, borrowing the definition from (Jaeger, 2001a):

Definition 1. *Assume that all inputs come from a compact set U. A network is said to have* ***echo states*** *if the current state $\mathbf{x}(t)$ is uniquely determined by left-infinite input history $\mathbf{u}^{-\infty} = (\ldots, u(t-1), u(t))$.*

This can be reformulated as follows: there exists an echo function $E = (e_1, \ldots, e_N)$, $e_i : U^{\mathbb{N}} \to \mathbb{R}$ such that for all left-infinite input histories the following equation holds:

$$\mathbf{x}(t) = \mathbf{E}(\ldots, \mathbf{u}(t-1), \mathbf{u}(t))$$

Necessary and sufficient condition for network echo-stateness are analyzed in (Jaeger, 2001a). It turns out that the existence of echo states has relation to the largest singular

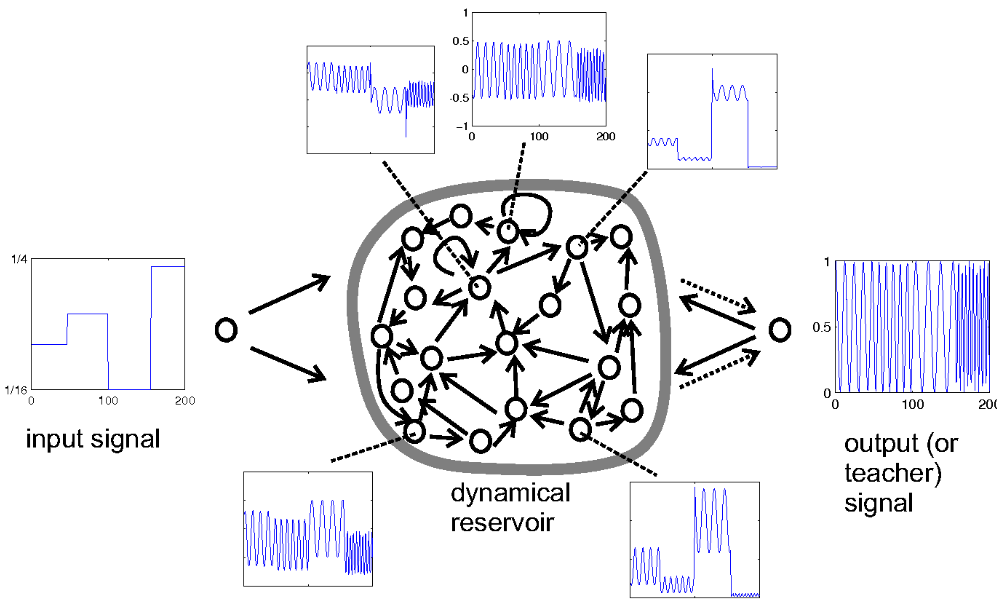


Figure 2: Illustration of ESN dynamics, taken from Jaeger (2007). Reservoir neurons create various echoes as a response to an input stimulus.

value $s_{\max}$ and the spectral radius ρ $(= |\lambda_{\max}|$, the largest eigenvalue in absolute value) of the reservoir matrix W. We will assume the tanh activation function and restate the propositions here:

Theorem 1 (sufficient condition)**.** *If $s_{max} < 1$, then the network has echo states.*

Theorem 2 (necessary condition)**.** *If $\rho > 1$, the input set contains the sequence* **0** *and the admissible state set is $[-1, 1]^N$, then the network has no echo states.*

Spectral radius ρ and the largest singular value $s_{\max}$ of an arbitrary matrix W automatically satisfies $0 \leq \rho \leq s_{\max}$, so regarding the existence of echo states we can differentiate between 3 cases, which are depicted in the following table:

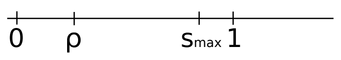
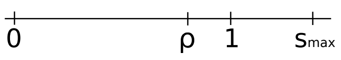
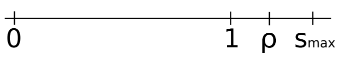
$0 < \rho < s_{\max} < 1$	Echo states exist	
$0 \leq \rho \leq 1 \leq s_{\max}$	Echo states may exist	
$0 < 1 < \rho < s_{\max}$	No echo states	

Table 2: How values of ρ and $s_{\max}$ determine the existence of echo states.

As noted in Jaeger (2001a, p. 8) a convenient strategy to obtain a reservoir matrix with echo states is to generate some matrix W and then use a global scaling using $\alpha \in \mathbb{R}^+$

$$\hat{W} := \alpha W$$

such that the spectral radius and the largest singular value lands into desired regions on real axis. Observe that ρ and $s_{\max}$ scale linearly with α,

$$\hat{\rho} = \alpha\rho \qquad \hat{s}_{\max} = \alpha s_{\max} \,.$$

This gives us an interesting scaling interval

$$(\alpha_{\min}, \alpha_{\max}) := \left(\frac{1}{s_{\max}}, \frac{1}{\rho}\right)$$

under which the echo states certainly exist, and above which there are no echo states (if the zero input sequence is an admissible input). Jaeger (2001a, p. 9) further notes that one obtains echo states in most cases even if $\alpha \in (\alpha_{\min}, \alpha_{\max})$.

1.3 Echo state network learning

Eventually, we would like the ESN to be able to solve problems, by which we mean the following: after receiving a given sequence of inputs, the ESN should produce certain desired outputs, given by some goal function $G(\ldots, \mathbf{u}(t-1), \mathbf{u}(t), \mathbf{x}(t))$.

This ability of ESN to model given goal function lies entirely in suitable choice of neuron connections' matrices W^{in}, W and W^{out} (and architectural choices as reservoir size and activation functions f, f^{out}). The approach, presented in (Jaeger, 2001a) is to **generate the input and the recurrent matrix** (more-or-less) **randomly**, and let the output matrix be determined by supervised learning on the training data (which can be thought of as a set of input–target pairs).

So, to find an optimal matrix W^{out} which approximates best the target function, we reduce our problem to:

$$f^{\text{out}}\left(W^{\text{out}} \cdot \mathbf{x}(n)\right) = \mathbf{d}(n) \qquad t \in \mathbb{Z} \tag{5}$$

which is equivalent to

$$W^{\text{out}} \cdot \mathbf{x}(t) = (f^{\text{out}})^{-1}(\mathbf{d}(t)) \qquad t \in \mathbb{Z} \tag{6}$$

where $\mathbf{d}(t)$ is the target output at time t.

Since there is (possibly) infinite number of linear equations with only finitely many variables (W is $N \times N$), we can solve this problem only approximately, reformulating the problem as a least squares problem and resorting to methods such as linear regression.

Method used by Boedecker et al. (2012) is to store the reservoir states into a long matrix as columns, to 'solve' an approximate equation:

$$W^{\text{out}} \cdot (\mathbf{x}(0), \mathbf{x}(1), \dots\ \mathbf{x}(1000)) = (\mathbf{d}(0), \mathbf{d}(1), \dots\ \mathbf{d}(1000))$$

and then taking the Moore-Penrose pseudoinverse to get the 'optimal' matrix W^{out}. This is also the approach taken during all experiments presented here which required ESN training.

1.4 Measures of ESN performance

The ESN initialization (choice of input and recurrent matrices) affects how well the ESN models the goal function. How to quantify this? We could give the ESN the training data input $\mathbf{u}(t)$, calculate the output $\mathbf{o}(t)$ and then compare it with desired output - by getting the mean square of their difference. So this error tells us how well this ESN instance performs in a specific task. But, are there any general measures of ESN performance?

There are quite a few. The measure we will be most concerned about is **short term memory capacity**, abbreviated MC. This notion introduced in Jaeger (2001b) measures the ability of the reservoir to store and recall previous inputs fed into the network. In this thesis, we will deviate a bit from Jeager's original definition, not allowing direct input-to-output neuron connections, and therefore we will restate the definition of memory capacity:

Definition 2. *Consider an ESN with $K = 1$ input nodes, N reservoir nodes and L output nodes.* ***Memory capacity*** *of this network is a real number defined as:*

$$MC := \sum_{k=1}^{\infty} MC_k := \sum_{k=1}^{\infty} \max_{W_k^{out}} \left[\rho^2 \left(u(t-k), o_k(t)\right)\right] \tag{7}$$

$\rho^2 \left(u(t-k), o_k(t)\right)$ is the squared correlation coefficient between activation of the input neuron and k-th ($k = 1, \ldots, N$) output neuron

$$o_k(t) = W_k^{out} \cdot \boldsymbol{x}(t) \tag{8}$$

Note: The underlying probability space used in ρ calculation is composed of all left-infinite sequences of inputs. Thanks to the echo state property, $\boldsymbol{x}(t)$ is uniquely determined by the left-infinite input sequence.

Another measure of reservoir performance is **Lyapunov exponent**. Lyapunov exponent is a term from dynamical systems theory and it can be used to describe criticality of an ESN, when looked upon as a dynamical system. For our purposes, Lyapunov exponent will be a real number λ assigned to an ESN. $\lambda > 0$ roughly means a chaotic system, while $\lambda < 0$ refers to a subcritical system. A point $\lambda \approx 0$ is called the critical point, or **the edge of chaos**.

As Boedecker et al. (2012) shows, the criticality of an ESN heavily correlates with memory capacity and overall network performance. The optimal performance is often achieved for system just below the edge of chaos.

The idea behind Lyapunov exponent is to measure sensitivity to perturbations in initial conditions (Zeng et al., 1991). The definition of Lyapunov exponent used in Boedecker et al. (2012) is:

Definition 3 (Lyapunov exponent)**.**

$$\lambda = \lim_{k\to\infty} \frac{1}{k} \ln\left(\frac{\gamma_k}{\gamma_0}\right)$$

where γ_0 is the initial distance between dynamical system state of perturbed and unperturbed instance. γ_k is the distance after k steps.

Since the Lyapunov exponent is defined as a limit, is can only be estimated in most cases. The measuring procedure is modified, to avoid numerical overflows.

1. Two identical neural network copies (with reservoir states $\mathbf{x}^1$ and $\mathbf{x}^2$) are taken.
2. A small perturbation to the neuron j activation value is introduced to the second network, such that $\|\mathbf{x}^1(t) - \mathbf{x}^2(t)\| = \gamma_0$, where $t = 0$.
3. The simulation is advanced one step further, calculating $\mathbf{x}^1(t+1)$ and $\mathbf{x}^2(t+1)$.
4. The new distance between states $\gamma_t := \|\mathbf{x}^1(t+1) - \mathbf{x}^2(t+1)\|$ is recorded and the second state is renormalized to the distance γ_0.

$$\mathbf{x}^2(t+1) \leftarrow \mathbf{x}^1(t+1) + (\gamma_0/\gamma_t)(\mathbf{x}^2(t+1) - \mathbf{x}^1(t+1))$$

5. Steps 3–4 are performed repeatedly.

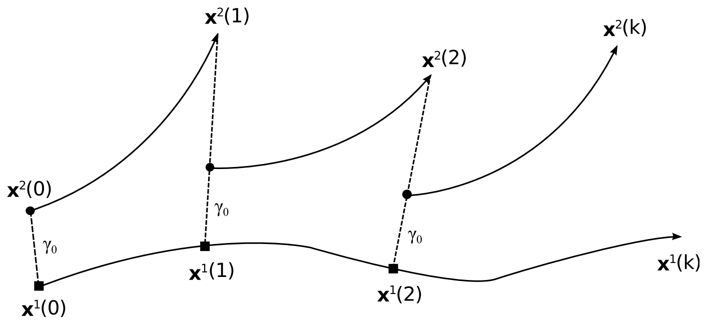


Figure 3: Estimation of the Lyapunov exponent, taken from Boedecker et al. (2012). After each network update, the distance between network state trajectories is reset to γ_0.

The Lyapunov exponent λ_j corresponding to the neuron j is then $\ln(\gamma_k/\gamma_0)$ averaged over all measured k. The resulting Lyapunov exponent λ is then taken as an average λ_j for echo neuron j in the reservoir.

In this work, we have not focused on measuring the Lyapunov exponent and analysing its relation to memory capacity because this was already done in Boedecker et al. (2012). The transition from ordered to chaotic dynamics (the edge of chaos) is often visible when tuning the reservoir parameters in the form of local maximum of memory capacity.

Another inspection of network parameters which affect the memory capacity, but with regard to Lyapunov exponent was done in Barančok and Farkaš (2014). Both structured and unstructured input data were tested. The investigated network parameters were the reservoir sparsity and input data shift (in case of unstructured, random input).

Boedecker et al. (2012, chapter 5), defines more advanced measures of reservoir performance such as *information storage* and *information transfer*, but these are beyond the focus of this thesis.

1.5 Motivation for this work

The effects of various ESN parameters on memory capacity have not yet been systematically investigated. In this thesis, we search the parameter space for values which achieve the greatest memory capacity. Furthermore, we suggest procedures, such as matrix orthogonalization that contribute to an increase of memory capacity. Various alterations of original reservoir initialization procedure are investigated as well.

2 Previous results

This chapter will cover some theoretical facts about echo state networks and memory capacity. The results concerning memory capacity were taken mostly from section 3 of Jaeger (2001b).

2.1 Echo states equivalent conditions

It turns out that the property of having echo states can be equivalently defined using the notions *uniformly state contracting*, *state forgetting* and *input forgetting network.* These provide a better understanding of the echo state property. Their definitions can be found in Jaeger (2001a).

Theorem 3. *Assume that the input comes from a compact set U and that the network update function $T : (u(t), x(t)) \mapsto x(t+1)$ is continuous. Then the following conditions are equivalent.*

1. *The network has echo states.*
2. *The network is uniformly state contracting.*
3. *The network is state forgetting.*
4. *The network is input forgetting.*

2.2 Memory capacity bound

A note about MC definition: the propositions **of this section** were proven, assuming a network architecture allowing neural connections directly between input- and output-layer neurons. This means that the output matrix W^{out} is $(K + N) \times L$ and that the output layer is updated according to:

$$\mathbf{o}(t) = W^{\text{out}} \cdot \begin{pmatrix} \mathbf{u}(t) \\ \mathbf{x}(t) \end{pmatrix} \tag{9}$$

In section 3 of this thesis, that contains experiments which measure memory capacity, we use an MC definition according to Boedecker et al. (2012). This definition does not allow input–output neural connections and was chosen such that we can compare results.

The difference in definition, does not render the following proposition completely invalid, since one definition can be seen as a special case of the other. A $(N + 1)$ unit reservoir with the last neuron serving as a mere delay of the input can mimic the behaviour of a reservoir which allows direct input–output connections. This means that if using Boedecker's definition, we expect to achieve maximal MC only $N - 1$ (or $N - MC_{k=0} \approx N - 1$ to be precise).

The memory capacity definition (7) involves an infinite sum. At first glance, it is not clear whether the memory capacity is a finite number. It turns out, that under reasonable circumstances, the memory capacity is not only finite, but also bounded by N, where N is the reservoir size. We will restate the proposition here:

Theorem 4. *The memory capacity for recalling an i.i.d. (independent, identically distributed) input by an N-unit ESN with identity activation function is bounded by N.*

The condition $f(x) = x$ allowed the theorem to be proven by means of linear algebra.

Another useful proposition is one that states the conditions under which is the full memory capacity achieved. Note that the sizes of matrices W and W^{in} are $N \times N$ and $1 \times N$ respectively.

Theorem 5. *The memory capacity of an ESN with identity activations functions is exactly N, iff the matrix $M_N = (W^1 W^{\text{in}} \dots W^N W^{\text{in}})$ has full rank.*

Two notes to this theorem:

1. In generic case, the full-rank condition is satisfied. By generic case, we mean that the space of matrices which do not satisfy the condition has measure 0.
2. In computer-simulated neural networks, the full memory capacity is rarely achieved, especially for large reservoirs. This is due to numerical errors.

2.3 Reservoir initialization guidelines

The papers Lukoševičius and Jaeger (2009) and Lukoševičius (2012) provide a nice and clear overview of practical tips on reservoir initialization. In the following paragraphs we will emphasize their most important suggestions.

In Lukoševičius and Jaeger (2009), the authors have suggested to generate **big**, **sparsely** and **randomly connected** reservoir. Many reservoir neurons ensure that there are many input signal transformations, which can be linearly combined into a larger family of functions. The reservoir sparsity makes the activation signals only loosely coupled, and random connections ensure that the neural activations are different.

As for the input matrix, it is advised to be dense and scaling should be adjusted according to amount of non-linearity we expect from the function, which the neural network is trying to perform. Larger input weights cause the tanh activation function to operate in its non-linear range. A bias input added to neurons has a similar effect.

The scaling of the reservoir (recurrent) matrix can be done, by setting its spectral radius $\rho = \rho(W)$. It should be less than 1, so that the reservoir has the echo state property. The closeness to 1 can affect the memory the network has and the non-linearity it can perform. Values of ρ close to 1 have longer memory and drive the activations into non-linear regions of tanh activation function.

Further, different reservoir topologies were suggested, as well as reservoir decomposed into modules (Lukoševičius and Jaeger, 2009). A modification of the neuron activation update formula can be made introducing a sort of a momentum term – these are called *leaky integrator neurons.*

A disadvantage of the standard reservoir architecture is that it has only one hidden layer. So in the tasks where little memory is needed, but the target output $y_d(t)$ is a complex transform of the input $u(t)$, the ESN can fail, where a feed-forward multilayer NN would succeed. To fix this, one has to delay the target output $y_d(t)$ by k steps, so that the signal passes multiple iterations before reaching the output. Another possibility is to update the reservoir (or its part) k times each iteration.

Reservoir pre-training tips which are taking the concrete task data into account are presented in Lukoševičius and Jaeger (2009).

3 Experiments

As indicated earlier, selecting the concrete neural network model requires both initialization of input/reservoir matrix and adaptation of output weights. We mentioned that the input and reservoir matrix are chosen 'more-or-less random'. But how random? In the following experiments, we try to find such initialization procedure and such parameters that maximize the network performance in terms of memory capacity.

As can be seen from memory capacity definition, the memory capacity is uniquely determined by known input matrix, recurrent matrix (inherently containing reservoir size and network topology), input signal distribution and the choice of activation function f (for us it will be always $f = \tanh$). In our experiments we consider an unstructured input: a sequence of independent, uniformly distributed real numbers from the interval $[-1, 1]$.

That leaves us with problem of generating input and reservoir matrices. Following Boedecker et al. (2012), we choose elements of the input matrix from $U(-\tau, \tau)$ uniform distribution and elements of the recurrent matrix from $N(0, \sigma^2)$ normal distribution. Parameters τ and σ are real and positive. So one task is to find an optimal σ and τ. Apart from that, several other reservoir modifications are tested, such as: scaling the reservoir to certain spectral radius (ρ) / largest singular value ($s_{\max}$); making the matrix sparse/orthogonal; changing the number of neurons in reservoir (reservoir size). A diagram of the presented reservoir initialization possibilities being the result of the problem analysis is shown in Figure 4.

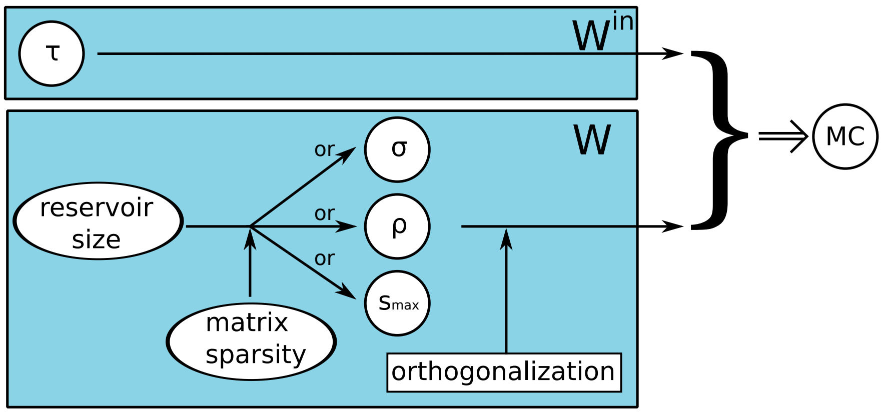


Figure 4: ESN initialization scheme.

3.1 Experimental setup

In a generic experiment we used an ESN composed of 1 input unit and 100 reservoir units. The number of output neurons was chosen 150, or 1.5 times the reservoir size, when the reservoir size was subject of testing and was different from 100. The parameter τ was chosen 0.01, apart from experiments where the effect of τ was investigated. The reservoir activation function used was $f = \tanh$ and output activation function f^{out} was an identity function: $f^{\text{out}}(x) \equiv x$.

In all figures, the y-axis represents the measured memory capacity. For each $\sigma/\rho/s_{\max}$ parameter value (on x-axis), multiple (≥ 1000) network instances were generated and their memory capacity was computed. Since a single $\sigma/\rho/s_{\max}$ parameter can yield different reservoir matrices, the memory capacity belonging to certain $\sigma/\rho/s_{\max}$ parameter value can be regarded as a random variable. The plot lines in the following graphs represent the average memory capacity for that particular parameter values; the vertical bars surrounding the plot lines represent the interval (memory capacity mean $\pm$ standard deviation).

3.2 Recurrent matrix scaling

In this experiment, we had fixed $\tau = 0.01$ and reservoir size and generated reservoir matrices for different values of σ to see which value of σ produced a reservoir with highest memory capacity:

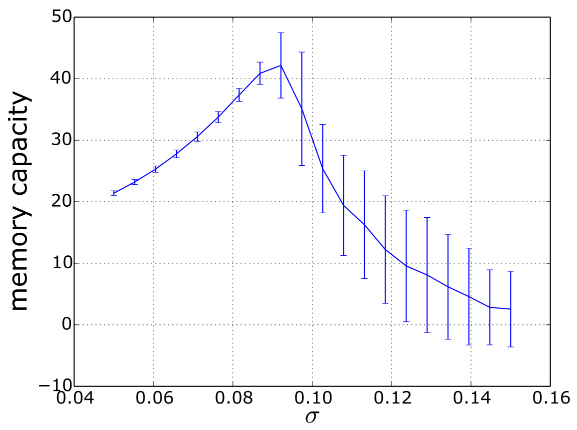


Figure 5: Memory capacity of the 100-neuron reservoir for different values of σ.

The memory capacity is maximized at the value $\sigma \approx 0.09$. As we will see later, this value has something to do with the *circular law* (which will be explained later, too).

Directly setting σ is not the only way of generating the reservoir. We can also generate the recurrent matrix elements from $N(0,1)$ distribution and then scale the matrix so that it has a specified spectral radius (or the largest singular value).

The generating algorithm would then be:

1. Generate a recurrent matrix W with elements selected from $N(0,1)$ distribution
2. Compute its spectral radius $\rho(W)$
3. Scale the matrix: $W \leftarrow W \cdot \dfrac{\rho_{\text{new}}}{\rho(W)}$
4. Now the matrix W has the desired spectral radius.

The same goes for the largest singular value $s_{\max}$ (which is equal to matrix operator norm).

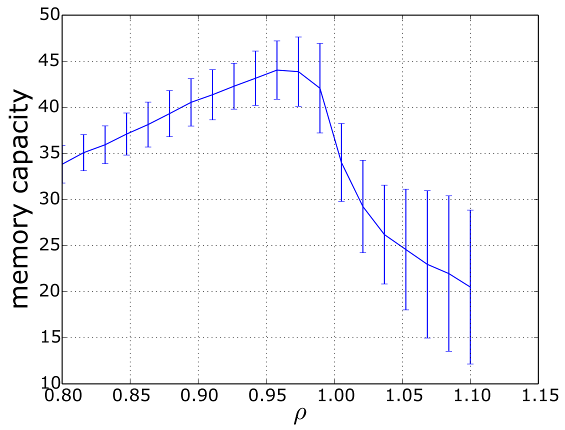


Figure 6: Memory capacity of the 100-neuron reservoir changing with spectral radius of the recurrent matrix.

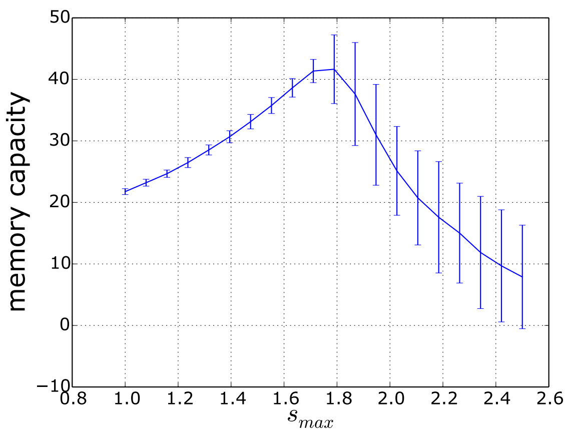


Figure 7: Memory capacity of the 100-neuron reservoir changing with the largest singular value of the recurrent matrix.

Figure 6 and Figure 7 show how the memory capacity changes for matrices scaled to have specified spectral radius (largest singular value, respectively).

All these three figures indicate that the memory capacity is maximized at certain optimal value, while declining when we distance ourselves from that optimal point.

The point of optimum is also recognizable by sudden increase of memory capacity variance. The large variance (and therefore the standard deviation as well) marks the transition from stable to chaotic reservoir dynamics. The change of memory capacity distribution at this optimal point is discussed in Appendix A.1.

3.2.1 Chaotic behaviour and eigenvalues

As we can see from Figure 6, the memory capacity is maximized slightly under the value $\rho = 1$. The largest absolute eigenvalue 1 is an important boundary value, because when taking vectors $\mathbf{x}(t)$ and $\mathbf{u}(t)$ very small, the reservoir update formula

$$\mathbf{x}(t+1) = \tanh(W^{\text{in}} \cdot \mathbf{u}(t+1) + W \cdot \mathbf{x}(t))\,, \tag{10}$$

changes to

$$\mathbf{x}(t+1) \approx W^{\text{in}} \cdot \mathbf{u}(t+1) + W \cdot \mathbf{x}(t) \tag{11}$$

because the tanh function operates in its 'linear' domain (close to zero). Several iterations operate like this:

$$\begin{aligned}\mathbf{x}(t+k) &\approx W^k \cdot \mathbf{x}(t) + W^{k-1} \cdot W^{\text{in}}\mathbf{u}(t+1) + W^{k-2} \cdot W^{\text{in}}u(t+2) + \dots \\ &+ W^1 \cdot W^{\text{in}}\mathbf{u}(t+k-1) + W^{\text{in}}\mathbf{u}(t+k) \qquad (12)\end{aligned}$$

The important part is $W^k\mathbf{x}$. If we take the Jordan normal form of the matrix $W = U\Lambda U^{-1}$ then having eigenvalues (in absolute value) greater than 1 causes the expression $W^k\mathbf{x} = U\Lambda^k U^{-1}\mathbf{x}$ to diverge. So small perturbations of the network state $\mathbf{x}$ around $\mathbf{x}_0 = 0$ are amplified. Although the bounded nonlinear activation function tanh prevents the reservoir activations from leaving the compact interval $[-1, 1]$ this situation still leads to chaotic behaviour of the dynamic system which has detrimental effect on memory capacity, as Boedecker et al. (2012) observed.

Moreover, having the spectral radius less or equal to 1 was a necessary condition for a reservoir to have echo states, as proven by Jaeger (2001a). And without echo states, the memory capacity is not well-defined (since it assumes, that the states are dependent only on left-infinite input history).

3.2.2 Optimal sigma and circular law

The reason why optimal value of reservoir scaling parameter σ is around 0.09 is tied to the optimal spectral radius (the largest absolute eigenvalue). The connection is namely a theorem called *the circular law.* It states that if we take a random $N \times N$ matrix whose elements are taken i.i.d. (independent and identically distributed) from distribution with mean 0 and variance $1/N$, the limiting distribution of its eigenvalues is the unit disc. This means that the limiting spectral radius is 1.

Since a multiple $a \cdot W$ of matrix W with eigenvalues λ_i has eigenvalues $a \cdot \lambda_i$, this with circular law yields, that the variance corresponding to the limit spectral radius 1 is $\sigma^2 = 1/N$. A $N = 100$-unit reservoir gives us corresponding parameter value $\sigma = 0.1$. Our measured optimum $\sigma = 0.09$ provides a spectral radius slightly less than 1. Note that the circular law gives us a hint how the optimal σ parameter changes with the reservoir size.

The loss of echo states would account for the decrease of memory capacity for $\sigma > \sigma_{\text{opt}}$, but why does the memory capacity fall for $\sigma < \sigma_{\text{opt}}$? That is, why is there a global maximum in Figure 5? The answer could be – numerical errors.

If we decrease the σ reservoir scaling parameter, we therefore decrease also the smallest singular value of the matrix. A singular value (for example 10^{-3}) represents a direction in state vector space which shrinks by 10^{-3} after one reservoir update. After a few iterations, the information in direction representing the k-past input can get very quickly under the floating point precision provided by the computer. For the standard data type `float64` the machine epsilon (which is the maximum relative rounding error when rounding a number to the nearest representable one) is $2^{-53} \approx 1.1 \cdot 10^{-16}$. So, if the average activation in the reservoir had the order of 0.1 and the smallest singular value would be 10^{-3}, in the worst case scenario, an information could be lost after 5 iterations. For illustration, the matrices W, W^{in} could be:

$$W^{in} = \begin{pmatrix} 0.2 \\ -0.15 \\ \vdots \\ -0.09 \end{pmatrix} \qquad W = \begin{pmatrix} 0.9 & 0 & 0 & \dots & 0 \\ 0 & 0.89 & 0 & \dots & 0 \\ 0 & 0 & 0.92 & \dots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & 0 & 10^{-3} \end{pmatrix}$$

This observation leads us to two conclusions:

1. Tuning N and τ parameters can affect the memory capacity, which is shown in the *input matrix scaling* paragraph below.

2. Finding the reason for a small memory capacity in too small singular values / eigenvalues, gives us a whole range of reservoir optimization techniques which try to manipulate singular values and eigenvalues. We will return to this in later paragraphs.

3.3 Reservoir size

Next we investigated the effect of number of reservoir neurons on memory capacity. Since the optimal reservoir scaling may vary with reservoir size, we plotted the memory capacity against the matrix-generating value (σ, ρ or $s_{\max}$) and represented different reservoir sizes as separate lines.

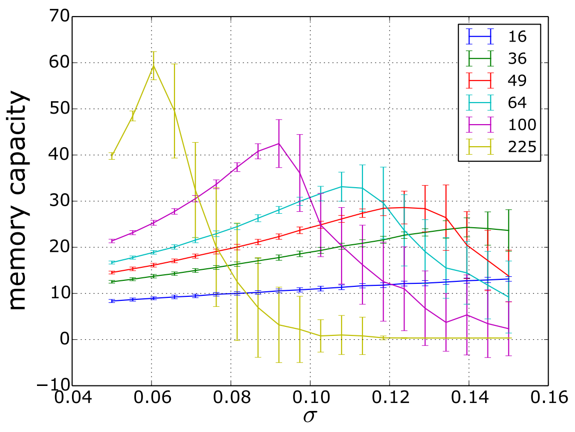


Figure 8: Memory capacity changing with σ for different reservoir sizes.

As Figure 8 shows, reservoirs of every size attain MC maximum at certain σ value and drift to zero MC for $\sigma \to 0$ or ∞. The position of the optimal σ value should move according to the circular law, discussed in the previous section.

Notice that the maximal MC value does not grow linearly with reservoir size. Larger reservoirs have worse ratio $MC_{\max}$/reservoir size.

N	16	36	49	64	100	225
maximal MC	14	25	29	33	42	64
$\frac{MC_{\max}}{\text{reservoir size}}$	86%	69%	59%	52%	42%	28%

Table 3: Memory capacity effectiveness for increasing reservoir size.

The reason behind the diminishing effectiveness lies again in numerical rounding errors. Large matrices may have a higher number of small singular values and therefore more directions in which the information strength shrinks considerably. Moreover, information about each past input signal is more distributed and very precise readout matrices may be needed to recover it.

Now, let us take a look at dependency on spectral radius based reservoir scaling:

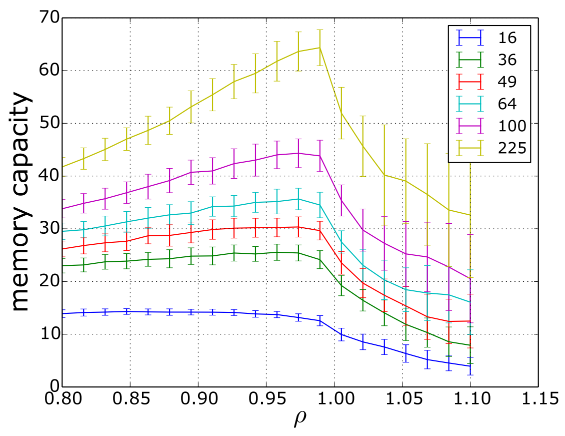


Figure 9: Memory capacity changing with spectral radius of the recurrent matrix for different reservoir sizes.

The optimal spectral radius for reservoir matrix moves less than optimal σ parameter, when reservoir size changes. Values close under 1 seem to be acceptable for large reservoirs.

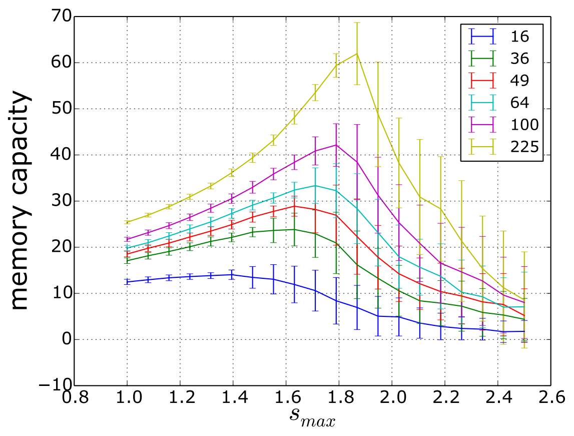


Figure 10: Memory capacity changing with largest singular value of the recurrent matrix for different reservoir sizes.

The dependence on the largest singular value depicted in Figure 10 look very similar to Figure 9, but with x-axis doubled. There is no apparent advantage of using the largest singular value as the scaling factor compared to using the spectral radius. The relationship between $s_{\max}$ and ρ in random matrices is discussed in the following paragraph.

3.3.1 Relationship between largest singular value and spectral radius

If we scale the reservoir according to largest singular value, how do we pick an optimal value? As the reservoir size increases, this optimum approaches the value 2. Let us plot the largest singular values and spectral radii of random matrices:

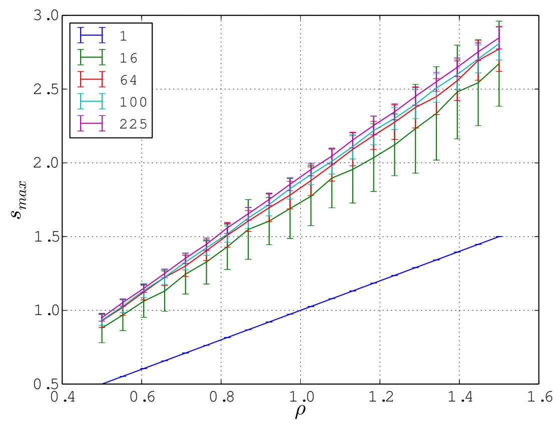


Figure 11: Largest singular value plotted against spectral radius for different matrix sizes. Each data point represents 100 matrix instances.

For 1×1 matrix, the only eigenvalue, the only singular value and the only matrix element coincide (in absolute value), and so the relationship between $s_{\max}$ and ρ is simple: $s_{\max} = \rho$.

For larger matrices the largest singular value and spectral radius cannot be determined from each other, but still, in average they tend to depend linearly on each other, forming a line $s_{\max} = k \cdot \rho, \;\; 1 \leq k \leq 2$. The limit line seems to be $s_{\max} = 2 \cdot \rho$ which is approached by large reservoirs (this was tested on 4000×4000 matrix).

3.4 Matrix sparsity

Interconnections between biological neurons are located in a three-dimensional euclidean space and do not generally form a complete graph. Therefore, it is biologically plausible that the matrix W representing their strength could be sparse. We investigate the effect the matrix sparsity has on memory capacity. Sparsity 0.8 means that 80% of the weights are set zero.

A 100-neuron reservoir with 0.8 sparsity would be generated as follows:

1. Generate the matrix using $N(0, \sigma^2)$ distribution.

2. Choose $\frac{80}{100} \cdot 100^2 = 8000$ weights which will be set zero.

3. Optionally, scale the matrix to have the desired spectral radius (or largest singular value).

Again, the optimal scaling may vary for different matrix sparsities, so the matrix scaling parameters were included in the plot.

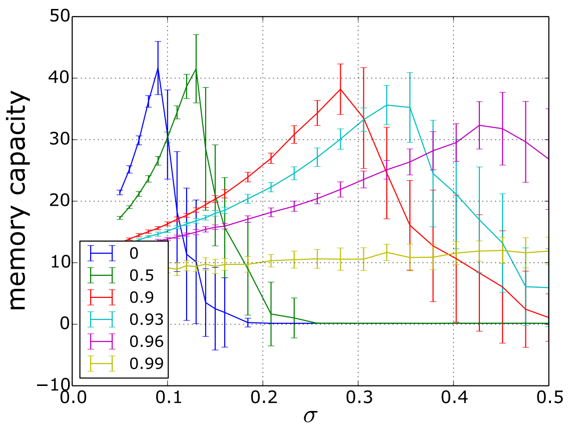


Figure 12: Memory capacity changing with σ for different matrix sparsities.

Memory capacity maxima are attained at different σ values. In particular, leaving more synapses zero means that we have to compensate for it with stronger synapse strengths to distribute the same amount of signal.

A pleasing observation is that the memory capacity falls very slowly with increasing sparsity, making sparse reservoir a viable option. Having a matrix more sparse does not increase the memory capacity, but it is generally advised (Lukoševičius and Jaeger, 2009) when processing structured data.

Moreover, matrix sparsity allows using sparse matrix representations and therefore efficient matrix multiplication algorithms. If we keep the neuron out-degree (number of neurons each neuron is connected to) fixed, the cost of reservoir update increases only linearly with the reservoir size, instead of quadratically.

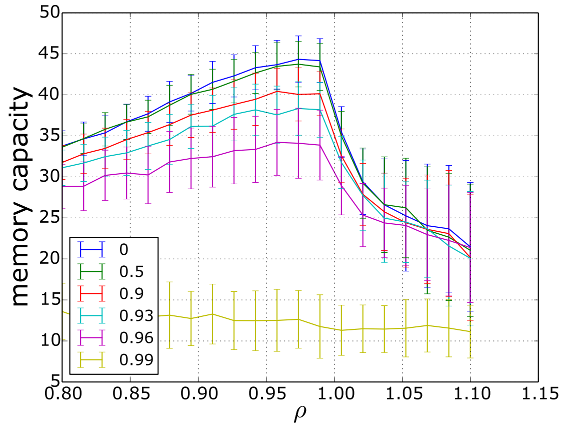


Figure 13: Memory capacity changing with spectral radius of the recurrent matrix for different matrix sparsities.

According to Figure 13, matrix sparsity does not affect the optimal spectral radius, which means we do not have to worry about this parameter.

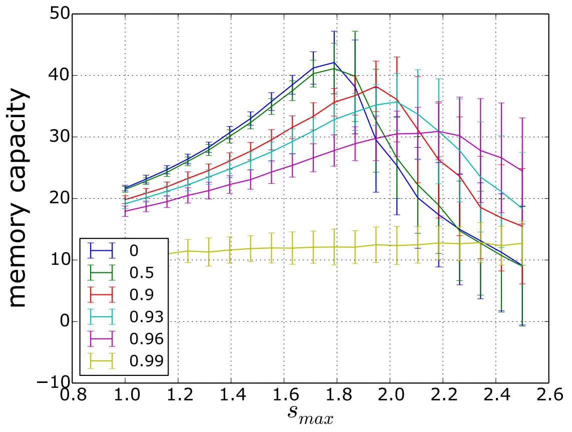


Figure 14: Memory capacity changing with largest singular value of the recurrent matrix for different matrix sparsities.

Surprisingly, the optimal $s_{\max}$ does not stay constant when changing matrix sparsity, and even gets beyond the 'boundary' value 2. The Figure 14 shows that singular values and eigenvalues behave differently when it comes to sparse matrices.

3.5 Orthogonalization procedure

Permutation matrices (which are a special case of orthogonal matrices) were tested by Boedecker et al. (2012) (and suggested by Hajnal and Lörincz (2006)) as a way of reservoir matrix initialization which outperforms other initialization methods in both memory capacity and in some structured data prediction (NARMA 30-th order system).

The matrix chosen in Boedecker et al. (2012) in fact is only 'almost' orthogonal – it is a permutation matrix scaled by factor of, for example 0.95 – to guarantee echo states in the ESN.

The observation that 'orthogonal' matrices perform so well in memory capacity task led us to the following, the orthogonalization process. The idea is to have a homotopic bridge between a randomly generated matrix to a matrix with columns forming an orthogonal basis (but not necessarily orthonormal basis). These 'quasi'-orthogonal matrices tend to have very high memory capacities.

The process itself is a gradient descent method in the space of $N \times N$ matrices based on energy function

$$E(V) := \|[M(V)]^T M(V)\|^2 ,$$

where $M(V)$ denotes matrix V whose columns have been normed and $\|\cdot\|$ is Frobenius norm. Differentiating this energy function leads to an update formula

$$\Delta v^{(i)} = -\eta \frac{4}{\|v^{(i)}\|} \left[I - m^{(i)} m^{(i)T}\right] \left(MM^T - I\right) m^{(i)} \tag{13}$$

$$\Delta V = \left(\Delta v^{(1)}, \Delta v^{(2)}, \ldots, \Delta v^{(n)}\right) \tag{14}$$

$$V \leftarrow V + \Delta V \tag{15}$$

with the learning speed η. Note that $v^{(i)}$ is the i-th column of matrix V.

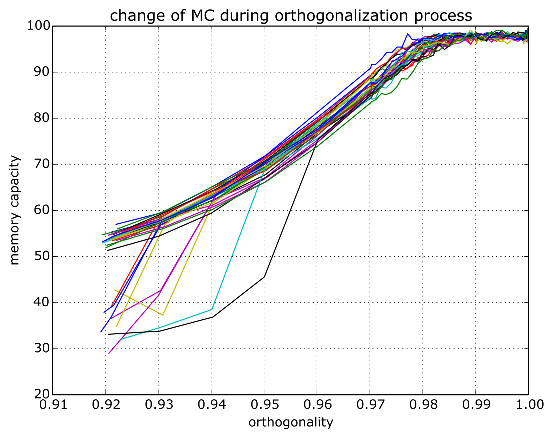


Figure 15: Memory capacity of 30 reservoir instances changing during orthogonalization process.

Figure 15 shows that using the orthogonalization process vastly improves the memory capacity of a randomly (normally-distributed) generated matrix.

The orthogonality axis depicted in Figure 15 is defined similarly to aforementioned energy function; It is computed as an average $1 - \cos(angle)$, where $angle$ is an angle between two matrix columns (when considered as vectors).

3.6 Input matrix scaling

It turns out that parameter τ has a profound effect on memory capacity: **The memory capacity increases with decreasing τ.** This is true for $\tau \geq 10^{-7}$. Then the MC growth stops.

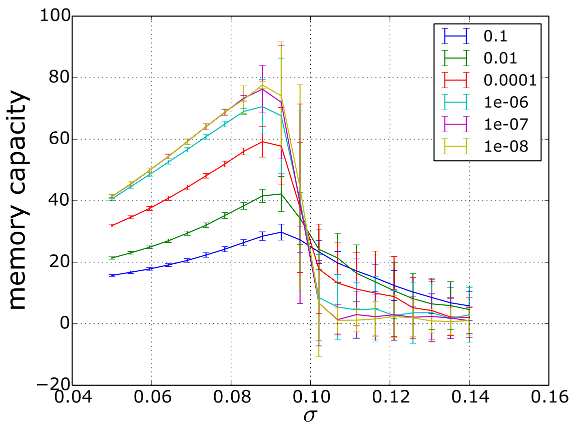


Figure 16: Memory capacity as a function of σ for different input matrix scaling.

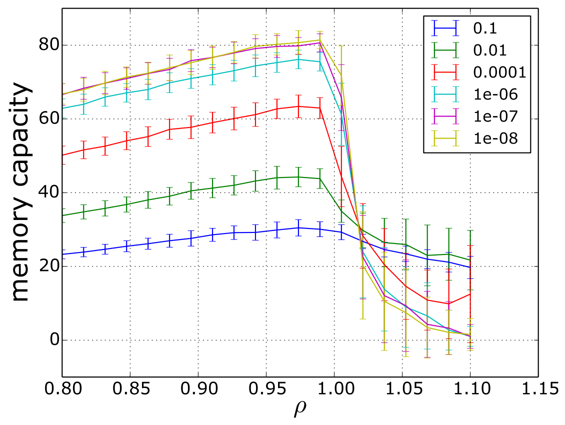


Figure 17: Memory capacity as a function of the spectral radius of the recurrent matrix for different input matrix scaling.

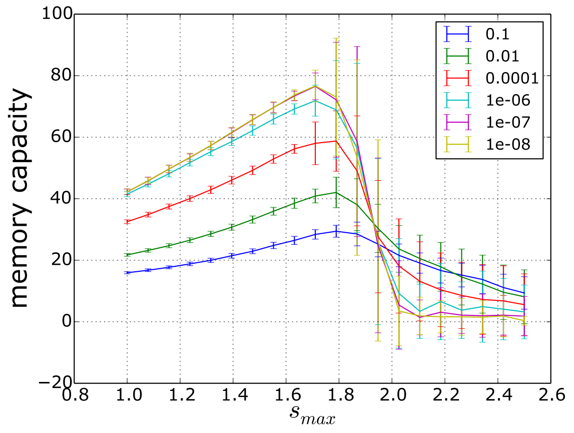


Figure 18: Memory capacity as a function of the largest singular value of the recurrent matrix for different input matrix scaling.

This surprising effect may be caused by the fact, that when τ is small, reservoir activations are low as well and the activation function (tanh) operates in its linear range. The diminishing effect might be caused by limited precision of computer floating-point operations as suggested by Jaeger (2001b).

3.7 Summary

The summary of previous sections is provided in short points:

1. Reservoir size increases the MC, but the increase is worse than linear.
2. Matrix sparsity decreases the MC slightly.
3. For all three approaches of generating the recurrent matrix (σ, ρ, $s_{\max}$) there exists an optimal value, for which the MC is maximal. The optimal σ value moves, while the optimal spectral radius (optimal largest singular value) approaches the value $\rho = 1$ (the value $s_{\max} = 2$ respectively). This makes ρ and $s_{\max}$ better alternatives for matrix scaling in matrix initialization.
4. The process of orthogonalization increases the MC greatly.
5. The MC increases with *decreasing* input matrix parameter τ. This is due to neuron activations operating in ‘linear’ range of tanh activation function.

4 Conclusion

This thesis provides an experimental overview of impact of various reservoir initialization parameters on memory capacity. We need to keep in mind that the high memory capacity is not the only thing we want from an ESN and that the parameters which yield optimal memory capacity can perform poorly on real-world, structured data.

Interesting is the revelation, how strong is the loss of memory capacity affected by rounding errors and the numerical precision of the floating-point representation of real numbers in computers. This opens the door of possibilities for using reservoir matrices close to orthogonal, for those who seek very large memory capacity. Here, the idea was developed only briefly (in form of the orthogonalization procedure) and provides space for further research.

References

Barančok, P. and Farkaš, I. (2014). Memory capacity of input-driven echo state networks at the edge of chaos. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*. Hamburg, Germany.

Boedecker, J., Obst, O., Lizier, J. T., Mayer, N. M., and Asada, M. (2012). Information processing in echo state networks at the edge of chaos. *Theory in Biosciences*, 131(3):205–213.

Doya, K. (1992). Bifurcations in the learning of recurrent neural networks. In *Circuits and Systems, 1992. ISCAS '92. Proceedings., 1992 IEEE International Symposium on*, volume 6, pages 2777–2780 vol.6.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

Hajnal, M. A. and Lörincz, A. (2006). Critical echo state networks. In Kollias, S. D., Stafylopatis, A., Duch, W., and Oja, E., editors, *ICANN (1)*, volume 4131 of *Lecture Notes in Computer Science*, pages 658–667. Springer.

Jaeger, H. (2001a). The "echo state" approach to analysing and training recurrent neural networks - with an erratum note. There is an Erratum note for this techreport at http://www.faculty.jacobs-university.de/hjaeger/pubs/EchoStatesTechRepErratum.pdf .

Jaeger, H. (2001b). Short term memory in echo state networks.

Jaeger, H. (2007). Echo state network. *Scholarpedia*, 2(9):2330. revision #143667.

Lukoševičius, M. (2012). *A practical guide to applying echo state networks*, volume 7700 of *Lecture Notes in Computer Science*, pages 659–686. Springer Berlin Heidelberg, 2 edition.

Lukoševičius, M. and Jaeger, H. (2009). Survey: Reservoir computing approaches to recurrent neural network training. *Comput. Sci. Rev.*, 3(3):127–149.

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

Zeng, X., Eykholt, R., and Pielke, R. A. (1991). Estimating the lyapunov-exponent spectrum from short time series of low precision. *Phys. Rev. Lett.*, 66:3229–3232.

Appendices

A Side results

A.1 Distribution of memory capacity at the edge of chaos

In chapter dedicated to recurrent matrix scaling parameters (σ, ρ and $s_{\max}$), we mentioned that at the optimum, the memory capacity variance suddenly increases. This can be observed on the vertical bars in most figures, which represent the standard deviation. The standard deviation, however, does not tell us about the actual distribution of the memory capacity.

The network setup was: a 100-neuron reservoir with tanh activation functions and input matrix initialisation parameter $\tau = 0.01$.

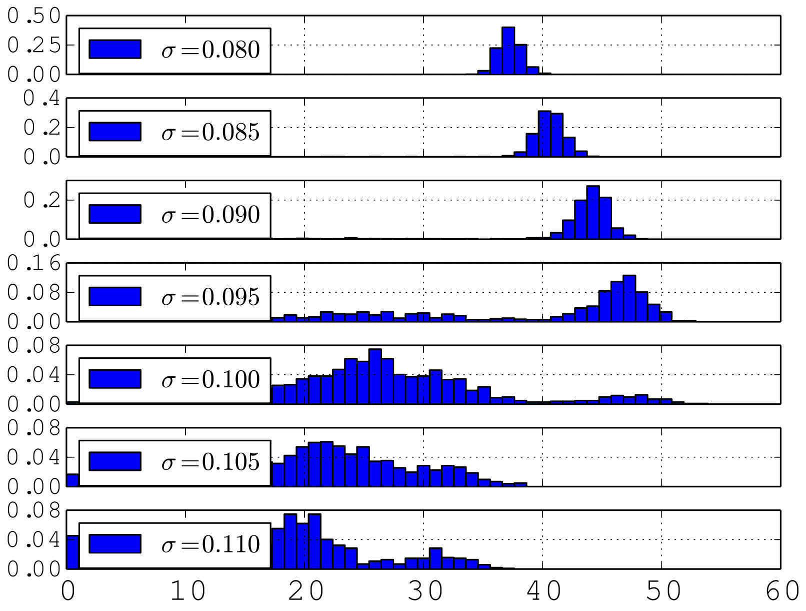


Figure 19: Memory capacity histograms for different parameters σ.

Figures 19 and 20 show, that the memory capacity distribution at the edge of chaos does not simply start shifting backwards, but instead, develops another peak. So, closely behind this turning point, both very good and very bad (in terms of memory capacity) reservoirs are generated.

This can make a difference if we do not care about performance of an average

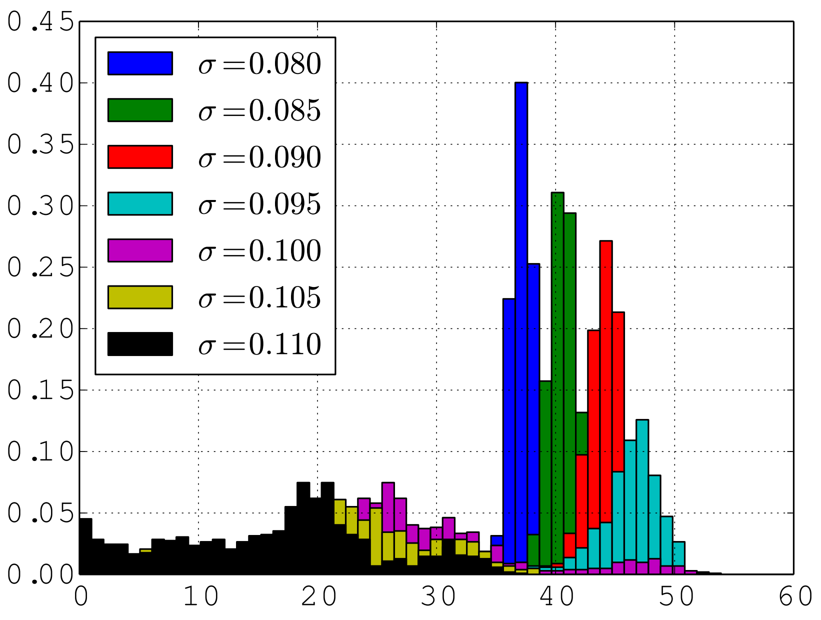


Figure 20: Memory capacity histograms for different parameters σ plotted on single image.

reservoir, but adopt an initialization strategy similar to *'pick best of 10 generated reservoirs'*.

A.2 MC forgetting curves

Recall that the memory capacity is defined as an infinite sum

$$MC = \sum_{k=1}^{\infty} MC_k \, .$$

The next value in a temporal sequence, which the network is supposed to predict does not necessarily depend on every past input. It may depend, for example, on only 10-th and 15-th past input. An example of such sequence could be one, generated by the equation

$$x(t) = x(t-10)x(t-15) + 0.2x(t-10) - 0.4x(t-15) \, .$$

In that case, individual MC_k are of interest (and not the whole sum).

In two following experiments, we set $\tau = 0.01$ and calculate the MC_k values for a 100-neuron reservoir on Figure 21 and for a 20-neuron reservoir on Figure 22.

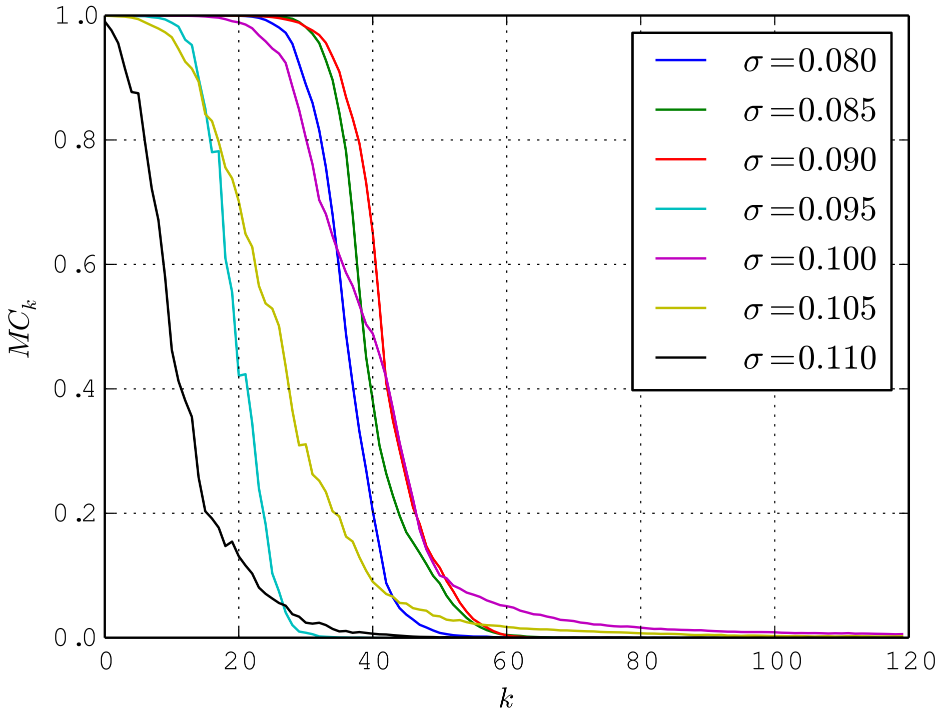


Figure 21: MC_k values of a 100-neuron reservoir for different values of parameter σ.

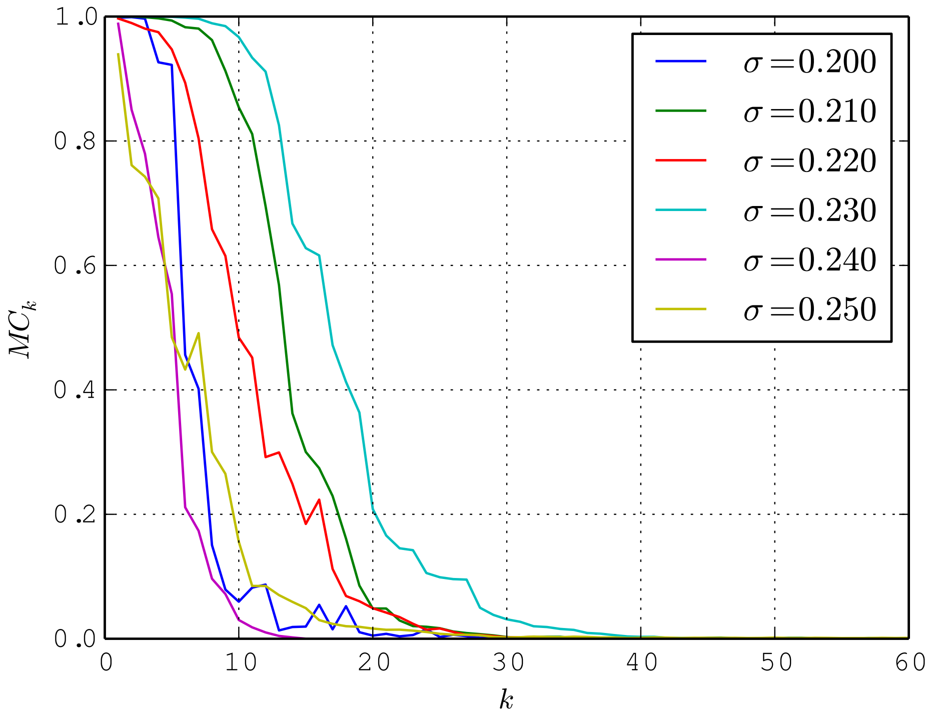


Figure 22: MC_k values of a 20-neuron reservoir for different values of parameter σ.

B Used software and graphics

All experiments were programmed in programming language **python**, using numerical library *numpy* and graphical library *matplotlib*.

All images (except Figure 2, Figure 3 and university logo in the title page) were created by me and are hereby released into public domain, according to the CC0 licence.

The source code for the python function `memory_capacity` used in most experiments is given in the code listing below.

```python
from numpy import random, zeros, tanh, dot, linalg, \
    corrcoef, average, std, sqrt, hstack
import scipy.linalg

def memory_capacity(W, WI, memory_max=None, iterations=1200,
    iterations_skipped=None, iterations_coef_measure=100,
    runs=1, input_dist=(-1., 1.),
    use_input=False, target_later=False):
    """Calculates memory capacity of an ESN
    [given by its input weights WI and reservoir weights W].
    W  = q x q matrix storing hidden reservoir weights
    WI = q x 1 vector storing input weights

    Returns: a non-negative real number MC
    MC: memory capacity sum for histories 1..MEMORY_MAX
    """
    # matrix shape checks
    if len(WI.shape) != 1:
        raise Exception("matrix WI must be vector-shaped!")
    q, = WI.shape
    if W.shape != (q, q):
        raise Exception("W and WI matrix sizes do not match")

    if memory_max is None:
        memory_max = q

    if iterations_skipped is None:
        iterations_skipped = max(memory_max, 100) + 1

    iterations_measured = iterations - iterations_skipped

    # vector initialization
    X = zeros(q)
    if use_input:
        S = zeros([q + 1, iterations_measured])
    else:
        S = zeros([q, iterations_measured])

    # generate random input
    u = random.uniform(input_dist[0], input_dist[1],
                       iterations)

    # run 2000 iterations and fill the matrices D and S
    for it in range(iterations):
        X = tanh(dot(W, X) + dot(WI, u[it]))

        if it >= iterations_skipped:
            # record the state of reservoir activations X
            # into S
            if use_input:
                S[:, it - iterations_skipped] = \
                    hstack([X, u[it]])
            else:
                S[:, it - iterations_skipped] = X

    # prepare matrix D of desired values
    # (that is, shifted inputs)
    assert memory_max < iterations_skipped
    D = zeros([memory_max, iterations_measured])
    if target_later:
        # if we allow direct input-output connections,
        # there is no point in measuring 0-delay corr. coef.
        # (it is always 1)
        for h in range(memory_max):
            D[h,:] = u[iterations_skipped - (h+1) \
                           : iterations - (h+1)]
    else:
        for h in range(memory_max):
            D[h,:] = u[iterations_skipped - h \
                           : iterations - h]


    # calculate pseudoinverse S+ and with it, the matrix WO
    S_PINV = scipy.linalg.pinv(S)
    WO = dot(D, S_PINV)

    # do a new run for an unbiased test of quality of our
    # newly trained WO. We skip memory_max iterations to
    # have large enough window
    MC = zeros([runs, memory_max]) # memory capacity
    for run in range(runs):
        u = random.uniform(input_dist[0], input_dist[1],
            iterations_coef_measure + memory_max)
        X = zeros(q)
        o = zeros([memory_max, iterations_coef_measure])
        for it in range(iterations_coef_measure + memory_max):
            X = tanh(dot(W, X) + dot(WI, u[it]))
            if it >= memory_max:
                # we calculate output nodes using WO
                if use_input:
                    o[:, it - memory_max] = \
                        dot(WO, hstack([X, u[it]]))
                else:
                    o[:, it - memory_max] = dot(WO, X)

        # correlate outputs with inputs (shifted)
        for h in range(memory_max):
            k = h + 1
            if target_later:
                uslice = u[memory_max - k : memory_max \
                    + iterations_coef_measure - k]
                cc = corrcoef(uslice, o[h, : ]) [0, 1]
            else:
                uslice = u[memory_max - h : memory_max \
                    + iterations_coef_measure - h]
                cc = corrcoef(uslice, o[h, : ]) [0, 1]
            MC[run, h] = cc * cc

    return sum(average(MC, axis=0))
```