JACOBS
UNIVERSITY

Reservoir Computing and Self-Organized Neural Hierarchies

by

Mantas Lukoševičius

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
in Computer Science

Approved, Dissertation Committee:

Prof. Dr. Herbert Jaeger, chair
Jacobs University Bremen

Prof. Dr. Helge Ritter
Bielefeld University

Dr. Mathias Bode
Jacobs University Bremen

Submitted: August 31, 2011; Defended: December 1, 2011; Published in 2012

School of Engineering and Science

Abstract

There is a growing understanding that machine learning architectures have to be much bigger and more complex to approach any intelligent behavior. There is also a growing understanding that purely supervised learning is inadequate to train such systems. A recent paradigm of artificial recurrent neural network (RNN) training under the umbrella-name Reservoir Computing (RC) demonstrated that training big recurrent networks (the *reservoirs*) differently than supervised readouts from them is often better. It started with Echo State Networks (ESNs) and Liquid State Machines ten years ago where the reservoir was generated randomly and only linear readouts from it were trained. Rather surprisingly, such simply and fast trained ESNs outperformed classical fully-trained RNNs in many tasks. While full supervised training of RNNs is problematic, intuitively there should also be something better than a random network. In recent years RC became a vivid research field extending the initial paradigm from *fixed random reservoir and trained output* into *using different methods for training the reservoir and the readout*. In this thesis we overview existing and investigate new alternatives to the classical supervised training of RNNs and their hierarchies. First we present a taxonomy and a systematic overview of the RNN training approaches under the RC umbrella. Second, we propose and investigate the use of two different neural network models for the reservoirs together with several unsupervised adaptation techniques, as well as unsupervisedly layer-wise trained deep hierarchies of such models. We rigorously empirically test the proposed methods on two temporal pattern recognition datasets, comparing it to the classical reservoir computing state of art.

Declaration

This thesis has been written independently and has not been submitted at any other university for the conferral of a degree.

Parts of the research reported in this thesis at their earlier stages have been published in [Lukoševičius and Jaeger, 2007, 2009] and [Lukoševičius, 2010].

The thesis is entirely written by me with an exception of Sections 2.1 and 2.3.6 that are mostly written by my supervisor and coauthor in the publications Herbert Jaeger, and have been retained for the continuity of the thesis.

Bremen, August 31, 2011 Mantas Lukoševičius

Acknowledgements

I would like to thank all my teachers, parents, and friends.

I first of all thank my supervisor Herbert Jaeger for his constant support, sharing the knowledge, mentoring, and scientific latitude granted, which shaped me as a researcher. I would also like to thank my colleagues Vytenis, Michael, Melanie, Manjunath, Jiwen, Narūnas, other professors, and the rest of Jacobs University community for creating the diverse and stimulating academic environment; also the whole reservoir computing community for warm and fruitful interactions.

Special thanks goes to Planet Intelligent Systems GmbH: Welf, Udo, Uli, Klaus, Donata, Hagen, and others, not only for the scientific cooperation, but also for their hospitality, adventurous company, and generosity which in part supported my research.

I am also grateful to Yoshua Bengio and all his lab in University of Montreal, where I did a summer internship in 2005, for my first serious exposure to the world and community of machine learning, and memorable personal experiences; also for the continuous contacts and advice.

In addition I want to thank Peter Tiňo and Laurenz Wiskott for stimulating discussions and valuable feedback on my research.

I am also thankful to my teachers and professors in Lithuania, especially in Kaunas University of Technology and its Gymnasium, for fostering my curiosity.

Last, but not least, I thank my parents for the upbringing and their unconditional and unwithering support and encouragement, as well as my personal friends. This support helps and means a lot to me.

This research is partially funded by the EC FP7 project Organic (FP7-231267).

Contents

Abbreviations and notation

List of abbreviations

AI	Artificial intelligence, page 2
APRL	Atiya-Parlos recurrent learning, page 22
BMU	Best matching unit, page 78
BP	Backpropagation, page 21
BPDC	Backpropagation-decorrelation, page 26
BPTT	Backpropagation through time, page 21
CPU	Central Processing Unit, page 4
CR	Classification rate, page 87
ESN	Echo state networks, page 23
Evolino	Evolution of recurrent systems with optimal linear output, page 25
EVS	Eigenvalue spread, page 39
FFNN	Feedforward neural networks, page 12
FORCE	First-order reduced and controlled error, page 27
IB	Information bottleneck, page 44
ICA	Independent component analysis, page 44
IP	Intrinsic plasticity, page 43
LI	Leaky integrator, page 36
LI-ESN	Leaky integrator (LI) echo state networks (ESNs), page 36
LMS	Least mean squares, page 55
LSM	Liquid state machines, page 24

LSTM	Long short-term memory, a type of RNN, page 25
ML	Machine learning, page 2
MLP	Multilayer perceptron, a type of FFNN, page 60
NG	Neural gas, page 79
NN	Neural networks, page 3
NRMSE	Normalized root-mean-square error, page 18
PCA	Principle component analysis, page 44
RBF	Radial basis function, page 75
RC	Reservoir computing, page 14
RKM	Recurrent kernel machine, page 62
RLS	Recursive least squares, page 55
RNN	Recurrent neural network, page 12
RSOM	Recursive self-organizing maps (SOMs), page 75
RTRL	Real-time recurrent learning, page 21
SFA	Slow Feature Analysis, page 48
SOM	Self-organizing maps, page 75
SOR	Self-organizing reservoir, page 75
STDP	Spike-time-dependent plasticity, page 43
SVM	Support vector machine, page 56
SVR	Support vector regression, page 56
USPS	United States Postal Service, in this context: a dataset originating from it, page 82
WDN	Weighted distance network, WeiDiNet, page 99
WSN	Weighted sum and nonlinearity, a type of neuron model, page 75

Mathematical notations

a, α, A	Scalars
$\mathbf{a}$	Vector
$\mathbf{A}$	Matrix
$\mathbf{A}^{\mathrm{T}}$	Matrix transpose
$\mathbb{R}^N$	A set of real-valued vectors of size N
$\mathbb{R}^{N \times M}$	A set of real-valued matrices of size $N \times M$
$[\cdot;\cdot]$	Vector or matrix vertical concatenation, page 24
$\|\cdot\|$	Euclidean distance (norm), page 18
$\langle\cdot\rangle$	Mean, page 18
n	Discrete time, page 18
N_{u}	Input signal dimension, number of input units, page 18
N_{x}	Internal state dimension, number of reservoir units, page 21
N_{y}	Output signal dimension, number of output units, page 18
$\mathbf{A}^{+}$	Pseudoinverse of matrix $\mathbf{A}$, page 53
$\rho(\mathbf{W})$	Spectral radius, i.e., the largest absolute eigenvalue, of $\mathbf{W}$, page 31
$\sigma_{\max}(\mathbf{W})$	The largest singular value of $\mathbf{W}$, page 32
T	The number of time steps in the dataset, page 18
$\mathbf{u}(n)$	Input signal, page 18
$\mathbf{W}$	Internal weight matrix, page 21
$\mathbf{W}^{\mathrm{in}}$	Input weight matrix, page 21
$\mathbf{W}^{\mathrm{ofb}}$	Output feedback weight matrix, page 21
$\mathbf{W}^{\mathrm{out}}$	Output weight matrix, page 19
$\mathbf{x}(n)$	Internal state signal, reservoir activations, page 21
$\mathbf{y}(n)$	Output signal, page 18
$\mathbf{y}_{\mathrm{target}}(n)$	Target (a.k.a. desired, teacher) output signal, page 18

Chapter 1

Introduction

The human desire for building thinking machines far predates the electronic age. It is closely related to the desire to understand what thinking, intelligence, consciousness actually are and what makes a human different from a machine. The question of whether or how these human characteristics can be replicated is of a central philosophical importance, as humans define themselves through them. This is the *sapiens* in *Homo sapiens*, or *cogito* in Descartes' famous *"cogito ergo sum"*.

From a long-term abstract perspective, this desire of humans might well be just a manifestation of an evolutionary tendency of intelligence to expand and replicate itself, current human condition being but an intermediate form of it.

Aside from philosophical aspirations of understanding intelligence by building it, and this understanding having enormous social implications, there are also obvious pragmatic reasons for having machines able to substitute humans in intellectual tasks.

1.1 Artificial intelligence

Formalizations of the thinking process in form of logics and mathematics dating back from antiquity were one of the most important developments and enablers in the history of science. Such thinking process reduced to "mechanical" manipulations of concepts could be implemented outside of a human in a machine that had sufficient computational capabilities.

The advent of electronics and digital programmable computers in the 20th century was so far the biggest technological step toward realizing this vision. In

1956 it sparkled the research field of Artificial Intelligence (AI). With continuous exponential explosion in computational power of computers and progress in algorithms, creating human-level AI seemed to be just around the corner. All what many deemed to be necessary was to codify human knowledge in a formal language and let good logic-based inference algorithms running on powerful computers solve any solvable problem requiring intelligence.

While all of these innovations have a tremendous impact and success in restricted domains, the general human-like intelligence turned out to be a much more elusive goal to achieve than initially thought. It became apparent, that not all intelligent reasoning can be expressed in logical rules and not all knowledge can be easily expressed in a formal representation. In fact the very definition of intelligence turned out to be quite problematic and multifaceted.

After the initial enthusiasm and later setback, the current state of affairs in the AI field is that it is highly fragmented, (quite successfully) focusing on much narrower problems and different aspects of intelligence, being influenced by different disciplines, views, and motivations. For example, it is a matter of perspective whether an artificial system should mimic human intelligence and how closely, if it can solve a given concrete task better without doing so – which is true for many engineered systems.

1.2 Machine learning

One of the most important aspects of intelligence is *learning*. Learning captures adaptivity of an intelligent agent to its environment which is one of the defining features of a living entity. In the realm of AI this aspect has been taken up by *Machine Learning* (ML). Loosely speaking, ML investigates how artificial systems can improve their performance with experience.

Learning is especially important because it does not only investigate this particular aspect of an existing intelligent system, but also mechanisms of how intelligence comes to be. ML allows artificial systems to cover areas of intelligent tasks that are not susceptible to efficient formal codification. In fact, it is most likely, that scaling AI systems any closer to the human intelligence level would not be feasible without extensive use of ML if only because of the enormous manual effort required otherwise. After all, even the most intelligent natural systems known largely acquire their intelligence through many years of continuous and

intense learning.

ML is by itself a highly non-homogeneous interdisciplinary field with many schools of thought, many different computational models and approaches to train them. Motivations of ML research range from achieving the best performance in a given task, to building more mathematically clean and tractable systems, developing a better mathematical understanding of them, explaining and modeling certain characteristics of natural intelligent systems, or trying to build more integrative and universal models, among others.

1.3 Neural networks and computation

This thesis is mostly concerned with a particular branch of machine learning dealing with artificial Neural Networks (NN), a computational model loosely inspired by the substrate of the natural intelligence, the brain. In particular, it is mainly dealing with Recurrent NNs (RNNs) and the Reservoir Computing (RC) approach to training them. All this, and the level of abstraction at which NNs are modeled, is explained in detail in Section 2.1.

Surely, NNs is not the only computational model, nor in every respect the best, to do ML. In particular, it is hard to interpret. A trained NN is most often treated as a black-box model of a system that has learned the required input-output mapping, but there is no good interpretation of *how* the problem is being solved. This is in contrast to some other ML methods (like decision tree learning or Bayesian networks) that extract intelligible rules which can be checked, corrected, or combined with knowledge of human experts. The parameters and structure of NNs have to be predefined largely based on experience and trial-and-error. Training is typically a long iterative process the global optimum of which is not guaranteed. It is hard to predict NN behavior with uncommon inputs. NNs usually do not offer probabilistic interpretations. Complex and recurrent NNs are also hard to analyze mathematically. Due to these characteristics, researchers in ML community often choose other techniques than NNs.

But NNs also have unique advantages. Mathematically they can approximate any function [Cybenko, 1989] or dynamical system (in the case of RNN) [Funahashi and Nakamura, 1993] arbitrarily close. Since natural intelligence is based on neural networks, we know that such level of intelligence can in principle be achieved with NNs (how much abstracted from the real brain they can be is a separate and

open question), while there is no such guarantee for AI techniques circumventing NNs. NNs research is also a two-way street between ML and (computational) neuroscience. It is essential in understanding the components of natural intelligence better and the way they can be replicated.

Even with all the rapid expansion of computational power – growing exponentially for the last 50 years, following Moore's law – computation still remains an issue in AI. By most estimates modern computers are still far behind the computational capacities of human brains. They also work very differently. A classical computer model has a single very fast central processing unit (CPU) that processes in a serial manner big amounts of information stored in a separate memory. In contrast, the brain is a relatively slow, but vastly parallel information processing system, where the separation between information storage and processing is not clear-cut. This makes emulation of big parallel NNs on serial CPUs computationally expensive, and thus less attractive than algorithms directly programmed and optimized for such computational architecture.

However, looking at the trends, the current CPU technology is already slowly approaching its technological limits. To keep up with Moore's law, CPU producers started to introduce more and more processing cores that work in parallel, while the speed of a single core is starting to fall behind the trend. Such approach was previously used only to build supercomputers that overtake in power best contemporary simple CPUs by many years. The importance of parallel computation is rising fast. Graphical processing units that are vastly parallel, even though with limited precision, have lately overtaken CPUs in computational power. Initially designed exclusively for 3D graphics, these devises are increasingly more often used for general-purpose computations, including ML and NNs, offering orders of magnitude of speedup.

It is likely that in the quest to maximize computational power and making the computing elements ever smaller at some point (and some instances) the strict determinism of the digital computing systems will be worth sacrificing. Such computers would require fault-tolerant algorithms that can cope with the "noise" of the hardware. Currently such faults are corrected at a lower level through redundancy in data storage, communication, and sometimes computation. But even in higher levels today, to gain computational power through massively parallel and distributed data processing in large applications, strict consistency of large databases is compromised or non-deterministic outcomes of interacting parallel

processes are accepted. The scientific community is also investigating different concepts of computation. There is a growing understanding that our digital computers is just one of many possible ways to do it. "Computation" in this broader perspective is now becoming an as a multifaceted term as "intelligence". The definitions of computation range from very narrow, where it can only be performed by a digital computer, i.e., Turing machine; to very broad "pancomputationalist" ones, calling basically any process in nature computation.

These trends make computers more similar to brains, and NN-like algorithms a very promising approach of utilizing their power. In fact, the paradigm of reservoir computing can be applied beyond NNs and digital, or even electronic, computers to make useful computations with different kinds of dynamical systems, like physical or optical, as explained in Section 2.3.7.

1.4 Limits of current machine learning

In contrast to the vision of general-purpose AI, ML is typically applied to a very restricted domain, for a very concrete particular task.

A sketch of a typical workflow applying ML is to:

- Manually choose a parameterizable computational *model* (for example, a rather universal one, like NN, but there is a wide variety) and its meta-parameters (like the number of neurons in NN, their connectivity).
- Manually choose a *training algorithm* and its parameters for the model. Often a multitude of algorithms exists for a single model.
- *Train* the model to perform a desired task by adapting its free parameters using the algorithm. Training is done using *data* as the experience from which the model learns. Technically, training is an optimization problem, where the free parameters are adapted to optimize the measure of how good the model is performing the task. This is put in formal terms in Section 2.2.1.
- *Exploit* the trained model for the designated task.

Since this typically leads to sub-optimal results, the workflow is often nonlinear, where multiple choices are tried and the one working best is selected. In addition,

performance can typically be improved by employing all kinds of task-specific tricks that incorporate additional knowledge about it. This, for example, includes deciding on the best way the data is coded or what kind of informative features are extracted from it. In the end there is typically a combination of several ML techniques custom-made for any nontrivial task.

This approach does work. In fact it is about the only way of applying ML. But it has serious limitations. The workflow requires many subjective choices and a lot of human supervision. The field of ML is being criticized, quoting this as the main reason for slow progress [Dou, 2007].

In response to this criticism, ML researchers are lately trying to make their approaches more autonomous, but this is not easy.

Simple approaches are often employed trying to automate these usually subjective choices to some extent by putting the depicted workflow into an additional external "metalearning" loop. They work reasonably well for choosing the parameters of a selected model and a learning algorithm. The sketch of such an approach is depicted as Algorithm 1.

Algorithm 1 Machine learning with meta-parameter optimization

```
Manually select a computational model and a learning algorithm;
Manually select the (meta-)parameters of the above to optimize, their ranges,
and an optimization strategy;
repeat
    Take a new set of parameters according to the strategy;
    Generate and train a model using these parameters;
    Validate the performance of the model with data left-out from the training;
until Parameters exhausted or sufficient performance achieved;
return The model trained that had the best performance.
```

This approach, however, still has serious limitations. It is typically computationally very expensive: the already-expensive learning step is put in the external loop, where it usually goes through many iterations. There are typically no good meta-parameter optimization strategies that would effectively reduce the number of iterations: often simple grid search or genetic strategies are utilized.

As a result, only a limited number of meta-parameters can effectively be optimized this way. In a simple exhaustive grid search strategy, the runtime grows exponentially with the number of the meta-parameters. Also, ranges and step sizes of the parameters have to be selected carefully to ensure computational re-

sources are used efficiently. This also means that the affordable model complexity and (related) training time are limited: for a fixed computational time the meta-parameter optimization is done at an expense of training a single much more powerful model.

Still, the manual supervision and choices required are substantial, if not equal. The choices of the model and learning algorithm an ML practitioner has to make can land them in very different subfields of ML with very different processes of learning, and things to take care of. These choices, thus, are often subjective, based on practitioner's background and experience. Also, the above-mentioned task-specific improvements, that are crucial in practice, remain a human expert prerogative. Attempts to automate these aspects usually just lead to over-complicated systems "wondering in the dark" while wasting computational resources. This can be seen as a non-tractable optimization problem.

The vision of "autonomous machine learning" is far from being achieved. Even if the above-mentioned aspects would be solved, it would not automatically mean, that such "autonomous" ML methods (or "the method") would scale up any higher toward human-level broader intelligence than the current highly human-powered state of art.

This difficulty is natural when looking back at the bigger picture. Learning is clearly not the only component of intelligence. As important, or even more, is the predisposed (manually configured) model structure and initial values of the parameters. How much of it can in principle be learned from the data is an interesting open question. This is related to the question of how much of behavior is determined by the genetically predisposed traits and how much by adaptation/learning in animals[1], or the "nature versus nurture" take on human psychology. For example, there is an ongoing debate in the scientific community on how much humans learn language – one of the main attributes of intelligence – purely from "data" and how much of it is pre-wired in the brain genetically (see, e.g., [Pullum and Scholz, 2002]).

ML methods also require a lot of specially prepared data. They need to learn from many instances of solved tasks, where both the input and the solved output are available. The required amount of data – and thus learning times – grow very fast with the difficulty of the task, unless some task-specific tricks/structure are employed that exploit the hidden structure and invariances in data. For example,

[1]Even though evolution in an abstract sense can also be seen as learning.

the amount of data needed to reasonably cover all cases grows exponentially with the number of input variables, which quite soon becomes infeasible – the so-called "curse of dimensionality". This excludes ML applications from areas where data is not abundant compared to the difficulty of task, unless the space of possible solutions is severely restricted by incorporating prior expert knowledge.

1.5 Learning with less supervision

To overcome these seemingly overwhelming limitations, researchers are looking for more fundamental universal principles in ML, following which the artificial systems could self-organize without so much manual intervention. Examples in natural intelligence apparently do not suffer from many of the same limitations, showing that they in principle can be overcome.

The vast majority of applied ML, depicted in Section 1.4, is *supervised*. This means, that we know exactly how the solution for the training examples should look like, and thus can rather directly adapt the parameters of the model to minimize the deviations of the obtained solution from the required one. This setup of supervised ML, put more formally in Section 2.2.1, is very powerful and technically transparent: it is a well-formulated optimization problem, where learning is by construction directly improving the goal performance. However, this setup is already part of the limitations.

For one, as mentioned, it needs a lot of already solved examples to learn from. If they are feasible to produce, then the ML technique can only offer the benefit of solving the problem in a more economical way, and if the solution examples are scarce it is often not applicable at all. This paradigm is also not easily applicable if there are several viable solutions.

A more general setup, still classified as supervised, is *reinforcement* learning, where good solutions are "rewarded" (or bad "punished"), but there is no a priori predefined unique "correct" solution. It can lead to unexpected creative solutions and allow for exploration. However, from a technical perspective it is a much harder learning problem. The parameters are optimized based on a much less informative (and possibly delayed) feedback signal, which tells how good a solution is, but not how to improve it.

Reinforcement learning paradigm is much closer to the learning which is happening in nature where the learning agent receives rewards or punishments from

the environment on which it learns to act. Learning from the environment with which the agent can interact, as opposed to from static data, is another very interesting aspect with a lot of potential for ML. It is pursued more in ML with relation to robotics and *embodied* cognition, and will not be the main topic of this thesis.

Unsupervised learning is an even less restricted setup, where the learning agent is only provided with the input data and has to "make sense" of it on its own. No other guidance is given. The benefit is that the agent can often be provided with abundant amounts of data by just "observing the world". Also, learning can be done locally, in every component that is receiving and processing information, as input is all what it needs. This type of local unsupervised adaptation is also known to be happening a lot in biological brains. In big learning architectures applying supervised learning is difficult, as the scarce feedback signal has to be shared among all the many components, including those that contribute to the final output in a complex and indirect way.

This setup, however, raises two important questions. The first is: **(i)** what the agent should learn from the data? In other words, what should be the purpose and goal of unsupervised learning? And, looking at this from the other end: **(ii)** is what the agent learns in some sense useful? Or, in which sense it is useful?

Other principles than minimizing error or maximizing reward in ML are proposed to answer the first (i) question. They include: data compression, clustering, reducing dimensionality while preserving the topology, learning the statistical distribution of the input, minimizing free energy, learning to predict the input, looking for slowly varying (close to invariant) components of the data, sparse representation, maximal information transmission while using minimal energy, etc. These ideas are often subject to trends in machine learning community. We will touch on most of them in the thesis.

Many of these principles have nice analytical properties and can explain (sometimes surprisingly) a lot of learning in nature, but none of them gives the full picture. They are also often difficult to reconcile with each other. If there is one meta-principle governing all learning in nature, it is likely to be evolution. And evolution tends to produce solutions of type "whatever works". Thus, it is likely that there is no single clean and simple analytical principle according to which all learning (or reasoning) happens. If there is some kind of helpful "shortcut" trick outside such a hypothetical principle, evolution was likely to employ it in the

brain. If this is the case, systems tending toward general human-level AI might need to be heterogeneous, not conceptually elegant, and, as a consequence, hard to analyze, not unlike the human brain. This is unless in the future a system much computationally superior to the brain will be feasible to build, which in addition to being as universally intelligent could also afford to be analytically transparent; and/or possibly posses a quite different type of intelligence; which are just speculations at this point in time.

There is no good general answer to the second (ii) question of whether the supervised learning benefits some concrete task. For the same input different tasks (in the form of required output) can be formulated. Different unsupervisedly learned representations of the input data can be useful for different tasks. There is often difficult to make a connection between what is learned unsupervisedly and what is needed for the task, mostly because the task is defined only through the empirical data. Thus the evaluation can often only be done empirically.

In fact, formal evaluation of how "good" the unsupervised learning is for some concrete task puts the system back into the supervised framework. In this case it is also rational to use at least some supervised training even if the system was mostly trained unsupervisedly. In this context the question (ii) can be reformulated as: does the unsupervised learning help the supervised learning?

As mentioned above, supervised training of big RNN models is often difficult: it converges slowly and usually gets stuck in a local minimum that depends on parameter initialization. Reservoir computing, as elaborated in Section 2.1, has demonstrated that sometimes less learning in RNNs can be beneficial. In particular, a much more powerful computational model can be used if not all of its parameters are supervisedly trained, leading to superior results. In classical reservoir computing most of the parameters remain randomly generated according to certain rules. This points out the limitations of the purely supervised learning, but a relevant question is whether these random structures can be improved upon in some other, potentially unsupervised, way.

These are the main questions addressed in this thesis.

1.6 Contributions of the thesis

The main contribution of this thesis is to systematically overview existing and investigate new alternatives to the classical supervised training of RNNs. These

alternatives roughly fall under the umbrella-name of Reservoir Computing. This includes generic RNN structures that can effectively do with only partial training. It also includes investigations in how other principles of learning, in particular unsupervised, can help in effectively building or improving such structures and complement the supervised learning. These alternative methods aim at circumventing the limitations of classical purely supervised training of RNNs and scaling RNNs up for more intelligent tasks. Some initial attempts of doing that, in particular training hierarchies of RNNs, are also presented.

Technically, the contribution of this thesis is two-fold: if offers a comprehensive survey of Reservoir Computing and investigates a novel type of network used as a reservoir, as well as its unsupervised training and layered hierarchies.

The thesis is organized in the following way. First, in Chapter 2 we introduce the concepts of Reservoir Computing and survey all the more important different RNN training and adaptation techniques under this umbrella-name. This review offers a natural conceptual classification of the techniques transcending the boundaries of the "brandnames" of reservoir computing methods, and thus aims to help in unifying the field and providing the reader with a detailed "map" of it. This chapter is a substantially updated version of a review which we previously published in [Lukoševičius and Jaeger, 2009].

In the connecting Chapter 3 we further motivate the importance of unsupervised learning and hierarchies.

In Chapter 4 we investigate the use of two different neural network models for the reservoir together with several unsupervised adaptation techniques and unsupervisedly layer-wise trained hierarchies of such models. We rigorously empirically test the proposed methods on two temporal pattern recognition datasets, comparing it to classical reservoir computing state of art. Some initial results of this research were published in [Lukoševičius, 2010].

We offer the final remarks and discussion in Chapter 5.

The longer Chapters 2 and 4 have the internal structure explained in their corresponding introducing Sections 2.1 and 4.1.

Chapter 2

Overview of Reservoir Computing

2.1 Introduction

Artificial *recurrent neural networks* (RNNs) represent a large and varied class of computational models that are designed by more or less detailed analogy with biological brain modules. In an RNN numerous abstract *neurons* (also called *units* or *processing elements*) are interconnected by likewise abstracted *synaptic connections* (or *links*), which enable *activations* to propagate through the network. The characteristic feature of RNNs that distinguishes them from the more widely used *feedforward neural networks* (FFNNs) is that the connection topology possesses cycles. The existence of cycles has a profound impact:

- An RNN may develop a self-sustained temporal activation dynamics along its recurrent connection pathways, even in the absence of input. Mathematically, this renders an RNN to be a *dynamical system*, while feedforward networks are *functions*.
- If driven by an input signal, an RNN preserves in its internal state a nonlinear transformation of the input history – in other words, it has a *dynamical memory*, and is able to process temporal context information.

This review concerns a particular subset of RNN-based research in two aspects:

- RNNs are used for a variety of scientific purposes, and at least two major classes of RNN models exist: they can be used for purposes of modeling biological brains, or as engineering tools for technical applications. The first usage belongs to the field of computational neuroscience, while the second

frames RNNs in the realms of machine learning, the theory of computation, and nonlinear signal processing and control. While there are interesting connections between the two attitudes, this survey focuses on the latter, with occasional borrowings from the first.

- From a dynamical systems perspective, there are two main classes of RNNs. Models from the first class are characterized by an energy-minimizing stochastic dynamics and symmetric connections. The best known instantiations are Hopfield networks [Hopfield, 2007, 1982], Boltzmann machines [Hinton, 2007; Ackley et al., 1985], and the recently emerging Deep Belief Networks [Hinton and Salakhutdinov, 2006]. These networks are mostly trained in some unsupervised learning scheme. Typical targeted network functionalities in this field are associative memories, data compression, the unsupervised modeling of data distributions, and static pattern classification, where the model is run for multiple time steps per single input instance to reach some type of convergence or equilibrium (but see e.g., [Taylor et al., 2007] for extension to temporal data). The mathematical background is rooted in statistical physics. In contrast, the second big class of RNN models typically features a deterministic update dynamics and directed connections. Systems from this class implement nonlinear filters, which transform an input time series into an output time series. The mathematical background here is nonlinear dynamical systems. The standard training mode is supervised. This survey is concerned only with RNNs of this second type, and when we speak of *RNNs* later on, we will exclusively refer to such systems.[1]

RNNs (of the second type) appear as highly promising and fascinating tools for nonlinear time series processing applications, mainly for two reasons. First, it can be shown that under fairly mild and general assumptions, such RNNs are universal approximators of dynamical systems [Funahashi and Nakamura, 1993]. Second, biological brain modules almost universally exhibit recurrent connection pathways too. Both observations indicate that RNNs should potentially be powerful tools for engineering applications.

Despite this widely acknowledged potential, and despite a number of successful academic and practical applications, the impact of RNNs in nonlinear modeling has remained limited for a long time. The main reason for this lies in the fact

[1] However, they can also be used in a converging mode, as shown at the end of Section 2.8.6.

that RNNs are difficult to train by gradient-descent-based methods, which aim at iteratively reducing the training error. While a number of training algorithms have been proposed (a brief overview in Section 2.2.5), these all suffer from the following shortcomings:

- The gradual change of network parameters during learning drives the network dynamics through bifurcations [Doya, 1992]. At such points, the gradient information degenerates and may become ill-defined. As a consequence, convergence cannot be guaranteed.
- A single parameter update can be computationally expensive, and many update cycles may be necessary. This results in long training times, and renders RNN training feasible only for relatively small networks (in the order of tens of units).
- It is intrinsically hard to learn dependences requiring long-range memory, because the necessary gradient information exponentially dissolves over time [Bengio et al., 1994] (but see the Long Short-Term Memory networks [Gers et al., 2000] for a possible escape).
- Advanced training algorithms are mathematically involved and need to be parameterized by a number of global control parameters, which are not easily optimized. As a result, such algorithms need substantial skill and experience to be successfully applied.

In this situation of slow and difficult progress, in 2001 a fundamentally new approach to RNN design and training was proposed independently by Wolfgang Maass under the name of *Liquid State Machines* [Maass et al., 2002] and by Herbert Jaeger under the name of *Echo State Networks* [Jaeger, 2001]. This approach, which had predecessors in computational neuroscience [Dominey, 1995] and subsequent ramifications in machine learning as the *Backpropagation-Decorrelation* [Steil, 2004] learning rule, is now increasingly often collectively referred to as *Reservoir Computing* (RC) [Verstraeten et al., 2007a]. The RC paradigm avoids the shortcomings of gradient-descent RNN training listed above, by setting up RNNs in the following way:

- A recurrent neural network is *randomly* created and remains unchanged during training. This RNN is called the *reservoir*. It is passively excited by

the input signal and maintains in its state a nonlinear transformation of the input history.

- The desired output signal is generated as a linear combination of the neuron's signals from the input-excited reservoir. This linear combination is obtained by linear regression, using the teacher signal as a target.

Figure 2.1 graphically contrasts previous methods of RNN training with the RC approach.

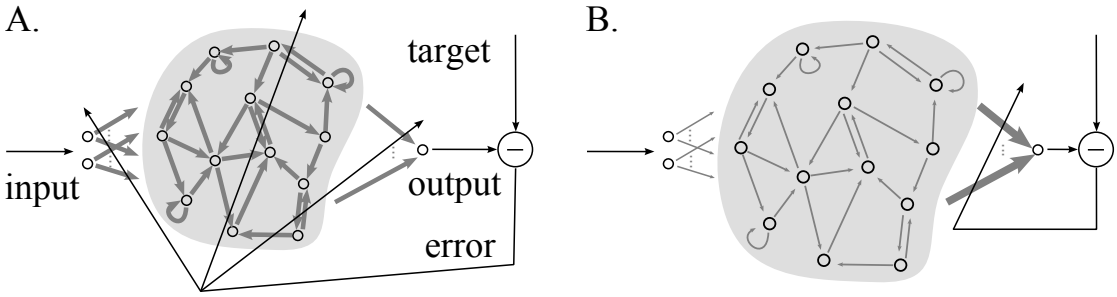


Figure 2.1: A. Traditional gradient-descent-based RNN training methods adapt all connection weights (bold arrows), including input-to-RNN, RNN-internal, and RNN-to-output weights. B. In Reservoir Computing, only the RNN-to-output weights are adapted.

Reservoir Computing methods have quickly become popular, as witnessed for instance by a theme issue of Neural Networks [Jaeger et al., 2007b], and today constitute one of the basic paradigms of RNN modeling [Jaeger, 2007b]. The main reasons for this development are the following:

Modeling accuracy. RC has starkly outperformed previous methods of nonlinear system identification, prediction and classification, for instance in predicting chaotic dynamics (three orders of magnitude improved accuracy [Jaeger and Haas, 2004]), nonlinear wireless channel equalization (two orders of magnitude improvement [Jaeger and Haas, 2004]), the Japanese Vowel benchmark (zero test error rate, previous best: 1.8% [Jaeger et al., 2007a]), financial forecasting (winner of the international forecasting competition NN3[2]), and in isolated spoken digits recognition (improvement of word error rate on benchmark from 0.6% of previous best system to 0.2%

[2] http://www.neural-forecasting-competition.com/NN3/index.htm

[Verstraeten et al., 2006], and further to 0% test error in recent unpublished work).

Modeling capacity. RC is computationally universal for continuous-time, continuous-value real-time systems modeled with bounded resources (including time and value resolution) [Maass et al., 2003, 2006].

Biological plausibility. Numerous connections of RC principles to architectural and dynamical properties of mammalian brains have been established. RC (or closely related models) provides explanations of why biological brains can carry out accurate computations with an "inaccurate" and noisy physical substrate [Buonomano and Merzenich, 1995; Haeusler and Maass, 2007], especially accurate timing [Karmarkar and Buonomano, 2007]; of the way in which visual information is superimposed and processed in primary visual cortex [Stanley et al., 1999; Nikolić et al., 2007]; of how cortico-basal pathways support the representation of sequential information; and RC offers a functional interpretation of the cerebellar circuitry [Kistler and Zeeuw, 2002; Yamazaki and Tanaka, 2007]. A central role is assigned to an RC circuit in a series of models explaining sequential information processing in human and primate brains, most importantly of speech signals [Dominey, 1995; Dominey et al., 2003; Blanc and Dominey, 2003; Dominey et al., 2006].

Extensibility and parsimony. A notorious conundrum of neural network research is how to extend previously learned models by new items without impairing or destroying previously learned representations (*catastrophic interference* [French, 2003]). RC offers a simple and principled solution: new items are represented by new output units, which are appended to the previously established output units of a given reservoir. Since the output weights of different output units are independent of each other, catastrophic interference is a non-issue.

These encouraging observations should not mask the fact that RC is still in its infancy, and significant further improvements and extensions are desirable. Specifically, just simply creating a reservoir at random is unsatisfactory. It seems obvious that when addressing a specific modeling task, a specific reservoir design that is adapted to the task will lead to better results than a naive random creation. Thus, the main stream of research in the field is today directed at understanding

the effects of reservoir characteristics on task performance, and at developing suitable reservoir design and adaptation methods. Also, new ways of reading out from the reservoirs, including combining them into larger structures, are devised and investigated. While shifting from the initial idea of having a fixed randomly created reservoir and training only the readout, the current paradigm of reservoir computing remains (and differentiates itself from other RNN training approaches) as producing/training the reservoir and the readout separately and differently.

This review offers a conceptual classification and a comprehensive survey of this research.

As is true for many areas of machine learning, methods in reservoir computing converge from different fields and come with different names. We would like to make a distinction here between these differently named "tradition lines", which we like to call *brands*, and the actual finer-grained ideas on producing good reservoirs, which we will call *recipes*. Since recipes can be useful and mixed across different brands, this review focuses on classifying and surveying them. To be fair, it has to be said that we associate themselves mostly with the Echo State Networks brand, and thus, willingly or not, are influenced by its mindset.

Overview of the chapter. We start by introducing a generic notational framework in Section 2.2. More specifically, we define what we mean by *problem* or *task* in the context of machine learning in Section 2.2.1. Then we define a general notation for expansion (or kernel) methods for both non-temporal (Section 2.2.2) and temporal (Section 2.2.3) tasks, introduce our notation for recurrent neural networks in Section 2.2.4, and outline classical training methods in Section 2.2.5. In Section 2.3 we detail the foundations of Reservoir Computing and proceed by naming the most prominent brands. In Section 2.4 we introduce our classification of the reservoir generation/adaptation recipes, which transcends the boundaries between the brands. Following this classification we then review universal (Section 2.5), unsupervised (Section 2.6), and supervised (Section 2.7) reservoir generation/adaptation recipes. In Section 2.8 we provide a classification and review the techniques for reading the outputs from the reservoirs reported in literature, together with discussing various practical issues of readout training. A final discussion (Section 2.10) wraps up the entire picture.

2.2 Formalism

2.2.1 Formulation of the problem

Let a *problem* or a *task* in our context of machine learning (ML) be defined as a problem of learning a functional relation between a given input $\mathbf{u}(n) \in \mathbb{R}^{N_\mathrm{u}}$ and a desired output $\mathbf{y}_\mathrm{target}(n) \in \mathbb{R}^{N_\mathrm{y}}$, where $n = 1, \ldots, T$, and T is the number of data points in the training *dataset* $\{(\mathbf{u}(n), \mathbf{y}_\mathrm{target}(n))\}$. This setup where exact required outputs $\mathbf{y}_\mathrm{target}(n)$ are known for the training data is called *supervised* ML. A *non-temporal* task is where the data points are independent of each other and the goal is to learn a function $\mathbf{y}(n) = y(\mathbf{u}(n))$ such that $E(\mathbf{y}, \mathbf{y}_\mathrm{target})$ is minimized, where E is an error measure, for instance, the normalized root-mean-square error (NRMSE)

$$E(\mathbf{y}, \mathbf{y}_\mathrm{target}) = \sqrt{\frac{\left\langle \|\mathbf{y}(n) - \mathbf{y}_\mathrm{target}(n)\|^2 \right\rangle}{\left\langle \|\mathbf{y}_\mathrm{target}(n) - \langle \mathbf{y}_\mathrm{target}(n) \rangle\|^2 \right\rangle}}, \tag{2.1}$$

where $\|\cdot\|$ stands for the Euclidean distance (or norm) and $\langle \cdot \rangle$ for mean.

A *temporal* task is where $\mathbf{u}$ and $\mathbf{y}_\mathrm{target}$ are signals in a discrete time domain $n = 1, \ldots, T$, and the goal is to learn a function $\mathbf{y}(n) = y(\ldots, \mathbf{u}(n-1), \mathbf{u}(n))$ such that $E(\mathbf{y}, \mathbf{y}_\mathrm{target})$ is minimized. Thus the difference between the temporal and non-temporal task is that the function $y(\cdot)$ we are trying to learn has memory in the first case and is memoryless in the second. In both cases the underlying assumption is, of course, that the functional dependence we are trying to learn actually exists in the data. For the temporal case this spells out as data adhering to an additive noise model of the form $\mathbf{y}_\mathrm{target}(n) = y_\mathrm{target}(\ldots, \mathbf{u}(n-1), \mathbf{u}(n)) + \boldsymbol{\theta}(n)$, where $y_\mathrm{target}(\cdot)$ is the relation to be learned by $y(\cdot)$ and $\boldsymbol{\theta}(n) \in \mathbb{R}^{N_\mathrm{y}}$ is a noise term, limiting the learning precision, i.e., the precision of matching the learned $\mathbf{y}(n)$ to $\mathbf{y}_\mathrm{target}(n)$.

Whenever we say that the task or the problem is learned *well*, or with good *accuracy* or *precision*, we mean that $E(\mathbf{y}, \mathbf{y}_\mathrm{target})$ is small. Normally one part of the T data points is used for training the model and another part (unseen during the training) for testing it. When speaking about output errors and *performance* or *precision* we will have *testing* errors in mind (if not explicitly specified otherwise). Also n, denoting the discrete time, will often be used omitting its range $1, \ldots, T$.

2.2.2 Expansions and kernels in non-temporal tasks

Many tasks cannot be accurately solved by a simple linear relation between the $\mathbf{u}$ and $\mathbf{y}_{\text{target}}$, i.e., a linear model $\mathbf{y}(n) = \mathbf{W}\mathbf{u}(n)$ (where $\mathbf{W} \in \mathbb{R}^{N_\text{y} \times N_\text{u}}$) gives big errors $E(\mathbf{y}, \mathbf{y}_{\text{target}})$ regardless of $\mathbf{W}$. In such situations one has to resort to *nonlinear* models. A number of generic and widely used approaches to nonlinear modeling are based on the idea of nonlinearly expanding the input $\mathbf{u}(n)$ into a high-dimensional feature vector $\mathbf{x}(n) \in \mathbb{R}^{N_\text{x}}$, and then utilizing those features using linear methods, for instance by linear regression or computing for a linear separation hyperplane, to get a reasonable $\mathbf{y}$. Solutions of this kind can be expressed in the form

$$\mathbf{y}(n) = \mathbf{W}^{\text{out}}\mathbf{x}(n) = \mathbf{W}^{\text{out}}x(\mathbf{u}(n)), \tag{2.2}$$

where $\mathbf{W}^{\text{out}} \in \mathbb{R}^{N_\text{y} \times N_\text{x}}$ are the trained output weights. Typically $N_\text{x} \gg N_\text{u}$, and we will often consider $\mathbf{u}(n)$ as included in $\mathbf{x}(n)$. There is also typically a constant *bias* value added to (2.2), which is omitted here and in other equations for brevity. The bias can be easily implemented, having one of the features in $\mathbf{x}(n)$ constant (e.g., $= 1$) and a corresponding column in $\mathbf{W}^{\text{out}}$. Some models extend (2.2) to

$$\mathbf{y}(n) = f_{\text{out}}\left(\mathbf{W}^{\text{out}}x[\mathbf{u}(n)]\right), \tag{2.3}$$

where $f_{\text{out}}(\cdot)$ is some nonlinear function (e.g., a sigmoid applied element-wise). For the sake of simplicity we will consider this definition as equivalent to (2.2), since $f_{\text{out}}(\cdot)$ can be eliminated from $\mathbf{y}$ by redefining the target as $\mathbf{y}'_{\text{target}} = {f_{\text{out}}}^{-1}(\mathbf{y}_{\text{target}})$ (and the error function $E(\mathbf{y}, \mathbf{y}'_{\text{target}})$, if desired). Note that (2.2) is a special case of (2.3), with $f_{\text{out}}(\cdot)$ being the identity.

Functions $x(\mathbf{u}(n))$ that transform an input $\mathbf{u}(n)$ into a (higher-dimensional) vector $\mathbf{x}(n)$ are often called *kernels* (and traditionally denoted $\phi(\mathbf{u}(n))$) in this context. Methods using kernels often employ the *kernel trick*, which refers to the option afforded by many kernels of computing inner products in the (high-dimensional, hence expensive) feature space of $\mathbf{x}$ more cheaply in the original space populated by $\mathbf{u}$. The term *kernel function* has acquired a close association with the kernel trick. Since here we will not exploit the kernel trick, in order to avoid confusion we will use the more neutral term of an *expansion* function for $x(\mathbf{u}(n))$, and refer to methods using such functions as *expansion methods*. These methods

then include *Support Vector Machines* (which standardly do use the kernel trick), *Feedforward Neural Networks*, *Radial Basis Function* approximators, *Slow Feature Analysis*, and various *Probability Mixture* models, among many others. Feedforward neural networks are also often referred to as *(multilayer) perceptrons* in the literature.

While training the output $\mathbf{W}^{\text{out}}$ is a well defined and understood problem, producing a good expansion function $x(\cdot)$ generally involves more creativity. In many expansion methods, e.g., Support Vector Machines, the function is chosen "by hand" (most often through trial-and-error) and is fixed.

2.2.3 Expansions in temporal tasks

Many temporal methods are based on the same principle. The difference is that in a temporal task the function to be learned depends also on the history of the input, as discussed in Section 2.2.1. Thus, the expansion function has memory: $\mathbf{x}(n) = x(\ldots, \mathbf{u}(n-1), \mathbf{u}(n))$, i.e., it is an expansion of the current input and its (potentially infinite) history. Since this function has an unbounded number of parameters, practical implementations often take an alternative, recursive, definition:

$$\mathbf{x}(n) = x(\mathbf{x}(n-1), \mathbf{u}(n)). \tag{2.4}$$

The output $\mathbf{y}(n)$ is typically produced in the same way as for non-temporal methods by (2.2) or (2.3).

In addition to the nonlinear expansion, as in the non-temporal tasks, such $\mathbf{x}(n)$ could be seen as a type of a spatial embedding of the temporal information of $\ldots, \mathbf{u}(n-1), \mathbf{u}(n)$. This, for example, enables capturing higher-dimensional dynamical attractors $\mathbf{y}(n) = y_{\text{target}}(\ldots, \mathbf{u}(n-1), \mathbf{u}(n)) = \mathbf{u}(n+1)$ of the system being modeled by $y(\cdot)$ from a series of lower-dimensional observations $\mathbf{u}(n)$ the system is emitting, which is shown to be possible by *Takens's theorem* [Takens, 1981].

2.2.4 Recurrent neural networks

The type of recurrent neural networks that we will consider most of the time in this review is a straightforward implementation of (2.4). The nonlinear expansion

with memory here leads to a *state vector* of the form

$$\mathbf{x}(n) = f\left(\mathbf{W}^{\text{in}}\mathbf{u}(n) + \mathbf{W}\mathbf{x}(n-1)\right), \quad n = 1, \ldots, T, \tag{2.5}$$

where $\mathbf{x}(n) \in \mathbb{R}^{N_\text{x}}$ is a vector of reservoir neuron activations at a time step n, $f(\cdot)$ is the neuron activation function, usually the symmetric $\tanh(\cdot)$, or the positive logistic (or Fermi) sigmoid, applied element-wise, $\mathbf{W}^{\text{in}} \in \mathbb{R}^{N_\text{x} \times N_\text{u}}$ is the input weight matrix and $\mathbf{W} \in \mathbb{R}^{N_\text{x} \times N_\text{x}}$ is a weight matrix of internal network connections. The network is usually started with the initial state $\mathbf{x}(0) = \mathbf{0}$. Bias values are again omitted in (2.5) in the same way as in (2.2). The readout $\mathbf{y}(n)$ of the network is implemented as in (2.3).

Some models of RNNs extend (2.5) as

$$\mathbf{x}(n) = f\left(\mathbf{W}^{\text{in}}\mathbf{u}(n) + \mathbf{W}\mathbf{x}(n-1) + \mathbf{W}^{\text{ofb}}\mathbf{y}(n-1)\right), \quad n = 1, \ldots, T, \tag{2.6}$$

where $\mathbf{W}^{\text{ofb}} \in \mathbb{R}^{N_\text{x} \times N_\text{y}}$ is an optional output feedback weight matrix.

2.2.5 Classical training of RNNs

The classical approach to supervised training of RNNs, known as *gradient descent*, is by iteratively adapting all weights $\mathbf{W}^{\text{out}}$, $\mathbf{W}$, $\mathbf{W}^{\text{in}}$, and possibly $\mathbf{W}^{\text{ofb}}$ (which as a whole we denote $\mathbf{W}^{\text{all}}$ for brevity in this section) according to their estimated gradients $\partial E/\partial \mathbf{W}^{\text{all}}$, in order to minimize the output error $E = E(\mathbf{y}, \mathbf{y}_{\text{target}})$. A classical example of such methods is Real-Time Recurrent Learning (RTRL) [Williams and Zipser, 1989], where the estimation of $\partial E/\partial \mathbf{W}^{\text{all}}$ is done recurrently, forward in time. Conversely, error BackPropagation (BP) methods for training RNNs, which are derived as extensions of the BP method for feedforward neural networks [Rumelhart et al., 1988], estimate $\partial E/\partial \mathbf{W}^{\text{all}}$ by propagating $E(\mathbf{y}, \mathbf{y}_{\text{target}})$ backwards through network connections and time. The BP group of methods is arguably the most prominent in classical RNN training, with the classical example in this group being BackPropagation Through Time (BPTT) [Werbos, 1990]. It has a runtime complexity of $O({N_\text{x}}^2)$ per weight update per time step for a single output $N_\text{y} = 1$, compared to $O({N_\text{x}}^4)$ for RTRL. An even simpler version of BP RNN training where error backpropagation is limited to a single time step was successfully used in [Elman, 1990].

A systematic unifying overview of many classical gradient descent RNN training methods is presented in [Atiya and Parlos, 2000]. The same contribution also proposes a new approach, often referred to by others as Atiya-Parlos Recurrent Learning (APRL). It estimates gradients with respect to neuron activations $\partial E/\partial \mathbf{x}$ (instead of weights directly) and gradually adapts the weights $\mathbf{W}^{\text{all}}$ to move the activations $\mathbf{x}$ into the desired directions. The method is shown to converge faster than previous ones. See Section 2.3.4 for more implications of APRL and bridging the gap between the classical gradient descent and the reservoir computing methods.

There are also other versions of supervised RNN training, formulating the training problem differently, such as using Extended Kalman Filters [Puškorius and Feldkamp, 1994] or the Expectation-Maximization algorithm [Ma and Ji, 1998], as well as dealing with special types of RNNs, such as Long Short-Term Memory [Hochreiter and Schmidhuber, 1997] modular networks capable of learning long-term dependences.

There are many more, arguably less prominent, methods and their modifications for RNN training that are not mentioned here, as this would lead us beyond the scope of this review. The very fact of their multiplicity suggests that there is no clear winner in all aspects. Despite many advances that the methods cited above have introduced, they still have multiple common shortcomings as pointed out in Section 2.1.

2.3 Reservoir methods

Reservoir computing methods differ from the "traditional" designs and learning techniques listed above in that they make a conceptual and computational separation between a dynamic *reservoir* – an RNN as a nonlinear temporal expansion function – and a recurrence-free (usually linear) *readout* that produces the desired output from the expansion.

This separation is based on the understanding (common with kernel methods) that $x(\cdot)$ and $y(\cdot)$ serve different purposes: $x(\cdot)$ expands the input history $\mathbf{u}(n), \mathbf{u}(n-1), \ldots$ into a rich enough reservoir state space $\mathbf{x}(n) \in \mathbb{R}^{N_\mathrm{x}}$, while $y(\cdot)$ combines the neuron signals $\mathbf{x}(n)$ into the desired output signal $\mathbf{y}_{\text{target}}(n)$. In the linear readout case (2.2), for each dimension y_i of $\mathbf{y}$ an output weight vector

$(\mathbf{W}^{\text{out}})_i$ in the same space $\mathbb{R}^{N_\text{x}}$ is found such that

$$(\mathbf{W}^{\text{out}})_i \mathbf{x}(n) = y_i(n) \approx y_{\text{target}_i}(n), \tag{2.7}$$

while the "purpose" of $\mathbf{x}(n)$ is to contain a rich enough representation to make this possible.

Since the expansion and the readout serve different purposes, training / generating them separately and even with different goal functions makes sense. The readout $\mathbf{y}(n) = y(\mathbf{x}(n))$ is essentially a non-temporal function, learning which is relatively simple. On the other hand, setting up the reservoir such that a "good" state expansion $\mathbf{x}(n)$ emerges is an ill-understood challenge in many respects. The "traditional" RNN training methods do not make the conceptual separation of a reservoir vs. a readout, and train both reservoir-internal and output weights in technically the same fashion. Nonetheless, even in traditional methods the ways of defining the error gradients for the output $\mathbf{y}(n)$ and the internal units $\mathbf{x}(n)$ are inevitably different, reflecting that an explicit target $\mathbf{y}_{\text{target}}(n)$ is available only for the output units. Analyses of traditional training algorithms have furthermore revealed that the learning dynamics of internal vs. output weights exhibit systematic and striking differences. This theme will be expanded in Section 2.3.4.

Currently, reservoir computing is a vivid fresh RNN research stream, which has recently gained wide attention due to the reasons pointed out in Section 2.1.

We proceed to review the most prominent "named" reservoir methods, which we call here *brands*. Each of them has its own history, a specific mindset, specific types of reservoirs, and specific insights.

2.3.1 Echo State Networks

Echo State Networks (ESNs) [Jaeger, 2007b] represent one of the two pioneering reservoir computing methods. The approach is based on the observation that if a random RNN possesses certain algebraic properties, training only a linear readout from it is often sufficient to achieve excellent performance in practical applications. The untrained RNN part of an ESN is called a *dynamical reservoir*, and the resulting states $\mathbf{x}(n)$ are termed *echoes* of its input history [Jaeger, 2001] – this is where reservoir computing draws its name from.

ESNs standardly use simple sigmoid neurons, i.e., reservoir states are computed by (2.5) or (2.6), where the nonlinear function $f(\cdot)$ is a sigmoid, usually

the $\tanh(\cdot)$ function. Leaky integrator neuron models represent another frequent option for ESNs, which is discussed in depth in Section 2.5.5. Classical recipes of producing the ESN reservoir (which is in essence $\mathbf{W}^{\text{in}}$ and $\mathbf{W}$) are outlined in Section 2.5.1, together with input-independent properties of the reservoir. Input-dependent measures of the quality of the activations $\mathbf{x}(n)$ in the reservoir are presented in Section 2.6.1.

The readout from the reservoir is usually linear (2.3), where $\mathbf{u}(n)$ is included as part of $\mathbf{x}(n)$, which can also be spelled out in (2.3) explicitly as

$$\mathbf{y}(n) = f_{\text{out}}\left(\mathbf{W}^{\text{out}}[\mathbf{u}(n); \mathbf{x}(n)]\right), \tag{2.8}$$

where $\mathbf{W}^{\text{out}} \in \mathbb{R}^{N_{\text{y}} \times (N_{\text{u}}+N_{\text{x}})}$ is the learned output weight matrix, $f_{\text{out}}(\cdot)$ is the output neuron activation function (usually the identity) applied component-wise, and $[\cdot;\cdot]$ stands for a vertical concatenation of vectors. The original and most popular batch training method to compute $\mathbf{W}^{\text{out}}$ is linear regression, discussed in Section 2.8.1.1, or a computationally cheap online training discussed in Section 2.8.1.2.

The initial ESN publications [Jaeger, 2001, 2002a,b, 2003; Jaeger and Haas, 2004] were framed in settings of machine learning and nonlinear signal processing applications. The original theoretical contributions of early ESN research concerned algebraic properties of the reservoir that make this approach work in the first place (the *echo state property* [Jaeger, 2001] discussed in Section 2.5.1) and analytical results characterizing the dynamical short-term memory capacity of reservoirs [Jaeger, 2002a] discussed in Section 2.6.1.

2.3.2 Liquid State Machines

Liquid State Machines (LSMs) [Maass et al., 2002] are the other pioneering reservoir method, developed independently from and simultaneously with ESNs. LSMs were developed from a computational neuroscience background, aiming at elucidating the principal computational properties of neural microcircuits [Maass et al., 2002, 2003; Natschläger et al., 2002; Maass et al., 2004]. Thus LSMs use more sophisticated and biologically realistic models of spiking integrate-and-fire neurons and dynamic synaptic connection models in the reservoir. The connectivity among the neurons often follows topological and metric constraints that are biologically motivated. In the LSM literature, the reservoir is often referred to as the *liquid*,

following an intuitive metaphor of the excited states as ripples on the surface of a pool of water. Inputs to LSMs also usually consist of spike trains. In their readouts LSMs originally used multilayer feedforward neural networks (of either spiking or sigmoid neurons), or linear readouts similar to ESNs [Maass et al., 2002]. Additional mechanisms for averaging spike trains to get real-valued outputs are often employed.

RNNs of the LSM-type with spiking neurons and more sophisticated synaptic models are usually more difficult to implement, to correctly set up and tune, and typically more expensive to emulate on digital computers[3] than simple ESN-type "weighted sum and nonlinearity" RNNs. Thus they are less widespread for engineering applications of RNNs than the latter. However, while the ESN-type neurons only emulate mean firing rates of biological neurons, spiking neurons are able to perform more complicated information processing, due to the time coding of the information in their signals (i.e., the exact timing of each firing also matters). Also findings on various mechanisms in natural neural circuits are more easily transferable to these more biologically-realistic models (there is more on this in Section 2.6.2).

The main theoretical contributions of the LSM brand to Reservoir Computing consist in analytical characterizations of the computational power of such systems [Maass et al., 2002, 2006] discussed in Sections 2.6.1 and 2.7.4.

2.3.3 Evolino

Evolino [Schmidhuber et al., 2007] transfers the idea of ESNs from an RNN of simple sigmoidal units to a Long Short-Term Memory (LSTM) type of RNNs [Hochreiter and Schmidhuber, 1997] constructed from units capable of preserving memory for long periods of time. In Evolino the weights of the reservoir are trained using evolutionary methods, as is also done in some extensions of ESNs, both discussed in Section 2.7.2.

2.3.4 Backpropagation-Decorrelation

The idea of separation between a reservoir and a readout function has also been arrived at from the point of view of optimizing the performance of the RNN training

[3]With a possible exception of event-driven spiking NN simulations, where the computational load varies depending on the amount of activity in the NN.

algorithms that use error backpropagation, as already indicated in Section 2.2.5. In an analysis of the weight dynamics of an RNN trained using the APRL learning algorithm [Schiller and Steil, 2005], it was revealed that the output weights $\mathbf{W}^{\text{in}}$ of the network being trained change quickly, while the hidden weights $\mathbf{W}$ change slowly and in the case of a single output $N_{\text{y}} = 1$ the changes are column-wise coupled. Thus in effect APRL decouples the RNN into a quickly adapting output and a slowly adapting reservoir. Inspired by these findings a new iterative/online RNN training method, called *BackPropagation-DeCorrelation* (BPDC), was introduced [Steil, 2004]. It approximates and significantly simplifies the APRL method, and applies it only to the output weights $\mathbf{W}^{\text{out}}$, turning it into an online RC method. BPDC uses the reservoir update equation defined in (2.6), where output feedbacks $\mathbf{W}^{\text{ofb}}$ are essential, with the same type of units as ESNs. BPDC learning is claimed to be insensitive to the parameters of fixed reservoir weights $\mathbf{W}$. BPDC boasts fast learning times and thus is capable of tracking quickly changing signals. As a downside of this feature, the trained network quickly forgets the previously seen data and is highly biased by the recent data. Some remedies for reducing this effect are reported in [Steil, 2005b]. Most of applications of BPDC in the literature are for tasks having one-dimensional outputs $N_{\text{y}} = 1$; however BPDC is also successfully applied to $N_{\text{y}} > 1$, as recently demonstrated in [Reinhart and Steil, 2008].

From a conceptual perspective we can define a range of RNN training methods that gradually bridge the gap between the classical BP and reservoir methods:

1. Classical BP methods, such as Backpropagation Through Time (BPTT) [Werbos, 1990];
2. Atiya-Parlos recurrent learning (APRL) [Atiya and Parlos, 2000];
3. BackPropagation-DeCorrelation (BPDC) [Steil, 2004];
4. Echo State Networks (ESNs) [Jaeger, 2007b].

In each method of this list the focus of training gradually moves from the entire network towards the output, and convergence of the training is faster in terms of iterations, with only a single "iteration" in case 4. At the same time the potential expressiveness of the RNN, as per the same number of units in the NN, becomes weaker. All methods in the list primarily use the same type of simple sigmoid neuron model.

2.3.5 FORCE Learning

A slightly similar recent addition to the reservoir computing methods called *First-Order Reduced and Controlled Error* (or FORCE) learning was introduced in [Sussillo and Abbott, 2009]. Just like BPTT, it is an iterative method designed to train RNN of sigmoid units with output feedbacks $\mathbf{W}^{\text{ofb}}$. It uses a powerful Least Mean Squares online training algorithm (mentioned in Section 2.8.1.2) to virtually instantaneously suppress the output errors by intensively adapting $\mathbf{W}^{\text{out}}$ during training. Thus, in contrast to most of the other iterative training methods, the output errors are small from the beginning, but instead the changes to the output weights $\mathbf{W}^{\text{out}}$ are gradually decreased until they settle in a final state. This solves the output feedback problem during the training which is discussed in Section 2.8.2. The close-to-target output $\mathbf{y}(n)$ which is fed-back through $\mathbf{W}^{\text{ofb}}$ is shown to effectively suppress autonomous chaotic activity in the reservoir during the FORCE learning and after.

2.3.6 Temporal Recurrent Networks

This summary of RC brands would be incomplete without a spotlight directed at Peter F. Dominey's decade-long research suite on cortico-striatal circuits in the human brain (e.g., [Dominey, 1995; Dominey et al., 2003, 2006], and many more). Although this research is rooted in empirical cognitive neuroscience and functional neuroanatomy and aims at elucidating complex neural structures rather than theoretical computational principles, it is probably Dominey who first clearly spelled out the RC principle: *"(...) there is no learning in the recurrent connections* [within a subnetwork corresponding to a reservoir]*, only between the State* [i.e., reservoir] *units and the Output units. Second, adaptation is based on a simple associative learning mechanism (...)"* [Dominey and Ramus, 2000]. It is also in this article where Dominey brands the neural reservoir module as a *Temporal Recurrent Network*. The learning algorithm, to which Dominey alludes, can be seen as a version of the Least Mean Squares discussed in Section 2.8.1.2. At other places, Dominey emphasizes the randomness of the connectivity in the reservoir: *"It is worth noting that the simulated recurrent prefrontal network relies on fixed randomized recurrent connections, (...)"* [Dominey, 2005]. Only in early 2008 did Dominey and "computational" RC researchers become aware of each other.

2.3.7 Other (exotic) types of reservoirs

As is clear from the discussion of the different reservoir methods so far, a variety of neuron models can be used for the reservoirs. Using different activation functions inside a single reservoir might also improve the richness of the echo states, as is illustrated, for example, by inserting some neurons with wavelet-shaped activation functions into the reservoir of ESNs [Wang et al., 2006]. A hardware-implementation-friendly version of reservoirs composed of stochastic bitstream neurons was proposed in [Verstraeten et al., 2005a]. A use of another hardware and parallelization friendly type of locally connected neural latices called Cellular Neural Networks [Chua and Yang, 1988] as reservoirs was investigated in [Verstraeten, 2009]. In Chapter 4 of this thesis, and in [Lukoševičius, 2010], an RNN of radial basis function units is proposed and investigated as a reservoir.

In fact the reservoirs do not necessarily need to be neural networks, governed by dynamics similar to (2.5). Other types of high-dimensional dynamical systems that can take an input $\mathbf{u}(n)$ and have an observable state $\mathbf{x}(n)$ (which does not necessarily fully describe the state of the system) can be used as well. In particular this makes the reservoir paradigm suitable for harnessing the computational power of unconventional hardware, such as analog electronics [Schürmann et al., 2005; Schrauwen et al., 2008a], biological neural tissue [Nikolić et al., 2007], optical [Vandoorne et al., 2008], quantum, or physical "computers". The last of these was demonstrated (taking the "reservoir" and "liquid" idea quite literally) by feeding the input via mechanical actuators into a reservoir full of water, recording the state of its surface optically, and successfully training a readout multilayer perceptron on several classification tasks [Fernando and Sojakka, 2003]. An idea of treating a computer-simulated gene regulation network of *Escherichia Coli* bacteria as the reservoir, a sequence of chemical stimuli as an input, and measures of protein levels and mRNAs as an output is explored in [Jones et al., 2007].

2.3.8 Other overviews of reservoir methods

The first experimental comparison of LSM, ESN, and BPDC reservoir methods with different neuron models, even beyond the standard ones used for the respective methods, and different parameter settings is presented in [Verstraeten et al., 2007a]. It is this article where the name "Reservoir Computing" was first proposed. A brief and broad overview of reservoir computing is presented in [Schrauwen et al.,

2007b], with an emphasis on applications and hardware implementations of reservoir methods. The editorial in the "Neural Networks" journal special issue on ESNs and LSMs [Jaeger et al., 2007b] offers a short introduction to the topic and an overview of the articles in the issue (most of which are also surveyed here). An older and much shorter part of this overview, covering only reservoir adaptation techniques, is available as a technical report [Lukoševičius and Jaeger, 2007]. Many different aspects of RC are discussed in David Verstraeten's PhD thesis on this subject [Verstraeten, 2009]. A recent brief summary of the LSM side of the RC research can by found in [Maass, 2011].

2.4 Our classification of reservoir recipes

The successes of applying RC methods to benchmarks (see the listing in Section 2.1) outperforming classical fully trained RNNs do not imply that randomly generated reservoirs are optimal and cannot be improved. In fact, "random" is almost by definition an antonym to "optimal". The results rather indicate the need for some novel methods of training/generating the reservoirs that are very probably not a direct extension of the way the output is trained (as in BP). Thus besides application studies (which are not surveyed here), the bulk of current RC research on reservoir methods is devoted to optimal reservoir design, or reservoir optimization algorithms.

It is worth mentioning at this point that the general "no free lunch" principle in supervised machine learning [Wolpert, 2001] states that there can exist no bias of a model which would universally improve the accuracy of the model for *all* possible problems. In our context this can be translated into a claim that no single type of reservoir can be optimal for all types of problems.

In this review we will try to survey all currently investigated ideas that help producing "good" reservoirs. We will classify those ideas into three major groups based on their universality:

- *Generic* guidelines/methods of producing good reservoirs irrespective of the task (both the input $\mathbf{u}(n)$ and the desired output $\mathbf{y}_{\text{target}}(n)$);
- *Unsupervised* pre-training of the reservoir with respect to the given input $\mathbf{u}(n)$, but not the target $\mathbf{y}_{\text{target}}(n)$;

- *Supervised* pre-training of the reservoir with respect to both the given input $\mathbf{u}(n)$ and the desired output $\mathbf{y}_{\text{target}}(n)$.

These three classes of methods are discussed in the following three sections. Note that many of the methods to some extend transcend the boundaries of these three classes, but will be classified according to their main principle.

2.5 Generic reservoir recipes

The most classical methods of producing reservoirs all fall into this category. All of them generate reservoirs randomly, with topology and weight characteristics depending on some preset parameters. Even though they are not optimized for a particular input $\mathbf{u}(n)$ or target $\mathbf{y}_{\text{target}}(n)$, a good manual selection of the parameters is to some extent task-dependent, complying with the "no free lunch" principle just mentioned.

These methods are outlined in Section 2.3, and include many different computational models of RNNs (and beyond), especially in Section 2.3.7. Here we will elaborate on the more concrete and specific approaches for manually building good reservoirs in the literature, that mostly focus on the conventional NNs with analog units and discrete time of type (2.5).

2.5.1 Classical ESN approach

Some of the most generic guidelines of producing good reservoirs were presented in the papers that introduced ESNs [Jaeger, 2001, 2002b]. Motivated by an intuitive goal of producing a "rich" set of dynamics, the recipe is to generate a **(i)** *big*, **(ii)** *sparsely* and **(iii)** *randomly* connected, reservoir. This means that (i) N_{x} is sufficiently large, with order ranging from tens to thousands, (ii) the weight matrix $\mathbf{W}$ is sparse, with several to 20 per cent of possible connections, and (iii) the weights of the connections are usually generated randomly from a uniform distribution symmetric around the zero value. This design rationale aims at obtaining *many*, due to (i), reservoir activation signals, which are only *loosely coupled*, due to (ii), and *different*, due to (iii).

The input weights $\mathbf{W}^{\text{in}}$ and the optional output feedback weights $\mathbf{W}^{\text{ofb}}$ are usually dense (they can also be sparse like $\mathbf{W}$) and generated randomly from a uniform distribution. The exact scaling of both matrices and an optional shift

of the input (a constant value added to $\mathbf{u}(n)$) are the few other free parameters that one has to choose when "baking" an ESN. The rules of thumb for them are the following. The scaling of $\mathbf{W}^{\text{in}}$ and shifting of the input depends on how much nonlinearity of the processing unit the task needs: if the inputs are close to 0, the tanh neurons tend to operate with activations close to 0, where they are essentially linear, while inputs far from 0 tend to drive them more towards saturation where they exhibit more nonlinearity. The shift of the input may help to overcome undesired consequences of the symmetry around 0 of the tanh neurons with respect to the sign of the signals. Similar effects are produced by scaling the bias inputs to the neurons (i.e., the column of $\mathbf{W}^{\text{in}}$ corresponding to constant input, which often has a different scaling factor than the rest of $\mathbf{W}^{\text{in}}$). The scaling of $\mathbf{W}^{\text{ofb}}$ is in practice limited by a threshold at which the ESN starts to exhibit an unstable behavior, i.e., the output feedback loop starts to amplify (the errors of) the output and thus enters a diverging generative mode. In [Jaeger, 2002b], these and related pieces of advice are given without a formal justification.

An important element for ESNs to work is that the reservoir should have the *echo state property* [Jaeger, 2001]. This condition in essence states that the effect of a previous state $\mathbf{x}(n)$ and a previous input $\mathbf{u}(n)$ on a future state $\mathbf{x}(n+k)$ should vanish gradually as time passes (i.e., $k \to \infty$), and not persist or even get amplified. For most practical purposes, the echo state property is assured if the reservoir weight matrix $\mathbf{W}$ is scaled so that its spectral radius $\rho(\mathbf{W})$ (i.e., the largest absolute eigenvalue) satisfies $\rho(\mathbf{W}) < 1$ [Jaeger, 2001]. The fact that $\rho(\mathbf{W}) < 1$ almost always ensures the echo state property has led to an unfortunate misconception which is expressed in many RC publications, namely, that $\rho(\mathbf{W}) < 1$ amounts to a necessary and sufficient condition for the echo state property. This is wrong. The mathematically correct connection between the spectral radius and the echo state property is that the latter is violated if $\rho(\mathbf{W}) > 1$ *in reservoirs using the* tanh *function as neuron nonlinearity, and for zero input.* Contrary to widespread misconceptions, the echo state property can be obtained even if $\rho(\mathbf{W}) > 1$ for non-zero input (including bias inputs to neurons), e.g., [Ozturk and Príncipe, 2005], or output feedback [Sussillo and Abbott, 2009], and it may be lost even if $\rho(\mathbf{W}) < 1$, although it is hard to construct systems where this occurs (unless $f'(0) > 1$ for the nonlinearity f), and in practice this does not happen.

The optimal value of $\rho(\mathbf{W})$ should be set depending on the profile of memory and nonlinearity that the given task requires. A rule of thumb, likewise discussed

in [Jaeger, 2001], is that $\rho(\mathbf{W})$ should be close to 1 for tasks that require long memory and accordingly smaller for the tasks where a too long memory might in fact be harmful. Larger $\rho(\mathbf{W})$ also have the effect of driving signals $\mathbf{x}(n)$ into more nonlinear regions of tanh units (further from 0) similarly to $\mathbf{W}^{\text{in}}$. Thus scalings of both $\mathbf{W}^{\text{in}}$ and $\mathbf{W}$ have a similar effect on nonlinearity of the ESN, with a difference that scaling up $\mathbf{W}$ makes reservoir unstable, while their difference determines the effect of current versus past inputs on the current state. An empirical study on how the reservoir parameters affect its stability and nonlinearity is presented in [Verstraeten et al., 2010].

A rather conservative rigorous *sufficient* condition of the echo state property for any kind of inputs $\mathbf{u}(n)$ (including zero) and states $\mathbf{x}(n)$ (with tanh nonlinearity) being $\sigma_{\max}(\mathbf{W}) < 1$, where $\sigma_{\max}(\mathbf{W})$ is the largest singular value of $\mathbf{W}$, was proved in [Jaeger, 2001]. Recently, a less restrictive sufficient condition, namely, $\inf_{\mathbf{D}\in\mathcal{D}} \sigma_{\max}(\mathbf{D}\mathbf{W}\mathbf{D}^{-1}) < 1$, where $\mathbf{D}$ is an arbitrary matrix, minimizing the so-called $\mathbf{D}$-norm $\sigma_{\max}(\mathbf{D}\mathbf{W}\mathbf{D}^{-1})$, from a set $\mathcal{D} \subset \mathbb{R}^{N_{\mathrm{x}}\times N_{\mathrm{x}}}$ of diagonal matrices, has been derived in [Buehner and Young, 2006]. This sufficient condition approaches the necessary $\inf_{\mathbf{D}\in\mathcal{D}} \sigma_{\max}(\mathbf{D}\mathbf{W}\mathbf{D}^{-1}) \to \rho(\mathbf{W})^{-}$, $\rho(\mathbf{W}) < 1$, e.g., when $\mathbf{W}$ is a normal or a triangular (permuted) matrix. A rigorous sufficient condition for the echo state property is rarely ensured in practice, with a possible exception being critical control tasks, where provable stability under any conditions is required.

2.5.2 Different topologies of the reservoir

There have been attempts to find topologies of the ESN reservoir different from sparsely randomly connected ones. Specifically, small-world [Watts and Strogatz, 1998], scale-free [Barabasi and Albert, 1999], and biologically inspired connection topologies generated by spatial growth [Kaiser and Hilgetag, 2004] were tested for this purpose in a careful study [Liebald, 2004], which we point out here due to its relevance although it was obtained only as a BSc thesis. The NRMS error (2.1) of $\mathbf{y}(n)$ as well as the eigenvalue spread of the cross-correlation matrix of the activations $\mathbf{x}(n)$ (necessary for a fast online learning described in Section 2.8.1.2; see Section 2.6.1 for details) were used as the performance measures of the topologies. This work also explored an exhaustive brute-force search of topologies of tiny networks (motifs) of four units, and then combining successful motives (in terms of the eigenvalue spread) into larger networks. The investigation, unfortunately,

concludes that *"(. . .) none of the investigated network topologies was able to perform significantly better than simple random networks, both in terms of eigenvalue spread as well as testing error"* [Liebald, 2004]. This, however, does not serve as a proof that similar approaches are futile. An indication of this is the substantial variation in ESN performance observed among randomly created reservoirs, which is, naturally, more pronounced in smaller reservoirs (e.g., [Jiang et al., 2008b]).

In contrast, LSMs often use a biologically plausible connectivity structure and weight settings. In the original form they model a single cortical microcolumn [Maass et al., 2002]. Since the model of both the connections and the neurons themselves is quite sophisticated, it has a large number of free parameters to be set, which is done manually, guided by biologically observed parameter ranges, e.g., as found in the rat somatosensory cortex [Maass et al., 2004]. This type of model also delivers good performance for practical applications of speech recognition [Maass et al., 2004], [Verstraeten et al., 2005b] (and many similar publications by the latter authors). Since LSMs aim at accuracy of modeling natural neural structures, less biologically plausible connectivity patterns are usually not explored.

It has been demonstrated that much more detailed biological neural circuit models, which use anatomical and neurophysiological data-based laminar (i.e., cortical layer) connectivity structures and Hodgkin-Huxley model neurons, improve the information-processing capabilities of the models [Haeusler and Maass, 2007]. Such highly realistic (for present-day standards) models *"perform significantly better than control circuits (which are lacking the laminar structures but are otherwise identical with regard to their components and overall connection statistics) for a wide variety of fundamental information-processing tasks"* [Haeusler and Maass, 2007].

Different from this direction of research, there are also explorations of using even simpler topologies of the reservoir than the classical ESN. It has been demonstrated that the reservoir can even be an unstructured feed-forward network with time-delayed connections if the finite limited memory window that it offers is sufficient for the task at hand [Čerňanský and Makula, 2005]. This, or an even simpler chain of neurons connected effectively into a delay line, gives results somewhat comparable to normal ESNs in several tasks, especially for online-trained outputs [Čerňanský and Tiňo, 2008]. The delay line, a bi-directionally connected line, or a ring topology, with weights deterministically selected from much more restricted distributions were shown to be comparable to regular ESNs in many respects,

having an additional benefit of being easier to analyze [Rodan and Tiňo, 2011]. A degenerate case of a "reservoir" composed of linear units and a diagonalized $\mathbf{W}$ (only self recurrence) and unitary inputs $\mathbf{W}^{\text{in}}$ was considered in [Fette and Eggert, 2005]. A one-dimensional lattice (ring) topology was used for a reservoir, together with an adaptation of the reservoir discussed in Section 2.6.2, in [Verstraeten et al., 2007b]. A special kind of excitatory and inhibitory neurons connected in a one-dimensional spatial arrangement was shown to produce interesting chaotic behavior in [Lourenço, 2006].

A tendency that higher ranks of the connectivity matrix $\mathbf{W}^{\text{mask}}$ (where ${w^{\text{mask}}}_{i,j} = 1$ if $w_{i,j} \neq 0$, and $= 0$ otherwise, for $i, j = 1, \ldots, N_{\text{x}}$) correlate with lower ESN output errors was observed in [Bush and Tsendjav, 2005]. Connectivity patterns of $\mathbf{W}$ such that $\mathbf{W}^{\infty} \equiv \lim_{k\to\infty} \mathbf{W}^k$ ($\mathbf{W}^k$ standing for "$\mathbf{W}$ to the power k" and approximating weights of the cumulative indirect connections by paths of length k among the reservoir units) is neither fully connected, nor all-zero, are claimed to give a broader distribution of ESN prediction performances, thus including best performing reservoirs, than random sparse connectivities in [Hajnal and Lőrincz, 2006]. A permutation matrix with a medium number and different lengths of connected cycles, or a general orthogonal matrix, are suggested as candidates for such $\mathbf{W}$s.

Reservoirs with orthogonal $\mathbf{W}$ were also shown to posses a better memory capacity measure in [White et al., 2004; Hermans and Schrauwen, 2010a], but all normal $\mathbf{W}$ a bad different measure in [Ganguli et al., 2008], details in Section 2.6.1.1.

2.5.3 Modular reservoirs

One of the shortcomings of conventional ESN reservoirs is that even though they are sparse, the activations are still coupled so strongly that the ESN is poor in dealing with different time scales simultaneously, e.g., predicting several superimposed generators. This problem was successfully tackled by dividing the reservoir into decoupled sub-reservoirs and introducing inhibitory connections among all the sub-reservoirs [Xue et al., 2007]. For the approach to be effective, the inhibitory connections must predict the activations of the sub-reservoirs one time step ahead. To achieve this the inhibitory connections are heuristically computed from (the rest of) $\mathbf{W}$ and $\mathbf{W}^{\text{ofb}}$, or the sub-reservoirs are updated in a sequence and the real

activations of the already updated sub-reservoirs are used.

The Evolino approach introduced in Section 2.3.3 can also be classified as belonging to this group, as the LSTM RNN used for its reservoir consists of specific small memory-holding modules (which could alternatively be regarded as more complicated units of the network).

Approaches relying on combining outputs from several separate reservoirs will be discussed in Section 2.9.

2.5.4 Time-delayed vs. instantaneous connections

Another time-related limitation of the classical ESNs pointed out in [Lukoševičius, 2007] is that no matter how many neurons are contained in the reservoir, it (like any other fully recurrent network with all connections having a time delay) has only a single layer of neurons (Figure 2.2). This makes it intrinsically unsuitable for some types of problems. Consider a problem where the mapping from $\mathbf{u}(n)$ to $\mathbf{y}_{\text{target}}(n)$ is a very complex, nonlinear one, and the data in neighboring time steps are almost independent (i.e., little memory is required), as e.g., the "meta-learning" task in [Prokhorov et al., 2002] [4]. Consider a single time step n: signals from the input $\mathbf{u}(n)$ propagate only through one untrained layer of weights $\mathbf{W}^{\text{in}}$, through the nonlinearity f influence the activations $\mathbf{x}(n)$, and reach the output $\mathbf{y}(n)$ through the trained weights $\mathbf{W}^{\text{out}}$ (Figure 2.2). Thus ESNs are not capable of producing a very complex *instantaneous* mapping from $\mathbf{u}(n)$ to $\mathbf{y}(n)$ using a realistic number of neurons, which could (only) be effectively done by a multilayer FFNN (not counting some non-NN-based methods). Delaying the target $\mathbf{y}_{\text{target}}$ by k time steps would in fact make the signals coming from $\mathbf{u}(n)$ "cross" the non-linearities $k+1$ times before reaching $\mathbf{y}(n+k)$, but would mix the information from different time steps in $\mathbf{x}(n), \ldots, \mathbf{x}(n+k)$, breaking the required virtually independent mapping $\mathbf{u}(n) \rightarrow \mathbf{y}_{\text{target}}(n+k)$, if no special structure of $\mathbf{W}$ is imposed.

As a possible remedy Layered ESNs were introduced in [Lukoševičius, 2007], where a part (up to almost half) of the reservoir connections can be instantaneous and the rest take one time step for the signals to propagate as in normal ESNs. Randomly generated Layered ESNs, however, do not offer a consistent improve-

[4] ESNs have been shown to perform well in a (significantly) simpler version of the "meta-learning" in [Oubbati et al., 2005].

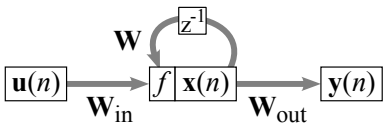


Figure 2.2: Signal flow diagram of the standard ESN.

ment for large classes of tasks, and pre-training methods of such reservoirs have not yet been investigated.

The issue of standard ESNs not having enough trained layers is also discussed and addressed in a broader context in Section 2.9.

2.5.5 Leaky integrator neurons and speed of dynamics

In addition to the basic sigmoid units, leaky integrator (LI) neurons were suggested to be used in ESNs from the point of their introduction [Jaeger, 2001]. This type of neuron performs a leaky integration of its activation from previous time steps. Today a number of versions of leaky integrator neurons are often used in ESNs, which we will call here *leaky integrator ESNs* (LI-ESNs) where the distinction is needed. The main two groups are those using leaky integration before application of the activation function $f(\cdot)$, and after. One example of the latter (in the discretized time case) has reservoir dynamics governed by

$$\mathbf{x}(n) = (1 - a\Delta t)\mathbf{x}(n-1) + \Delta t f(\mathbf{W}^{\text{in}}\mathbf{u}(n) + \mathbf{W}\mathbf{x}(n-1)), \tag{2.9}$$

where Δt is a compound time gap between two consecutive time steps divided by the time constant of the system and a is the decay (or leakage) rate [Lukoševičius et al., 2006]. Another popular (and we believe, preferable) design can be seen as setting $a = 1$ and redefining Δt in (2.9) as the leaking rate a to control the "speed" of the dynamics,

$$\mathbf{x}(n) = (1 - a)\mathbf{x}(n-1) + a f(\mathbf{W}^{\text{in}}\mathbf{u}(n) + \mathbf{W}\mathbf{x}(n-1)), \tag{2.10}$$

which in effect is an exponential moving average, has only one additional parameter and the desirable property that neuron activations $\mathbf{x}(n)$ never go outside the boundaries defined by $f(\cdot)$. Note that the simple ESN (2.5) is a special case of LI-ESNs (2.9) or (2.10) with $a = 1$ and $\Delta t = 1$. As a corollary, an LI-ESN with

a good choice of the parameters can always perform *at least* as well as a corresponding simple ESN. With the introduction of the new parameter a (and Δt), the condition for the echo state property is redefined [Jaeger, 2001]. A natural constraint on the two new parameters is $a\Delta t \in [0, 1]$ in (2.9), and $a \in [0, 1]$ in (2.10) – a neuron should neither retain, nor leak, more activation than it had. An investigation of stability of LI-ESNs, their applications, and a scheme to optimize their global parameters through gradient descent was presented in [Jaeger et al., 2007a]. Applying the leaky integration in different places of the model or resampling the signals as an alternative and their effects on a speech recognition task was investigated in [Schrauwen et al., 2007a]. The speed of dynamics of a reservoir can be adapted online, for example to deal with time-warping of the input [Lukoševičius et al., 2006; Jaeger et al., 2007a].

The additional parameters of the LI-ESN control the "speed" of the reservoir dynamics. Small values of a and Δt result in reservoirs that react slowly to the input. By changing these parameters it is possible to shift the effective interval of frequencies in which the reservoir is working. See Section 2.6.4 for a method of dealing with time-warping in input implemented along these lines.

From a signal processing point of view, the exponential moving average on the neuron activation (2.10) does a simple *low-pass* filtering of its activations with the cutoff frequency

$$f_c = \frac{a}{2\pi(1-a)\Delta t}, \tag{2.11}$$

where Δt is the discretization time step. This makes the neurons average out the frequencies above f_c and enables tuning the reservoirs for particular frequencies. Elaborating further on this idea, *high-pass* neurons, that produce their activations by subtracting from the unfiltered activation (2.5) the low-pass filtered one (2.10), and *band-pass* neurons, that combine the low-pass and high-pass ones, were introduced [Siewert and Wustlich, 2007]. The authors also suggested mixing neurons with different pass-bands inside a single ESN reservoir, and reported that a single reservoir of such kind is able to predict/generate signals having structure on different timescales. A variational Bayesian method for training it is presented in [Zechner and Shutin, 2010].

Following this line of thought, Infinite Impulse Response (IIR) band-pass filters having sharper cutoff characteristics were tried on neuron activations in ESNs with success in several types of signals [Holzmann and Hauser, 2010]. Since the

filters often introduce an undesired phase shift to the signals, a time delay for the activation of each neuron was learned and applied before the linear readout from the reservoir. A successful application of Butterworth band-pass filters in ESNs is reported in [wyffels et al., 2008b].

Connections between neurons that have different time delays (more than one time step) can actually also be used inside the recurrent part, which enables the network to operate on different timescales simultaneously and learn longer-term dependences [Hihi and Bengio, 1996]. Similarly, neurons proposed in [Sutskever and Hinton, 2010] have exponentially decaying connections to previous states of input and reservoir activations. They in effect act as leaky integration of the neuron activation before applying the nonlinearity $f(\cdot)$. These two ideas have been tried for RNNs trained by error backpropagation, but could also be useful for multi-timescale reservoirs. Long-term dependences can also be learned using the reservoirs mentioned in Section 2.3.3.

2.6 Unsupervised reservoir adaptation

In this section we describe reservoir training/generation methods that try to optimize some measure defined on the activations $\mathbf{x}(n)$ of the reservoir, for a given input $\mathbf{u}(n)$, but regardless of the desired output $\mathbf{y}_{\text{target}}(n)$. We see this as a very important research direction, which will will further motivate in Chapter 3 of this thesis and introduce some new contributions in Chapter 4. This section is organized in the following way. In Section 2.6.1 we survey measures that are used to estimate the quality of the reservoir, irrespective of the methods optimizing them. Then local, Section 2.6.2, and global, Section 2.6.3, unsupervised reservoir training methods are surveyed.

2.6.1 "Goodness" measures of the reservoir activations

The classical feature that reservoirs should possess is the echo state property, defined in Section 2.5.1. Even though this property depends on the concrete input $\mathbf{u}(n)$, usually in practice its existence is not measured explicitly, and only the spectral radius $\rho(\mathbf{W})$ is selected to be < 1 irrespective of $\mathbf{u}(n)$, or just tuned for the final performance.

The two necessary and sufficient conditions for LSMs to work were introduced

in [Maass et al., 2002]. A *separation property* measures the distance between different states $\mathbf{x}$ caused by different input sequences $\mathbf{u}$. The measure is refined for binary ESN-type reservoirs in [Bertschinger and Natschläger, 2004] with a generalization in [Schrauwen et al., 2009]. An *approximation property* measures the capability of the readout to produce a desired output $\mathbf{y}_{\text{target}}$ from $\mathbf{x}$, and thus is not an unsupervised measure, but is included here for completeness.

Methods for estimating the computational power and generalization capability of neural reservoirs were presented in [Maass et al., 2005]. The proposed measure for computational power, or *kernel quality*, is obtained in the following way. Take k different input sequences (or segments of the same signal) $\mathbf{u}^i(n)$, where $i = 1, \ldots, k$, and $n = 1, \ldots, T_k$. For each input i take the resulting reservoir state $\mathbf{x}^i(n_0)$, and collect them into a matrix $\mathbf{M} \in \mathbb{R}^{k \times N_x}$, where n_0 is some fixed time after the appearance of $\mathbf{u}^i(n)$ in the input. Then the rank r of the matrix $\mathbf{M}$ is the measure. If $r = k$, this means that all the presented inputs can be separated by a linear readout from the reservoir, and thus the reservoir is said to have a *linear separation property*. For estimating the generalization capability of the reservoir, the same procedure can be performed with s ($s \gg k$) inputs $\mathbf{u}^j(n)$, $j = 1, \ldots, s$, that represent the set of all possible inputs. If the resultant rank r is substantially smaller than the size s of the training set, the reservoir generalizes well. These two measures are more targeted to tasks of time series classification, but can also be revealing in predicting the performance of regression [Legenstein and Maass, 2007b].

A much-desired measure to minimize is the eigenvalue spread (EVS, the ratio of the maximal eigenvalue to the minimal eigenvalue) of the cross-correlation matrix of the activations $\mathbf{x}(n)$. A small EVS is necessary for an online training of the ESN output by a computationally cheap and stable stochastic gradient descent algorithm outlined in Section 2.8.1.2 (see, e.g., [Farhang-Boroujeny, 1998], chapter 5.3, for the mathematical reasons that render this mandatory). In classical ESNs the EVS sometimes reaches 10^{12} or even higher [Jaeger, 2005], which makes the use of stochastic gradient descent training unfeasible. Other commonly desirable features of the reservoir are small pairwise correlation of the reservoir activations $x_i(n)$, or a large entropy of the $\mathbf{x}(n)$ distribution (e.g., [Jaeger, 2005]). The latter is a rather popular measure, as discussed later in this review. A criterion for maximizing the local information transmission of each individual neuron was investigated in [Triesch, 2005b] (more in Section 2.6.2).

2.6.1.1 Memory capacity

One of the most important functions of the reservoir is to keep the memory of the previous inputs. A correlation-based measure of short-term *memory capacity*, evaluating how well $\mathbf{u}(n)$ can be reconstructed by the reservoir as $\mathbf{y}(n+k)$ after various delays k, was introduced in [Jaeger, 2002a]. It was shown that the total memory capacity, i.e., a sum over different k, does not exceed the number of reservoir units N_x and is maximal for linear reservoir.

The maximum was shown to be easier achievable and more robust to noise in linear RNNs where $\mathbf{W}$ is a scaled orthogonal matrix in [White et al., 2004].

An alternative memory capacity measure based on Fisher information for networks with added Gaussian noise to the activations was proposed in [Ganguli et al., 2008]. It was shown that the maximal total such memory capacity, also of size N_x, for linear units can only be achieved when the N_x units are connected in a single feedforward string. For units that have restricted activation range, like those with sigmoid activation function, the maximal capacity is only proportional to $\sqrt{N_\mathrm{x}}$, and can be achieved by a feedforward branching-out tree-like network topology. Recurrent topologies, and in particular random ones, compare unfavorably under this setup. Contrary to the previous memory measure in [White et al., 2004], linear RNNs with normal $\mathbf{W}$ can only achieve maximal total Fisher memory capacity of 1.

A similar to [Jaeger, 2002a] correlation-based memory capacity measure was investigated for continuous-time linear RNNs in [Hermans and Schrauwen, 2010a]. It was concluded that continuous-time counterparts of the reservoirs constructed according to [Ozturk et al., 2007] (discussed in more detail in Section 2.6.3) tend to have a higher total memory capacity, and those of orthogonal reservoirs [White et al., 2004] also in addition more robustness to noise, compared to random reservoirs.

All the memory measures above considered only one-dimensional $N_\mathrm{u} = 1$ input $\mathbf{u}(n)$. Memory of a multidimensional input in the reservoir was investigated in [Hermans and Schrauwen, 2010b]. Results show that the shape of the memory curve depends on the spectral radius $\rho(\mathbf{W})$: reservoirs with small $\rho(\mathbf{W})$ have precise memory of the recent input which drops sharply with delay k, while those with big $\rho(\mathbf{W})$ have a more extended memory at the expense of precision. The same limit N_x of the correlation-based memory capacity applies, which input dimensions have to share. The individual principal components of the input have memory ca-

pacity roughly proportional to the square root of their variance, indicating that a lot of memory is spent for non-principal components.

The memory capacity measure for ESNs with discrete quantized activations was investigated in [Büsing et al., 2010] and found to be proportional to $\log N_x$.

2.6.1.2 Edge of stability and chaos

For the real-valued reservoirs of sigmoid units the edge of chaos should not be confused with the edge of stability. They are both related to the echo state property (Section 2.5.1). The real-valued sigmoid networks tend to first exhibit multiple fixed-point and periodic attractors, when increasing the spectral radius of $\mathbf{W}$ beyond the stable regime, and only for bigger values of the spectral radius chaotic attractors appear [Ozturk and Príncipe, 2005; Verstraeten et al., 2010]. Thus, the stability (and echo state property) is often lost well before the chaos begins. The latter happens only when the sufficient amount of nonlinearity for chaos is reached. In spiking networks and some other highly nonlinear systems, however, these two boundaries tend to coincide: the unstable dynamics are often immediately chaotic.

Even though stronger inputs $\mathbf{u}(n)$ can push the dynamics of the reservoirs out of the chaotic regime and thus make them useful for computation, no reliable benefit of such a mode of operation was found in the last contribution.

In contrast to ESN-type reservoirs of real-valued units, simple binary threshold units exhibit a more immediate transition from damped to chaotic behavior without intermediate periodic oscillations [Bertschinger and Natschläger, 2004]. This difference between the two types of activation functions, including intermediate *quantized* ones, in ESN-type reservoirs was investigated more closely in [Schrauwen et al., 2009; Büsing et al., 2010]. The investigation showed that reservoirs of binary units are more sensitive to the topology and the connection weight parameters of the network in their transition between damped and chaotic behavior, and computational performance, than the real-valued ones. This difference can be related to the similar apparent difference in sensitivity of the ESNs and LSM-type reservoirs of firing units, discussed in Section 2.5.2.

The so-called *edge of chaos* is a region of parameters of a dynamical system at which it operates at the boundary between the chaotic and non-chaotic behavior. It is often claimed (but not undisputed; see, e.g., [Mitchell et al., 1994]) that at the edge of chaos many types of dynamical systems, including binary systems and reservoirs, possess high computational power [Bertschinger and Natschläger,

2004; Legenstein and Maass, 2007a]. It is intuitively clear that the edge of chaos in reservoirs can only arise when the effect of inputs on the reservoir state does not die out quickly; thus such reservoirs can potentially have high memory capacity, which is also demonstrated in [Legenstein and Maass, 2007a]. However, this does not universally imply that such reservoirs are optimal [Legenstein and Maass, 2007b].

The edge of chaos can be empirically detected (even for biological networks) by measuring Lyapunov exponents [Legenstein and Maass, 2007a], even though such measurements are not trivial (and often involve a degree of expert judgment) for high-dimensional noisy systems. For reservoirs of simple binary threshold units this can be done more simply by computing the Hamming distances between trajectories of the states [Bertschinger and Natschläger, 2004].

Lyapunov exponents for reservoirs of real-valued sigmoid units can be computed more efficiently by making use of their Jacobian matrices [Verstraeten, 2009]. There is also an empirical observation that, while changing different parameter settings of a reservoir, the best performance in a given task correlates with a maximal Lyapunov exponent specific to that task [Verstraeten et al., 2007a]. The optimal exponent is related to the amount of memory needed for the task as discussed in Section 2.5.1. A later more close investigation suggests that this optimum coincides with the situation when the minimal Lyapunov exponent is largest [Verstraeten, 2009].

2.6.2 Unsupervised local methods

A natural strategy for improving reservoirs is to mimic biology (at a high level of abstraction) and count on *local* adaptation rules. "Local" here means that parameters pertaining to some neuron i are adapted on the basis of no other information than the activations of neurons directly connected with neuron i. In fact all local methods are almost exclusively unsupervised, since the information on the performance E at the output is unreachable in the reservoir.

First attempts to decrease the eigenvalue spread in ESNs by classical Hebbian [Hebb, 1949] (inspired by synaptic plasticity in biological brains) or Anti-Hebbian learning gave no success [Jaeger, 2005]. A modification of Anti-Hebbian learning, called *Anti-Oja* learning is reported to improve the performance of ESNs in [Babinec and Pospíchal, 2007].

On the more biologically realistic side of the RC research with spiking neurons, local unsupervised adaptations are very natural to use. In fact, LSMs had used synaptic connections with realistic short-term dynamic adaptation, as proposed by [Markram et al., 1998], in their reservoirs from the very beginning [Maass et al., 2002].

The Hebbian learning principle is usually implemented in spiking NNs as *spike-time-dependent plasticity* (STDP) of synapses. STDP is shown to improve the separation property of LSMs for real-world speech data, but not for random inputs $\mathbf{u}$, in [Norton and Ventura, 2006]. The authors however were uncertain whether manually optimizing the parameters of the STDP adaptation (which they did) or the ones for generating the reservoir would result in a larger performance gain for the same effort spent. STDP is shown to work well with time-coded readouts from the reservoir in [Paugam-Moisy et al., 2008].

Biological neurons are widely observed to adapt their intrinsic excitability, which often results in exponential distributions of firing rates, as observed in visual cortex (e.g., [Baddeley et al., 1997]). This homeostatic adaptation mechanism, called *intrinsic plasticity* (IP) has recently attracted a wide attention in the reservoir computing community. Mathematically, the exponential distribution maximizes the entropy of a non-negative random variable with a fixed mean; thus it enables the neurons to transmit maximal information for a fixed metabolic cost of firing. An IP learning rule for spiking model neurons aimed at this goal was first presented in [Stemmler and Koch, 1999].

For a more abstract model of the neuron, having a continuous Fermi sigmoid activation function $f : \mathbb{R} \to (0, 1)$, the IP rule was derived as a proportional control that changes the steepness and offset of the sigmoid to get an exponential-like output distribution in [Triesch, 2005a]. A more elegant gradient IP learning rule for the same purpose was presented in [Triesch, 2005b], which is similar to the information maximization approach in [Bell and Sejnowski, 1995]. Applying IP with Fermi neurons in reservoir computing significantly improves the performance of BPDC-trained networks [Steil, 2007b; Wardermann and Steil, 2007], and is shown to have a positive effect on offline trained ESNs, but can cause stability problems for larger reservoirs [Steil, 2007b]. An ESN reservoir with IP-adapted Fermi neurons is also shown to enable predicting several superimposed oscillators [Steil, 2007a].

An adaptation of the IP rule to tanh neurons ($f : \mathbb{R} \to (-1, 1)$) that re-

sults in a zero-mean Gaussian-like distribution of activations was first presented in [Verstraeten et al., 2007b] and investigated more in [Schrauwen et al., 2008b]. The IP-adapted ESNs were compared with classical ones, both having Fermi and tanh neurons, in the latter contribution. IP was shown to (modestly) improve the performance in all cases. It was also revealed that ESNs with Fermi neurons have significantly smaller short-term memory capacity (as in Section 2.6.1) and worse performance in a synthetic NARMA prediction task, while having a slightly better performance in a speech recognition task, compared to tanh neurons. The same type of tanh neurons adapted by IP aimed at Laplacian distributions are investigated in [Boedecker et al., 2009]. In general, IP gives more control on the working points of the reservoir nonlinearity sigmoids. The slope (first derivative) and the curvature (second derivative) of the sigmoid at the point around which the activations are centered by the IP rule affect the effective spectral radius and the nonlinearity of the reservoir, respectively. Thus, for example, centering tanh activations around points other than 0 is a good idea if no quasi-linear behavior is desired. IP has recently become employed in reservoirs as a standard practice by several research groups.

Overall, an information-theoretic view on adaptation of spiking neurons has a long history in computational neuroscience. Even better than maximizing just any information in the output of a neuron is maximizing *relevant* information. In other words, in its output the neuron should encode the inputs in such a way as to preserve maximal information about some (local) target signal. This is addressed in a general information-theoretical setting by the *Information Bottleneck* (IB) method [Tishby et al., 1999]. A learning rule for a spiking neuron that maximizes mutual information between its inputs and its output is presented in [Toyoizumi et al., 2005]. A more general IB learning rule, transferring the general ideas of IB method to spiking neurons is introduced in [Klampfl et al., 2007] and [Klampfl et al., 2008]. Two *semi-local* training scenarios are presented in these two contributions. In the first, a neuron optimizes the mutual information of its output with outputs of some neighboring neurons, while minimizing the mutual information with its inputs. In the second, two neurons reading from the same signals maximize their information throughput, while keeping their inputs statistically independent, in effect performing Independent Component Analysis (ICA). A simplified online version of the IB training rule with a variation capable of performing Principle Component Analysis (PCA) was recently introduced in [Buesing

and Maass, 2008]. In addition, it assumes slow semi-local target signals, which is more biologically plausible. The approaches described in this paragraph are still waiting to be tested in the reservoir computing setting.

It is also of great interest to understand how different types of plasticity observed in biological brains interact when applied together and what effect this has on the quality of reservoirs. The interaction of the IP with Hebbian synaptic plasticity in a single Fermi neuron is investigated in [Triesch, 2005a] and further in [Triesch, 2007]. The synergy of the two plasticities is shown to result in a better specialization of the neuron that finds heavy-tail directions in the input. An interaction of IP with a neighborhood-based Hebbian learning in a layer of such neurons was also shown to maximize information transmission, perform nonlinear ICA, and result in an emergence of orientational Gabor-like receptive fields in [Butko and Triesch, 2007].

The interaction of STDP with IP in an LSM-like reservoir with a fixed number of units that spike at each time step[5] was investigated in [Lazar et al., 2007]. The interaction turned out to be a non-trivial one, resulting in networks more robust to perturbations of the state $\mathbf{x}(n)$ and having a better short-time memory and time series prediction performance. An RNN model with discrete-time spiking excitatory and inhibitory units with STDP, IP, and synaptic scaling plasticity mechanisms simultaneously acting on subsets of the different connections was investigated in [Lazar et al., 2010; Lazar, 2010] under a name of *self-organizing recurrent neural network*. Such reservoirs were shown to be better on synthetic counting and occlusion tasks with symbol sequences by learning to have a longer task-specific memory compared to random reservoirs. All three types of plasticity were shown to be essential for the model to work. Recently, a connection of such learning with Bayesian priors was discussed in [Lazar et al., 2011].

An RNN of radial basis function units adapted by generalized *self-organizing maps* or *neural gas* learning methods is proposed and investigated as a reservoir in the Chapter 4 of this thesis, with an earlier version published in [Lukoševičius, 2010]. The learning mechanisms also make the approach a not purely local one, because global information is needed for organizing the learning.

A recent approach of combining STDP with a biologically plausible reinforcement signal is discussed in Section 2.7.5, as it is not unsupervised.

[5]This makes the method a not purely local, because global information is needed to determine which units fire.

2.6.3 Unsupervised global methods

Here we review unsupervised methods that optimize reservoirs based on *global* information of the reservoir activations induced by the given input $\mathbf{u}(x)$, but irrespective of the target $\mathbf{y}_{\text{target}}(n)$, like for example the measures discussed in Section 2.6.1. The intuitive goal of such methods is to produce good representations of (the history of) $\mathbf{u}(n)$ in $\mathbf{x}(n)$ for any (and possibly several) $\mathbf{y}_{\text{target}}(n)$.

A biologically inspired unsupervised approach with a reservoir trying to predict itself is proposed in [Mayer and Browne, 2004]. An additional output $\mathbf{z}(n) \in \mathbb{R}^{N_{\text{x}}}$, $\mathbf{z}(n) = \mathbf{W}^{\text{z}}\mathbf{x}(n)$ from the reservoir is trained on the target $\mathbf{z}_{\text{target}}(n) = \mathbf{x}'(n+1)$, where $\mathbf{x}'(n)$ are the activations of the reservoir before applying the neuron transfer function $\tanh(\cdot)$, i.e., $\mathbf{x}(n) = \tanh(\mathbf{x}'(n))$. Then, in the application phase of the trained networks, the original activations $\mathbf{x}'(n)$, which result from $\mathbf{u}(n)$, $\mathbf{W}^{\text{in}}$, and $\mathbf{W}$, are mixed with the self-predictions $\mathbf{z}(n-1)$ obtained from $\mathbf{W}^{\text{z}}$, with a certain mixing ratio $(1-\alpha) : \alpha$. The coefficient α determines how much the reservoir is relying on the external input $\mathbf{u}(n)$ and how much on the internal self-prediction $\mathbf{z}(n)$. With $\alpha = 0$ we have the classical ESN and with $\alpha = 1$ we have an "autistic" reservoir that does not react to the input. Intermediate values of α close to 1 were shown to enable reservoirs to generate slow, highly nonlinear signals that are hard to get otherwise.

An algebraic unsupervised way of generating ESN reservoirs was proposed in [Ozturk et al., 2007]. The idea is to linearize the ESN update equation (2.5) locally around its current state $\mathbf{x}(n)$ at every time step n to get a linear approximation of (2.5) as $\mathbf{x}(n+1) = \mathbf{A}\mathbf{x}(n) + \mathbf{B}\mathbf{u}(n)$, where $\mathbf{A}$ and $\mathbf{B}$ are time (n)-dependent matrices corresponding to $\mathbf{W}$ and $\mathbf{W}^{\text{in}}$ respectively. The approach aims at distributing the predefined complex eigenvalues of $\mathbf{A}$ uniformly within the unit circle on the $\mathbb{C}$ plane. The reservoir matrix $\mathbf{W}$ is obtained analytically from the set of these predefined eigenvalues and a given input $\mathbf{u}(n)$. The motivation for this is, as for Kautz filters [Kautz, 1954] in linear systems, that if the target $\mathbf{y}_{\text{target}}(n)$ is unknown, it is best to have something like an orthogonal basis in $\mathbf{x}(n)$, from which any $\mathbf{y}_{\text{target}}(n)$ could, on average, be constructed well. The spectral radius of the reservoir is suggested to be set by hand (according to the correlation time of $\mathbf{u}(n)$, which is an indication of a memory span needed for the task), or by adapting the bias value of the reservoir units to minimize the output error (which actually renders this method *supervised*, as in Section 2.7). Reservoirs generated this way are shown to yield higher average entropy of $\mathbf{x}(n)$ distribution, higher

short-term memory capacity (both measures mentioned in Section 2.6.1), and a smaller output error on a number of synthetic problems, using relatively small reservoirs ($N_{\mathrm{x}} = 20, 30$). However, a more extensive empirical comparison of this type of reservoir with the classical ESN one is still lacking.

It has been recently demonstrated that reservoir activations $\mathbf{x}(n)$ obtained by running a regular ESN with random weights can be successfully reimplemented by learning new $\mathbf{W}^{\mathrm{in}}$ and $\mathbf{W}$ from $\mathbf{x}(n)$ using the same type of linear or ridge regression which is used to learn $\mathbf{W}^{\mathrm{out}}$ (see Section 2.8.1.1) [Reinhart and Steil, 2011b]. Even though the obtained activations $\mathbf{x}(n)$ are virtually the same, $\mathbf{W}$ are very different and on average much smaller, subject to the regularization parameter in the ridge regularization. As a benefit, regularizing $\mathbf{W}$ in such a way was shown to improve stability of ESNs with output feedbacks [Reinhart and Steil, 2011a], a problem discussed in Section 2.8.2.

2.6.4 Online reservoir control

In contrast to the universal "goodness" measures of the reservoirs discussed in Section 2.6.1, for some applications the desired properties of the reservoir are quite specific. Similar to the local homeostatic self-regulation of neurons discussed in Section 2.6.2 we might want the reservoir to posses some global invariances. There are several approaches in the literature that use simple unsupervised control mechanisms for the reservoirs to achieve the desired invariances.

A common distortion in speech or handwriting signals is the so-called time-warping, which is a varying speed of signal changes along the time domain. These temporal distortions have a negative effect on most neural-dynamics-based speech or handwriting recognition methods, thus it would be desirable to make them time-warping invariant. As a potential solution, a simple mechanism dynamically adapting the leaking rates of an ESN reservoir depending on the rate of change was presented in [Lukoševičius et al., 2006; Jaeger et al., 2007a]. This architecture varies Δt on-the-fly in (2.9), directly depending on the speed at which the input $\mathbf{u}(n)$ is changing. It in effect dynamically adapts the time of the reservoir such that the reservoir "sees" the input as changing at a uniform rate, and effectively deals with even a high degree of time-warping. As a drawback, such method would probably be sensitive to high frequency noise in the input. Also the constant-rate-of-change interpretation of, especially low-dimensional, inputs might discard some

important information.

Other distortions need to be addressed in different ways. A possible way to deal with widely but slowly varying input modulation is presented in [Jaeger, 2010]. It uses the relative slowness of the distortion as its distinguishing characteristic, similar to Slow Feature Analysis (SFA) [Wiskott et al., 2011], by controlling the ESN reservoir to suppress the slowest components in its activations. This way the reservoirs can be trained to compensate for slow input distortions that can be quite big. After training, controller can even be integrated into the reservoir weights $\mathbf{W}$ eliminating any computational overhead. A related method for controlling features of an ESN-based pattern generator is presented in [Li and Jaeger, 2010]. A similar approach for extracting any predefined distortion invariant features from an ESN reservoir is presented in [Šakėnas, 2010].

An additional binary input to the reservoir is able to switch between two learned behaviors in the output, as, e.g., demonstrated in [Antonelo et al., 2008].

2.7 Supervised reservoir pre-training

In this section we discuss methods for training reservoirs to perform a specific given task, i.e., not only the concrete input $\mathbf{u}(n)$, but also the desired output $\mathbf{y}_{\text{target}}(n)$ is taken into account. Since a linear readout from a reservoir is quickly trained, the suitability of a candidate reservoir for a particular task (e.g., in terms of NRMSE (2.1)) is inexpensive to check. Notice that even for most methods of this class the explicit target signal $\mathbf{y}_{\text{target}}(n)$ is not technically required for training the reservoir itself, but only for evaluating it in an outer loop of the adaptation process.

2.7.1 Optimization of global reservoir parameters

In Section 2.5.1 we discussed guidelines for the manual choice of global parameters for reservoirs of ESNs. This approach works well only with experience and a good intuitive grasp on nonlinear dynamics. A systematic gradient descent method of optimizing the global parameters of LI-ESNs (recalled from Section 2.5.5) to fit them to a given task is presented in [Jaeger et al., 2007a]. The investigation shows that the error surfaces in the combined global parameter and $\mathbf{W}^{\text{out}}$ spaces may have very high curvature and multiple local minima. Thus, gradient descent

methods are not always practical.

2.7.2 Evolutionary methods

As one can see from the previous sections of this review, optimizing reservoirs is generally challenging, and breakthrough methods remain to be found. On the other hand checking the performance of a resulting ESN is relatively inexpensive, as said. This brings in *evolutionary* methods for the reservoir pre-training as a natural strategy.

Recall that the classical method generates a reservoir randomly; thus the performance of the resulting ESN varies slightly (and for small reservoirs not so slightly) from one instance to another. Then indeed, an "evolutionary" method as naive as "generate k reservoirs, pick the best" will outperform the classical method ("generate a reservoir") with probability $(k-1)/k$, even though the improvement might be not striking.

Several evolutionary approaches on optimizing reservoirs of ESNs are presented in [Ishii et al., 2004]. The first approach was to carry out an evolutionary search on the parameters for generating $\mathbf{W}$: N_x, $\rho(\mathbf{W})$, and the connection density of $\mathbf{W}$. Then an evolutionary algorithm [Holland, 1992] was used on individuals consisting of all the weight matrices $(\mathbf{W}^\mathrm{in}, \mathbf{W}, \mathbf{W}^\mathrm{ofb})$ of small $(N_\mathrm{x} = 5)$ reservoirs. A variant with a reduced search space was also tried where the weights, but not the topology, of $\mathbf{W}$ were explored, i.e., elements of $\mathbf{W}$ that were zero initially always stayed zero. The empirical results of modeling the motion of an underwater robot showed superiority of the methods over other state-of-art methods, and that the topology-restricted adaptation of $\mathbf{W}$ is almost as effective as the full one.

Another approach of optimizing the reservoir $\mathbf{W}$ by a greedy evolutionary search is presented in [Bush and Tsendjav, 2005]. Here the same idea of separating the topology and weight sizes of $\mathbf{W}$ to reduce the search space was independently used, but the search was, conversely, restricted to the connection topology. This approach also was demonstrated to yield on average 50% smaller (and much more stable) error in predicting the behavior of a mass–spring–damper system with small $(N_\mathrm{x} = 20)$ reservoirs than without the genetic optimization.

Yet another way of reducing the search space of the reservoir parameters is constructing a big reservoir weight matrix $\mathbf{W}$ in a fractal fashion by repeatedly applying *Kronecker* self-multiplication to an initial small matrix, called the *Kro-*

necker kernel [Rad et al., 2008]. This contribution showed that among $\mathbf{W}$s constructed in this way some yield ESN performance similar to the best unconstrained $\mathbf{W}$s; thus only the good weights of the small Kronecker kernel need to be found by evolutionary search for producing a well-performing reservoir.

Evolino [Schmidhuber et al., 2007], introduced in Section 2.3.3, is another example of adapting a reservoir (in this case an LSTM network) using a genetic search.

It has been recently demonstrated that by adapting only the slopes of the reservoir unit activation functions $f(\cdot)$ by a state-of-art evolutionary algorithm, and having $\mathbf{W}^{\text{out}}$ random and fixed, a prediction performance of an ESN can be achieved close to the best of classical ESNs [Jiang et al., 2008b].

In addition to (or instead of) adapting the reservoirs, an evolutionary search can also be applied in training the readouts, such as readouts with no explicit $\mathbf{y}_{\text{target}}(n)$ as discussed in Section 2.8.4.

2.7.3 Other types of supervised reservoir tuning

A greedy pruning of neurons from a big reservoir has been shown in a recent initial attempt [Dutoit et al., 2007] to often give a (bit) better classification performance for the same final N_{x} than just a randomly created reservoir of the same size. The effect of neuron removal to the reservoir dynamics, however, has not been addressed yet.

Combining RC approach and error backpropagation methods for RNN training briefly discussed in Section 2.2.5 is also a naturally interesting approach investigated in RC literature. Fine-tuning a trained small ($N_{\text{x}} = 10$) ESN with a computationally expensive RTRL [Williams and Zipser, 1989] algorithm substantially increases its performance in simple tasks, resulting in compact reasonably well performing RNNs [Erhan, 2004]. ESNs, however, allow for and usually work best with much larger reservoir sizes, for which RTRL algorithm would have a prohibitively long runtime.

A simple one time step BPTT [Werbos, 1990] like in Elman [Elman, 1990] networks was applied to train a RNN of ESN type in [Hermans and Schrauwen, 2010c]. This training was performed on either only $\mathbf{W}^{\text{out}}$, which is a classical online ESN training (Section 2.8.1.2), on $\mathbf{W}^{\text{out}}$ and $\mathbf{W}^{\text{in}}$, or on all weights including $\mathbf{W}$. Training more weights improved performance on spoken digit recognition, with

training just $\mathbf{W}^{\text{out}}$ and $\mathbf{W}^{\text{in}}$, and leaving $\mathbf{W}$ fixed being a good compromise. Batch learning of $\mathbf{W}^{\text{out}}$ via linear regression (Section 2.8.1.1) is, however, often much more accurate than the investigated online learning.

2.7.4 Trained auxiliary feedbacks

While reservoirs have a natural capability of performing complex real-time analog computations with fading memory [Maass et al., 2002], an analytical investigation has shown that they can approximate any k-order differential equation (with persistent memory) if extended with k trained feedbacks [Maass et al., 2006, 2007]. This is equivalent to simulating any Turing machine, and thus also means universal digital computing. In the presence of noise (or finite precision) the memory becomes limited in such models, but they still can simulate Turing machines with finite tapes.

This theory has direct implications for reservoir computing; thus different ideas on how the power of ESNs could be improved along its lines are explored in [Lukoševičius, 2007]. It is done by defining auxiliary targets, training additional outputs of ESNs on these targets, and feeding the outputs back to the reservoir. Note that this can be implemented in the usual model with feedback connections (2.6) by extending the original output $\mathbf{y}(n)$ with additional dimensions that are trained before training the original (final) output. The auxiliary targets are constructed from $\mathbf{y}_{\text{target}}(n)$ and/or $\mathbf{u}(n)$ or some additional knowledge of the modeled process. The intuition is that the feedbacks could shift the internal dynamics of $\mathbf{x}(n)$ in the directions that would make them better linearly combinable into $\mathbf{y}_{\text{target}}(n)$. The investigation showed that for some types of tasks there are natural candidates for such auxiliary targets, which improve the performance significantly. Unfortunately, no universally applicable methods for producing auxiliary targets are known such that the targets would be both easy to learn and improve the accuracy of the final output $\mathbf{y}(n)$. In addition, training multiple outputs with feedback connections $\mathbf{W}^{\text{ofb}}$ makes the whole procedure more complicated, as cyclical dependences between the trained outputs (one must take care of the order in which the outputs are trained) as well as stability issues discussed in Section 2.8.2 arise. Despite these obstacles, we perceive this line of research as having a big potential.

2.7.5 Reinforcement learning

In the line of biologically inspired local unsupervised adaptation methods discussed in Section 2.6.2, an STDP modulated by a reinforcement signal has recently emerged as a powerful learning mechanism, capable of explaining some famous findings in neuroscience (biofeedback in monkeys), as demonstrated in [Legenstein et al., 2008a,b] and references thereof. The learning mechanism is also well biologically motivated as it uses a local unsupervised STDP rule and a reinforcement (i.e., reward) feedback, which is present in biological brains in a form of chemical signaling, e.g., by the level of dopamine. A similar rule has also been recently devised for analog neurons [Legenstein et al., 2010]. In the RC framework these learning rules have been successfully applied for training readouts from the reservoirs so far in [Legenstein et al., 2008b, 2010; Hoerzer, 2010] (see Section 2.8.4), but could in principle be applied inside the reservoir too.

Overall we believe that reinforcement learning methods are natural candidates for reservoir adaptation, as they can immediately exploit the knowledge of how well the output is learned inside the reservoir without the problems of error back-propagation. They can also be used in settings where no explicit target $\mathbf{y}_{\text{target}}(n)$ is available. We expect to see more applications of reinforcement learning in reservoir computing in the future.

2.8 Readouts from the reservoirs

Conceptually, training a readout from a reservoir is a common supervised non-temporal task of mapping $\mathbf{x}(n)$ to $\mathbf{y}_{\text{target}}(n)$. This is a well investigated domain in machine learning, much more so than learning temporal mappings with memory. A large choice of methods is available, and in principle any of them can be applied. Thus we will only briefly go through the ones reported to be successful in the literature.

2.8.1 Single-layer readout

By far the most popular readout method from the ESN reservoirs is the originally proposed [Jaeger, 2001] simple linear readout, as in (2.3) (we will consider it as equivalent to (2.8), i.e., $\mathbf{u}(n)$ being part of $\mathbf{x}(n)$). It is shown to be often sufficient, as reservoirs provide a rich enough pool of signals for solving many application-

relevant and benchmark tasks, and is very efficient to train, since optimal solutions can be found analytically.

2.8.1.1 Linear regression

In batch mode, learning of the output weights $\mathbf{W}^{\text{out}}$ (2.2) can be phrased as solving a system of linear equations

$$\mathbf{W}^{\text{out}}\mathbf{X} = \mathbf{Y}_{\text{target}} \tag{2.12}$$

with respect to $\mathbf{W}^{\text{out}}$, where $\mathbf{X} \in \mathbb{R}^{N\times T}$ are all $\mathbf{x}(n)$ produced by presenting the reservoir with $\mathbf{u}(n)$, and $\mathbf{Y}_{\text{target}} \in \mathbb{R}^{N_{\text{y}}\times T}$ are all $\mathbf{y}_{\text{target}}(n)$, both collected into respective matrices over the training period $n = 1, \ldots, T$. Usually $\mathbf{x}(n)$ data from the beginning of the training run are discarded (they come before $n = 1$), since they are contaminated by initial transients.

Since typically the goal is minimizing a quadratic error $E(\mathbf{Y}_{\text{target}}, \mathbf{W}^{\text{out}}\mathbf{X})$ as in (2.1) and $T > N$, to solve (2.12) one usually employs methods for finding *least square* solutions of *overdetermined* systems of linear equations (e.g., [Björck, 1996]), the problem also known as *linear regression*. One direct method is calculating the Moore-Penrose pseudoinverse $\mathbf{X}^{+}$ of $\mathbf{X}$, and $\mathbf{W}^{\text{out}}$ as

$$\mathbf{W}^{\text{out}} = \mathbf{Y}_{\text{target}}\mathbf{X}^{+}. \tag{2.13}$$

Direct pseudoinverse calculations exhibit high numerical stability, but are expensive memory-wise for large state-collecting matrices $\mathbf{X} \in \mathbb{R}^{N\times T}$, thereby limiting the size of the reservoir N and/or the number of training samples T.

This issue is resolved in the *normal equations* formulation of the problem:[6]

$$\mathbf{W}^{\text{out}}\mathbf{X}\mathbf{X}^{\text{T}} = \mathbf{Y}_{\text{target}}\mathbf{X}^{\text{T}}. \tag{2.14}$$

A naive solution of it would be

$$\mathbf{W}^{\text{out}} = \mathbf{Y}_{\text{target}}\mathbf{X}^{\text{T}}(\mathbf{X}\mathbf{X}^{\text{T}})^{-1}. \tag{2.15}$$

Note that in this case $\mathbf{Y}_{\text{target}}\mathbf{X}^{\text{T}} \in \mathbb{R}^{N_{\text{y}}\times N}$ and $\mathbf{X}\mathbf{X}^{\text{T}} \in \mathbb{R}^{N\times N}$ do not depend on the length T of the training sequence in their sizes, and can be calculated incrementally

[6] Note that our matrices are transposed compared to the conventional notation.

while the training data are passed through the reservoir. Thus, having these two matrices collected, the solution complexity of (2.15) does not depend on T either in time or in space. Also, intermediate values of $\mathbf{W}^{\text{out}}$ can be calculated in the middle of running through the training data, e.g., for an early assessment of the performance, making this a "semi-online" training method.

The method (2.15) has lower numerical stability, compared to (2.13), but the problem can be mitigated by using the pseudoinverse $(\mathbf{X}\mathbf{X}^{\text{T}})^{+}$ instead of the real inverse $(\mathbf{X}\mathbf{X}^{\text{T}})^{-1}$ (which usually also works faster). In addition, this method enables one to introduce *ridge*, or *Tikhonov*, regularization elegantly:

$$\mathbf{W}^{\text{out}} = \mathbf{Y}_{\text{target}}\mathbf{X}^{\text{T}}(\mathbf{X}\mathbf{X}^{\text{T}} + \alpha^2\mathbf{I})^{-1}, \tag{2.16}$$

where $\mathbf{I} \in \mathbb{R}^{N\times N}$ is the identity matrix and α is a regularization factor. In addition to improving the numerical stability, the regularization in effect reduces the magnitudes of entries in $\mathbf{W}^{\text{out}}$, thus mitigating sensitivity to noise and overfitting; see Section 2.8.2 for more details. All this makes (2.16) a highly recommendable choice for learning outputs from the reservoirs.

Another alternative for solving (2.14) is decomposing the matrix $\mathbf{X}\mathbf{X}^{\text{T}}$ into a product of two triangular matrices via Cholesky or LU decomposition, and solving (2.14) by two steps of substitution, avoiding (pseudo-)inverses completely. The Cholesky decomposition is the more numerically stable of the two.

Weighted regression can be used for training linear readouts by multiplying both $\mathbf{x}(n)$ and the corresponding $\mathbf{y}_{\text{target}}(n)$ by different weights over time, thus emphasizing some time steps n over others. Multiplying certain recorded $\mathbf{x}(n)$ and corresponding $\mathbf{y}_{\text{target}}(n)$ by $\sqrt{k}$ has the same emphasizing effect as if they appeared in the training sequence k times.

When the reservoir is made from spiking neurons and thus $\mathbf{x}(n)$ becomes a collection of spike trains, smoothing by low-pass filtering may be applied to it before doing the linear regression, or it can be done directly on $\mathbf{x}(n)$ [Maass et al., 2002]. For more on linear regression based on spike train data, see [Carnell and Richardson, 2005].

Evolutionary search for training linear readouts can also be employed. State-of-art evolutionary methods are demonstrated to be able to achieve the same record levels of precision for supervised tasks as with the best applications of linear regression in ESN training [Jiang et al., 2008b]. Their much higher computational

cost is justifiable in settings where no explicit $\mathbf{y}_{\text{target}}(n)$ is available, discussed in Section 2.8.4.

2.8.1.2 Online adaptive output weight training

Some applications require online model adaptation, e.g., in online adaptive channel equalization [Jaeger and Haas, 2004]. In such cases one typically minimizes an error that is exponentially discounted going back in time. $\mathbf{W}^{\text{out}}$ here acts as an adaptive linear combiner. The simplest way to train $\mathbf{W}^{\text{out}}$ is to use stochastic gradient descent. The method is familiar as the *Least Mean Squares* (LMS) algorithm in linear signal processing [Farhang-Boroujeny, 1998], and has many extensions and modifications. Its convergence performance is unfortunately severely impaired by large eigenvalue spreads of $\mathbf{X}\mathbf{X}^{\text{T}}$, as mentioned in Section 2.6.1.

An alternative to LMS, known in linear signal processing as the *Recursive Least Squares* (RLS) algorithm, is insensitive to the detrimental effects of eigenvalue spread and boasts a much faster convergence because it is a second-order method. The downside is that RLS is computationally more expensive (order $O(N^2)$ per time step instead of $O(N)$ for LMS, for $N_{\text{y}} = 1$) and notorious for numerical stability issues. Demonstrations of RLS are presented in [Jaeger and Haas, 2004; Jaeger, 2003]. A careful and comprehensive comparison of variants of RLS is carried out in a Master's thesis [Küçükemre, 2006], which we mention here because it will be helpful for practitioners.

The BackPropagation-DeCorrelation (BPDC) and FORCE learning algorithms discussed in Sections 2.3.4 and 2.3.5 respectively are two other powerful methods for online training of single-layer readouts with feedback connections from the reservoirs.

Simple forms of adaptive online learning, such as LMS, are also more biologically plausible than batch-mode training. From spiking neurons a firing time-coded (instead of a more common firing rate-coded) output for classification can also be trained by only adapting the delays of the output connections [Paugam-Moisy et al., 2008]. And firing rate-coded readouts can be trained by a biologically-realistic reward-modulated STDP [Legenstein et al., 2008b], mentioned in Section 2.6.2.

2.8.1.3 SVM-style readout

Continuing the analogy between the temporal and non-temporal expansion methods, discussed in Section 2.2, the reservoir can be considered a temporal kernel, and the standard linear readout $\mathbf{W}^{\text{out}}$ from it can be trained using the same loss functions and regularizations as in Support Vector Machines (SVMs) or Support Vector Regression (SVR). Different versions of this approach are proposed and investigated in [Shi and Han, 2007].

A standard SVM (having its own kernel) can also be used as a readout from a continuous-value reservoir [Schmidhuber et al., 2006]. Similarly, special kernel types could be applied in reading out from spiking (LSM-type) reservoirs [Schrauwen and Campenhout, 2006] (and references therein).

2.8.2 Feedbacks and stability issues

Stability issues (with reservoirs having the echo state property) usually only occur in *generative* setups where a model trained on (one step) signal prediction is later run in a generative mode, looping its output $\mathbf{y}(n)$ back into the input as $\mathbf{u}(n+1)$. Note that this is equivalent to a model with output feedbacks $\mathbf{W}^{\text{ofb}}$ (2.6) and no input at all ($N_{\text{u}} = 0$), which is usually trained using *teacher forcing* (i.e., feeding $\mathbf{y}_{\text{target}}(n)$ as $\mathbf{y}(n)$ for the feedbacks during the training run) and later is run freely to generate signals as $\mathbf{y}(n)$. $\mathbf{W}^{\text{in}}$ in the first case is equivalent to $\mathbf{W}^{\text{ofb}}$ in the second one. Models having feedbacks $\mathbf{W}^{\text{ofb}}$ may also suffer from instability while driven with external input $\mathbf{u}(n)$, i.e., not in a purely generative mode.

The reason for these instabilities is that even if the model can predict the signal quite accurately, going through the feedback loop of connections $\mathbf{W}^{\text{out}}$ and $\mathbf{W}^{\text{ofb}}$ (or $\mathbf{W}^{\text{in}}$) small errors get amplified, making $\mathbf{y}(n)$ diverge from the intended $\mathbf{y}_{\text{target}}(n)$.

One way to look at this for trained linear outputs is to consider the feedback loop connections $\mathbf{W}^{\text{out}}$ and $\mathbf{W}^{\text{ofb}}$ as part of the reservoir $\mathbf{W}$. Putting (2.6) and (2.2) together we get

$$\mathbf{x}(n) = f(\mathbf{W}^{\text{in}}\mathbf{u}(n) + [\mathbf{W} + \mathbf{W}^{\text{ofb}}\mathbf{W}^{\text{out}}]\mathbf{x}(n-1)), \tag{2.17}$$

where $\mathbf{W} + \mathbf{W}^{\text{ofb}}\mathbf{W}^{\text{out}}$ forms the "extended reservoir" connections, which we will call $\mathbf{W}^*$ for brevity (as in [Lukoševičius, 2007] Section 3.2). If the spectral radius

of the extended reservoir $\rho(\mathbf{W}^*)$ is very large we can expect unstable behavior. A more detailed analysis using Laplace transformations and a sufficient condition for stability is presented in [Steil, 2005a]. On the other hand, for purely generative tasks, $\rho(\mathbf{W}^*) < 1$ would mean that the generated signal would die out, which is not desirable in most cases. Thus producing a generator with stable dynamics is often not trivial.

Quite generally, models trained with clean (noise-free) data for the best one-time-step prediction diverge fast in the generative mode, as they are too "sharp" and not noise-robust. A classical remedy is adding some noise to reservoir states $\mathbf{x}(n)$ [Jaeger, 2001] during the training. This way the generator forms a stable attractor by learning how to come to the desired next output $\mathbf{y}_{\text{target}}(n)$ from a neighborhood of the current state $\mathbf{x}(n)$, having seen it perturbed by noise during training. Setting the right amount of noise is a delicate balance between the sharpness (of the prediction) and the stability of the generator. Alternatively, adding noise to $\mathbf{x}(n)$ can be seen as a form of regularization in training, as it in effect also emphasizes the diagonal of matrix $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ in (2.16). A similar effect can be achieved using ridge regression (2.16) [wyffels et al., 2008a], or to some extent even pruning of $\mathbf{W}^{\text{out}}$ [Dutoit et al., 2008]. Ridge regression (2.16) is the least computationally expensive to do of the three, since the reservoir does not need to be rerun with the data to test different values of the regularization factor α.

Using different modifications of signals for teacher forcing, like mixing $\mathbf{y}_{\text{target}}(n)$ with noise, or in some cases using pure strong noise, during the training also has an effect on the final performance and stability, as discussed in Section 5.4 of [Lukoševičius, 2007].

2.8.3 Readouts for classification/recognition

The time series classification or temporal pattern detection tasks that need a category indicator (as opposed to real values) as an output can be implemented in two main ways. The most common and straightforward way is having a real-valued output for each class (or a single output and a threshold for the two-class classifier), and interpreting the strengths of the outputs as votes for the corresponding classes, or even class probabilities (several options are discussed in [Jaeger et al., 2007a]). Often the most probable class is taken as the decision. A simple target $\mathbf{y}_{\text{target}}$ for this approach is a constant $y_{\text{target}_i}(n) = 1$ signal for the right class i and 0 for the

others in the range of n where the indicating output is expected. More elaborate shapes of $\mathbf{y}_{\text{target}}(n)$ can improve classification performance, depending on the task (e.g., [Lukoševičius et al., 2006]). With spiking neurons the direct classification based on time coding can be learned and done, e.g., the class is assigned depending on which output fires first [Paugam-Moisy et al., 2008].

The main alternative to direct class indications is to use predictive classifiers, i.e., train different predictors to predict different classes and assign a class to a new example corresponding to the predictor that predicts it best. Here the quality of each predictor serves as the output strength for the corresponding class. The method is quite popular in automated speech recognition (e.g., Section 6 in [Tebelskis, 1995] for an overview). However, in Section 6.5 of [Tebelskis, 1995] the author argues against this approach, at least in its straightforward form, pointing out some weaknesses, like the lack of specificity, and negative practical experience.

For both approaches a weighting scheme can be used for both training (like in weighted regression) and integrating the class votes, e.g., putting more emphasis on the end of the pattern when sufficient information has reached the classifier to make the decision.

An advanced version of ESN-based predictive classifier, where for each class there is a set of competitively trained predictors and dynamic programming is used to find the optimal sequence of them, is reported to be much more noise robust than a standard Hidden Markov Model in spoken word recognition [Skowronski and Harris, 2007].

A phoneme recognizer accommodating a bigram phoneme language model and doing a Viterbi search for the most likely phonemic sequence as a readout from a large ESN reservoir was demonstrated to achieve state of art recognition performance in [Triefenbach et al., 2011].

See [Graves, 2008] for additional advanced options on recognition readouts from RNNs.

2.8.4 Readouts trained without target

Even though most of the readout types from reservoirs reported in the literature are trained in a purely supervised manner, i.e., making $\mathbf{y}(n)$ match an explicitly given $\mathbf{y}_{\text{target}}(n)$, the reservoir computing paradigm lends itself to settings where no $\mathbf{y}_{\text{target}}(n)$ is available. A typical such setting is *reinforcement learning* where

only a feedback on the model's performance is available. Note that an explicit $\mathbf{y}_{\text{target}}(n)$ is not required for the reservoir adaptation methods discussed in Sections 2.5 and 2.6 of this survey by definition. Even most of the adaptation methods classified as *supervised* in Section 2.7 do not need an explicit $\mathbf{y}_{\text{target}}(n)$, as long as one can evaluate the performance of the reservoir. Thus they can be used without modification, provided that unsupervised training and evaluation of the output is not prohibitively expensive or can be done simultaneously with reservoir adaptation. In this section we will give some pointers on training readouts using reinforcement learning.

A biologically inspired learning rule of Spike-Time-Dependent Plasticity (STDP) modulated by a reinforcement signal has been successfully applied for training a readout of firing neurons from the reservoirs of the same LSM-type in [Legenstein et al., 2008b].

A similar reinforcement-modulated Hebbian learning rule has recently been devised for analog neurons in [Legenstein et al., 2010] and shown to work as a learning mechanism for an ESN readout with feedbacks, a setup similar to FORCE learning (Section 2.3.5), in [Hoerzer, 2010].

Evolutionary algorithms are a natural candidate for training outputs in a non-fully-supervised manner. Using a genetic search with crossover and mutation to find optimal output weights $\mathbf{W}^{\text{out}}$ of an ESN is reported in [Xu et al., 2005]. Such an ESN is successfully applied for a hard reinforcement learning task of direct adaptive control, replacing a classical indirect controller.

ESNs trained with a simple "(1+1)" evolution strategy for an unsupervised artificial embryogeny (the, so-called, "flag") problem are shown to perform very well in [Devert et al., 2008].

An ESN trained with a state-of-art evolutionary continuous parameter optimization method (CMA-ES) shows comparable performance in a benchmark double pole balancing problem to the best RNN topology-learning methods in [Jiang et al., 2008b,a]. For this problem the best results are obtained when the spectral radius $\rho(\mathbf{W})$ is adapted together with $\mathbf{W}^{\text{out}}$. The same contributions also validate the CMA-ES readout training method on a standard supervised prediction task, achieving the same excellent precision (MSE of the order 10^{-15}) as the state-of-art with linear regression. Conversely, the best results for this task were achieved with $\rho(\mathbf{W})$ fixed and training only $\mathbf{W}^{\text{out}}$. An even more curious finding is that almost as good results were achieved by only adapting slopes of the reservoir activation

functions $f(\cdot)$ and having $\mathbf{W}^{\text{out}}$ fixed, as mentioned in Section 2.7.2.

A way of extracting features from an ESN reservoir invariant to specific distortions of input is presented in [Šakėnas, 2010].

A Slow Feature Analysis SFA and sparse coding layers stacked on top of an ESN reservoir fed with robot data sensors as input is shown to unsupervisedly learn the behavior of place cells: each neuron is only activated when the robot is in a particular place in its environment [Antonelo and Schrauwen, 2011].

2.8.5 Multilayer readouts

Multilayer perceptrons (MLPs) as readouts, trained by error backpropagation, were used from the very beginnings of LSMs [Maass et al., 2002] and ESNs (unpublished). They are theoretically more powerful and expressive in their instantaneous mappings from $\mathbf{x}(n)$ to $\mathbf{y}(n)$ than linear readouts, and are thus suitable for particularly nonlinear outputs, e.g., in [Bush and Anderson, 2005; Babinec and Pospíchal, 2006]. In both cases the MLP readouts are trained by error backpropagation. On the other hand they are significantly harder to train than an optimal single-layer linear regression, thus often giving inferior results compared to the latter in practice.

Some experience in training MLPs as ESN readouts, including network initialization, using stochastic, batch, and semi-batch gradients, adapting learning rates, and combining with regression-training of the last layer of the MLP, is presented in Section 5.3 of [Lukoševičius, 2007].

An approach of combining ESN reservoirs with random feed-forward layers of neurons is proposed in [Butcher et al., 2010].

2.8.6 Readouts with delays

While the readouts from reservoirs are usually recurrence-free, it does not mean that they may not have memory. In some approaches they do, or rather some memory is inserted between the reservoir and the readout.

Learning a delay for each neuron in an ESN reservoir $\mathbf{x}(n)$ in addition to the output weight from it is investigated in [Holzmann and Hauser, 2010]. One way of learning such readout is using a cross-correlation (simple or generalized) to optimally align activations of each neuron in $\mathbf{x}(n)$ with $\mathbf{y}_{\text{target}}(n)$, and then activations with the delays $\mathbf{x}_{\text{delayed}}(n)$ are used to find $\mathbf{W}^{\text{out}}$ using linear regression.

Another proposed way is using a kind of Expectation-Maximization to iteratively correlate each dimension $x_i(n)$ of $\mathbf{x}(n)$ with the target residue $\mathbf{y}_{\text{target}}(n) - \mathbf{y}(n)$ to find its optimal delay and then its ${\mathbf{w}^{\text{out}}}_i$. The process is repeated until the delays converge and then the whole $\mathbf{W}^{\text{out}}$ is recomputed the usual way. This type of readout potentially enables utilizing the computational power of the reservoir more efficiently. In a time-coded output from a spiking reservoir the output connection delays can actually be the only thing that is learned [Paugam-Moisy et al., 2008].

For time series classification tasks the decision can be based on a readout from a joined reservoir state $\mathbf{x}_{\text{joined}} = [\mathbf{x}(n_1), \mathbf{x}(n_2), \ldots, \mathbf{x}(n_k)]$ that is a concatenation of the reservoir states from different moments $n_1, n_2, \ldots, n_k$ in time during the time series [Jaeger et al., 2007a]. This approach, compared to only using the last state of the given time series, moves the emphasis away from the ending of the series, depending on how the support times n_i are spread. It is also more expressive, since it has k times more trainable parameters in $\mathbf{W}^{\text{out}}$ for the same size of the reservoir N. As a consequence, it is also more prone to overfitting. It is also possible to integrate intervals of states in some way, e.g., use $\mathbf{x}^*(n_1) = \frac{1}{n_1 - n_0 + 1} \sum_{m=n_0}^{n_1} \mathbf{x}(m)$ instead of using a single snapshot of the states $\mathbf{x}(n_1)$.

An approach of treating a finite history of reservoir activations $\mathbf{x}(n)$ (similar to $\mathbf{X}$ in (2.12)) as a two-dimensional image, and training a minimum average correlations energy filter as the readout for dynamical pattern recognition is presented in [Ozturk and Príncipe, 2007].

2.8.7 Reservoir computing with non-temporal data

Even though in Section 2.1 we stated that the RNNs considered in this survey are used as nonlinear filters, which transform an input time series into an output time series, ESNs can also be utilized for non-temporal (defined in Section 2.2.1) tasks $\{(\mathbf{u}(n), \mathbf{y}_{\text{target}}(n))\}$ by presenting an ESN with the same input $\mathbf{u}(n)$ for many time steps letting the ESN converge to a fixed-point attractor state $\mathbf{x}^{\mathbf{u}(n)}(\infty)$ (which it does if it possesses echo state property) and reading the output from the attractor state $\mathbf{y}(n) = y(\mathbf{x}^{\mathbf{u}(n)}(\infty))$ [Embrechts et al., 2009; Reinhart and Steil, 2009].

An interesting approach of applying ESNs on tree-structured data is proposed in [Gallicchio and Micheli, 2010b]. For a binary tree (as a simple example) the states of the Tree-ESN are updated traversing the tree from the leaves to the root in every vertex v with information $\mathbf{u}(v)$ stored on it, the activation $\mathbf{x}(v)$ is

computed as $\mathbf{x}(v) = x\left(\mathbf{x}(v_l), \mathbf{x}(v_r), \mathbf{u}(v)\right)$, compared to (2.4), where v_l and v_r are the left and right children of v. Thus the current state is computed from two previous states with two "recurrent" connection matrices, traversing the edges of graph simultaneously toward the rood, instead of the time steps. The readout is done from the activation of the root node. Regular ESN can be seen as a special case of Tree-ESN where the input time series is considered as a directed tree where every node has just one child. The Tree-ESN approach can be extended to general graphs [Gallicchio and Micheli, 2010a].

2.8.8 Combining several readouts

Segmenting of the spatially embedded trajectory of $\mathbf{x}(n)$ by k-means clustering and assigning a separate "responsible" linear readout for each cluster is investigated in [Bush and Anderson, 2006]. This approach increases the expressiveness of the ESN by having k linear readouts trained and an online switching mechanism among them. Bigger values of k are shown to compensate for smaller sizes N_x of the reservoirs to get the same level of performance.

A benchmark-record-breaking approach of taking an average of outputs from many (1000) different instances of tiny ($N = 4$) trained ESNs is presented in Section 5.2.2 of [Jaeger et al., 2007a]. The approach is also combined with reading from different support times as discussed in Section 2.8.6 of this survey. Averaging outputs over 20 instances of ESNs was also shown to refine the prediction of chaotic time series in supporting online material of [Jaeger and Haas, 2004].

Using dynamic programing to find sequences in multiple sets of predicting readouts for classification [Skowronski and Harris, 2007] was already mentioned at the end of Section 2.8.3.

2.8.9 Reservoir as a kernel trick

Recently reservoir-inspired SVM-style Recurrent Kernel Machines (RKM) were proposed in [Hermans and Schrauwen, 2012]. They in effect work as infinite-dimensional RNNs, because an integral over all possible weight $\mathbf{W}^{\mathrm{in}}$ and $\mathbf{W}$ values is computed. As in SVMs, this is not done explicitly, but using the *kernel trick*: a dot product is computed between the input sequence $\mathbf{u}(n)$ and stored support sequences in the infinite-dimensional expansion space by means of the recurrent kernel function. Standard SVM output training techniques can be used with

recurrent kernels, such as maximum-margin classification or regression. Output weights correspond to the dot products with the support sequences, instead of the dimensions of the reservoir (which would be infinitely many in this case). Standard for SVMs Gaussian radial basis function as well as arcsine kernels were considered. The first corresponds to an infinite-dimensional RNN of Gaussian radial basis function neurons, and the second is very similar to an infinite-dimensional RNN of standard tanh sigmoid neurons, extending the idea of integrating over infinite-dimensional FFNNs in [Cho and Saul, 2010].

Despite the different readout mechanism, the authors of [Hermans and Schrauwen, 2012] consider ESN as a Monte Carlo approximation of an infinite-dimensional RNN represented by the arcsine RKM. Their empirical experiments show that the performance of ESN quickly approaches that of an RKM when increasing the number of units N_x. This indicates that ESN is still a more practical thing to use as even with big N_x it requires much smaller computational costs compared to RKMs. Nonetheless, we consider RKM a very interesting and important new development in machine learning.

2.9 Hierarchies

Following the analogy between the ESNs and non-temporal kernel methods, ESNs would be called "type-1 shallow architectures" according to the classification proposed in [Bengio and LeCun, 2007]. The reservoir adaptation techniques reviewed in our article would make ESNs "type-3 shallow architectures", which are more expressive. However, the authors in [Bengio and LeCun, 2007] and [Bengio, 2009] argue that any type of *shallow* (i.e., non-hierarchical) architectures is incapable of learning really complex intelligent tasks. Also in practice best performing systems on demanding tasks tend to be hierarchical, e.g., [LeCun et al., 1998; Hinton and Salakhutdinov, 2006; Graves and Schmidhuber, 2009; Cireşan et al., 2010]. This suggests that for demandingly complex tasks the adaptation of a single reservoir might not be enough and a hierarchical architecture within or of reservoirs might be needed. This is elaborated more in Chapter 3.

An example of such is presented in [Jaeger, 2007a]. Here the outputs of a higher level in the hierarchy serve as coefficients of mixing (or voting on) outputs from a lower one. The structure can have an arbitrary number of layers. Only the outputs from the reservoirs of each layer are trained simultaneously, using stochas-

tic gradient descent and error backpropagation through the layers. The structure is demonstrated to discover features on different timescales in an unsupervised way when being trained for predicting a synthetic time series of interchanging generators. On the downside, such hierarchies require many epochs to train, and suffer from a similar problem of vanishing gradients, as deep feedforward neural networks or gradient-descent methods for fully trained RNNs, because of the deep multilayered structure and recurrent bottom-up and top-down dependences. At the present form they do not scale-up yet to real-world demanding problems.

A bottom-up connected hierarchy of layer-wise greedily unsupervisedly trained radial basis function RNNs is proposed and investigated in Section 4.6 of this thesis, and was earlier in [Lukoševičius, 2010]. This hierarchy is also so far only shown to work good on a synthetic task.

A similarly bottom-up connected and greedily supervisedly trained hierarchy of large ESN reservoirs was shown to significantly improve phoneme recognition rates and achieve state-of-art performance in [Triefenbach et al., 2011]. Each layer was trained on the same target $\mathbf{y}_{\text{target}}(n)$. Upper layers learned to correct errors of previous layers by receiving just their imprecise output.

2.10 Discussion

The striking success of the original RC methods in outperforming fully trained RNNs in many (though not all) tasks, established an important milestone, or even a turning point, in the research of RNN training. The fact that a randomly generated fixed RNN with only a linear readout trained consistently outperforms state-of-art RNN training methods had several consequences:

- First of all it revealed that we do not really know how to train RNNs well, and something new is needed. The error backpropagation methods, which had caused a breakthrough in feedforward neural network training (up to a certain depth), and had also become the most popular training methods for RNNs, are hardly unleashing their full potential.

- Neither are the classical RC methods yet exploiting the full potential of RNNs, since they use a random RNN, which is unlikely to be optimal, and a linear readout, which is quite limited by the quality of the signals it is

combining. But they give a quite tough performance reference for more sophisticated methods.

- The separation between the RNN reservoir and the readout provides a good *platform* to try out all kinds of RNN adaptation methods in the reservoir and see how much they can actually improve the performance over randomly created RNNs. This is particularly well suited for testing various biology-inspired RNN adaptation mechanisms, which are almost exclusively local and unsupervised, in how they can improve learning of a supervised task.

- In parallel, it enables all types of powerful non-temporal methods to be applied for reading out of the reservoir.

This platform is the current *paradigm of RC*: using different methods for **(i)** producing/adapting the reservoir, and **(ii)** training different types of readouts. It enables looking for good (i) and (ii) methods independently, and combining the best practices from both research directions. The platform has been actively used by many researchers, ever since the first ESNs and LSMs appeared. This research in both (i) and (ii) directions, together with theoretical insights, like what characterizes a "good" reservoir, constitutes the modern field of RC.

In this review, together with motivating the new paradigm, we have provided a comprehensive survey of all this RC research. We introduced a natural taxonomy of the reservoir generation/adaptation techniques (i) with three big classes of methods (generic, unsupervised, and supervised), depending on their universality with respect to the input and desired output of the task. Inside each class, methods are also grouped into major directions of approaches, taking different inspirations. We have also surveyed all types of readouts from the reservoirs (ii) reported in the literature, including the ones containing several layers of nonlinearities, combining several time steps, or several reservoirs, among others. We also briefly discussed some practical issues of training the most popular types of readouts in a tutorial-like fashion.

The survey is transcending the boundaries among several traditional methods that fall under the umbrella of RC, generalizing the results to the whole RC field and pointing out relations, where applicable.

Even though this review is quite extensive, we tried to keep it concise, outlining only the basic ideas of each contribution. We did not try to include *every* contribution relating to RC in this survey, but only the ones highlighting the main

research directions. Publications only reporting applications of reservoir methods, but not proposing any interesting modifications of them, were left out. Since this review is aimed at a (fast) moving target, which RC is, some (especially very new) contributions might have been missed unintentionally.

In general, the RC field is still very young, but very active and quickly expanding. While the original first RC methods made an impact that could be called a small revolution, current RC research is more in a phase of a (rapid) evolution. The multiple new modifications of the original idea are gradually increasing the performance of the methods. While with no striking breakthroughs lately, the progress is steady, establishing some of the extensions as common practices to build on further. There are still many promising directions to be explored, hopefully leading to breakthroughs in the near future.

While the tasks for which RNNs are applied nowadays often are quite complex, hardly any of them could yet be called truly *intelligent*, as compared to the human level of intelligence. The fact that RC methods perform well in many of these simple tasks by no means indicates that there is little space left for their improvement. More complex tasks and adequate solutions are still to meet each other in RC. We further provide some of our (subjective, or even speculative) outlooks on the future of RC.

The elegant simplicity of the classical ESNs gives many benefits in these simple applications, but it also has some intrinsic limitations (as, for example, discussed in Section 2.5.4) that must be overcome in some way or other. Since the RNN model is by itself biologically inspired, looking at real brains is a natural (literally) source of inspiration on how to do that. RC models may reasonably explain some aspects of how small portions of the brain work, but if we look at the bigger picture, the brain is far from being just a big blob of randomly connected neurons. It has a complex structure that is largely predefined before even starting to learn. In addition, there are many learning mechanisms observed in the real brain, as briefly outlined in Section 2.6.2. It is very probable that there is no single easily implementable underlying rule which can explain all learning.

The required complexity in the context of RC can be achieved in two basic ways: either **(i)** by giving the reservoir a more complex internal structure, like that discussed in Section 2.5.3 or **(ii)** externally building structures combining several reservoirs and readouts, like those discussed in Section 2.9. Note that the two ways correspond to the above-mentioned dichotomy of the RC research

and are not mutually exclusive. An "externally" (ii) built structure can also be regarded as a single complex reservoir (i) and a readout from it all can be trained.

An internal auto-structuring of the reservoir (i) through an (unsupervised) training would be conceptually appealing and nature-like, but not yet quite feasible at the current state of knowledge. A robust realization of such a learning algorithm would signify a breakthrough in the generation/training of artificial NNs. Most probably such an approach would combine several competing learning mechanisms and goals, and require a careful parameter selection to balance them, and thus would not be easy to successfully apply. In addition, changing the structure of the RNN during the adaptive training would lead to bifurcations in the training process, as in [Doya, 1992], which makes learning very difficult.

Constructing external architectures of several reservoirs can be approached as more of an engineering task. The structures can be hand-crafted, based on the specifics of the application, and, in some cases, trained entirely supervised, each reservoir having a predefined function and a target signal for its readout. While such approaches are successfully being applied in practice, they are very case-specific, and not quite in the scope of the research reviewed here, since in essence they are just applications of (several instances of) the classical RC methods in bigger settings.

However, generic structures of multiple reservoirs (ii) that can be trained with no additional information, such as discussed in Section 2.9, are of high interest. Despite their current state being still an "embryo", and the difficulties pointed out earlier, we see this direction as highly promising. We propose a contribution towards this direction in Chapter 4 of this thesis.

Biological inspiration and progress of neuroscience in understanding how real brains work are beneficial for both (i) and (ii) approaches. Well understood natural principles of local neural adaptation and development can be relatively easily transfered to artificial reservoirs (i), and reservoirs internally structured to more closely resemble cortical microcolumns in the brain have been shown to perform better [Haeusler and Maass, 2007]. Understanding how different brain areas interact could also help in building external structures of reservoirs (ii) better suited for nature-like tasks.

In addition to processing and "understanding" multiple scales of time and abstraction in the data, which hierarchical models aim to solve, other features still lacking in the current RC (and overall RNN) methods include robustness and

stability of pattern generation. A possible solution to this could be a homeostasis-like self-regulation in the RNNs. Other intelligence-tending features as selective longer-term memory or active attention are also not yet well incorporated.

In short, RC is not the end, but an important stepping-stone in the big journey of developing RNNs, ultimately leading towards building artificial and comprehending natural intelligence.

Chapter 3

Reservoirs, Hierarchies, and Learning

As noted in Chapter 1, the current state in ML is nowhere close in its capabilities to human level intelligence. We see two important related reasons for that:

- The current artificial ML systems are just too simple.
- Purely supervised learning is an inadequate paradigm.

We are not claiming that these two reasons are the only one. In fact, we have pointed out more in Chapter 1.

Let us elaborate on these two points in the following two sections.

3.1 Toward more complex ML architectures

The current "AI" systems are just too simple to approach real intelligence. Looking at the scale and complexity of brains of higher animals and of the current ML systems, the gap is obvious.

Realizing this, a current trend in state of art ML is from simple monolithic architectures to more complex ones having more layers and/or components. The trend brings machine learning closer to the field of cognitive architectures.

There is a growing consensus that the old type "shallow" architectures are insufficient for more demanding tasks in machine learning that tend toward artificial intelligence and multilayer "deep" architectures are needed [Bengio and LeCun, 2007; Bengio, 2009].

From a computational perspective, the deep models are much more expressive and capable of learning less trivial (non-local) generalizations, or features, concepts, from the data [Bengio, 2009]. This multilayer "deep" approach also matches our basic knowledge about natural brains, for example the visual processing stream in higher animals. There is also an abstract correspondence to the way humans think in hierarchies of concepts.

Also empirically, almost all the best performing neural image processing methods are deep [LeCun et al., 1998; Hinton and Salakhutdinov, 2006; Graves and Schmidhuber, 2009; Cireşan et al., 2010].

One of the main reasons why large complex ML architectures have not been successfully applied in the past is that they are a big challenge for learning algorithms. Classical supervised learning algorithms based on following error gradients are often inefficient, because the error gradients are only easy to estimate close to the output points of an architecture where the desired "target" signals is available (the error is just the difference between the actual and the desired output signals). The further from these points the error gradients are back-propagated the more they get "diluted" and the less effective the learning becomes in these areas, as already hinted at in Section 1.5.

It has been recently shown that deep FFNNs can successfully be trained by first training them unsupervisedly in a greedy layer-by-layer fashion from bottom, where the input comes, up and then fine-tuning with a classical supervised error backpropagation (BP) method from top, where the output is produced, down [Hinton and Salakhutdinov, 2006]. Such approach is more efficient and gives better final results than training only with the supervised BP [Erhan et al., 2010]. A few examples of successfully fully supervisedly trained deep networks usually employ other tricks, like severe restriction of the number of trainable parameters by the convolutional application of the network [LeCun et al., 1998; Graves and Schmidhuber, 2009].

Recurrence is another type of complexity which has to be present for truly intelligent behavior, because intelligent agents, mathematically speaking, are not memoryless input-output mapping functions, but rather dynamical systems embedded and acting in time.

It is worth mentioning at this point that the state of art deep neural learning architectures are largely still non-recurrent [LeCun et al., 1998; Hinton and Salakhutdinov, 2006; Cireşan et al., 2010]. In particular, few of them are ade-

quately dealing with temporal signals.

Quite some of the mentioned learning challenges are common between the deep FFNNs and the recurrent systems. In fact, a recurrent NN "unfolded" in time can be seen as an infinitely deep FFNN. Despite the commonalities, the combined complexity of the two makes deep recurrent neural architectures particularly challenging.

A few temporal hierarchical systems are mentioned in Section 2.9. A successful example of fully supervisedly trained one was recently presented in [Graves and Schmidhuber, 2009]. In Chapter 4 we will propose another, unsupervisedly trained one.

3.2 Why unsupervised learning is important

Let us summarize the reasons why unsupervised learning is important for training intelligent ML systems:

- Such systems will be too big and complex to be efficiently trained by purely supervised methods, as we explained in Section 3.1.
- Current large RNNs and deep FFNNs are already trained sub-optimally by purely supervised methods as demonstrated by RC (Chapter 2) and e.g. [Erhan et al., 2010].
- Labeled data is too scarce to only learn from it in most domains tending to general-purpose AI, as pointed out in Section 1.5.
- There are hardly any purely supervised tasks in nature. Intelligent agent has to "learn about the world" to be able to perform many different tasks on demand. It would be impossible to train a general-purpose AI in a purely supervised way.
- Purely supervised learning does not give rise to any "creative" original solutions. It is also not generative: it does not help to create the structure of the model, only to fine-tune its parameters towards correct output.
- Natural neural systems are well known for their unsupervised adaptivity. They are self-regulating by many mechanisms, on many timescales and for

several purposes: optimizing performance, dynamical stabilization, homeostasis, etc. Even neurons scattered on a dish are known to organize themselves into living networks. Many of these mechanisms remain largely unknown.

This list is, again, not exhaustive. In particular, for now, in this thesis, we are interested in unsupervised learning as means of training more complex RNN architectures.

We have reviewed many alternative approaches for training RNNs in Chapter 2, including unsupervised ones in Section 2.6, but they are not powerful enough for this purpose (or at least not reported to be). In particular, we do not know any unsupervisedly fully trained deep recurrent neural architectures reported in the literature, probably the closest of such being [Jaeger, 2007a].

We will propose and investigate a use of a new type of network together with powerful unsupervised learning algorithms to train RNN reservoirs as building blocks for deep recurrent architectures in Chapter 4

Chapter 4

Self-organized Reservoirs and their Hierarchies

4.1 Introduction

Despite the biological plausibility and theoretical computational universality of artificial Recurrent Neural Networks (RNNs), their practical applications are still scarce. This should arguably be attributed to their training being far from trivial. While many algorithms for training RNNs exist, they usually require a high level of expertise and do not scale up well to large networks. There are likely even theoretical limitations to using gradient descent training techniques in such networks [Doya, 1992; Bengio et al., 1994]. One fresh strain of RNN training approaches has even abandoned training the recurrent part of the network at all. The strain was pioneered by Echo State Networks (ESNs) [Jaeger, 2001] in machine learning and Liquid State Machines [Maass et al., 2002] in computational neuroscience and is increasingly referred to as Reservoir Computing (RC), as reviewed in Chapter 2. The fact that a simple ESN having a *randomly* generated recurrent part (called *reservoir*) and only a readout from it trained is outperforming sophisticated RNN training algorithms in many tasks [Jaeger and Haas, 2004; Jaeger et al., 2007a; Verstraeten et al., 2006] is in a way odd if not embarrassing. Intuitively, there should be something better than a random network. It is also not well understood what makes a reservoir good for a particular task. Dissatisfied with this situation many researchers are looking for such qualities and for novel RNN (reservoir) adaptation algorithms (see Chapter 2 for an overview).

A part of our contribution in this chapter is along the lines of these efforts reviewed in Section 2.6. We focus our attention on an *unsupervised* pre-training of the reservoir followed by a supervised training of the readout. These are the middle grounds between a fixed reservoir and a fully in a supervised way trained RNN, hopefully mitigating the shortcomings of both. Among the benefits of the unsupervised pre-training are the ability to utilize unlabeled data (input signals with no corresponding target outputs) that are abundant in many applications and the ability to use such reservoirs as building blocks for more complex hierarchical architectures since they do not require an external error signal back-propagated through other complex components and diluted in the process (See Section 3.2 for more).

The latter is the main original motivation to this research and is further explored in this chapter. We investigate architectures of multiple layers of such reservoirs that are greedily trained in an unsupervised way. It has been recently argued [Bengio, 2009] and widely believed in the machine learning community that such deep architectures are necessary for more challenging tasks.

The chapter is organized as follows. We specify our self-organizing reservoir in Section 4.2, together with a short discussion on the properties of the neuron model used in Section 4.2.2, and the training algorithms used in Section 4.3, both the unsupervised training in Section 4.3.1 and the readout mechanism in Section 4.3.2. We compare the self-organizing reservoirs with random networks of the same type and simple ESNs in Section 4.4 and show that they are consistently better in the investigated two tasks. More concretely, we specify the baseline ESN in Section 4.4.1, the two tasks used in Sections 4.4.2 and 4.4.3, and the technical details of the simulations in Section 4.4.4. We then analyze the results of the comparison in Section 4.4.5. We further investigate the scalability of our computational model with random weights in Section 4.4.6, and point out a possible limitation. Then in Section 4.5 we propose an alternative generalized model that we call weighted difference network, that alleviates these limitations, discuss a few alternatives for a better unsupervised training for the model in Section 4.5.2, and demonstrate that the model offers superior performance in Section 4.5.3. Utilizing the benefits of the self-organizing reservoirs we build multi-layered hierarchies of them in Section 4.6, giving the technical details in Section 4.6.1 and analyzing the success in Section 4.6.2.

4.2 Computational model

4.2.1 Self-organizing reservoir

There are quite some unsupervised adaptation techniques for the "weighted sum and nonlinearity" (WSN) type of neurons suggested and recently investigated in the RC context, however often the improvements they offer are minute and the adaptation is only partial, not modifying all the free parameters (see Section 2.6 for an overview). Instead, we focused our attention to the tradition of Self-Organizing (also called Kohonen) Maps (SOMs) [Kohonen, 1982], probably the most classic unsupervised neural network training method, and their recurrent extensions. While there are many extensions of SOMs to temporal data suggested (see [Hammer et al., 2004] for a systematic overview), the one truly fully recurrent model, as in normal WSN fully recurrent networks, was introduced as Recursive Self-Organizing Maps (RSOMs) in [Voegtlin, 2002]. In this work we use a similar model for our Self-Organizing Reservoir (SOR) with the state update equations

$$\tilde{x}_i(n) = \exp\left(-\alpha\left\|\mathbf{v}^{\text{in}}{}_i - \mathbf{u}(n)\right\|^2 - \beta\|\mathbf{v}_i - \mathbf{x}(n-1)\|^2\right), \quad i = 1, \ldots, N_{\text{x}}, \tag{4.1}$$

$$\mathbf{x}(n) = (1-\gamma)\mathbf{x}(n-1) + \gamma\tilde{\mathbf{x}}(n), \tag{4.2}$$

where $\mathbf{u}(n) \in \mathbb{R}^{N_{\text{u}}}$ is the input signal, $\mathbf{x}(n) \in \mathbb{R}^{N_{\text{x}}}$ is a vector of reservoir neuron activations and $\tilde{\mathbf{x}}(n) = [\tilde{x}_1(n), \ldots, \tilde{x}_{N_{\text{x}}}(n)]^{\text{T}} \in \mathbb{R}^{N_{\text{x}}}$ is its update, all at time step n, $\|\cdot\|$ stands for the Euclidean norm, $\mathbf{v}^{\text{in}}{}_i \in \mathbb{R}^{N_{\text{u}}}$ is the ith column of the input weight matrix $\mathbf{V}^{\text{in}} \in \mathbb{R}^{N_{\text{u}} \times N_{\text{x}}}$, $\mathbf{v}_i \in \mathbb{R}^{N_{\text{x}}}$ is the ith column of the recurrent weight matrix $\mathbf{V} \in \mathbb{R}^{N_{\text{x}} \times N_{\text{x}}}$, $\gamma \in (0, 1]$ is the leaking rate, and α and β are scaling parameters for the input and the recurrent distances respectively.

Our SORs are different from RSOMs of [Voegtlin, 2002], in that we use leaky integration (4.2), which makes our model also resemble the earlier temporal Kohonen networks [Chappell and Taylor, 1993] and recurrent self-organizing maps [Varsta et al., 1997] that use leaky integration as the only type of recurrence. The unit activation function (4.1) is a Gaussian and in fact can be seen as a Radial Basis Function (RBF). However, to the best of our knowledge, this type of fully recurrent systems has not been investigated in the RBF literature, the closest of such being the recurrent RBF network [Mak, 1995] which is similar to the temporal Hebbian SOM [Koutník, 2007]. There seem to not be any citations between

the two communities.

A recent biologically-motivated contribution with similar objectives was introduced in [Lazar et al., 2010; Lazar, 2010]. Also, RSOMs are used as a pre-processor in the context of reservoir computing in [Farkaš and Crocker, 2007].

4.2.2 Properties of the neuron

The difference between this model and a more conventional RNN as, for example, in ESNs is the model of neuron: the WSN type $x = f(\mathbf{wu})$ versus the RBF type $x = f(\|\mathbf{v} - \mathbf{u}\|)$, where $f(\cdot)$ stands for a nonlinearity, typically a $\tanh(\cdot)$ sigmoid in the first case and a Gaussian in the second. Even more specifically it is how the inputs to the neurons are combined. As a result RBF neurons have some very different properties from the WSN neurons:

- Locality. By the virtue of calculating Euclidean distance $\|\mathbf{v} - \mathbf{u}\|$ between the vectors of its inputs $\mathbf{u}$ and the input weights $\mathbf{v}$ (as opposed to dot product $\mathbf{wu}$ in WSN units), responses of RBF units are intrinsically local. The response is centered around $\mathbf{v}$ as the "prototype" input pattern that excites the unit most. The excitation drops sharply when $\mathbf{u}$ moves away from $\mathbf{v}$.

- Prototype input pattern. The prototype input pattern $\mathbf{u} = \mathbf{v}$ is bounded for RBF units as opposed to WSN units where it is asymptotic: the bigger the scalar projection of $\mathbf{u}$ on $\mathbf{v}$, the bigger the output.

- Quasi-sparse coding. A group of RBF units receiving the same input $\mathbf{u}$ and having different weights $\mathbf{v}$ produces a sparse spatial coding of $\mathbf{u}$ in a sense that only a few units, $\mathbf{v}$ of which are closest to $\mathbf{u}$, have higher activation. This however is not sparse coding in the strict sense of the term [Foldiak and Endres, 2008] but something in between a sparse coding and a local coding. On one hand, units are excited by their local prototype patterns, on the other hand they have continuous activations and in general any input can be reconstructed from their activations as long as the number of units in the population is greater than the dimensionality of the input and the units are not on a lower-dimensional hyperplane in the input space.

- Greater nonlinearity. The response of a RBF unit is more nonlinear than

that of the sigmoidal activation function of WSN units, which is monotonic and changes only along one direction of the input space.

- Signal value invariance. Prototype inputs of RBF units can be placed anywhere in the input space, treating any value on the axis of real numbers like any other. For example, an RBF unit can have its maximal response to the input with all values 0 (if $\mathbf{v} = \mathbf{0}$), while an WSN unit will have a zero activation independent of its $\mathbf{w}$. This gives more flexibility and requires less care when encoding the inputs.

- No signal energy metaphor. A closely related issue is that with RBF units the notion of signal energy is lost. Higher energy, or amplitude, input signals $\mathbf{u}$ do not automatically result in higher energy, or amplitude, output signals $\mathbf{x}$ and vice versa. In particular, because the zero input signal has no special meaning, there is no "dying-out" of the signals due to zero input.

Some of these features are desirable in a reservoir while others are daunting. In particular, recurrent networks of such units are hard to analyze analytically, because many insights on the WSN type of reservoirs, such as the role of the spectral radius of the recurrent connection matrix [Jaeger, 2001], are based on their local linearization, which is hard to apply here. The derived stability conditions for RSOMs [Tiňo et al., 2006] are not as elegant as for reservoirs of WSN units and depend on particular inputs $\mathbf{u}(n)$: RSOM produces a contractive mapping if

$$\beta < \frac{e}{2 \sum_{i=1}^{N_\mathrm{x}} \exp\left(-2\alpha \|{\mathbf{v}^{\mathrm{in}}}_i - \mathbf{u}(n)\|^2\right)}. \tag{4.3}$$

Since this condition is for RSOM, it does not take leaking rate γ (4.2) into account (is only valid for $\gamma = 1$).

In a way, the RBF units in a recurrent network work as detectors of the state of the dynamical system while at the same time constituting the next state by representing the dimensions of it. Each of them "detects" a certain situation in the combined space of the input and its history. This makes such a reservoir a natural candidate for temporal pattern detection or classification tasks.

To test whether and by how much this different computational model affects the reservoir computations compared to standard ESNs, we will include random networks of both kinds in our empirical simulations.

The specific properties of the RBF units allow them to be trained by some state-of-art unsupervised methods.

4.3 Training

The training of our model consists of the unsupervised pre-training of the reservoir (4.1) and a supervised training of a readout from it to assess the effects of unsupervised adaptation.

4.3.1 Unsupervised training of the reservoir

Based on the fact that input weights of the RBF units "dwell" in the same space as their inputs, there is a large range of unsupervised training methods available for them. Most of them combine competitive and collaborative aspects of learning to make the units nicely span the input space. Virtually all such methods for static data are also applicable to our recurrent model (4.1). One natural option for training (4.1) is the classical SOM learning algorithm [Kohonen and Honkela, 2007]:

$$\begin{aligned} \mathbf{v}^{\text{in}}{}_i(n+1) &= \mathbf{v}^{\text{in}}{}_i(n) + \eta(n) h(i,n) \left(\mathbf{u}(n) - \mathbf{v}^{\text{in}}{}_i(n) \right), \\ \mathbf{v}_i(n+1) &= \mathbf{v}_i(n) + \eta(n) h(i,n) \left(\mathbf{x}(n) - \mathbf{v}_i(n) \right), \end{aligned} \tag{4.4}$$

with $\eta(n)$ being the learning rate and the learning gradient distribution function

$$h(i,n) = \exp\left(-\frac{d_h(i, \text{bmu}(n))^2}{b_h(n)^2} \right), \tag{4.5}$$

where $\text{bmu}(n) = \arg\max_i (x_i(n))$ is the index of the "best matching unit" (BMU), $d_h(i,j)$ is the distance between units with indices i and j in the additionally defined topology for reservoir units, and $b_h(n)$ is the neighborhood width of the gradient distribution. In our experiments we use a 2D rectangular lattice where $d_h(i,j)$ is the Manhattan distance between nodes i and j on it. Intuitively, $h(i,n)$ distributes the error gradient in (4.4) so that the BMU is updated with the biggest individual learning rate (since $h(\text{bmu}(n), n) \equiv 1$) and this rate drops as a smooth Gaussian going further away from the BMU in the defined topology of units. Note, that we are using $\mathbf{x}(n)$ for finding $\text{bmu}(n)$ as in recurrent SOMs [Varsta et al., 1997]

as opposed to using $\tilde{\mathbf{x}}(n)$ as in temporal Kohonen networks [Chappell and Taylor, 1993]. $\eta(n)$ and $b_h(n)$ control the *schedule of the training* process by varying the overall learning rate and amount of learning done outside the BMU at each time step respectively.

Neural gas (NG) [Martinetz and Schulten, 1991] is another closely related alternative learning method to SOMs that we tried. It differs only in the gradient distribution function, which instead of (4.5) is

$$h_{ng}(i,n) = \exp\left(-\frac{d_{ng}(i,n)}{b_h(n)}\right), \tag{4.6}$$

where $d_{ng}(i,n)$ is the zero-based index of the node i in the descending ordering of nodes by their $x_i(n)$. As in the case of SOM, $d_{ng}(\mathrm{bmu}(n),n) \equiv 0$ and $h(\mathrm{bmu}(n),n) \equiv 1$. In our experiments that are reported later we got similar results with both SOM and NG training.

4.3.2 Supervised training of the readouts

After unsupervised adaptation of such a reservoir to the given input $\mathbf{u}(n)$ a readout $\mathbf{y}(n) \in \mathbb{R}^{N_\mathrm{y}}$ from the reservoir can be trained in a supervised way to match a desired output $\mathbf{y}_\mathrm{target}(n) \in \mathbb{R}^{N_\mathrm{y}}$. A natural and straightforward option is to use a linear readout

$$\mathbf{y}(n) = \mathbf{W}^\mathrm{out}[1;\mathbf{x}(n)], \tag{4.7}$$

where $[\cdot;\cdot]$ stands for a vertical vector concatenation. The output weight matrix $\mathbf{W}^\mathrm{out} \in \mathbb{R}^{N_\mathrm{y}\times(N_\mathrm{x}+1)}$ is learned using linear regression, a standard technique in reservoir computing [Jaeger, 2007b]. The input $\mathbf{u}(n)$ can also be concatenated to $[1;\mathbf{x}(n)]$ in (4.7), making $\mathbf{W}^\mathrm{out} \in \mathbb{R}^{N_\mathrm{y}\times(N_\mathrm{u}+N_\mathrm{x}+1)}$.

In this work we will not put much emphasis on designing elaborate output schemes for particular applications (which would be important for tasks like classification or detection), but rather use simple linear outputs trained on simple targets to estimate the quality of the unsupervised adaptation in $\mathbf{x}(n)$.

4.4 Empirical comparison of SORs and ESNs

We made a systematic comparison between self-organizing reservoirs, both randomly and unsupervisedly pre-trained, and classical random echo state networks.

We will first specify the ESNs used here for the sake of completeness in Section 4.4.1, describe the data on which all the experiments are run in Section 4.4.2, give the technical details of the numerical simulations in Section 4.4.4, and analyze the results in Section 4.4.5.

4.4.1 Baseline ESN architecture

We compare our self-organizing reservoirs presented in Section 4.2.1 to reservoirs of classical echo state networks [Jaeger, 2001, 2007b] with the update equation

$$\tilde{\mathbf{x}}(n) = f\left(\mathbf{W}^{\text{in}}[1; \mathbf{u}(n)] + \mathbf{W}\mathbf{x}(n-1)\right), \tag{4.8}$$

where $f(\cdot) = \tanh(\cdot)$ is the neuron activation function applied element-wise, instead of (4.1). Here the input weight matrix $\mathbf{W}^{\text{in}} \in \mathbb{R}^{N_{\text{x}} \times (N_{\text{u}}+1)}$ is a randomly-generated matrix with elements uniformly distributed in a range set by the *input scaling* parameter. We denote input scaling "in.s." for brevity: elements of $\mathbf{W}^{\text{in}}$ are uniformly distributed in the interval $[-\text{in.s.}, +\text{in.s.}]$. The recurrent weight matrix $\mathbf{W} \in \mathbb{R}^{N_{\text{x}} \times N_{\text{x}}}$ is a random sparse matrix with 20% connectivity and scaled to have a predefined spectral radius $\rho(\mathbf{W})$. The rest of the model, including leaky integration (4.2) and readout (4.7), is the same.

4.4.2 Synthetic smooth temporal pattern dataset

To investigate the effects of unsupervised pre-training in more controlled conditions we use a synthetically generated smooth temporal pattern dataset (Figure 4.1) in our simulations, the same as in some of our previous work [Lukoševičius et al., 2006; Jaeger et al., 2007a]. It is essentially a multidimensional red noise background signal with smoothly embedded short temporal patterns. Both the background signal and the patterns are generated in the same way by low-pass filtering white noise. Both the background signal and the patterns have the same amplitude and frequency makeup. The patterns are embedded into the background signal by applying smooth envelopes that also have the same frequency makeup. At the places where the patterns are embedded the background signal is suppressed by the envelope and the pattern is accordingly multiplied by it, producing smooth cross-fades between the background and the pattern. The pattern signals have the same dimensionality as the background and appear in all dimen-

sions of the background simultaneously. All the dimensions in the background and inside the patterns are generated as independent signals. The patterns are chosen randomly with equal probabilities and embedded at randomly varying intervals in the background. The patterns do not overlap. Almost half on the final signal is constituted by the patterns, the rest is the background. Thus, the more different patterns there are, the rarer they appear in the signal. The average length of the pattern in the final signal is about 20 time steps. The whole signal in addition is moderately time-warped: the original time step of size 1 can obtain values from the interval $[0.5, 1.5]$ during the transformation. See section 6.3 in [Lukoševičius et al., 2006] for more technical details on how the data were produced.

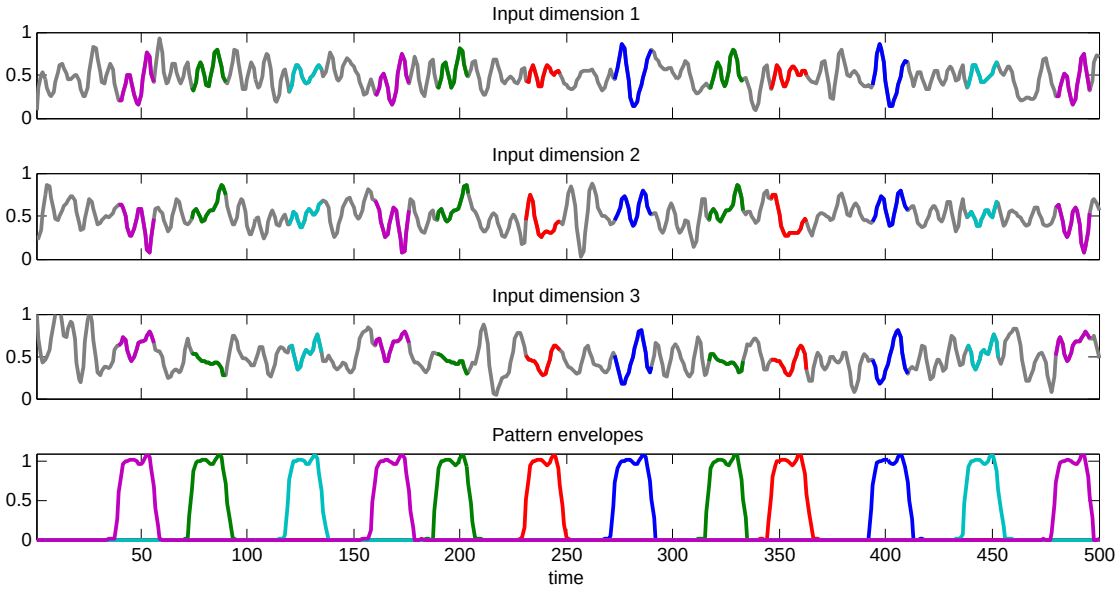


Figure 4.1: A sample of a three-dimensional smooth pattern dataset with five different patterns highlighted in colors and their corresponding envelopes.

The dataset was originally designed having handwriting in mind. The background can be seen as the unknown handwriting characters and the patterns as frequently reappearing letters to be recognized in it. Everything can be seen as encoded in, for example, pen movement data, or features extracted from sequentially scanning the handwriting and treating it as a time series.

In our experiments for unsupervised training we only use the data for input $\mathbf{u}(n)$ with no targets. The general idea is to test how well the unsupervised models learn the structure of the data. To evaluate this we estimate how well the patterns are separable form the rest of the signal in the reservoir activation

space $\mathbf{x}(n)$. More concretely, we test how well the envelopes of the patterns can be recovered from it. For this we train a supervised output (4.7) using a signal containing the envelopes with which the N_y patterns were inserted as the training target $\mathbf{y}_\mathrm{target}(n) \in \mathbb{R}^{N_\mathrm{y}}$.

This data is difficult in several aspects:

- The background can literally become very similar to the pattern. This makes perfect learning impossible. The lower the dimensionality of the signal the higher the probability of this happening.
- Because the background signal and the patterns (including transitions) have the same amplitude and frequency makeup there are no "cheap tricks" to spot them without looking at the exact shape of the signals. In fact the patterns are not easy to see in the signal by the naked eye if they are not highlighted.
- The background signal is very information-rich because it is not repeating itself. This can be a problem for some unsupervised models.
- Time warping and relative slowness of the signal can be problematic for some models.

4.4.3 Concatenated USPS digits dataset

To investigate the effects of the unsupervised pre-training on a more real-life-like data, we generate long strings of handwritten digits. The individual digits were taken from the well-known USPS dataset[1] [LeCun et al., 1989]. The dataset consists of digits "0" through "9" with 1100 examples of each class. They are in fact originally cut out from longer scanned handwritten digit sequences and preprocessed. Each image consists of 16×16 8-bit gray-scale pixels that were scaled to the $[0, 1]$ interval.

The input strings are generated by concatenating the digit images horizontally, selecting each digit class and image instance randomly with equal probability. Every digit image is placed after the preceding image with an up to ± 3 pixels random uniformly distributed displacement, such that the images can overlap or leave a gap of up to three pixels. The gaps are filled with zero-valued background

[1] Downloaded from Sam Roweis' homepage http://www.cs.nyu.edu/~roweis/data.html.

and overlapping images are merged by taking maxima of the pixel values. The vertical size of the images corresponds to the $N_{\mathrm{u}} = 16$ dimensions of the input $\mathbf{u}(n)$, and the x-axis is treated as time n, pixels corresponding to time steps, scanning the images from left to right.

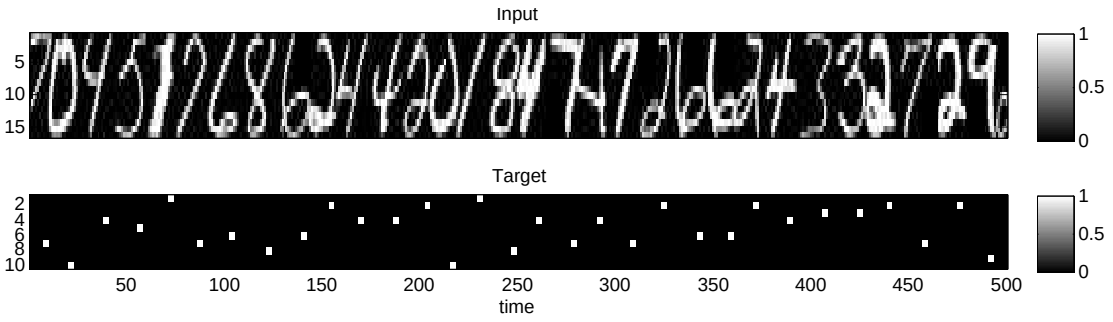


Figure 4.2: A sample of USPS concatenated dataset.

To test how the unsupervised pre-training aids digit recognition task, we also generate targets corresponding to the input signals. The target signals $\mathbf{y}_{\mathrm{target}}(n)$ have $N_{\mathrm{y}} = 10$ dimensions, one for every digit class "1",...,"9","0". The target signal $\mathbf{y}_{\mathrm{target}}(n)$ is raised to the value one in one of the ten channels whenever a digit of the corresponding class appears in the input and is equal to zero everywhere else. Each such class indicator lasts three time steps, corresponding to the 12th to 14th step out of the 16 columns of the digit image. These values were found to work well with all the investigated models through some experimentation.

A sample of the generated input data together with a corresponding target signal is presented in Figure 4.2.

4.4.4 Simulation details

We will first describe the simulation details that are common for the both datasets introduced in Sections 4.4.2 and 4.4.3, and then delve into specifics in the respective Subsections 4.4.4.1 and 4.4.4.2.

Since both datasets are generated randomly and the difficulty of the task depends on the concrete instance, we generate ten samples of each data and run all the experiments on these ten different instances of the data to get the statistics of the performance.

We use long enough data sequences so that overfitting was found not to be an

issue. Thus, selection of the best parameters is done based on the performance on the training sets, averaged over the ten data instances.

To keep things fast and manageable we use reservoirs of size $N_\mathrm{x} = 50$ for both the proposed SOR model and the regular ESN. Since the readout part of the model is always (4.7) with the concatenated input, the models in every task have the same number of parameters – namely $\mathbf{W}^\mathrm{out} \in \mathbb{R}^{N_\mathrm{y} \times (N_\mathrm{x}+N_\mathrm{u}+1)}$ – that are trained in the same supervised way.

Parameters trained in a supervised way can not by directly compared with the ones trained in an unsupervised way, because information about the desired output is not available in unsupervised training. In fact, it is not self-evident that unsupervised training helps the supervised one at all—this is that we want to investigate. Thus, we only compare the models with equal number of supervisedly trained parameters despite some of them having more parameters trained unsupervisedly.

To investigate how much the performance is influenced by the different computational model of SOR (4.1), compared to (4.8), and how much by the unsupervised training (4.4), we tested this model with both randomly generated weights and unsupervisedly pre-trained. For the random model, which we refer to as "random SOR", the input $\mathbf{V}^\mathrm{int}$ and recurrent $\mathbf{V}$ weights were all drawn from a uniform $[0, 1]$ distribution.

Both SOR and ESN computational models have a number of parameters that need to be set that influence their performance. To have a fair comparison where none of the models is disadvantaged by parameter misconfiguration, we have run grid searches over the three parameters that affect the performance most in both of the models. This is a common technique in machine learning, as sketched in Section 1.4. For the SOR these three parameters are: input and recurrent distance scalings α and β in (4.1), and the leaking rate γ in (4.2). For the regular ESNs the three similar parameters are: the scaling of the input matrix $\mathbf{W}^\mathrm{in}$ which we denote "in.s." for brevity (such that the elements of $\mathbf{W}^\mathrm{in}$ are uniformly distributed in $[-\text{in.s.}, +\text{in.s.}]$), the spectral radius of the recurrent connection matrix $\rho(\mathbf{W})$ (4.8), and the same leaking rate γ in (4.2).

The $N_\mathrm{x} = 50$ units of the self-organizing reservoirs are conceptually arranged into a 2D rectangular lattice topology of 10×5 units for the SOM type (4.5) of unsupervised training. The training schedule parameters for the self-organizing reservoirs are set to the values that were found to be reasonable through some

manual experimentation.

The input weight matrix $\mathbf{V}^{\text{in}}$ of the self-organizing reservoir (4.1) is initialized so that the two dimensions of the 10×5 unit lattice are aligned with the two principal components of the input $\mathbf{u}(n)$ within the input space, with input vectors $\mathbf{v}^{\text{in}}{}_i$ of the units equally spaced along these dimensions, spreading five times the standard deviation of the input along the corresponding principal component, in both positive and negative directions from the mean. This is a classical SOM initialization method that makes the network capture the two principal components of the input before the training. We use the same initialization also when training with NG algorithm, even though the concept of the 2D lattice is not used in NG training itself. The recurrent connections $\mathbf{V}$ were simply initialized as the identity matrix $\mathbf{V}(0) = \mathbf{I}_{N_\text{x}}$ in both cases, biasing the network for slower activation dynamics.

The specifics of the simulations for the two datasets follow.

4.4.4.1 Smooth synthetic patterns

In addition to the ten data instances we also run the experiments with a different number of patterns (corresponding to the dimensionality of the target N_y) in the data, ranging from one to five. The input dimension of the data is always $N_\text{u} = 3$.

The training data of 50 000 time steps is used, of which the first 500 are discarded from training and used to "wash out" the initialization transient in the reservoirs. The best parameters were chosen the ones that give the smallest training error. Testing was also performed on continuations of the datasets of length 10 000. Testing errors match very closely the training errors (both presented in the thesis) proving that overfitting is not happening. The exact same setup was used for both the proposed and the standard ESN models.

All the three dimensions of input data are normalized to have 0.01 variance and zero mean. For self-organizing reservoirs the input data is then shifted to have a 0.5 mean, such that it lies almost entirely inside the $[0, 1]$ interval. This is done for convenience since activations $\mathbf{x}(n)$ and inputs $\mathbf{u}(n)$, and thus subsequently the weights $\mathbf{V}$ and $\mathbf{V}^{\text{in}}$ stay roughly in the same $[0, 1]$ interval. For ESNs the data was left at zero mean, as this was found to give slightly better results.

As the performance criterion for all the models we use the normalized root mean square error (NRMSE) (2.1) of the pattern envelopes reconstructed from $\mathbf{x}(n)$ and $\mathbf{u}(n)$ using linear regression (4.7), as mentioned in Section 4.4.2. This

way both models have the same number $N_x + N_u + 1 = 54$ of parameters per pattern that are trained in a supervised way. The pattern envelopes on which the outputs are trained (and performance evaluated) are delayed by one time step, as such shift was found to give the best performance.

The unsupervised training is done for either SOM or NG by passing through the training data once, that is in 49 500 time steps. The learning rate $\eta(n)$ (4.4) follows a geometric progression from 0.04 to 0.01. The neighborhood width $b_h(n)$ for the SOM training algorithm (4.5) follows a geometric progression from 2 to 0.001 and for the NG (4.6) algorithm from 10 to 0.002. NG usually requires broader gradient distribution neighborhoods $b_h(n)$ than SOM, because the NG neighborhood d_{ng} is in effect one-dimensional, while d_h of SOM is two-dimensional.

The readouts are trained by passing through the training data once (again) for all the models: the random ESN reservoirs or the unsupervisedly trained ones.

4.4.4.2 Concatenated USPS digits

We generate ten training and ten testing sequences of 64 000 time steps in a way described in Section 4.4.3. Each sequence on average contains 4 000 digit images. First 160 time steps in all models are used to "wash out" the initialization transient in the reservoirs.

The unsupervised training is done for either SOM or NG by passing through the training data once, that is in 63 840 time steps. The learning rate $\eta(n)$ (4.4) follows a geometric progression from 0.05 to 0.001. The neighborhood width $b_h(n)$ for the SOM training algorithm (4.5) again follows a geometric progression from 2 to 0.001 and for the NG (4.6) algorithm from 10 to 0.002.

The same type of readouts (4.7) from all the models are used again, each having $N_y \times (N_x + N_u + 1) = 5170$ parameters that are trained supervisedly by linear regression, running through the same training data.

We use two different performance criteria for all the models with this data. The first one is the same NRMSE (2.1) between the model output $\mathbf{y}(n)$ and the target $\mathbf{y}_{\text{target}}(n)$ containing the class indicators. This criterion is explicitly minimized by the supervised readout training.

We use an additional *classification rate* criterion to better evaluate how well the output signal $\mathbf{y}(n)$ is suited for making the discrete classification decisions. More specifically, extracting the actual sequence of digits $s(m) \in \{1, \ldots, 10\}$, where $m = 1, ..., T_s$ is the position of the symbol in the sequence from $\mathbf{y}(n)$, where

$n = 1, ..., T_{\mathrm{y}}$. Note, that m is a much slower time scale than n, in our case $T_{\mathrm{y}} \approx 16 T_{\mathrm{s}}$. We employ a simple heuristic to produce the symbol sequence $s(m)$ from $\mathbf{y}(n)$ and compute the classification rate by comparing this sequence with the correct one $s_{\mathrm{target}}(m)$ produced from $\mathbf{y}_{\mathrm{target}}(n)$.

Since the digit image placement is nondeterministic, neither the exact position of the image, nor its class are known to the classification mechanism. Both have to be determined from $\mathbf{y}(n)$. For this purpose we implement a heuristic which exploits the fact that the variation of the distance Δn between two consecutive patterns (and their indicators in $\mathbf{y}_{\mathrm{target}}(n)$) is limited. It assumes the ranges $\Delta n_{\min} = 11$ and $\Delta n_{\max} = 21$ and proceeds as specified in Algorithm 2.

Algorithm 2 Calculate $s(m)$ from $\mathbf{y}(n)$, $\Delta n_{\min}$, $\Delta n_{\max}$

$n \leftarrow 1$
$m \leftarrow 1$
$\Delta n'_{\min} \leftarrow 0$
while $(n \leq T_{\mathrm{y}} - \Delta n_{\max})$ **do**
  $\hat{c},\ \Delta\hat{n} \leftarrow \underset{\tilde{c}\in[1,10],\ \Delta\tilde{n}\in[\Delta n'_{\min}, \Delta n_{\max}]}{\arg\max} y_{\tilde{c}}(n + \Delta\tilde{n})$
  $s(m) \leftarrow \hat{c}$
  $m \leftarrow m + 1$
  $n \leftarrow n + \Delta\hat{n}$
  $\Delta n'_{\min} \leftarrow \Delta n_{\min}$
end while
return $s(m)$

In short, the Algorithm 2 in each iteration finds the maximal value of the output in the allowed interval from the current position n, records the symbol by the channel in which the maximum was found and advances the current position to the time step where the maximum was found. Note that such approach only works well if $\Delta n_{\max} < 2\Delta n_{\min}$, otherwise two class indicators could fall inside the interval and a weaker one could be missed.

This simple classification heuristic serves good enough for our purpose of comparing different reservoir models, but for building a state-of-art sequence classifier much more sophisticated techniques can be employed, see, e.g., [Graves, 2008].

The classification rate (CR) is computed by

$$CR(s, s_{\mathrm{target}}) = 1 - \frac{d_{\mathrm{L}}(s, s_{\mathrm{target}})}{T_{s_{\mathrm{target}}}}, \tag{4.9}$$

where $d_{\mathrm{L}}(s, s_{\mathrm{target}})$ stands for Levenshtein edit distance, i.e., the minimal number of remove, add, or replace symbol operations needed to transform the sequence $s(m)$ into $s_{\mathrm{target}}(m)$ (or vice versa, d_{L} is symmetric). $s_{\mathrm{target}}(m)$ is produced from $\mathbf{y}_{\mathrm{target}}(n)$ by the same Algorithm 2. Note, that the lengths of the two sequences are not necessary equal: $T_{\mathrm{s}_{\mathrm{target}}} \neq T_{\mathrm{s}}$. The classification rate indicates the proportion of symbols that are wrongly recognized (or missed) and is equal to 1.0 for a perfect recognition.

4.4.5 Simulation results

To answer the question whether and by how much the use of the self-organizing reservoir and its unsupervised pre-training benefits the pattern detection task, we compare it with the regular type of reservoirs (4.8). We use the same number of neurons $N_{\mathrm{x}} = 50$ in both types of reservoirs, so that the reservoir signal space $\mathbf{x}(n)$ has the same dimensionality.

As explained in Section 4.4.4, the main three parameters of ESNs: leaking rate γ, $\mathbf{W}^{\mathrm{in}}$ scaling "in.s.", and spectral radius $\rho(\mathbf{W})$; and of SOR: leaking rate γ, input α and recurrent β distance scalings, are selected through grid search. We take eight values of each parameter over a reasonable interval. It took multiple trials to get the parameter ranges right. The ranges, step sizes, and the best found values of the parameters are presented in Tables 4.1 and 4.2 for the two datasets in the corresponding Subsections 4.4.5.1 and 4.4.5.2.

4.4.5.1 Smooth synthetic patterns

The details and results of parameter grid search with synthetic pattern data (Section 4.4.2) are presented in Table 4.1.

We found that good α and β values in (4.1) for this data should account for the normalization over the dimensionality of the input N_{u} and the reservoir N_{x}. This is logical, because the two $\|\cdot\|^2$ terms in (4.1) are summations over N_{u} and N_{x} dimensions respectively, and a bigger dimensionality gives a bigger sum. As a result the intervals of α and β look very different in Table 4.1 because they are normalized. If we denormalize them, we see that αN_{u} and βN_{x} have the exact same intervals of $[25, 200]$. Best parameters for both SOM and NG trained SORs are very similar in all cases.

Table 4.1: Grid search parameters and best values found with the synthetic data.

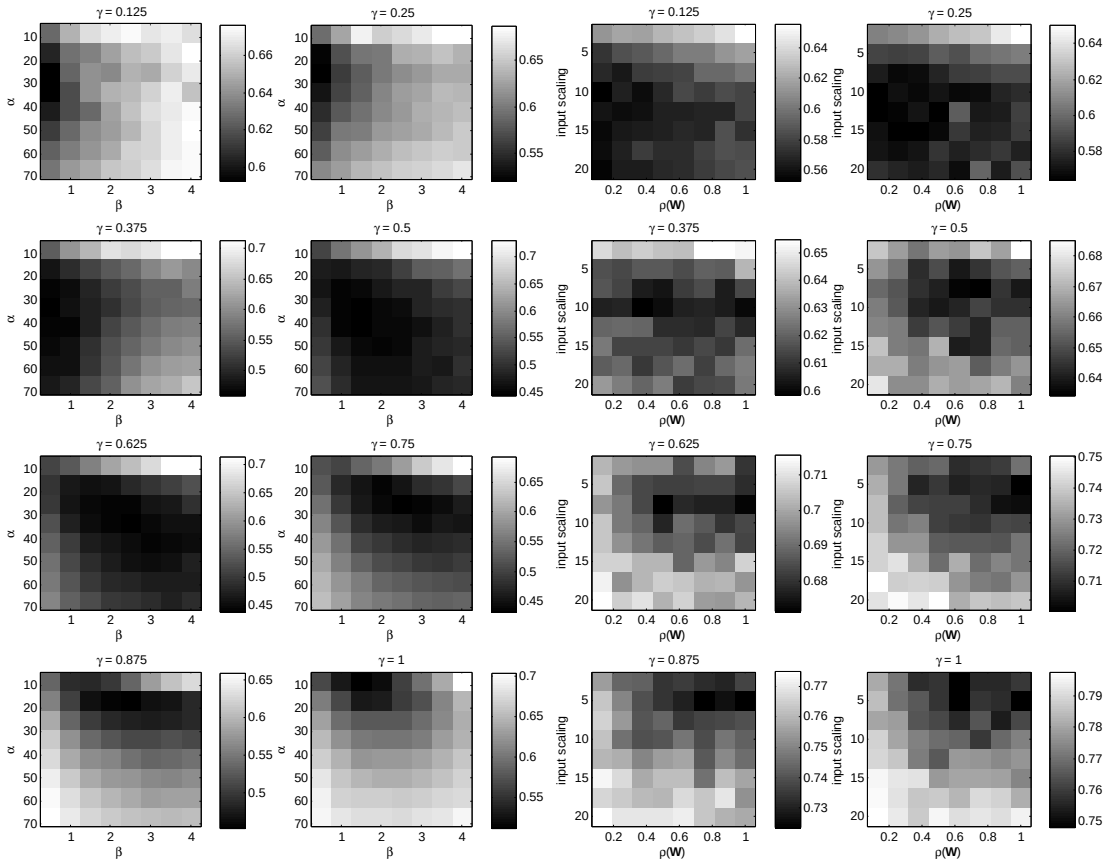
Model	**ESN**			**Self-organizing reservoir**								
Algorithm				Random			SOM			NG		
Parameter	γ	in.s.	$\rho(\mathbf{W})$	γ	α	β	γ	α	β	γ	α	β
Min. value	0.125	2.5	0.125	0.1	10/3	0.1	0.125	25/3	0.5	0.125	25/3	0.5
Step size	0.125	2.5	0.125	0.1	10/3	0.1	0.125	25/3	0.5	0.125	25/3	0.5
Max. value	1	20	1	0.8	80/3	0.8	1	200/3	4	1	200/3	4
Best values												
1 pattern	0.25	10	0.75	0.25	20/3	0.2	0.75	100/3	2.5	0.75	100/3	2.5
2 patterns	0.25	10	0.125	0.25	10/3	0.1	0.75	75/3	2	0.875	75/3	3
3 patterns	0.125	10	0.125	0.125	40/3	0.5	0.75	75/3	2.5	0.75	75/3	3
4 patterns	0.125	15	0.125	0.125	50/3	0.2	0.5	150/3	2	0.5	125/3	1.5
5 patterns	0.125	20	0.125	0.125	60/3	0.1	0.5	100/3	1.5	0.5	100/3	1.5

For random SORs these two values had to be considerably smaller. Since vectors in $\mathbf{V}^{\text{in}}$ and $\mathbf{V}$ are distributed randomly and are not centered on the typical respective $\mathbf{u}(n)$ and $\mathbf{x}(n)$ values by unsupervised training, the distances in (4.1) are greater. Thus, they have to be scaled with smaller coefficients for the neurons to reach reasonable working activation levels.

A trend for both of the random models can be observed in Table 4.1, that with a bigger number of different patterns, the influence of immediate input should be bigger. This could be explained by the effect, that while few patterns can be distinguished by subtitle changes in the reservoir activation space $\mathbf{x}(n)$, with more patterns these changes should be stronger and more immediate. Self-organized reservoirs, apparently, learn to react to different patterns more sensitively.

ESNs gave the best performance with quite surprisingly big input scalings, together with small leaking rates and spectral radii.

Training and testing of a self-organizing network took about 7.6 seconds on an average modern computer, so it is quite fast. Training and testing an ESN or a random SOR took about 3.0 seconds. Thus self-organizing reservoirs received more pure computational time. On the other hand we have selected not only the best parameters from the grid search for the random models (ESNs and random SORs), but took the same ten randomly generated reservoirs with these parameters, that gave the best average performance, for testing. Thus, the performance fluctuations caused by randomness of the reservoirs were used to the advantage of the random models.



(a) Self-organizing reservoir (SOM) (b) ESN

Figure 4.3: Mean training error surfaces for data with three different patterns.

Figure 4.3 illustrates in more details how the mean training error (averaged over the ten runs) depends on the three parameters set by the grid search for the SOM-trained SOR and for the ESN. The case with three patterns is shown here. The surfaces for ESNs are a bit more rough because the use of random reservoirs introduces some additional variance in performance. But the ranges of the ESN performance are smaller. The self-organizing reservoirs are a bit more sensitive to the set parameters, at least in these ranges. We can see that the mean error surfaces are rather smooth, not abrupt, indicating that the right parameters do not need to be picked very accurately to get reasonably good performance.

The pattern separation errors for the two models and different numbers of patterns in the data are presented in Figure 4.4. The best parameters found using the grid search (Table 4.1) were used for every number of patterns in both models.

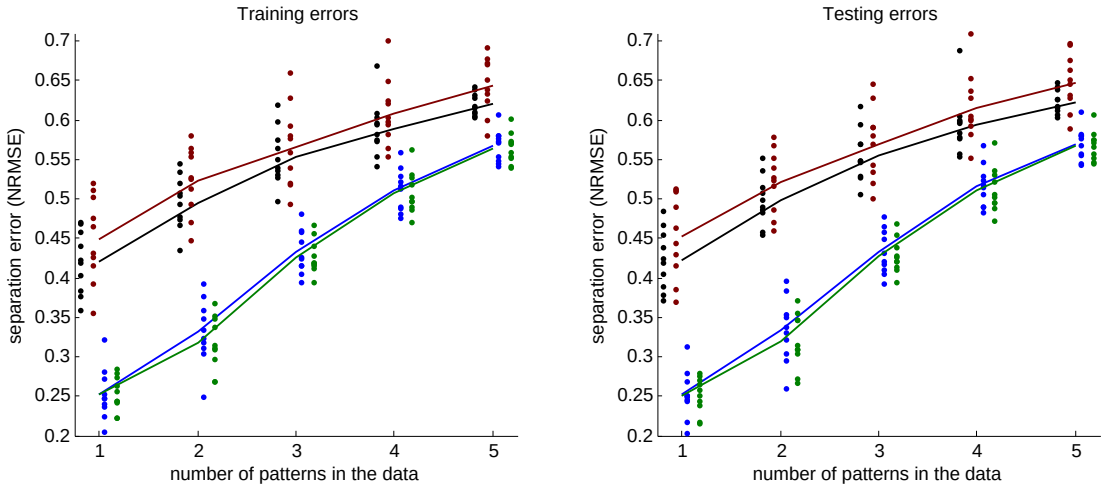


Figure 4.4: Training (left) and testing (right) separation errors with best parameter values of regular ESNs (black) self-organizing reservoirs random (dark red), trained with SOM (blue) and NG (green) for different number of patterns in the data. The values of the ten data instances are represented by dots (slightly horizontally displaced for visual clarity) and mean values by lines.

The bigger spread of the ESN errors can be explained by the the use of randomly generated reservoirs.

The performance of the random SOR model is similar, albeit slightly worse, to the classical ESN, showing it as a viable alternative.

Figure 4.4 shows clearly that unsupervised training benefits the patterns separation in the reservoir space. This improvement is statistically significant, present with different numbers of patterns in the data, and, because of the grid search, is not caused by parameter settings favoring one of the methods. The performance of both SOM and NG training are virtually indistinguishable, showing that exact details of the training do not matter in this case. We also see that training and testing errors are almost identical in all the cases, justifying choosing the parameters based on the training error.

Looking at Figure 4.4, we see that the benefit of the self-organizing reservoirs is bigger in the cases where there are fewer different patterns in the data. The reason for this is that given the limited capacity of the self-organizing reservoir it can learn to better represent fewer different patterns than more, while the random reservoirs of ESN or SOR are universal and the readouts for the different patterns are virtually independent. The drop in performance with the number of different patterns is only due to the fact that each pattern appears more rarely in the input

and thus is harder to learn.

To visualize how different number of patterns are represented in the signal space $\mathbf{x}(n)$ of the self-organizing reservoir, in Figure 4.5 we plot the two principal components of $\mathbf{x}(n)$ with activations corresponding to the patterns highlighted in colors.

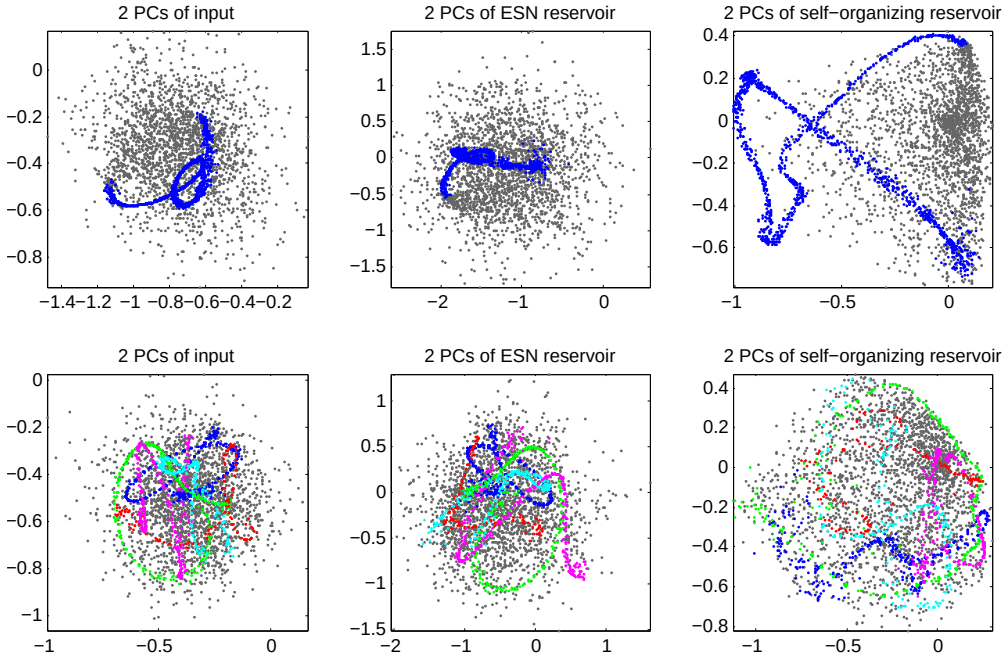


Figure 4.5: 1 pattern and 5 patterns highlighted in the two principal components of $\mathbf{u}(n)$, and in $\mathbf{x}(n)$ of both ESN and SOM-trained SOR.

We can see that a single pattern gets a special representation in $\mathbf{x}(n)$ of the self-organizing reservoir which is already clearly visible in the two principal components. With more patterns we can see that they are spread more than in the ESN or in $\mathbf{u}(n)$ but are in no way easily separable.

4.4.5.2 Concatenated USPS digits

The details and results of parameter grid search with concatinated USPS data (Section 4.4.4.2) are presented in Table 4.2. They were selected according to two distinct performance criteria: the NRMSE of the output signal and the classification rate of the final recognized symbol sequence (4.9), as explained in Section 4.4.4.2.

Table 4.2: Grid search parameters and best values found with the USPS data.

Model	**ESN**			**Self-organizing reservoir**								
Algorithm				Random			SOM			NG		
Parameter	γ	in.s.	$\rho(\mathbf{W})$	γ	α	β	γ	α	β	γ	α	β
Min. value	0.125	0.1	0.2	0.125	0.0125	0.025	0.125	0.1	0.1	0.125	0.1	0.1
Step size	0.125	0.1	0.2	0.125	0.0125	0.025	0.125	0.1	0.1	0.125	0.1	0.1
Max. value	0.5	0.8	1.6	0.5	0.1	0.2	0.5	0.8	0.8	0.5	0.8	0.8
Best values												
NRMSE	0.25	0.3	0.8	0.25	0.1	0.1	0.25	0.3	0.2	0.25	0.4	0.3
Class. rate	0.25	0.2	0.8	0.25	0.025	0.15	0.125	0.1	0.3	0.125	0.1	0.1

Contrary to the previous case with the synthetic data in Table 4.1, normalization of good α and β by N_u and N_x respectively was not needed. Possibly, because the difference between $N_\mathrm{u} = 16$ and $N_\mathrm{x} = 50$ is not as big in this case.

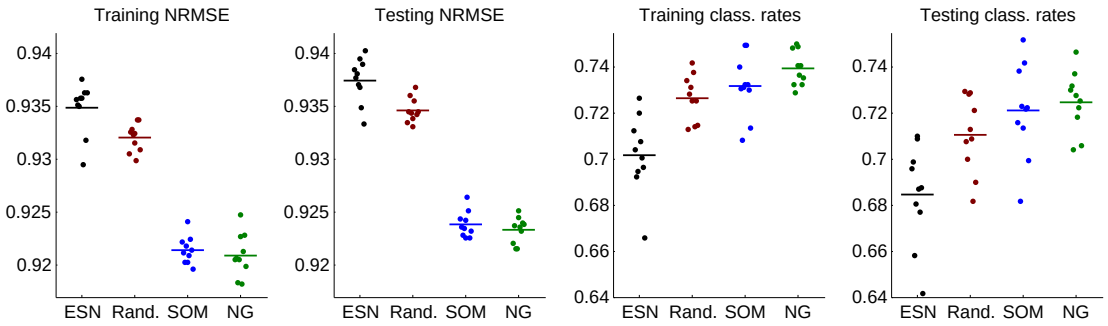


Figure 4.6: Output errors and classification rates with best parameter values of regular ESNs, self-organizing reservoirs with random weights, and trained with SOM and NG on the concatenated USPS data. The values of the ten data instances are represented by dots (slightly horizontally displaced for visual clarity) and mean values by lines.

The NRMSE and classification rate (CR) performance criteria for the two models are presented in Figure 4.6. We can see, that self-organizing reservoirs perform better than simple ESNs in both respects, with NG pre-training performing marginally better than SOM. A possible explanation for this is that the enforced 2D structure of the SOM lattice becomes a slight hindrance with this more complex data. It is also noticeable that compared to the performance dispersions the superiority of SORs is a bit more pronounced for the NRMSE criterion. While testing results are naturally slightly worse than training, they are consistent across the both datasets, overfitting was only observed in some degenerate cases.

Interestingly, the performance of random SOR is consistently better than that of ESN, showing that this computational model alone is beneficial in this particular setting, and is even more enhanced by the unsupervised training.

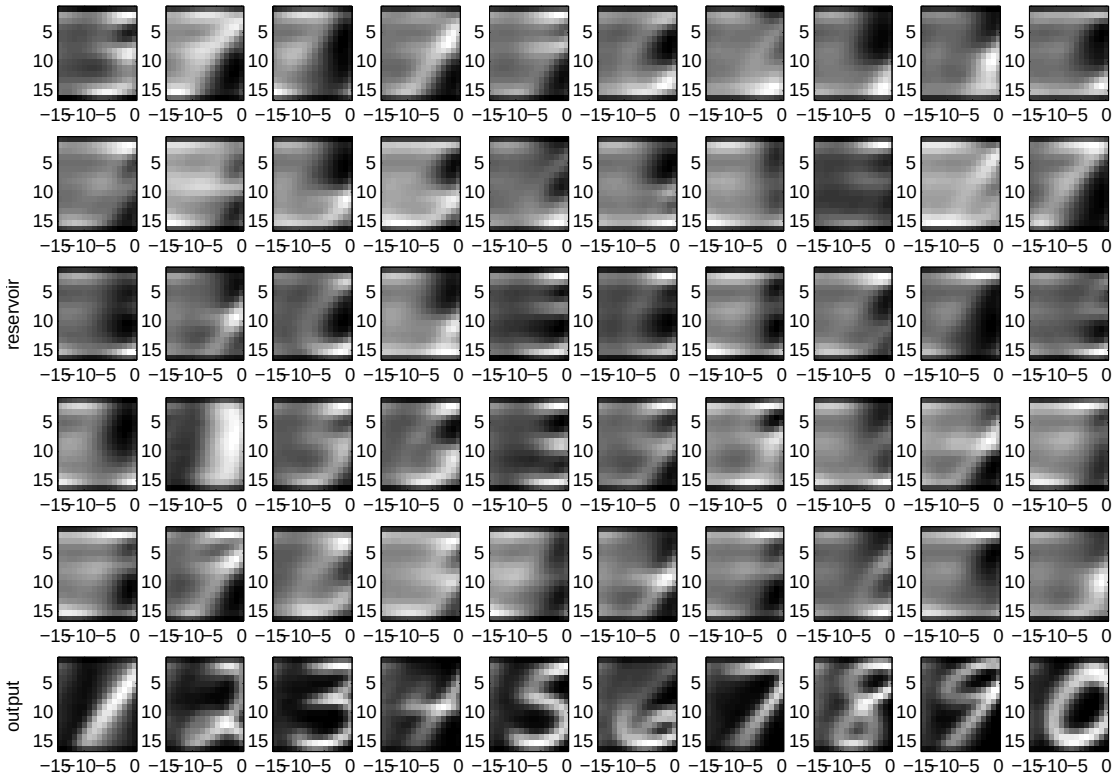


Figure 4.7: Average input histories for every reservoir and output neuron in a SOM-trained SOR, that excite the neuron most.

One way to look at what the unsupervised pre-training does is presented in Figure 4.7. It shows input history (up to $\mathbf{u}(n-15)$) averages weighted by the activation to the 7th power $x_i^7(n)$ of each neuron i in a fully SOM-trained SOR. Similar to simple SOMs, each neuron in the reservoir learns to respond to a specific common situation in the input. The difference here is that each neuron reacts to a particular *history* of the input. Due to SOM training, the neighboring units tend to be excited most by similar temporal patterns. The output neurons (that are supervisedly trained linear combinations of the reservoir units) show clear "prototypical" digit patterns that excite them most.

A curious thing to notice in Table 4.2 is that the parameters of SORs are different for the best NRMSE and CR, while the ones of ESNs are almost the same. To illustrate this better, average training CRs vs. NRMSEs are plotted

for the NG-trained SORs and ESNs with different parameter settings in Figure 4.8. In general, lower NRMSEs correspond to higher CRs, which is expected, but this is not always the case, especially for SORs. While there are quite some SORs better than ESNs according to the both criteria, best in each are different. Results with testing data and with SOM pre-training are very similar to those with NG, and the general shape of the ESN point cloud very similar to the Random SOR.

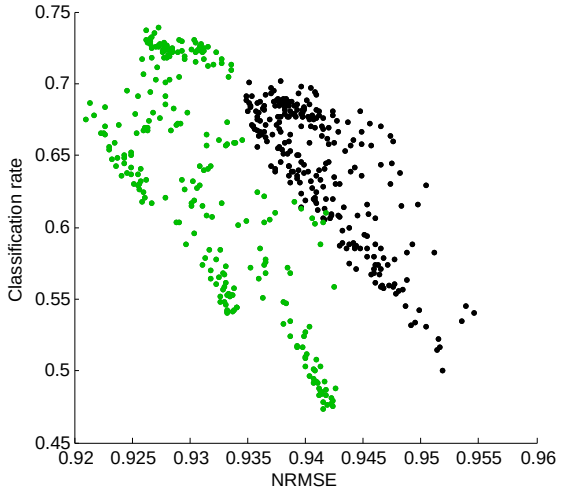


Figure 4.8: Training classification rates vs. training NRMSE, for different parameter settings, averaged over ten runs. SOR trained by NG is in green and ESN is in black.

This illustrates that different parameters bias the model to learning different unsupervised representations of the input data. Which of the representations is favored depends on the ultimate goal of the supervised task. We see here that even a slight change in the final performance measure favors quite different unsupervised representations. The "no free lunch" principle of supervised learning [Wolpert, 2001] is also applicable to unsupervised: there are no unsupervisedly learned representations of data that are universally best for any supervisedly measured performance criterion.

Having said that, we do believe that there are unsupervised representations that are good for many "naturally" arising tasks. Models that learn statistical structure of the data should be good with tasks that correlate well with this structure. We still observe in Figure 4.8, that the correlation between the two criteria is strong.

These differences between the two criteria are also partially due to the powerful

readout training algorithm and a rather special ad-hoc symbol sequence extraction Algorithm 2 that we use. A closer look at the SOR models trained with the best parameters for the CR reveals that these are quite degenerate cases (also hinted by the extreme parameter settings), where typically only few neurons in the reservoir are reasonably active, and the output is produced by big $\mathbf{W}^{\text{out}}$ values, exploiting tiny differences in these minute activations. This performance would be destroyed with regularization of $\mathbf{W}^{\text{out}}$ (2.16), but nonetheless, in this setting it gives the best training and testing classification rates.

4.4.6 Scaling up the reservoirs

To explore the limits of the proposed methods and have a better grasp on them, we conducted a number of additional investigations.

RNNs of $N_{\text{x}} = 50$ units are big by most standards in ML, but not in RC. To investigate the scalability of our methods, we also ran the same experiments on the concatenated USPS data as in Section 4.4.5.2 with reservoirs of size $N_{\text{x}} = 100$.

We had to adapt the model parameter ranges and some training parameters for that. Reflecting the twofold increase in reservoir size, the initial values of the neighborhood widths $b_h(1)$ for the SOM (4.5) and the NG (4.6) training algorithms were increased twice to the values of 4 and 20, respectively. For the sake of completeness the grid search parameters and results are presented in Table 4.3.

Table 4.3: Grid search parameters and best values found with the USPS data and $N_{\text{x}} = 100$.

Model	**ESN**			**Self-organizing reservoir**								
Algorithm				Random			SOM			NG		
Parameter	γ	in.s.	$\rho(\mathbf{W})$	γ	α	β	γ	α	β	γ	α	β
Min. value	0.125	0.1	0.2	0.125	0.005	0.0125	0.125	0.05	0.05	0.125	0.05	0.05
Step size	0.125	0.1	0.2	0.125	0.005	0.0125	0.125	0.05	0.05	0.125	0.05	0.05
Max. value	0.5	0.8	1.6	0.5	0.04	0.1	0.5	0.4	0.4	0.5	0.4	0.4
Best values												
NRMSE	0.25	0.5	1	0.25	0.04	0.075	0.25	0.3	0.1	0.25	0.3	0.1
Class. rate	0.125	0.6	1	0.25	0.005	0.0875	0.125	0.05	0.15	0.125	0.05	0.15
Best values with $N_{\text{x}} = 50$ from Table 4.2												
NRMSE	0.25	0.3	0.8	0.25	0.1	0.1	0.25	0.3	0.2	0.25	0.4	0.3
Class. rate	0.25	0.2	0.8	0.25	0.025	0.15	0.125	0.1	0.3	0.125	0.1	0.1

Due to the increased dimensionality, good β values of SORs are smaller. Good

parameters of ESN remain similar, as this model, in particular with a fixed average number of connections to a reservoir unit, is not sensitive to these changes.

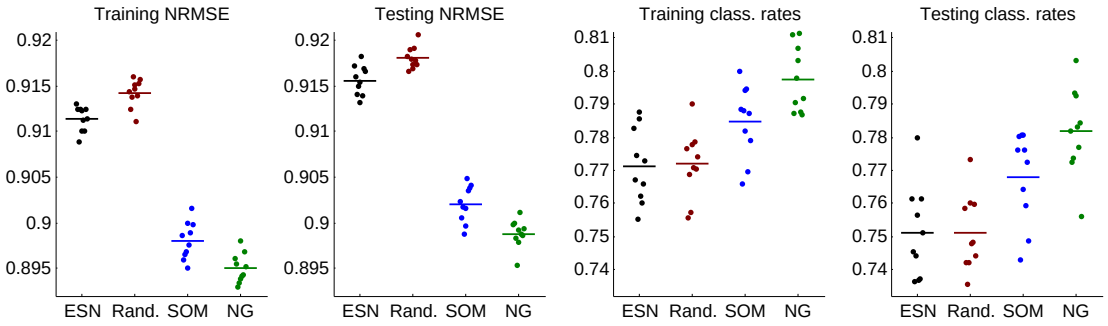


Figure 4.9: Output errors and classification rates with best parameter values and $N_x = 100$ of regular ESNs, self-organizing reservoirs with random weights, and trained with SOM and NG on the concatenated USPS data. The values of the ten data instances are represented by dots (slightly horizontally displaced for visual clarity) and mean values by lines.

NRMSE errors and classification rates with the best parameters and $N_x = 100$ are presented in Figure 4.9. The first thing to observe is that the performances are much better with bigger reservoirs. Dimensionality of the reservoir, and thus the number of supervisedly trained parameters, is essential for performance, this is why we keep them the same for the different models.

Differences between training and testing performances are also more pronounced, as more powerful models get a bit closer to overfitting (but still far from it).

We also see that the SOM-trained SORs fall a bit further behind the NG ones. A plausible explanation is that the imposed 2D structure (10×10) of the SOM lattice becomes an even bigger hindrance when trying to cover the 100D space of $\mathbf{x}(n)$.

We also observe in Figure 4.9 that random SORs (at least by the NRMSE criterion) are falling behind the ESNs. This might be an indication that the RBF model of SORs does not scale as well as that of an ESN. To further investigate this hypothesis, we ran experiments with increasing sizes of the reservoirs for both ESNs and random SORs. The parameters were taken from the experiments for the best NRMSE with $N_x = 50$ (Table 4.2) and kept the same, except that β was corrected for N_x. The results are presented in Figure 4.10.

We can again see that performance increases a lot with N_x, and indeed the

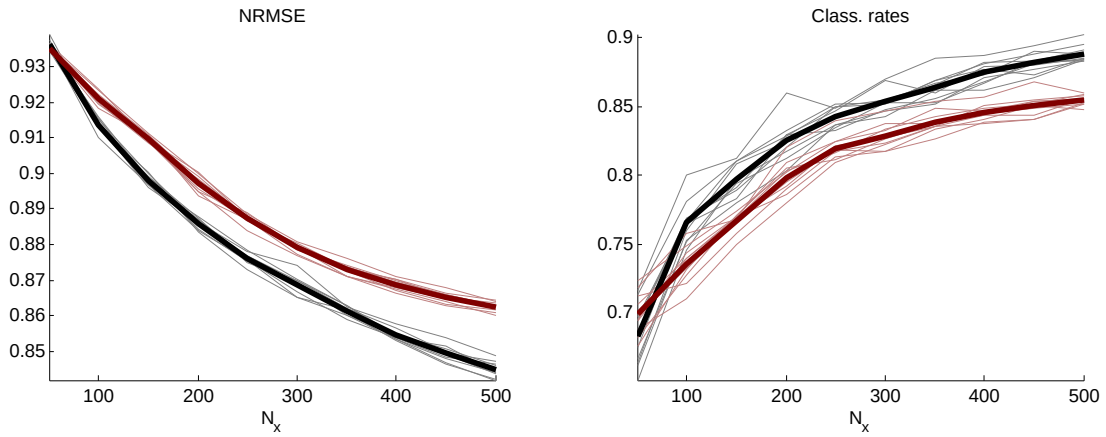


Figure 4.10: Output testing errors and classification rates of ESNs (black) and random SORs (dark red) with different network sizes N_{x}. The parameters are selected with $N_{\mathrm{x}} = 50$ (Table 4.2). Averages are shown by thick and instances by thin lines.

random SORs scale worse than ESNs. Part of this relative drop in performance can be attributed to the fact that random SORs have quite different best parameters for different sizes (as seen in Table 4.3), but even with selected best parameters their NRMSE is worse with $N_{\mathrm{x}} = 100$ (Figure 4.9).

We did not run the same simulations with SOM or NG trained bigger reservoirs, as they would computationally expensive and we suspect that they would suffer from the same limitations of the model.

4.5 Weighted distance networks

4.5.1 Computational model

We have seen so far that the SOR-type RNN (4.1) reservoirs perform better than ESN (4.8) for up to certain sizes (Figure 4.10), but do not scale well beyond. We believe, that there is a fundamental limitation at work here. With every added new unit the reservoir signal $\mathbf{x}(n)$ space increases by an additional dimension. Since RNN has to model the input $\mathbf{u}(n)$ as well as its previous state $\mathbf{x}(n-1)$ in its activation $\mathbf{x}(n)$, the total input dimension $N_{\mathrm{u}} + N_{\mathrm{x}}$ to the current state $\mathbf{x}(n)$ also increases. But there is only one extra unit in the reservoir to model this bigger space. This is not a problem for the WSN type of units (4.8), because every unit "reacts" to a direction in its input space defined by the weight vector and adding

one more (random) direction covers the added dimension in the space. For RBF units, however, this creates a problem. Such unit reacts to a particular area of its input space locally centered around the weight vector. The number of such local units required to cover the input space with the same density in fact grows exponentially with the number of input dimensions. Adding only one unit to cover an additional dimension is inadequate and enriches the signal space $\mathbf{x}(n)$ less in the RBF case. In a way it could be said that the SOR reservoir starts to suffer from its own "curse of dimensionality".

Another, related, limitation of the SOR model is that each neuron in it tries to model the complete state and input space and is invariant to none of its dimensions. In addition, the activation of the neuron is dominated by the biggest distances in (4.1), in other words the things that the neuron fails to model. This makes any independent dynamics in the reservoir impossible and prevents the reservoir state $\mathbf{x}(n)$ from a more efficient information coding of distributed representations and a richer signal space.

We can alleviated both of these limitations if we make the RBF RNN model *sparse*. In particular, we generalize the model (4.1) into

$$\tilde{x}_i(n) = \exp\left(-{\mathbf{w}^{\text{in}}}_i^{\text{T}} \left(\mathbf{v}^{\text{in}}{}_i - \mathbf{u}(n)\right)^2 - \mathbf{w}_i^{\text{T}} \left(\mathbf{v}_i - \mathbf{x}(n-1)\right)^2\right), \quad i = 1, \ldots, N_{\text{x}}, \tag{4.10}$$

where the dimensions in the distances are weighted with the additional weight parameters $\mathbf{W}^{\text{in}} \in \mathbb{R}^{N_{\text{u}} \times N_{\text{x}}}$ and $\mathbf{W} \in \mathbb{R}^{N_{\text{x}} \times N_{\text{x}}}$ ($\mathbf{w}^{\text{in}}{}_i \in \mathbb{R}^{N_{\text{u}}}$ stands for the ith column of $\mathbf{W}^{\text{in}}$ and $\mathbf{w}_i \in \mathbb{R}^{N_{\text{x}}}$ of $\mathbf{W}$), and $(\cdot)^2$ is applied element-wise. We will call this computational model Weighted Distance Network (WDN or WeiDiNet).

In this model each connection (i, j) is parametrized by two types of weights: v_{ij} and w_{ij}. It can be seen as a combination of RBF (4.1) and WSN (4.8) type of RNNs. Note that if all the elements of $\mathbf{W}^{\text{in}}$ are equal to α and $\mathbf{W}$ to β, we fall back to (4.1).

Equation (4.10) can be rewritten in a more compact form

$$\tilde{x}_i(n) = \exp\left(-{\mathbf{w}^{\text{all}}}_i^{\text{T}} \left(\mathbf{v}^{\text{all}}{}_i - [\mathbf{u}(n); \mathbf{x}(n-1)]\right)^2\right), \quad i = 1, \ldots, N_{\text{x}}, \tag{4.11}$$

where $\mathbf{V}^{\text{all}} = [\mathbf{V}^{\text{in}}; \mathbf{V}]$ and $\mathbf{W}^{\text{all}} = [\mathbf{W}^{\text{in}}; \mathbf{W}]$.

In fact the WDN model could be even further generalized by using a Mahalanobis-

like distance:

$$\tilde{x}_i(n) = \exp\left(-\left(\mathbf{v}^{\text{all}}{}_i - [\mathbf{u}(n); \mathbf{x}(n-1)]\right)^{\text{T}} \mathbf{W}^{\text{all}}{}_i \left(\mathbf{v}^{\text{all}}{}_i - [\mathbf{u}(n); \mathbf{x}(n-1)]\right)\right), \quad i = 1, \ldots, N_{\text{x}}, \tag{4.12}$$

where $\mathbf{W}^{\text{all}}{}_i \in \mathbb{R}^{(N_{\text{u}}+N_{\text{x}})\times(N_{\text{u}}+N_{\text{x}})}$ would be a matrix of weights for every unit i in the reservoir. This would be an even more powerful model, but would require much more computations, more memory, and also much more parameters to train if the structure of $\mathbf{W}^{\text{all}}{}_i$ is unrestricted. One obvious option would be to make all $\mathbf{W}^{\text{all}}{}_i$ approximate the inverse of covariance matrix of $[\mathbf{u}(n); \mathbf{x}(n-1)]$ to have the true Mahalanobis distance in (4.12). We leave this most general model (4.12) only as a theoretical possibility at this point.

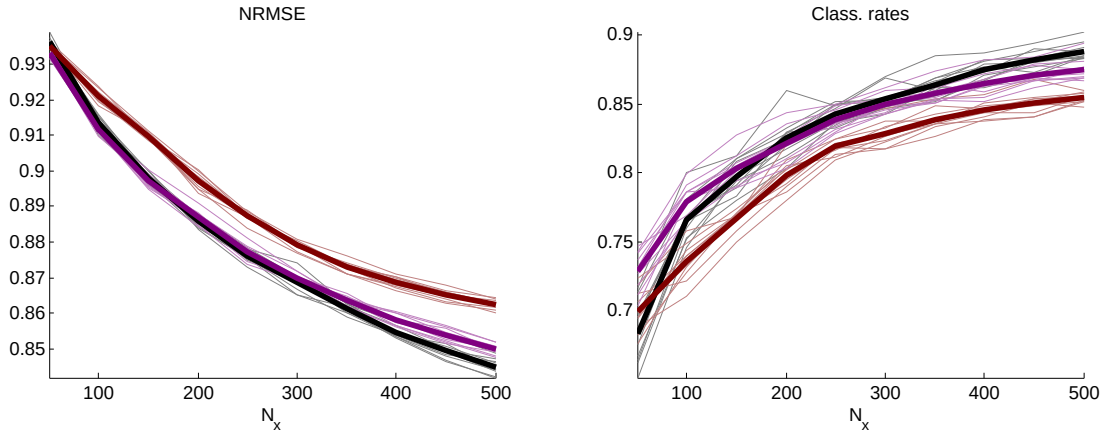


Figure 4.11: Output testing errors and classification rates of ESNs (black) and random SORs (dark red) and sparse random WDNs (magenta) with different network sizes N_{x}. The parameters are selected with $N_{\text{x}} = 50$ (Table 4.2). Averages are shown by thick and instances by thin lines.

To test whether sparseness of network helps it scale better, we run the same experiments as in Figure 4.10 with a sparse random WDN, where parameters are copied from the random SOR: $\gamma = 0.25$, all the elements of $\mathbf{W}^{\text{in}}$ are equal to $\alpha = 0.1$, but $\mathbf{W}$ has on average only $5N_{\text{x}}$ nonzero elements (5 incoming reservoir connections per unit), that are all equal to $50\beta/5 = 1$ to preserve the same average activation. No scaling of $\mathbf{W}$ for different N_{x} is required in this case. The results are presented in Figure 4.11.

We can see here that the sparse WDN gives an immediate boost in performance compared to RSOM, even with no special parameter selection, and scales similarly

to ESNs at least up to size 300.

4.5.2 WDN training options

Figure 4.11 illustrates that the WDN model has a potential when used with random weights $\mathbf{V}^{\text{all}}$ and sparse $\mathbf{W}$, similarly to ESNs. This is one option.

But can we train it in an unsupervised way, instead of just using random weights?

We have in fact tried multiple unsupervised approaches of training $\mathbf{V}^{\text{all}}$ and $\mathbf{W}^{\text{all}}$ weights simultaneously. For example, $\mathbf{V}^{\text{all}}$ were trained with the same SOM or NG algorithm, and $\mathbf{W}^{\text{all}}$ were trained with a combination of SOM or NG, Hebbian learning, and intrinsic plasticity (IP). It has not produced fully satisfactory results at this stage.

There are a couple of problems with this approach:

- $\mathbf{W}^{\text{all}}$ learned by Hebbian learning make neurons prefer inputs that are easy to predict and weaken connections from those that carry more information.
- When neurons have different inputs (different ${\mathbf{w}^{\text{all}}}_i$ values), the competitive-collaborative learning algorithms such as SOM or NG have no clear meaning. Comparison and competition of the units, like for the best matching one (4.5), are no longer on equal grounds.
- We have observed that sparse $\mathbf{W}^{\text{all}}$ matrices are usually better than just random. But learning sparseness is problematic for most algorithms. A training algorithm with an L1-norm regularization might be an option.

These are interesting open unsupervised ML questions that have to be addressed to produce a good training algorithm for the WDN model.

One possible simple option would be to just set rows of $\mathbf{W}^{\text{in}}$ and $\mathbf{W}$ inversely proportional to the standard deviations of the corresponding dimensions of $\mathbf{u}(n)$ and $\mathbf{x}(n)$, respectively, and have the normalized Euclidean distances in (4.10).

To further explore the potential of the WDN model, and having no satisfactory more sophisticated training algorithm at this point, we just ignored the problems mentioned above and ran the unmodified SOM and NG algorithms on sparse WDNs with adaptable $\mathbf{V}^{\text{all}}$ and fixed (sparse) $\mathbf{W}^{\text{all}}$. The results are presented in the next section.

4.5.3 Simulation results

We did experiments in the same setup and same reservoir sizes $N_x = 50$ as in Section 4.4.5.2 with the following modifications.

All elements of $\mathbf{W}^{in}$ were set to a parameter α found by grid search. Only 10% of random elements in $\mathbf{W}$ were nonzero (on average 5 incoming reservoir connections per unit) and were all equal to parameter β found by grid search. The grid search of the leaking rate γ was restricted to just two values 0.125 and 0.25 that gave best results with SOR models, to save the computational time.

The grid search parameters and results are presented in Table 4.4.

Table 4.4: Grid search parameters and best values found with the USPS data for the WDN model.

Model	**Sparse WDN**								
Algorithm	Random			SOM			NG		
Parameter	γ	α	β	γ	α	β	γ	α	β
Min. value	0.125	0.02	0.25	0.125	0.05	0.25	0.125	0.05	0.25
Step size	0.125	0.02	0.25	0.125	0.05	0.25	0.125	0.05	0.25
Max. value	0.25	0.16	2	0.25	0.4	2	0.25	0.4	2
Best values									
NRMSE	0.25	0.12	0.75	0.25	0.05	0.5	0.25	0.15	0.5
Class. rate	0.25	0.06	1.25	0.25	0.05	1.75	0.25	0.05	1.5

The NRMSE and classification rate performances with the sparse WDN model and best parameters are presented in Figure 4.12. ESN results are reused from Section 4.4.5.2.

Comparing the results in Figures 4.6 and 4.12 (they are also summarized in a numerical form in Table 4.5), it is clear that the sparse topology of the WDN network significantly improves the performance across all the WDN models.

The results also show that the simple-minded unsupervised approach of training sparse WDNs disregarding their sparseness and training only $\mathbf{V}^{all}$ weights, albeit not conceptually clean, as explained in Section 4.5.2, does work: the unsupervisedly trained reservoirs are much better than random. Note that in this case the number of unsupervisedly adapted parameters remains the same.

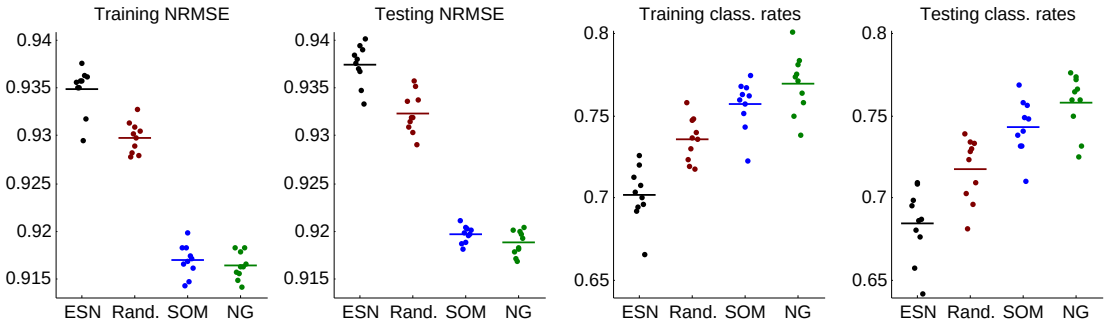


Figure 4.12: Output errors and classification rates with best parameter values of regular ESNs, and sparse WDNs: random and trained with SOM and NG on the concatenated USPS data. The values of the ten data instances are represented by dots (slightly horizontally displaced for visual clarity) and mean values by lines.

4.6 Hierarchies of self-organizing reservoirs

As mentioned in Chapter 3, one of the main benefits of unsupervised learning is that components trained this way can be easier assembled into more complex architectures. Here we investigate a simple layered hierarchy of such reservoirs where the bottom reservoir receives the external input $\mathbf{u}(n)$ and every reservoir above receives the activations $\mathbf{x}(n)$ of the reservoir directly below it as the input. Such an architecture features only bottom-up interactions and can be trained in a greedy layer-by-layer way starting from the bottom. Since every layer is trained independently from the rest, this hierarchical structure in essence does not introduce additional difficulties in training, except more of it needs to be done, because there are more layers.

When comparing a hierarchy to a single reservoir, a natural question to ask is whether it is better to invest the additional effort in training many layers of the hierarchy or in better training of the single reservoir. More concretely, we take the training time measured in epochs as the "effort". As a generalization of this question, we investigate how the performance depends on the number of layers in the hierarchy for a given fixed training effort. By this we mean that if a hierarchy has k layers and the fixed training effort is l epochs, then each layer receives l/k epochs of training.

4.6.1 Simulation details

For the experiments with the hierarchies we used the same synthetic data described in Section 4.4.2. The same ten data instances with $N_y = 5$ temporal patterns in it were reused as in Section 4.4.4 with the same normalization and splitting into the initialization, training, and testing sequences. In these experiments, however, we have gone through the 49 500 time steps of training data multiple times (*epochs*), each time first initializing the model with the initialization sequence of 500 time steps during which the training was not happening.

We again used reservoirs of $N_x = 50$ units and all the other parameters: $\gamma = 0.75$, $\alpha = 100/N_u$, and $\beta = 50/N_x = 1$, the same in every layer. Note, however, that $N_u = 3$ for the bottom layer and $N_u = N_x = 50$ for the others, which affects α accordingly.

For training *all* of our reservoirs we used the same SOM algorithm (4.4)(4.5) with the reservoir units again organized into a 10×5 lattice, the a bit faster weight initialization, and the same but slightly more subtle training schedule, where $\eta(n)$ followed a geometric progression from 0.01 to 0.001 and the neighborhood width $b_h(n)$ again from 2 to 0.001. The same training schedule was used independently of the length of the training: if the training is taking more epochs, the learning parameters are simply changing slower, but the end-points remain the same.

The same performance criterion is also used: a linear readout (4.7) is trained on the pattern envelopes as the teacher and the error (NRMSE) of the reconstruction computed. In this case the input $\mathbf{u}(n)$ was not included as part of $\mathbf{x}(n)$ in (4.7). For every architecture the readout was trained only from the activations of the top-most layer. This way the signal space from which the pattern separation is learned always have the same dimensionality $N_x = 50$ and every model has the same amount of parameters trained in a supervised way, namely $\mathbf{W}^{\text{out}} \in \mathbb{R}^{N_y \times (N_x+1)}$ (or $N_x + 1 = 51$ parameters per pattern). The target signal $\mathbf{y}_{\text{target}}(n)$ for each layer was delayed by the number of time steps equal to the number of layer (1 for the bottom layer, 2 for the second, and so on) as this was found to be optimal at least for a couple of first layers.

4.6.2 Simulation results

The results showing how different numbers of layers and different numbers of training epochs per layer affect the testing performance are presented in Figure

4.13. The performance is plotted against the total number of epochs spent in training. Each curve here represents a hierarchy trained with the same amount of epochs per layer. The points on the curves represent the mean test separation errors in different layers. Every tenth layer is annotated. The hierarchies are trained with 1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60, or 120 epochs per layer. They are colored from blue (the top-most curve, 120 layers, each trained with 1 epoch) to red (a single point in the middle right, a single layer trained with 120 epochs) as the two extreme cases.

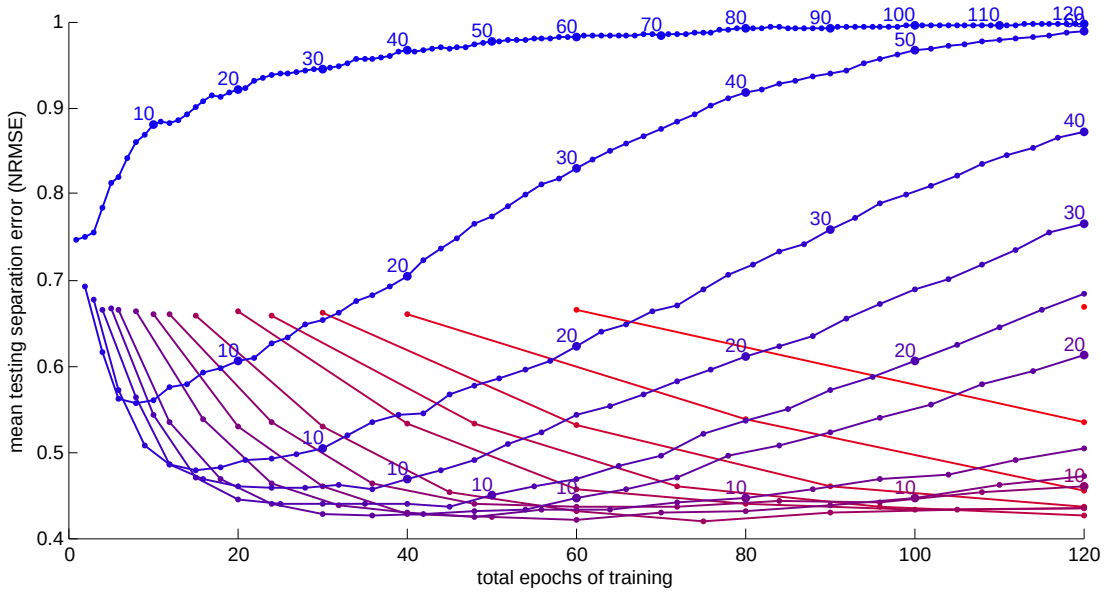


Figure 4.13: Mean testing separation errors in layers of differently trained hierarchies plotted against the total epochs of training. See text for the details.

We can see that if the layers are not trained well, stacking them in a hierarchy is not going to improve the result. The extreme case with each layer only trained in one epoch is the top-most blue curve. We can see that in this case the performance decreases with every additional layer and is approaching the worst NRMSE of 1. If a layer is not able to learn a good representation of its input, this bad representation is passed to the upper layers, information from the input is lost and the performance only decreases. Because we use quite small learning rates here the training time of one epoch per layer might simply be not enough.

However, when we give each layer enough time to learn a good representation of the input, we observe that adding additional layers improves the performance. The better the individual layers are trained, the better the improvement in the

upper layers.

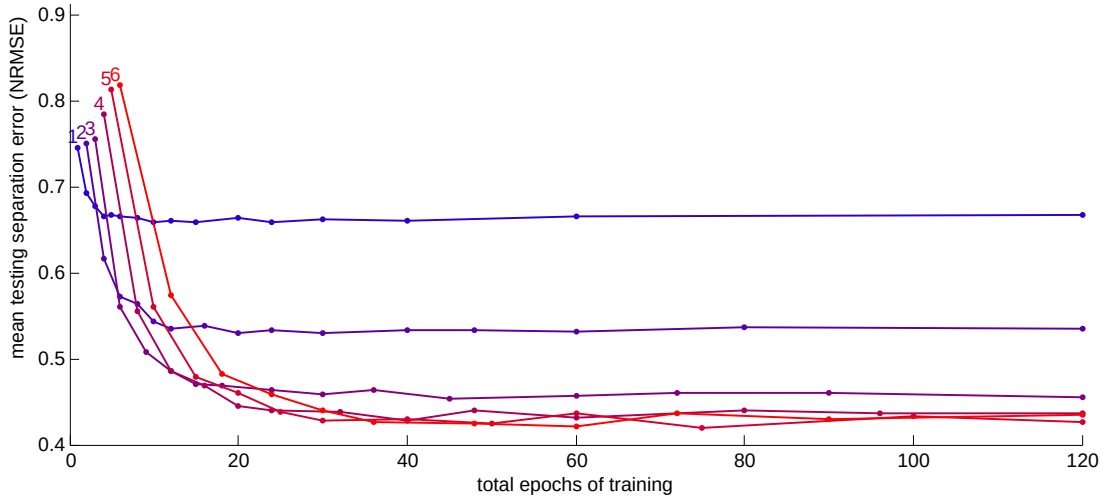


Figure 4.14: Errors in the first six layers across differently trained hierarchies plotted against the total epochs of training. See text for the details.

We can visualize the data of Figure 4.13 from a slightly different perspective. In Figure 4.14 we connect the dots representing layers of the same level across differently trained hierarchies. Here the six curves represent the errors in the first six layers across the architectures. This way we can see that putting more effort in training a single layer (the top-most blue curve) does improve the performance but only to a point where additional effort does not help anymore. The additional layers are able to break this performance ceiling achieving much smaller separation errors than a single layer could reach. We observe that when going up into the higher layers there is a significant drop in the error till about the fourth layer. This shows that with the same total number of training epochs hierarchical architectures clearly outperform a single reservoir.

The fact that the additional effort in training yields better results is not at all trivial in this case, because we train our reservoirs in an unsupervised way and test them on a supervised task. This proves that in this case the unsupervised pre-training does indeed improve the performance of a supervised task and there is a positive correlation between the quality of the unsupervised pre-training and the performance on the supervised task.

To have a more detailed view, we single out one case from the Figure 4.13 where every layer is trained using eight epochs of learning and present it in Figure

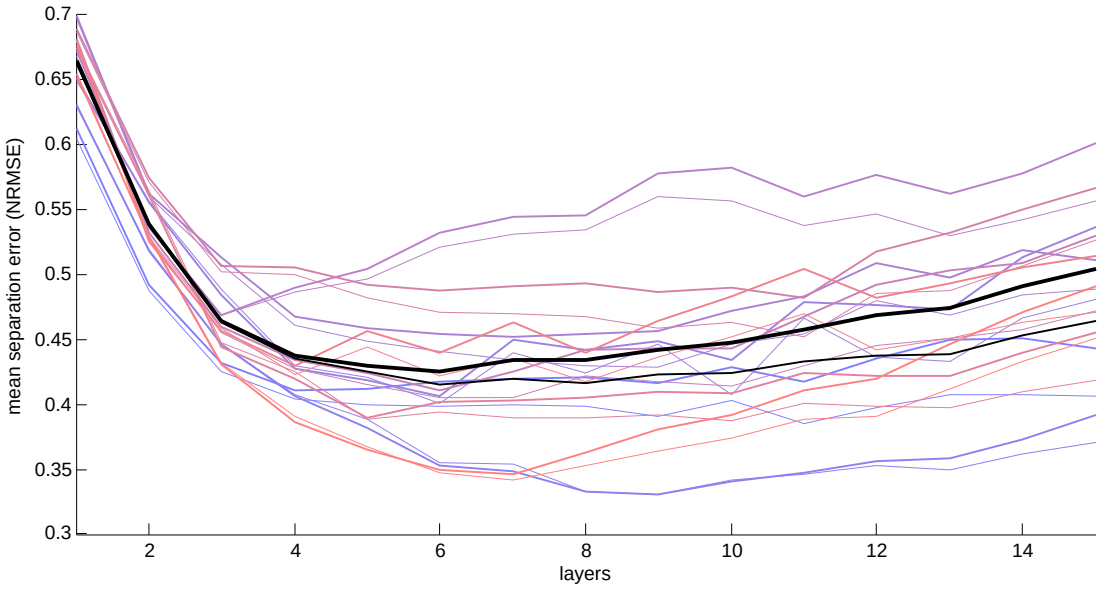


Figure 4.15: Errors in the layers each trained with eight epochs. Training errors are shown in thin and testing errors in bold lines. The mean values are shown in black and the ten separate instances are shown in light colors.

4.15. Here both the training and testing errors are shown in all the fifteen layers. The mean values are presented as well as the ten individual cases with the data instances to indicate the variability among them.

We see, that while the difference between the training and testing errors increases going up in the layers, it still remains quite small, and there is no real overfitting, because both errors reach minimal values at the same layer (6 in this case). One reason for this could be that we use long enough training data with small enough models. Another reason could be that most of the training we do is unsupervised, thus the model could overfit the input data but there is no training target to overfit. It might well be that the input data is too rich and without the target too open for interpretations to overfit.

The representation of five patterns in the two principal components of the activations $\mathbf{x}(n)$ of the first six layers in a hierarchy is presented in Figure 4.16. We can see that the special representation of the patterns in data gets more expressed in the principal components of $\mathbf{x}(n)$ when going up in the hierarchy.

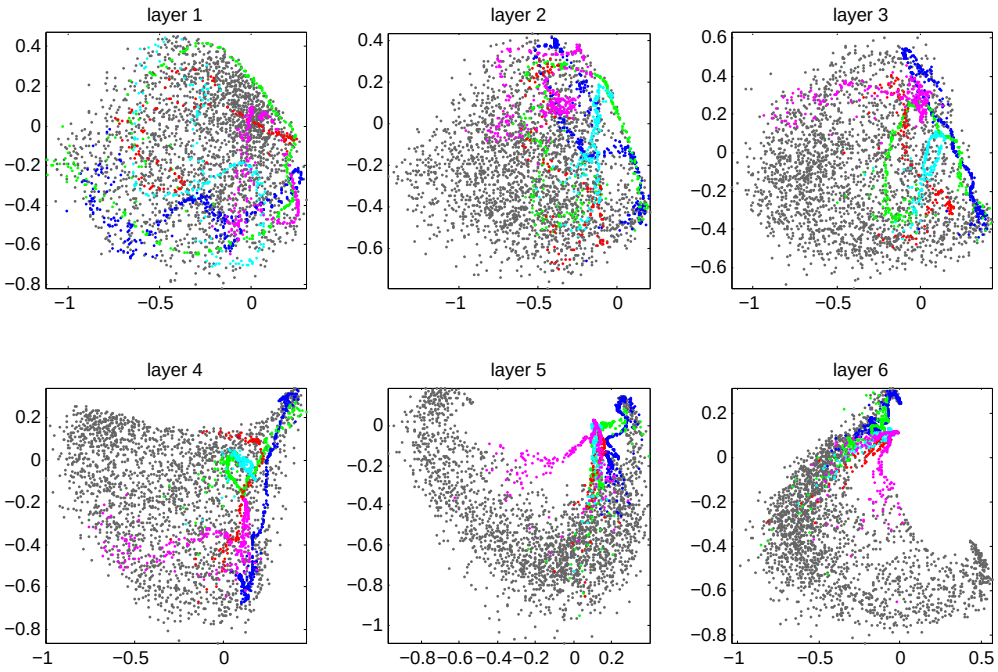


Figure 4.16: Patterns highlighted in the two principal components of $\mathbf{x}(n)$ in the first six layers of a hierarchy.

4.7 Discussion

We have demonstrated that the type of self-organized recurrent neural network investigated here can learn in an unsupervised way the representations of the temporal input data that enable better results in a supervised task such as separating repeated slow patterns or recognizing handwritten digits in the input signal. The self-organized networks were rigorously compared to structurally identical networks and classical echo state networks that produces random rich representations of the same dimensionality and found to be better. Parameter sweeps and averaging of results over different instances of data were performed for all models to exclude the possibility of an unfair comparison. The results are summarized in Table 4.5.

We also showed that longer unsupervised training results in better supervised performance, establishing a positive correlation between the two (see, e.g., Figure 4.14).

We do not have a rigorous explanation or analytical proof of this correlation, but can offer some intuitions. The competitive nature of the self-organizing learn-

Table 4.5: Summary of the test performances with the synthetic and USPS datasets and single reservoirs: mean ±standard deviation.

Model / Algorithm	**ESN**	**Self-organizing reservoir**		
		Random	SOM	NG
Synthetic data, NRMSE, $N_x = 50$				
1 pattern	0.42265 ±0.038441	0.45275 ±0.052971	0.25169 ±0.030817	0.25091 ±0.022701
2 patterns	0.49752 ±0.031056	0.52169 ±0.040179	0.33299 ±0.040864	0.31917 ±0.034875
3 patterns	0.55517 ±0.032841	0.57009 ±0.046305	0.43235 ±0.028593	0.42731 ±0.021406
4 patterns	0.59328 ±0.037655	0.61567 ±0.043534	0.51625 ±0.026512	0.51144 ±0.028712
5 patterns	0.62290 ±0.015118	0.64801 ±0.035877	0.56950 ±0.020516	0.56726 ±0.019220
Concatenated USPS data, $N_x = 50$				
NRMSE	0.93751 ±0.0021096	0.93464 ±0.0011451	0.92386 ±0.0012337	0.92333 ±0.0012275
Class. rate	0.68436 ±0.021519	0.71072 ±0.016717	0.72089 ±0.020555	0.72473 ±0.013017
	(the same)	**Sparse weighted distance network**		
NRMSE	0.93751 ±0.0021096	0.93240 ±0.0021419	0.91961 ±0.0008993	0.91872 ±0.0012659
Class. rate	0.68436 ±0.021519	0.71778 ±0.019281	0.74327 ±0.016493	0.75764 ±0.017270
Concatenated USPS data, $N_x = 100$				
NRMSE	0.91565 ±0.0016558	0.91812 ±0.0012217	0.90208 ±0.0020162	0.89884 ±0.0015389
Class. rate	0.75097 ±0.014010	0.75112 ±0.011407	0.76808 ±0.013785	0.78175 ±0.013120

ing diversifies the responses of the units in such a reservoir. Each unit during the training is "looking for" its "niche" input pattern to which it produces a high response, as shown in Figure 4.7. The reservoir tries to "inhabit" the input dynamics and intrinsically looks for patterns in it. The parts of the data that are more predictable get a more expressed representation in the reservoir space, as shown in Figure 4.5. The reservoir also can be seen as trying to predict the inputs, as the unit that is trained to best match (predict) the reservoir state at each moment in time is also trained to best match (predict) the input.

To summarize the paragraph above, there are at least three intuitive interpretations of what the self-organized reservoir is doing:

- Detecting common temporal patterns in input (e.g., Figure 4.7);
- "Amplifying" the power of predictable dynamics of input in its activation signal space (e.g. Figure 4.5);
- Learning to intrinsically predict the input.

The unsupervised training algorithms SOM and NG used here to train RNNs are not prone to bifurcations in training. These are indicated by the events during training where the average activation level of a group of neurons becomes much higher in a relatively short period of time and that of an another group simultaneously becomes low (Figure 4.17). However, these bifurcations are usually

"squashed" by the training schedule and have no big negative impact. The success of training depends a lot on the setting of the model parameters α, β, and γ, as well as the training schedule. Overall, this unsupervised training is relatively fast (compared to other RNN training techniques), in our case capable of training the reasonably big models in a single pass through the data.

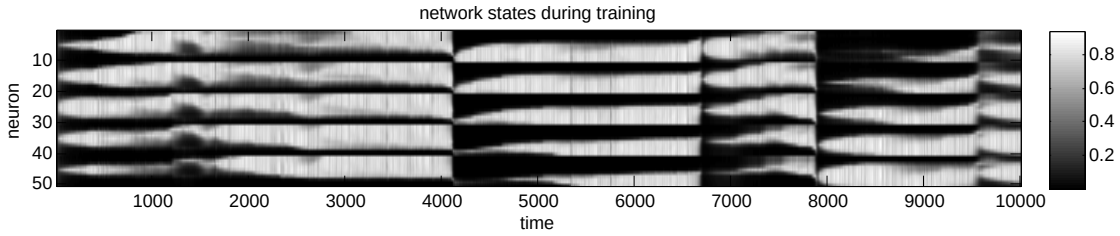


Figure 4.17: An example of bifurcations in an RBF reservoir trained by SOM algorithm with a static train schedule and bad model parameters.

Along the way, we have also introduced too alternative RNN models to the classical ESNs and demonstrated them to work better even with random weights for moderate network sizes: the SOR model (4.1) and a more general WDN model (4.10). Yet both of these models were demonstrated to work even better when trained in an unsupervised way.

We have exposed potential limitations of the computational model of SOR when dimensionality increases and the potential for better unsupervised training algorithms for the WDN model.

Sparseness of the models, realized by the WDN, was found to be an important aspect for both performance with moderately sized networks (see the sparse WDN results in Table 4.5) and scalability to bigger sizes.

Last, but not least, we have also demonstrated that hierarchies of our unsupervised reservoirs improve the performance by a large margin. The exact reasons for this need further investigation. Figure 4.16 gives an insight that the patterns in the data get more expressed in the principal components of the reservoir activations when going up in the layers. One reason for this could be that with more layers we simply get more powerful models having more unsupervisedly trained parameters. As a future work, it would be interesting to check if a single reservoir with a comparable number of parameters could achieve comparable performance. We have only compared the training effort so far. It could also be that the deep structure produces a more compact representation of the input which can not

easily be achieved with a single layer (examples of such are presented in [Bengio, 2009]). On one hand there is no really deep structure in our data, but on the other hand the fact that we get better representations in the reservoirs of the same dimension when going up the hierarchy is still spectacular.

Successful training of the same type of unsupervised hierarchies with the concatenated USPS data was not easily achievable at least with the small $N_{\mathrm{x}} = 50$ reservoirs. Due to a richer nature of the data, such bottom reservoirs are most probably inadequate to learn a good representation, and thus classification from the upper reservoirs deteriorates similarly to the inadequately trained hierarchies represented by blue curves in Figure 4.13.

There are still many other improvements that could be done to the introduced models, both to the structure and learning, as well as understanding it better. For example, an interesting and biologically motivated modification would be to also have top-down interactions in the hierarchies. It would also be interesting to see how the model scales up to even more challenging real-world data.

Chapter 5

Conclusions and open questions

In this thesis we have motivated, surveyed, and proposed a wide variety of ways for generating, training, and adapting recurrent neural networks that are alternatives to fully supervised training.

Most of the reviewed and proposed approaches are along the lines of reservoir computing paradigm. We have extensively and systematically reviewed this paradigm in Chapter 2.

This research direction is important, because less supervision in training is a prerequisite much needed both conceptually and technically for significant advancements in the field of machine learning. In particular, the lack of such algorithms hinders development and training of more complex, including deep and recurrent, ML architectures that could approach intelligence.

While these are big challenges, we try to contribute a few small steps toward training unsupervised recurrent hierarchical architectures capable of naturally processing temporal data, a challenge which is still largely untackled.

The self-organized reservoirs and their hierarchies, that we propose in Section 4, while mostly successful in the investigated settings, are still in the "proof of concept" stage. Their scalability to real world problems still remains an issue. In fact we have pointed out some of their limitations.

We believe that the key to scaling up the unsupervised RNN models and making them more powerful is reducing the dependences among its all components. This was successfully demonstrated with the sparse WDN example in Section 4.5. A possible future approach could be to further systematically disentangle big networks into interacting locally coherent sub-networks that could be trained mostly independently of each other. Unsupervised, or in fact any, training of such

networks is still largely unsolved.

Global unsupervised learning methods intrinsically do not scale very well to really big networks. They are also not biologically realistic, as they require complete global knowledge of the system. Physical learning systems are limited by their structure in what information is available where. This also becomes an issue in artificial ML systems when scaling them up and running on vastly parallel computers. For efficient implementations in such cases there should be some correspondence between the conceptual models and their physical computational materialization.

On the other hand, neurons in the network can not learn completely independently of each other, because no specialization can effectively be achieved without coordination. Striking the balance here is important. The natural biological mechanisms of how real neurons do this are still mostly unknown.

Our proposed unsupervised recurrent hierarchies in Section 4.6 are also at this stage just toy examples. One important feature needed in a powerful deep hierarchy is recurrence between the layers, i.e., connections going both up and down. There is a strong both conceptual and biological motivation to have them: they enable crucial phenomena such as attention or active perception. Bidirectional connections between layers, however, complicate the training a lot. Such hierarchy has to be trained simultaneously which usually leads to bifurcations during the learning that disrupt the training and functioning of the whole system. How natural systems overcome such limitations is, again, largely unknown.

One more fundamental question still not well answered is the relation between unsupervised and supervised training. While on one hand, universally good unsupervised representations seem plausible in the intuitive form of "understanding" the data, it is also clear that for the same input data very different output tasks can be formulated, and that not all the representations of the input data will be equally well suited for the tasks at hand. We have witnessed this in the Section 4.4.5.2, where a slight change in the formulation of the final task and performance, favors quite different unsupervised representations. A general more systematic mathematical account for the relation between the unsupervised and supervised goals in training is not available, as these goals are purely defined by the empirical data.

There is still a long road ahead but we hope to have contributed a step or two in understanding the issues and possible directions to tackle them in this thesis.

Bibliography

Future challenges for the science and engineering of learning. NSF Final workshop report, July 2007. URL http://cnl.salk.edu/Media/NSFWorkshopReport.v4.pdf.

David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

Eric A. Antonelo and Benjamin Schrauwen. Learning slow features with reservoir computing for biologically-inspired robot localization. *Neural Networks*, in press, 2011.

Eric A. Antonelo, Benjamin Schrauwen, and Dirk Stroobandt. Modeling multiple autonomous robot behaviors and behavior switching with a single reservoir computing network. In *Proc. IEEE Int. Conf. Systems, Man and Cybernetics SMC 2008*, pages 1843–1848, 2008. doi: 10.1109/ICSMC.2008.4811557.

Amir F. Atiya and Alexander G. Parlos. New results on recurrent network training: unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks*, 11(3):697–709, 2000.

Štefan Babinec and Jiří Pospíchal. Merging echo state and feedforward neural networks for time series forecasting. In *Proceedings of the 16th International Conference on Artificial Neural Networks (ICANN 2006)*, volume 4131 of *LNCS*, pages 367–375. Springer, 2006.

Štefan Babinec and Jiří Pospíchal. Improving the prediction accuracy of echo state neural networks by anti-Oja's learning. In *Proceedings of the 17th International Conference on Artificial Neural Networks (ICANN 2007)*, volume 4668 of *LNCS*, pages 19–28. Springer, 2007.

Roland Baddeley, Larry F. Abbott, Michael C. A. Booth, Frank Sengpeil, Toby Freeman, Edward A. Wakeman, and Edmund T. Rolls. Responses of neurons in primary and inferior temporal visual cortices to natural scenes. *Proc. R. Soc. Lond. B*, 264:1775–1783, 1997.

Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286:509, 1999.

Anthony J. Bell and Terrence J. Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7(6): 1129–1159, 1995.

Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009. doi: 10.1561/2200000006.

Yoshua Bengio and Yann LeCun. Scaling learning algorithms toward AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*. MIT Press, Cambridge, MA, 2007.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

Nils Bertschinger and Thomas Natschläger. Real-time computation at the edge of chaos in recurrent neural networks. *Neural Computation*, 16(7):1413–1436, 2004. ISSN 0899-7667.

Åke Björck. *Numerical Method for Least Squares Problems*. SIAM, Philadelphia, PA, USA, 1996.

Jean-Marc Blanc and Peter F. Dominey. Identificationof prosodic attitudes by atemporal recurrent network. *Cognitive Brain Research*, 17:693–699, 2003.

Joschka Boedecker, Oliver Obst, Norbert Michael Mayer, and Minoru Asada. Studies on reservoir initialization and dynamics shaping in echo state networks. In *Proceedings of the 17th European Symposium on Artificial Neural Networks (ESANN 2009)*, pages 227–232, 2009.

Michael Buehner and Peter Young. A tighter bound for the echo state property. *IEEE Transactions on Neural Networks*, 17(3):820–824, 2006.

Lars Buesing and Wolfgang Maass. Simplified rules and theoretical analysis for information bottleneck optimization and PCA with spiking neurons. In *Advances in Neural Information Processing Systems 20 (NIPS 2007)*, pages 193–200. MIT Press, Cambridge, MA, 2008.

Dean V. Buonomano and Michael M. Merzenich. Temporal information transformed into a spatial code by a neural network with realistic properties. *Science*, 267:1028–1030, 1995.

Keith Bush and Charles Anderson. Modeling reward functions for incomplete state representations via echo state networks. In *Proceedings of the IEEE International Joint Conference on Neural Networks, 2005 (IJCNN 2005)*, volume 5, pages 2995–3000, 2005.

Keith Bush and Charles Anderson. Exploiting iso-error pathways in the N,k-plane to improve echo state network performance, 2006.

Keith Bush and Batsukh Tsendjav. Improving the richness of echo state features using next ascent local search. In *Proceedings of the Artificial Neural Networks In Engineering Conference*, pages 227–232, St. Louis, MO, 2005.

Lars Büsing, Benjamin Schrauwen, and Robert Legenstein. Connectivity, dynamics, and memory in reservoir computing with binary and analog neurons. *Neural Computation*, 22(5):1272–1311, 2010. doi: http://dx.doi.org/10.1162/neco.2009.01-09-947.

John Butcher, David Verstraeten, Benjamin Schrauwen, Charles Day, and Peter Haycock. Extending reservoir computing with random static projections: a hybrid between extreme learning and RC. In *Proceedings of the 18th European Symposium on Artificial Neural Networks (ESANN 2010)*, pages 303–308, 2010.

Nicholas J. Butko and Jochen Triesch. Learning sensory representations with intrinsic plasticity. *Neurocomputing*, 70(7-9):1130–1138, 2007.

Andrew Carnell and Daniel Richardson. Linear algebra for time series of spikes. In *Proceedings of the 13th European Symposium on Artificial Neural Networks (ESANN 2005)*, pages 363–368, 2005.

Michal Čerňanský and Matej Makula. Feed-forward echo state networks. In *Proceedings of the IEEE International Joint Conference on Neural Networks, 2005 (IJCNN 2005)*, volume 3, pages 1479–1482, 2005.

Michal Čerňanský and Peter Tiňo. Predictive modeling with echo state networks. In *Proceedings of the 18th International Conference on Artificial Neural Networks (ICANN 2008)*, volume 5163 of *LNCS*, pages 778–787. Springer, 2008. ISBN 978-3-540-87535-2. doi: http://dx.doi.org/10.1007/978-3-540-87536-9_80.

Geoffrey J. Chappell and John G. Taylor. The temporal Kohonen map. *Neural Networks*, 6(3):441–445, 1993. ISSN 0893-608. doi: DOI:10.1016/0893-6080(93)90011-K.

Youngmin Cho and Lawrence K. Saul. Large-margin classification in infinite neural networks. *Neural Computation*, 22:2678–2697, 2010. doi: 10.1162/NECO_a_00018.

Leon O. Chua and Lin Yang. Cellular neural networks: theory. *IEEE Transactions on Circuits and Systems*, 35(10):1257–1272, 1988. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7600.

Dan C. Cireşan, Ueli Meier, Luca M. Gambardella, and Jürgen Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition. *Neural Computation*, 22(12):3207–3220, 2010. doi: 10.1162/NECO_a_00052.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

Alexandre Devert, Nicolas Bredeche, and Marc Schoenauer. Unsupervised learning of echo state networks: a case study in artificial embryogeny. In *Proceedings of the 8th International Conference on Artificial Evolution (EA 2007)*, volume 4926 of *LNCS*, pages 278–290. Springer, 2008.

Peter F. Dominey. Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning. *Biological Cybernetics*, 73:265–274, 1995.

Peter F. Dominey. From sensorimotor sequence to grammatical construction: evidence from simulation and neurophysiology. *Adaptive Behaviour*, 13(4):347–361, 2005.

Peter F. Dominey and Franck Ramus. Neural network processing of natural language: I. sensitivity to serial, temporal and abstract structure of language in the infant. *Language and Cognitive Processes*, 15(1):87–127, 2000.

Peter F. Dominey, Michel Hoen, Jean-Marc Blanc, and Taïssia Lelekov-Boissard. Neurological basis of language and sequential cognition: evidence from simulation, aphasia, and ERP studies. *Brain and Language*, 86:207–225, 2003.

Peter F. Dominey, Michel Hoen, and Toshio Inui. A neurolinguistic model of grammatical construction processing. *Journal of Cognitive Neuroscience*, 18 (12):2088–2107, 2006.

Kenji Doya. Bifurcations in the learning of recurrent neural networks. In *Proceedings of IEEE International Symposium on Circuits and Systems 1992*, volume 6, pages 2777–2780, 1992.

Xavier Dutoit, Hendrik Van Brussel, and Marnix Nutti. A first attempt of reservoir pruning for classification problems. In *Proceedings of the 15th European Symposium on Artificial Neural Networks (ESANN 2007)*, pages 507–512, 2007.

Xavier Dutoit, Benjamin Schrauwen, Jan Van Campenhout, Dirk Stroobandt, Hendrik Van Brussel, and Marnix Nuttin. Pruning and regularization in reservoir computing: a first insight. In *Proceedings of the 16th European Symposium on Artificial Neural Networks (ESANN 2008)*, pages 1–6, 2008.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14:179211, 1990.

Mark Embrechts, Luis Alexandre, and Jonathan Linton. Reservoir computing for static pattern recognition. In *Proceedings of the 17th European Symposium on Artificial Neural Networks (ESANN 2009)*, pages 245–250, 2009.

Dumitru Erhan. Exploration of combining ESN learning with gradient-descent RNN learning techniques. Bachelor's thesis, Jacobs University Bremen, 2004. Available online at http://www.eecs.jacobs-university.de/archive/bsc-2004/erhan.pdf.

Dumitru Erhan, Yoshua Bengio, Aaron C. Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, 2010. doi: 10.1145/1756006.1756025.

Behrouz Farhang-Boroujeny. *Adaptive Filters: Theory and Applications*. Wiley, 1998.

Igor Farkaš and Matthew W. Crocker. Systematicity in sentence processing with a recursive self-organizing neural network. In *Proceedings of the 15th European Symposium on Artificial Neural Networks (ESANN 2007)*, pages 49–54, 2007.

Chrisantha Fernando and Sampsa Sojakka. Pattern recognition in a bucket. In *Proceedings of the 7th European Conference on Advances in Artificial Life (ECAL 2003)*, volume 2801 of *LNCS*, pages 588–597. Springer, 2003.

Georg Fette and Julian Eggert. Short term memory and pattern matching with simple echo state network. In *Proceedings of the 15th International Conference on Artificial Neural Networks (ICANN 2005)*, volume 3696 of *LNCS*, pages 13–18. Springer, 2005.

Peter Foldiak and Dominik Endres. Sparse coding. *Scholarpedia*, 3(1):2984, 2008. URL http://www.scholarpedia.org/article/Sparse_coding.

Robert M. French. Catastrophic interference in connectionist networks. In L. Nadel, editor, *Encyclopedia of Cognitive Science*, volume 1, pages 431–435. Nature Publishing Group, 2003.

Ken-ichi Funahashi and Yuichi Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6:801–806, 1993.

Claudio Gallicchio and Alessio Micheli. Graph echo state networks. In *Proc. Int Neural Networks (IJCNN) Joint Conf*, pages 1–8, 2010a. doi: 10.1109/IJCNN.2010.5596796.

Claudio Gallicchio and Alessio Micheli. TreeESN: a preliminary experimental analysis. In *Proceedings of the 18th European Symposium on Artificial Neural Networks (ESANN 2010)*, pages 333–338, 2010b.

Surya Ganguli, Dongsung Huh, and Haim Sompolinsky. Memory traces in dynamical systems. *Proceedings of the National Academy of Sciences*, 105(48):18970–18975, 2008. doi: 10.1073/pnas.0804451105. URL http://www.pnas.org/content/105/48/18970.abstract.

Felix A. Gers, Jürgen Schmidhuber, and Fred A. Cummins. Learning to forget: continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000.

Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. PhD thesis, Technical University Munich, Munich, Germany, 2008.

Alex Graves and Jürgen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in Neural Information Processing Systems 21 (NIPS 2008)*, pages 545–552. MIT Press, Cambridge, MA, 2009.

Stefan Haeusler and Wolfgang Maass. A statistical analysis of information-processing properties of lamina-specific cortical microcircuit models. *Cerebral Cortex*, 17(1):149–162, 2007. ISSN 1047-3211. doi: http://dx.doi.org/10.1093/cercor/bhj132.

Márton A. Hajnal and András Lőrincz. Critical echo state networks. In *Proceedings of the 16th International Conference on Artificial Neural Networks (ICANN 2006)*, volume 4131 of *LNCS*, pages 658–667. Springer, 2006.

Barbara Hammer, Alessio Micheli, Alessandro Sperduti, and Marc Strickert. Recursive self-organizing network models. *Neural Networks*, 17(8-9):1061 – 1085, 2004. ISSN 0893-6080. doi: DOI:10.1016/j.neunet.2004.06.009.

Donald O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Wiley, New York, 1949.

Michiel Hermans and Benjamin Schrauwen. Memory in linear recurrent neural networks in continuous time. *Neural Networks*, 23(3):341–355, 2010a. ISSN 0893-6080. doi: DOI:10.1016/j.neunet.2009.08.008. URL http://www.sciencedirect.com/science/article/pii/S0893608009002305.

Michiel Hermans and Benjamin Schrauwen. Memory in reservoirs for high dimensional input. In *Proceedings of the IEEE International Joint Conference on Neural Networks, 2010 (IJCNN 2010)*, pages 1–7, 2010b. doi: 10.1109/IJCNN.2010.5596884.

Michiel Hermans and Benjamin Schrauwen. One step backpropagation through time for learning input mapping in reservoir computing applied to speech recog-

nition. In *Proc. IEEE Int Circuits and Systems (ISCAS) Symp*, pages 521–524, 2010c. doi: 10.1109/ISCAS.2010.5537568.

Michiel Hermans and Benjamin Schrauwen. Recurrent kernel machines: Computing with infinite echo state networks. *Neural Computation*, 24(1):104–133, 2012. doi: http://dx.doi.org/10.1162/NECO_a_00200.

Salah El Hihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in Neural Information Processing Systems 8 (NIPS 1995)*, pages 493–499. MIT Press, Cambridge, MA, 1996.

Geoffrey E. Hinton. Boltzmann machine. *Scholarpedia*, 2(5):1668, 2007. URL http://www.scholarpedia.org/article/Boltzmann_machine.

Geoffrey E. Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. URL http://www.sciencemag.org/cgi/content/abstract/313/5786/504.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Gregor M. Hoerzer. Reward-modulated Hebbian learning is able to induce coherent patterns of activity and simple memory functions in initially chaotic recurrent neural networks. In *CONAS workshop*, Ghent, Belgium, 2010. URL http://organic.elis.ugent.be/sites/organic.elis.ugent.be/files/conas2010_finalrevision.pdf.

John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992. ISBN 0262082136.

Gregor Holzmann and Helmut Hauser. Echo state networks with filter neurons and a delay and sum readout. *Neural Networks*, 23(2):244–256, 2010. doi: 10.1016/j.neunet.2009.07.004.

John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of National Academy of Sciences USA*, 79: 2554–2558, 1982.

John J. Hopfield. Hopfield network. *Scholarpedia*, 2(5):1977, 2007. URL http://www.scholarpedia.org/article/Hopfield_network.

Kazuo Ishii, Tijn van der Zant, Vlatko Bečanović, and Paul Plöger. Identification of motion with echo state network. In *Proceedings of the OCEANS 2004 MTS/IEEE – TECHNO-OCEAN 2004 Conference*, volume 3, pages 1205–1210, 2004.

Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology, 2001. URL http://www.faculty.jacobs-university.de/hjaeger/pubs/EchoStatesTechRep.pdf.

Herbert Jaeger. Short term memory in echo state networks. Technical Report GMD Report 152, German National Research Center for Information Technology, 2002a. URL http://www.faculty.jacobs-university.de/hjaeger/pubs/STMEchoStatesTechRep.pdf.

Herbert Jaeger. A tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the "echo state network" approach. Technical Report GMD Report 159, German National Research Center for Information Technology, 2002b. URL http://www.faculty.jacobs-university.de/hjaeger/pubs/ESNTutorialRev.pdf.

Herbert Jaeger. Adaptive nonlinear system identification with echo state networks. In *Advances in Neural Information Processing Systems 15 (NIPS 2002)*, pages 593–600. MIT Press, Cambridge, MA, 2003.

Herbert Jaeger. Reservoir riddles: suggestions for echo state network research. In *Proceedings of the IEEE International Joint Conference on Neural Networks, 2005 (IJCNN 2005)*, volume 3, pages 1460–1462, 2005.

Herbert Jaeger. Discovering multiscale dynamical features with hierarchical echo state networks. Technical Report No. 9, Jacobs University Bremen, 2007a.

Herbert Jaeger. Echo state network. *Scholarpedia*, 2(9):2330, 2007b. URL http://www.scholarpedia.org/article/Echo_state_network.

Herbert Jaeger. Reservoir self-control for achieving invariance against slow input distortions. Technical Report No. 23, Jacobs University Bremen, 2010.

Herbert Jaeger and Harald Haas. Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667): 78–80, 2004. doi: 10.1126/science.1091277. URL http://www.faculty.jacobs-university.de/hjaeger/pubs/ESNScience04.pdf.

Herbert Jaeger, Mantas Lukoševičius, Dan Popovici, and Udo Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Networks*, 20(3):335–352, 2007a.

Herbert Jaeger, Wolfgang Maass, and José C. Príncipe. Special issue on echo state networks and liquid state machines — Editorial. *Neural Networks*, 20(3): 287–289, 2007b.

Fei Jiang, Hugues Berry, and Marc Schoenauer. Unsupervised learning of echo state networks: balancing the double pole. In *Proceedings of the 10th Genetic and Evolutionary Computation Conference (GECCO 2008)*, pages 869–870. ACM, 2008a.

Fei Jiang, Hugues Berry, and Marc Schoenauer. Supervised and evolutionary learning of echo state networks. In *Proceedings of 10th International Conference on Parallel Problem Solving from Nature (PPSN 2008)*, volume 5199 of *LNCS*, pages 215–224. Springer, 2008b.

Ben Jones, Dov Stekelo, Jon Rowe, and Chrisantha Fernando. Is there a liquid state machine in the bacterium Escherichia coli? In *Procedings of the 1st IEEE Symposium on Artificial Life (ALIFE 2007)*, pages 187–191, 2007.

Marcus Kaiser and Claus C. Hilgetag. Spatial growth of real-world networks. *Physical Review E*, 69:036103, 2004.

Uma R. Karmarkar and Dean V. Buonomano. Timing in the absence of clocks: encoding time in neural network states. *Neuron*, 53(3):427–438, 2007.

William H. Kautz. Transient synthesis in the time domain. *IRE Transactions on Circuit Theory*, 1(3):29–39, 1954.

Werner M. Kistler and Chris I. De Zeeuw. Dynamical working memory and timed responses: the role of reverberating loops in the olivo-cerebellar system. *Neural Computation*, 14:2597–2626, 2002.

Stefan Klampfl, Robert Legenstein, and Wolfgang Maass. Information bottleneck optimization and independent component extraction with spiking neurons. In *Advances in Neural Information Processing Systems 19 (NIPS 2006)*, pages 713–720. MIT Press, Cambridge, MA, 2007.

Stefan Klampfl, Robert Legenstein, and Wolfgang Maass. Spiking neurons can learn to solve information bottleneck problems and to extract independent components. *Neural Computation*, 21(4):911–959, 2008.

Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, January 1982. doi: http://dx.doi.org/10.1007/BF00337288.

Teuvo Kohonen and Timo Honkela. Kohonen network. *Scholarpedia*, 2(1):1568, 2007. URL http://www.scholarpedia.org/article/Kohonen_network.

Jan Koutník. Inductive modelling of temporal sequences by means of self-organization. In *Proceeding of Internation Workshop on Inductive Modelling (IWIM 2007)*, pages 269–277. CTU in Prague, 2007. ISBN 978-80-01-03881-9.

Ali U. Küçükemre. Echo state networks for adaptive filtering. Master's thesis, University of Applied Sciences Bohn-Rhein-Sieg, Germany, 2006. Available online at http://reservoir-computing.org/publications/2006-echo-state-networks-adaptive-filtering.

Andreea Lazar. *Self-organizing recurrent neural networks*. PhD thesis, Goethe University Frankfurt, Frankfurt am Main, Germany, 2010.

Andreea Lazar, Gordon Pipa, and Jochen Triesch. Fading memory and time series prediction in recurrent networks with different forms of plasticity. *Neural Networks*, 20(3):312–322, 2007.

Andreea Lazar, Gordon Pipa, and Jochen Triesch. SORN: a self-organizing recurrent neural network. *Frontiers in Computational Neuroscience*, 4(0):12, 2010. ISSN 1662-5188. doi: 10.3389/neuro.10.023.2009.

Andreea Lazar, Gordon Pipa, and Jochen Triesch. Emerging Bayesian priors in a self-organizing recurrent network. In *Proceedings of the 21st International Conference on Artificial Neural Networks (ICANN 2011)*, volume 6792 of *LNCS*, pages 127–134. Springer, 2011.

Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. doi: 10.1162/neco.1989.1.4.541.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998. doi: 10.1109/5.726791.

Robert Legenstein and Wolfgang Maass. What makes a dynamical system computationally powerful? In S. Haykin, J. Príncipe, T. Sejnowski, and J. McWhirter, editors, *New Directions in Statistical Signal Processing: From Systems to Brain*, pages 127–154. MIT Press, 2007a.

Robert Legenstein, Dejan Pecevski, and Wolfgang Maass. Theoretical analysis of learning with reward-modulated spike-timing-dependent plasticity. In *Advances in Neural Information Processing Systems 20 (NIPS 2007)*, pages 881–888. MIT Press, Cambridge, MA, 2008a.

Robert Legenstein, Dejan Pecevski, and Wolfgang Maass. A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Computational Biology*, 4(10):e1000180, 2008b.

Robert Legenstein, Steven M. Chase, Andrew B. Schwartz, and Wolfgang Maass. A reward-modulated hebbian learning rule can explain experimentally observed network reorganization in a brain control task. *Journal of Neuroscience*, 30(25): 8400–8410, Jun 2010. doi: 10.1523/JNEUROSCI.4284-09.2010.

Robert A. Legenstein and Wolfgang Maass. Edge of chaos and prediction of computational performance for neural circuit models. *Neural Networks*, 20(3):323–334, 2007b.

Jiwen Li and Herbert Jaeger. Minimal energy control of an ESN pattern generator. Technical Report No. 26, Jacobs University Bremen, 2010.

Benjamin Liebald. Exploration of effects of different network topologies on the ESN signal crosscorrelation matrix spectrum. Bachelor's thesis, Jacobs University Bremen, 2004. Available online at http://www.eecs.jacobs-university.de/archive/bsc-2004/liebald.pdf.

Carlos Lourenço. Dynamical reservoir properties as network effects. In *Proceedings of the 14th European Symposium on Artificial Neural Networks (ESANN 2006)*, pages 503–508, 2006.

Mantas Lukoševičius. Echo state networks with trained feedbacks. Technical Report No. 4, Jacobs University Bremen, February 2007. URL http://jpubs.jacobs-university.de/bitstream/579/959/1/tfbesn_iubtechreport.pdf.

Mantas Lukoševičius. On self-organizing reservoirs and their hierarchies. Technical Report No. 25, Jacobs University Bremen, October 2010. URL http://minds.jacobs-university.de/sites/default/files/uploads/papers/2396_Lukosevicius10.pdf.

Mantas Lukoševičius and Herbert Jaeger. Overview of reservoir recipes. Technical Report No. 11, Jacobs University Bremen, 2007. URL http://jpubs.jacobs-university.de/bitstream/579/138/1/reservoiroverview_techreport11.pdf.

Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3): 127–149, August 2009. ISSN 1574-0137. doi: 10.1016/j.cosrev.2009.03.005. URL http://www.faculty.jacobs-university.de/hjaeger/pubs/2261_LukoseviciusJaeger09.pdf.

Mantas Lukoševičius, Dan Popovici, Herbert Jaeger, and Udo Siewert. Time warping invariant echo state networks. Technical Report No. 2, Jacobs University Bremen, May 2006. URL http://jpubs.jacobs-university.de/bitstream/579/149/1/twiesn_iubtechreport.pdf.

Sheng Ma and Chuanyi Ji. Fast training of recurrent networks based on the EM algorithm. *IEEE Transactions on Neural Networks*, 9(1):11–26, 1998. ISSN 1045-9227. doi: 10.1109/72.655025.

Wolfgang Maass. Liquid state machines: motivation, theory, and applications. In B. Cooper and A. Sorbi, editors, *Computability in Context: Computation and Logic in the Real World*. Imperial College Press, 2011.

Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: a new framework for neural computation based on

perturbations. *Neural Computation*, 14(11):2531–2560, 2002. ISSN 0899-7667. doi: http://dx.doi.org/10.1162/089976602760407955.

Wolfgang Maass, Thomas Natschläger, and Henry Markram. A model for real-time computation in generic neural microcircuits. In *Advances in Neural Information Processing Systems 15 (NIPS 2002)*, pages 213–220. MIT Press, Cambridge, MA, 2003.

Wolfgang Maass, Thomas Natschläger, and Henry Markram. Computational models for generic cortical microcircuits. In *Computational Neuroscience: A Comprehensive Approach*, pages 575–605. Chapman & Hall/CRC, 2004.

Wolfgang Maass, Robert A. Legenstein, and Nils Bertschinger. Methods for estimating the computational power and generalization capability of neural microcircuits. In *Advances in Neural Information Processing Systems 17 (NIPS 2004)*, pages 865–872. MIT Press, Cambridge, MA, 2005.

Wolfgang Maass, Prashant Joshi, and Eduardo D. Sontag. Principles of real-time computing with feedback applied to cortical microcircuit models. In *Advances in Neural Information Processing Systems 18 (NIPS 2005)*, pages 835–842. MIT Press, Cambridge, MA, 2006.

Wolfgang Maass, Prashant Joshi, and Eduardo D. Sontag. Computational aspects of feedback in neural circuits. *PLoS Computational Biology*, 3(1):e165+, 2007.

Man Wai Mak. A learning algorithm for recurrent radial basis function networks. *Neural Processing Letters*, 2(1):27–31, 1995. ISSN 1370-4621. doi: http://dx.doi.org/10.1007/BF02312380.

Henry Markram, Yun Wang, and Misha Tsodyks. Differential signaling via the same axon of neocortical pyramidal neurons. *Proceedings of National Academy of Sciences USA*, 95(9):5323–5328, 1998. ISSN 0027-8424. URL http://view.ncbi.nlm.nih.gov/pubmed/9560274.

Thomas Martinetz and Klaus Schulten. A "neural-gas" network learns topologies. *Artificial Neural Networks*, 1:397–402, 1991.

Norbert M. Mayer and Matthew Browne. Echo state networks and self-prediction. In *Revised Selected Papers of Biologically Inspired Approaches to Advanced Information Technology (BioADIT 2004)*, pages 40–48, 2004.

Melanie Mitchell, James P. Crutchfield, and Peter T. Hraber. Dynamics, computation, and the "edge of chaos": a re-examination. In G. Cowan, D. Pines, and D. Melzner, editors, *Complexity: Metaphors, Models, and Reality*, pages 497–513. Addison-Wesley, Reading, MA, 1994.

Thomas Natschläger, Henry Markram, and Wolfgang Maass. Computer models and analysis tools for neural microcircuits. In R. Kötter, editor, *A Practical Guide to Neuroscience Databases and Associated Tools*, chapter 9. Kluver Academic Publishers (Boston), 2002.

Danko Nikolić, Stefan Haeusler, Wolf Singer, and Wolfgang Maass. Temporal dynamics of information content carried by neurons in the primary visual cortex. In *Advances in Neural Information Processing Systems 19 (NIPS 2006)*, pages 1041–1048. MIT Press, Cambridge, MA, 2007.

David Norton and Dan Ventura. Preparing more effective liquid state machines using hebbian learning. *Proceedings of the IEEE International Joint Conference on Neural Networks, 2006 (IJCNN 2006)*, pages 4243–4248, 2006.

Mohamed Oubbati, Paul Levi, and Michael Schanz. Meta-learning for adaptive identification of non-linear dynamical systems. In *Proceedings of the IEEE International Joint Symposium on Intelligent Control*, pages 473–478, 2005.

Mustafa C. Ozturk and José C. Príncipe. Computing with transiently stable states. In *Proceedings of the IEEE International Joint Conference on Neural Networks, 2005 (IJCNN 2005)*, volume 3, pages 1467–1472, 2005.

Mustafa C. Ozturk and José C. Príncipe. An associative memory readout for ESNs with applications to dynamical pattern recognition. *Neural Networks*, 20 (3):377–390, 2007.

Mustafa C. Ozturk, Dongming Xu, and José C. Príncipe. Analysis and design of echo state networks. *Neural Computation*, 19(1):111–138, 2007.

Hélène Paugam-Moisy, Regis Martinez, and Samy Bengio. Delay learning and polychronization for reservoir computing. *Neurocomputing*, 71(7-9):1143–1158, 2008.

Danil V. Prokhorov, Lee A. Feldkamp, and Ivan Yu. Tyukin. Adaptive behavior with fixed weights in RNN: an overview. In *Proceedings of the IEEE International Joint Conference on Neural Networks, 2002 (IJCNN 2002)*, pages 2018–2023, 2002.

Geoffrey K. Pullum and Barbara C. Scholz. Empirical assessment of stimulus poverty arguments. *The Linguistic Review*, 19:9–50, 2002. URL http://ling.ed.ac.uk/~gpullum/bcscholz/Assessment.pdf.

Gintaras V. Puškorius and Lee A. Feldkamp. Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks. *IEEE Transactions on Neural Networks*, 5(2):279–297, 1994. ISSN 1045-9227.

Ali Ajdari Rad, Mahdi Jalili, and Martin Hasler. Reservoir optimization in recurrent neural networks using Kronecker kernels. In *Proceedings of IEEE International Symposium on Circuits and Systems 2008*, pages 868–871. IEEE, 2008.

Felix R. Reinhart and Jochen J. Steil. Recurrent neural autoassociative learning of forward and inverse kinematics for movement generation of the redundant PA-10 robot. In *Proceedings of the ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems (LAB-RS)*, volume 1, pages 35–40. IEEE Computer Society Press, 2008.

Felix R. Reinhart and Jochen J. Steil. Attractor-based computation with reservoirs for online learning of inverse kinematics. In *Proceedings of the 17th European Symposium on Artificial Neural Networks (ESANN 2009)*, pages 257–262, 2009.

Felix R. Reinhart and Jochen J. Steil. Reservoir regularization stabilizes learning of echo state networks with output feedback. In *Proceedings of the 19th European Symposium on Artificial Neural Networks (ESANN 2011)*, 2011a. In Press.

Felix R. Reinhart and Jochen J. Steil. A constrained regularization approach for input-driven recurrent neural networks. *Differential Equations and Dynamical Systems*, 19:27 – 46, 2011b. doi: 10.1007/s12591-010-0067-x.

Ali Rodan and Peter Tiňo. Minimum complexity echo state network. *IEEE Transactions on Neural Networks*, 22(1):131–144, 2011.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In *Neurocomputing: Foundations of research*, pages 673–695. MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-01097-6.

Vytenis Šakėnas. Distortion invariant feature extraction with echo state networks. Technical Report No. 24, Jacobs University Bremen, 2010.

Ulf D. Schiller and Jochen J. Steil. Analyzing the weight dynamics of recurrent learning algorithms. *Neurocomputing*, 63C:5–23, 2005.

Jürgen Schmidhuber, Matteo Gagliolo, Daan Wierstra, and Faustino J. Gomez. Evolino for recurrent support vector machines. In *Proceedings of the 14th European Symposium on Artificial Neural Networks (ESANN 2006)*, pages 593–598, 2006.

Jürgen Schmidhuber, Daan Wierstra, Matteo Gagliolo, and Faustino J. Gomez. Training recurrent networks by Evolino. *Neural Computation*, 19(3):757–779, 2007.

Benjamin Schrauwen and Jan Van Campenhout. Linking non-binned spike train kernels to several existing spike train metrics. In M. Verleysen, editor, *Proceedings of the 14th European Symposium on Artificial Neural Networks (ESANN 2006)*, pages 41–46, Evere, 2006. d-side publications.

Benjamin Schrauwen, Jeroen Defour, David Verstraeten, and Jan M. Van Campenhout. The introduction of time-scales in reservoir computing, applied to isolated digits recognition. In *Proceedings of the 17th International Conference on Artificial Neural Networks (ICANN 2007)*, volume 4668 of *LNCS*, pages 471–479. Springer, 2007a.

Benjamin Schrauwen, David Verstraeten, and Jan Van Campenhout. An overview of reservoir computing: theory, applications and implementations. In *Proceedings of the 15th European Symposium on Artificial Neural Networks (ESANN 2007)*, pages 471–482, 2007b.

Benjamin Schrauwen, Michiel D'Haene, Davfid Verstraeten, and Dirk Stroobandt. Compact hardware liquid state machines on FPGA for real-time speech recognition. *Neural Networks*, 21(2-3):511–523, 2008a.

Benjamin Schrauwen, Marion Wardermann, David Verstraeten, Jochen J. Steil, and Dirk Stroobandt. Improving reservoirs using intrinsic plasticity. *Neurocomputing*, 71(7-9):1159–1171, 2008b.

Benjamin Schrauwen, Lars Büsing, and Robert Legenstein. On computational power and the order-chaos phase transition in reservoir computing. In *Advances in Neural Information Processing Systems 21 (NIPS 2008)*, pages 1425–1432. MIT Press, Cambridge, MA, 2009.

Felix Schürmann, Karlheinz Meier, and Johannes Schemmel. Edge of chaos computation in mixed-mode VLSI - a hard liquid. In *Advances in Neural Information Processing Systems 17 (NIPS 2004)*, pages 1201–1208. MIT Press, Cambridge, MA, 2005.

Zhinwei Shi and Min Han. Support vector echo-state machine for chaotic time-series prediction. *IEEE Transactions on Neural Networks*, 18(2):359–72, 2007.

Udo Siewert and Welf Wustlich. Echo-state networks with band-pass neurons: towards generic time-scale-independent reservoir structures. Internal status report, PLANET intelligent systems GmbH, 2007. Available online at `http://reslab.elis.ugent.be/node/112`.

Mark D. Skowronski and John G. Harris. Automatic speech recognition using a predictive echo state network classifier. *Neural Networks*, 20(3):414–423, 2007.

Garrett B. Stanley, Fei F. Li, and Yang Dan. Reconstruction of natural scenes from ensemble responses in the lateral genicualate nucleus. *Journal of Neuroscience*, 19(18):8036–8042, 1999.

Jochen J. Steil. Backpropagation-decorrelation: recurrent learning with O(N) complexity. In *Proceedings of the IEEE International Joint Conference on Neural Networks, 2004 (IJCNN 2004)*, volume 2, pages 843–848, 2004.

Jochen J. Steil. Stability of backpropagation-decorrelation efficient O(N) recurrent learning. In *Proceedings of the 13th European Symposium on Artificial Neural Networks (ESANN 2005)*, pages 43–48, 2005a.

Jochen J. Steil. Memory in backpropagation-decorrelation O(N) efficient online recurrent learning. In *Proceedings of the 15th International Conference on Ar-*

tificial Neural Networks (ICANN 2005), volume 3697 of *LNCS*, pages 649–654. Springer, 2005b.

Jochen J. Steil. Several ways to solve the MSO problem. In *Proceedings of the 15th European Symposium on Artificial Neural Networks (ESANN 2007)*, pages 489–494, 2007a.

Jochen J. Steil. Online reservoir adaptation by intrinsic plasticity for backpropagation-decorrelation and echo state learning. *Neural Networks*, 20 (3):353–364, 2007b.

Martin Stemmler and Christof Koch. How voltage-dependent conductances can adapt to maximize the information encoded by neuronal firing rate. *Nature Neuroscience*, 2(6):521–527, 1999.

David Sussillo and Larry F. Abbott. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4):544–557, 2009. doi: DOI:10.1016/j.neuron.2009.07.018.

Ilya Sutskever and Geoffrey Hinton. Temporal-kernel recurrent neural networks. *Neural Networks*, 23(2):239–243, 2010. ISSN 0893-6080. doi: DOI:10.1016/j.neunet.2009.10.009.

Floris Takens. Detecting strange attractors in turbulence. In *Proceedings of a Symposium on Dynamical Systems and Turbulence*, volume 898 of *LNM*, pages 366–381. Springer, 1981.

Graham W. Taylor, Geoffrey E. Hinton, and Sam Roweis. Modeling human motion using binary latent variables. In *Advances in Neural Information Processing Systems 19 (NIPS 2006)*, pages 1345–1352. MIT Press, Cambridge, MA, 2007.

Joe Tebelskis. *Speech Recognition using Neural Networks*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1995.

Peter Tiňo, Igor Farkaš, and Jort van Mourik. Dynamics and topographic organization of recursive self-organizing maps. *Neural Computation*, 18(10):2529–2567, 2006.

Naftali Tishby, Fernando C. Pereira, and William Bialek. The information bottleneck method. In *Proceedings of the 37th Annual Allerton Conference on Communication, Control and Computing*, pages 368–377, 1999.

Taro Toyoizumi, Jean-Pascal Pfister, Kazuyuki Aihara, and Wulfram Gerstner. Generalized Bienenstock-Cooper-Munro rule for spiking neurons that maximizes information transmission. *Procedings of National Academy of Sciences USA*, 102:5239–5244, 2005. URL http://infoscience.epfl.ch/getfile.py?mode=best&recid=97821.

Fabian Triefenbach, Azarakhsh Jalalvand, Benjamin Schrauwen, and Jean-Pierre Martens. Phoneme recognition with large hierarchical reservoirs. In *Advances in Neural Information Processing Systems 23 (NIPS 2010)*, pages 2307–2315. MIT Press, Cambridge, MA, 2011.

Jochen Triesch. Synergies between intrinsic and synaptic plasticity in individual model neurons. In *Advances in Neural Information Processing Systems 17 (NIPS 2004)*, pages 1417–1424. MIT Press, Cambridge, MA, 2005a.

Jochen Triesch. A gradient rule for the plasticity of a neuron's intrinsic excitability. In *Proceedings of the 15th International Conference on Artificial Neural Networks (ICANN 2005)*, volume 3696 of *LNCS*, pages 65–70. Springer, 2005b.

Jochen Triesch. Synergies between intrinsic and synaptic plasticity mechanisms. *Neural Computation*, 19(4):885–909, 2007.

Kristof Vandoorne, Wouter Dierckx, Benjamin Schrauwen, David Verstraeten, Roel Baets, Peter Bienstman, and Jan Van Campenhout. Toward optical signal processing using photonic reservoir computing. *Optics Express*, 16(15):11182–11192, 2008.

Markus Varsta, José Del R. Millán, and Jukka Heikkonen. A recurrent self-organizing map for temporal sequence processing. In *Proceedings of the 7th International Conference on Artificial Neural Networks (ICANN 1997)*, volume 1327 of *LNCS*, pages 421–426. Springer, 1997. ISBN 978-3-540-63631-1. doi: 10.1007/BFb0020191.

David Verstraeten. *Reservoir computing: computation with dynamical systems*. PhD thesis, Ghent University. Faculty of Engineering, Ghent, Belgium, 2009.

David Verstraeten, Benjamin Schrauwen, and Dirk Stroobandt. Reservoir computing with stochastic bitstream neurons. In *Proceedings of the 16th Annual ProRISC Workshop*, pages 454–459, Veldhoven, The Netherlands, 2005a.

David Verstraeten, Benjamin Schrauwen, Dirk Stroobandt, and Jan Van Campenhout. Isolated word recognition with the liquid state machine: a case study. *Information Processing Letters*, 95(6):521–528, 2005b.

David Verstraeten, Benjamin Schrauwen, and Dirk Stroobandt. Reservoir-based techniques for speech recognition. In *Proceedings of the IEEE International Joint Conference on Neural Networks, 2006 (IJCNN 2006)*, pages 1050 – 1053, 2006.

David Verstraeten, Benjamin Schrauwen, Michiel D'Haene, and Dirk Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks*, 20(3):391–403, 2007a.

David Verstraeten, Benjamin Schrauwen, and Dirk Stroobandt. Adapting reservoirs to get Gaussian distributions. In *Proceedings of the 15th European Symposium on Artificial Neural Networks (ESANN 2007)*, pages 495–500, 2007b.

David Verstraeten, Joni Dambre, Xavier Dutoit, and Benjamin Schrauwen. Memory versus non-linearity in reservoirs. In *Proc. Int Neural Networks (IJCNN) Joint Conf*, pages 1–8, 2010. doi: 10.1109/IJCNN.2010.5596492.

Thomas Voegtlin. Recursive self-organizing maps. *Neural Networks*, 15(8-9):979–991, 2002. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/S0893-6080(02)00072-2.

Se Wang, Xiao-Jian Yang, and Cheng-Jian Wei. Harnessing non-linearity by sigmoid-wavelet hybrid echo state networks (SWHESN). *The 6th World Congress on Intelligent Control and Automation (WCICA 2006)*, 1:3014– 3018, 2006.

Marion Wardermann and Jochen J. Steil. Intrinsic plasticity for reservoir learning algorithms. In *Proceedings of the 15th European Symposium on Artificial Neural Networks (ESANN 2007)*, pages 513–518, 2007.

Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

Paul J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. ISSN 0018-9219.

Olivia L. White, Daniel D. Lee, and Haim Sompolinsky. Short-term memory in orthogonal neural networks. *Physical Review Letters*, 92(14):148102, Apr 2004. doi: 10.1103/PhysRevLett.92.148102.

Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.

Laurenz Wiskott, Pietro Berkes, Mathias Franzius, Henning Sprekeler, and Niko Wilbert. Slow feature analysis. *Scholarpedia*, 6(4):5282, 2011. URL http://www.scholarpedia.org/article/Slow_feature_analysis.

David H. Wolpert. The supervised learning no-free-lunch theorems. In *Proceedings of the 6th Online World Conference on Soft Computing in Industrial Applications (WSC 2006)*, pages 25–42, 2001.

Francis wyffels, Benjamin Schrauwen, and Dirk Stroobandt. Stable output feedback in reservoir computing using ridge regression. In *Proceedings of the 18th International Conference on Artificial Neural Networks (ICANN 2008)*, volume 5163 of *LNCS*, pages 808–817. Springer, 2008a.

Francis wyffels, Benjamin Schrauwen, David Verstraeten, and Dirk Stroobandt. Band-pass reservoir computing. In Z. Hou and N. Zhang, editors, *Proceedings of the IEEE International Joint Conference on Neural Networks, 2008 (IJCNN 2008)*, pages 3204–3209, Hong Kong, 2008b.

Dongming Xu, Jing Lan, and José C. Príncipe. Direct adaptive control: an echo state network and genetic algorithm approach. In *Proceedings of the IEEE International Joint Conference on Neural Networks, 2005 (IJCNN 2005)*, volume 3, pages 1483–1486, 2005.

Yanbo Xue, Le Yang, and Simon Haykin. Decoupled echo state networks with lateral inhibition. *Neural Networks*, 20(3):365–376, 2007.

Tadashi Yamazaki and Shigeru Tanaka. The cerebellum as a liquid state machine. *Neural Networks*, 20(3):290–297, 2007.

Christoph Zechner and Dmitriy Shutin. Bayesian learning of echo state networks with tunable filters and delay&sum readouts. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 1998–2001, march 2010. doi: 10.1109/ICASSP.2010.5495225.

Number of bibliography items: 209.