Reservoir Computing: rekenen met dynamische systemen

Reservoir Computing: Computation with Dynamical Systems

David Verstraeten

Promotoren: prof. dr. ir. D. Stroobandt, dr. ir. B. Schrauwen
Proefschrift ingediend tot het behalen van de graad van
Doctor in de Ingenieurswetenschappen: Computerwetenschappen

Vakgroep Elektronica en Informatiesystemen
Voorzitter: prof. dr. ir. J. Van Campenhout
Faculteit Ingenieurswetenschappen
Academiejaar 2009 - 2010

ISBN 978-90-8578-309-1
NUR 984
Wettelijk depot: D/2009/10.500/67

Dankwoord

Dit proefschrift vertegenwoordigt een grote persoonlijke inspanning, maar tegelijk zou dit werk niet mogelijk zijn geweest zonder de ondersteuning van vele anderen. Ik wens iedereen die betrokken was bij mijn onderzoek daarom van harte te bedanken, ook zij die hier niet bij naam genoemd zijn.

Eerst en vooral bedank ik mijn promotoren dr. Benjamin Schrauwen en prof. Dirk Stroobandt. Benjamin, bedankt voor de begeleiding van de voorbije jaren en het opzetten van een zeer aangename werkomgeving. Ik heb onderzoek kunnen doen in een stimulerende omgeving samen met een jong en gemotiveerd team van enthousiaste onderzoekers en dat is jouw verdienste. Onze beste ideeën zijn ontstaan op conferenties en die momenten hebben me altijd gemotiveerd om 'goeie science' te produceren. Dirk, bedankt om de kans en de ondersteuning te bieden die mij in staat hebben gesteld dit doctoraat te doen.

Ik wil ook prof. Joni Dambre en prof. Jan Van Campenhout bedanken voor het grondige nalezen (soms meerdere keren zelfs) en voor de waardevolle raad die dit werk hebben opgetild tot het huidige niveau. Jullie visie, vaak vanuit een heel andere invalshoek dan ik gewend was, heeft me gedwongen om mijn werk op andere manieren te bekijken en dat heeft zeker bijgedragen tot de kwaliteit van dit proefschrift.

Verder ook een bedanking voor mijn collega's om te helpen om van het Reservoir Lab niet alleen een professioneel onderzoeksteam, maar in de eerste plaats een leuke werkomgeving te maken. Merci, zowel aan de 'anciens' Michiel D'Haene, Francis wyffels, Eric Antonelo, Pieter Buteneers, Michiel Hermans en Xavier Dutoit, als aan de nieuwelingen: Antonio Rebordao, Tim Waegeman, Philemon Brakel, Fionntan O'Donnell. Ook bedankt aan de collega's van HES om van de groepstweedaagse telkens een leuke uitstap te maken: Fatma Abouelella, Peter Bertels, Karel Bruneel,

Harald Devos, Tom Davidson, Wim Heirman en Craig Moore.

Bedankt ook aan de collega-onderzoekers van de Photonics groep in INTEC: prof. Peter Bienstman, Martin Fiers en Kristof Vandoorne, voor de goede samenwerking rond een veelbelovende en opwindende volgende stap in het research rond Reservoir Computing.

I also wish to thank the people at Planet GmbH, not only for offering the opportunity to do research in an industrial setting and to work on real-world engineering problems, but also for providing a very welcoming and hospitable working and living environment during my stay there. Welf, Udo, Klaus, Ulli, Hagen, Donata and the other Planeteers: thanks a lot, I enjoyed my 'German episode' very much.

Bedankt aan Wim Meeus, Ronny Blomme en Michiel Ronsse voor de logistieke steun en de hulp bij het opzetten van allerhande ICT-gerelateerde constructies die mijn onderzoek mee hebben mogelijk gemaakt. Ik wens ook de partners in het HomeMATE project te bedanken voor de goede samenwerking en ik hoop dat we dit kunnen voortzetten in de toekomst.

Verder ook dank aan de leden van mijn examencommissie, en in het bijzonder prof. Tom Dhaene, prof. Herbert Peremans en prof. Herbert Jaeger voor de waardevolle raadgevingen en opmerkingen over mijn proefschrift.

Ik wil verder ook zeker mijn moeder, zus en (schoon-)familie en vrienden bedanken. Hoewel jullie niet rechtstreeks bij mijn onderzoek betrokken geweest zijn, hebben jullie ook bijgedragen tot dit werk door jullie aanhoudende steun, aanmoedigingen en interesse.

Tot slot, maar eigenlijk in de eerste plaats, wil ik mijn vrouw Stans en onze zoon Louis bedanken voor de voorbije jaren. Jullie hebben het de voorbije jaren mogelijk gemaakt om dit werk te doen door van ons huis een echte thuis te maken. Bedankt om dit met mij te delen.

David Verstraeten
Gent, 30 september 2009

Dit werk is ondersteund door het Instituut voor de Aanmoediging van Innovatie door Wetenschap en Technologie Vlaanderen (IWT Vlaanderen).

This work was supported by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT Vlaanderen).

Examencommissie

Prof.	Rik Van de Walle, voorzitter Academisch secretaris, Faculteit Ingenieurswetenschappen Universiteit Gent
Prof.	Joni Dambre, secretaris Vakgroep ELIS, Faculteit Ingenieurswetenschappen Universiteit Gent
Prof.	Dirk Stroobandt, promotor Vakgroep ELIS, Faculteit Ingenieurswetenschappen Universiteit Gent
Dr. ir.	Benjamin Schrauwen, promotor Vakgroep ELIS, Faculteit Ingenieurswetenschappen Universiteit Gent
Prof.	Jan Van Campenhout Vakgroep ELIS, Faculteit Ingenieurswetenschappen Universiteit Gent
Prof.	Tom Dhaene Vakgroep INTEC, Faculteit Ingenieurswetenschappen Universiteit Gent
Prof.	Herbert Peremans Active Perception Lab, Vakgroep Milieu, Technologie en technologiemanagement, Faculteit Toegepaste Economische Wetenschappen Universiteit Antwerpen

Prof. Herbert Jaeger
School of Engineering and Science
Jacobs University

Eerste (interne) verdediging: 22 september 2009, 15h00
Openbare verdediging: 13 oktober 2009, 18h00

Samenvatting

Reservoir Computing

Een groot aantal uitdagende en interessante problemen voor ingenieurs kunnen niet worden opgelost met heuristische methoden of expliciet geprogrammeerde algoritmen. Deze problemen zijn kandidaten bij uitstek om aangepakt te worden met machine learning-methoden. Deze methoden hebben alle de eigenschap dat ze leren uit voorbeelden, en dat ze deze voorbeelden kunnen generalizeren op een 'intelligente' manier naar nieuwe, ongeziene invoer. Er bestaan veel machine learning methoden, en een grote subklasse wordt gevormd door de Neurale Netwerken (NN). NN zijn zeer abstracte, connectionistische modellen van de manier waarop het brein "rekent". Ze bestaan uit netwerken van eenvoudige, niet-lineaire rekenknopen die waarden communiceren langs gewogen verbindingen. Door deze gewichten aan te passen (te trainen) op basis van voorbeelden kan het gewenste gedrag van het netwerk verkregen worden. Als het netwerk een recurrente structuur heeft (i.e., terugkoppellussen bevat), dan zal het een geheugen hebben van invoer uit het verleden, waardoor het temporele signalen kan verwerken en waardoor deze netwerken krachtige niet-lineaire temporele rekenmethodes worden. Deze recurrente neurale netwerken zijn echter berucht vanwege hun moeilijke trainbaarheid.

Recent werd een nieuw leerparadigma geïntroduceerd met de naam Reservoir Computing (RC). Deze methode stelt ons in staat om recurrente neurale netwerken te gebruiken zonder een lange en moeilijke trainingsfaze. Dezelfde basisidee is onafhankelijk geïntroduceerd als Echo State Netwerken en Liquid State Machines. In beide gevallen bestaat het systeem uit twee delen: een recurrent netwerk van neuronen dat het reservoir genoemd wordt en dat willekeurig geconstrueerd wordt en verder niet getraind wordt, en een aparte lineaire uitleeslaag die getraind wordt met eenvoudige eenstapsmethodes. De niet-lineaire transformatie en het

korte-termijngeheugen van het reservoir versterkt het rekenvermogen van de eenvoudige, geheugenloze lineare uitvoerlaag.

Deze thesis

Sinds zijn introductie in 2002 heeft Reservoir Computing veel aandacht getrokken binnen de onderzoeksgemeenschap rond neurale netwerken door de combinatie van zijn eenvoud van gebruik en zijn uitstekende prestatie op een brede waaier van toepassingen. In dit doctoraat zullen wij aantonen dat RC kan uitgebreid worden van zijn oorspronkelijke neurale implementaties naar een geheel nieuwe manier om te rekenen met generieke, niet-lineare dynamische media. Niettemin, het blind toepassen van het RC paradigma op nieuwe reservoir types is niet mogelijk omdat de noodzakelijke theoretische onderbouw ontbreekt. Deze thesis geeft een experimentele validatie van de bewering dat RC kan toegepast worden op generieke, niet-neurale netwerken en introduceert een aantal hulpmiddelen die de ontwerper van een RC systeem kan helpen om de optimale reservoir parameters te kiezen.

Klassieke RC en toepassingen

We beginnen met een beschrijving van de methodologie om een ESN netwerk te bouwen, simuleren en trainen. Dit reservoirtype is sterk aanwezig in deze thesis. Daarnaast bespreken we drie methodologische principes die essentieel zijn voor een correcte evaluatie van de prestatie van het systeem, en die helpen om deze prestatie te optimalizeren. Deze klassieke, 'neurale' RC systemen worden dan gebruikt om zowel de brede toepasbaarheid als de uitstekende prestatie van RC aan te tonen, door het toe te passen op een herkenningstaak van gesproken cijfers, een industriële signaalclassificatie taak en een biomedisch detectieprobleem.

Nieuwe implementaties van Reservoir Computing

Hoewel RC stamt uit het onderzoeksgebied rond neurale netwerken, kunnen zijn fundamentele concepten overgedragen worden op andere exciteerbare media en technologieën. Wanneer deze transitie naar nieuwe reservoirimplementaties gemaakt wordt, rijst de vraag hoe tijd moet voorgesteld worden in deze systemen, en hoe de overgang tussen de verschillende tijdsdomeinen (invoer, reservoir en uitvoer) gedaan kan worden. Door de onderlinge verhouding tussen de tijdschalen in deze verschillende domeinen te regelen kan men de geheugeneigenschappen van het reservoir aanpassen aan de taak die bestudeerd wordt. We tonen aan dat het bijstellen van deze tijdschalen cruciaal is voor het optimalizeren van de prestatie, door

de effecten te bestuderen in de context van een gesproken cijferherkenningstaak.

Vervolgens onderbouwen wij de bewering dat RC toepasbaar is op meer generieke dynamische systemen verder, door verschillende nieuwe reservoir implementaties voor te stellen en te onderzoeken. Deze implementaties vertonen gradueel minder en minder overeenkomsten met de oorspronkelijk neurale reservoirs. We bespreken banddoorlaat reservoirs, Cellulaire Niet-lineaire reservoirs en fotonische reservoirs en we tonen aan dat al deze implementaties kunnen gebruikt worden in de context van het RC-raamwerk.

Het quantificeren van reservoir dynamica

Het gebruik van nieuwe reservoirtypes leidt tot de vraag hoe deze systemen moeten afgeregeld worden voor optimale prestatie. De klassieke, stationaire methodes zijn niet meer toepasbaar op nieuwe reservoirs, dus de nood ontstaat voor een nieuwe methode om het dynamisch regime te quantificeren. We introduceren en onderzoeken een maat van reservoir dynamica die gesteund is op concepten uit niet-lineaire dynamische systeemtheorie. Deze maat neemt het eigenlijke werkingspunt in rekening, en we tonen aan dat de maat kan dienen als accurate predictor van prestatie voor verschilllende taken. Het voordeel van deze maat is dat zij toepasbaar is op nieuwe reservoir implementaties en dat zij een nauwkeuriger quantificatie biedt van het dynamische regime.

Regelen van de reservoir dynamica

Nadat we een manier hebben voorgesteld om de dynamica te meten, introduceren en onderzoeken wij een generiek regelmechanisme, Intrinsieke Plastiticiteit, dat actief de dynamica van het reservoir bijregelt. Het mechanisme is gebaseerd op informatie-theoretische principes en regelt de dynamica van het reservoir op een ongesuperviseerde, autonome en biologisch plausibele manier. De voorgestelde adaptatieregel is generisch: hij kan gebruikt worden om individuele adaptatieregels te instantiëren voor een gegeven reservoir type. We tonen aan voor twee verschillende reservoirtypes aan dat dit adaptatiemechanisme automatisch te parameters van de knopen bijregelt in het reservoir en dat het optimale dynamische regime bereikt wordt.

Summary

Reservoir Computing

Many challenging and interesting problems in engineering are unsolvable using heuristic methods or explicitly programmed algorithms. These problems are prime candidates for applying machine learning methods. These methods share the common property that they learn by example and can generalize these examples in an 'intelligent' way to new, unseen inputs. Many machine learning techniques exist, and a large subclass of these is formed by Neural Networks (NN). NN are very abstract connectionist models of the way the brain does computation. They consist of networks of simple, nonlinear computational nodes that communicate values across weighted connections. By training the values of these weights based on examples, the desired behaviour of the network is attained. If the network has a recurrent structure (i.e., feedback loops), then it will have a memory of past inputs, which enables it to do processing of temporal signals rendering them powerful nonlinear computational methods. These recurrent neural networks are however notoriously difficult to train.

A novel learning paradigm called Reservoir Computing (RC) has been recently introduced that enables the use of recurrent neural networks without the lenghty and difficult training stage. The same basic idea was introduced independently as Echo State Networks and Liquid State Machines. In both cases, the architecture consists of a recurrent network of neurons called the reservoir, which is constructed randomly and left untrained, and a separate linear output layer that is trained using simple one-shot methods. The nonlinear mapping and fading memory provided by the reservoir boosts the power of the simple memoryless linear output.

This thesis

Since its introduction in 2002, Reservoir Computing has attracted much attention in the neural networks community due to the combination of its simplicity of use and its very good performance on a variety of difficult benchmark tasks. In this doctoral thesis, we will provide experimental evidence for the claim that RC can be extended beyond its original neural implementations to a novel way of doing computation with generic, nonlinear dynamical media. However, blindly applying the RC paradigm to novel reservoir types is not feasible since the necessary theoretical insights are lacking. This thesis presents an experimental validation of the claim that RC is applicable to generic, non-neural media and introduces some tools that can help a designer of an RC system select the optimal reservoir parameters.

Standard RC and Applications

We describe the methodology of constructing, simulating and training ESN networks, which are featured prominently throughout this thesis. In addition, we discuss three methodological principles that are essential for a correct evaluation of the performance of the system, and that help to optimize the performance. These standard, 'neural' RC systems are then used to demonstrate both the wide applicability and good performance of RC on a spoken digit recognition task, an industrial signal classification task and a biomedical detection problem.

Towards novel implementations of reservoir computing

While RC is rooted in the research field of neural networks, its fundamental concepts can be transposed to other excitable media and technologies. When making this transition to novel reservoir implementations, the issue arises of how time should be represented and how the transition between the different time domains (input, reservoir and output) can be done. By adjusting the relationship between the time scales in these different domains, one can tune the memory properties of the reservoir to the task at hand. We show that tuning these time scales is crucial for optimizing the performance by investigating its effects on a spoken digit recognition task.

Next, we further substantiate the claim that RC is applicable to more general dynamical systems by presenting and investigating several reservoir implementations that show progressively less similarities to the original neural reservoirs. We discuss bandpass reservoirs, Cellular Nonlinear

reservoirs and photonic reservoirs and show that all these implementations can be used in the RC framework.

Quantifying reservoir dynamics

The use of novel reservoir types leads to the question of how to tune these systems for optimal performance. Traditional, stationary measures no longer apply to novel reservoirs, so the need arises for a method to quantify the dynamical regime. We introduce and investigate a measure of the reservoir dynamics that is based on nonlinear dynamical systems theory. This measure takes the actual operating point into account, and we show that this can be used as an accurate predictor of performance for several tasks. The advantage of this measure is that it is applicable to new reservoir implementations and offers a more accurate quantification of the dynamical regime.

Tuning reservoir dynamics

After developing a way to quantify the dynamics, we introduce and investigate a generalized adaptation mechanism called Intrinsic Plasticity that actively tunes the dynamics of the reservoir. The mechanism is based on information theoretic principles, and tunes the dynamics of the reservoir in an unsupervised, autonomous and biologically plausible way. The presented adaptation rule is generic: from it, one can derive individual adaptation rules for a given reservoir type. We show that this adaptation mechanism automatically adjusts the parameters of the nodes in the reservoir and that it reaches the optimal dynamical regime by applying it to two different reservoir types.

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
APRL	Atiya-Parlos Recurrent Learning
AUC	Area Under Curve
BPDC	Backpropagation Decorrelation
BPTT	Backpropagation Through Time
CA	Cellular Automaton
CNN	Cellular Nonlinear/Neural Network
ESN	Echo State Network
ESP	Echo State Property
FFT	Fast Fourier Transform
IP	Intrinsic Plasticity
KS	Kolmogorov-Sinai
LIF	Leaky Integrate and Fire
LLE	Local Lyapunov Exponent
LMS	Least Mean Squares
LSM	Liquid State Machine
LSV	Largest Singular Value
MFCC	Mel-Frequency Cepstral Coefficients
ML	Machine learning
NN	Neural Network
NMSE	Normalized Mean Square Error

NRMSE	Normalized Root Mean Square Error
RC	Reservoir Computing
RCT	Reservoir Computing Toolbox
ROC	Receiver Operating Characteristic
RLS	Recursive Least Squares
RTRL	RealTime Recurrent Learning
SISO	Single Input Single Output
SOA	Semiconductor Optical Amplifier
SSV	Structured Singular Value
SV	Singular Value
TLG	Threshold Logic Gate
TM	Turing Machine
WER	Word Error Rate

Mathematic notations

$\langle \mathbf{x}, \mathbf{z} \rangle$	inner product
$I(X;Y)$	mutual information of stochastic variables X and Y
η_{IP}	learning rate for the IP learning rule
$\hat{h}_k$	kth local lyapunov exponent (LLE)
$\hat{h}_{max}$	maximal local lyapunov exponent (LLE)
$H(X)$	entropy of stochastic variable X
J_f	Jacobian of a map f
μ_{SSV}	Structured singular value
M	number of input nodes
N	number of reservoir nodes
P	number of output nodes
D	number of samples in the dataset
D_{KL}	Kullback-Leibler divergence
$\rho(W)$	spectral radius of matrix W
ρ_{eff}	effective spectral radius
$s(t)$	signal in continuous time, indicated with t
$s[k]$	signal in discrete time, indexed with k
T_d	number of timesteps in the dth sample
$\mathbf{u}[k]$	input vector at timestep k
W_{res}	reservoir weight matrix
W_{in}	input weight matrix

W_{out}	output weight matrix
W_{fb}	feedback weight matrix
$\mathbf{x}[k]$	vector of reservoir states at time k

Contents

1 Introduction

1.1 Automating information processing

Mankind has since long looked for ways to automate certain computational tasks, from the Jaquard loom at the beginning of 19th century to the current proliferation of computers and electronic devices that have become an indispensible part of our everyday lives. Both scientific and economic progress has shifted over the past century from *industrialisation* (the automation of industrial processes) to *informatisation* (the automation of information processing). This shift is captured in the term 'information age' which indicates a transition to a society in which information is the main commodity – in a similarly fundamental way as the industrial revolution changed the society. This transition is still ongoing, and its momentum is increasing.

In the same way that industrial processes try to automate the shaping and combination of the basic resources provided by nature, information processing transforms and combines information – information that can be available in the external world (environment) or stored digitally somewhere. The processes that perform the actual information processing are almost always based on some form of computation. As both the amount of information and its importance in society are increasing, the need for methods to automatically process this information also grows. This increasing interest in computation has motivated scientists from a wide variety of fields (such as logic, mathematics, physics and others) to start thinking about the nature of computation and how humans can construct systems that automate this process. Fueled by the rapidly accelerating transition to an information society and the accompanying technological advances in hardware, these individual research lines have converged in

the past few decades in a whole new research field called computer science. Taking into account the fact that computer science has computation and information processing in its core, the term *science of computation* was perhaps more fitting. Edsger Dijkstra, one of the founders of computer science, summarized this in the well-known quotation: "Computer Science is no more about computers than astronomy is about telescopes."

There are quite a few theories of computation available. Most theories of computation have focused on algorithmic computation, i.e., which requirements of the computation device (the *computer* in a generic sense) are needed to execute certain algorithms, and what can be said about the properties of those algorithms. Arguably the most important theory of computation that has been described in this context has been the theoretical framework described by Alan Turing (Turing, 1936). Turing described an abstract device (the *Turing machine*) which is a conceptually simple but powerful computation device with an infinite storage capacity. The Turing machine has had a fundamental impact on the theory of computation: it is a system that can compute any algorithm that eventually terminates. So, instead of reasoning about algorithms, the theorists now have an actual device (albeit an abstract one) that executes these algorithms which they can reason about.

Turing machines are devices that describe *how* algorithms can be executed. They consist of a processing unit (the *head*) that contains a program which defines its behaviour and that operates on a separate storage device for symbols (the *tape*). This basic architecture – separating storage and processing units – can be made more general by storing the program in the same memory as the data (which a.o. allows the device to modify its own program). This architecture is called the universal Turing machine, because it can simulate any given Turing machine. This architecture was later adopted by Von Neumann as the basis for some of the first discrete-symbol computers (as opposed to analog computers), and as such it forms the basis for every modern personal computer. While many variations have been invented and constructed, this fundamental architecture is until now by far the most prevalent in any information processing device.

One of the implications of the Turing architecture is the fact that the behaviour of the processing unit – what its output should be, given a certain input – is explicitly programmed by a human. Moreover, its fundamental mode of operation is timestep-based, and it reads and writes discrete symbols instead of continuous values. Almost all modern computers also have these three (programmed, timestep-based and digital) properties. The popularity of this approach is due to various technological and historical reasons, but it is by no means the only way to do

computation. Indeed, an important part of the message of this doctoral thesis is the fact that computation is also possible using non-discrete (both in time and state) systems that are not explicitly programmed.

The programmable computer is a very flexible and powerful device, which is part of the reason why it has pervaded our daily lives so quickly and profoundly and why today much of our economy relies on a vast information processing infrastructure. However, the complexity of the data and the increasingly challenging problems that arise in information processing are pushing the current computation paradigm to its limits. Novel ways of thinking about computation are needed to overcome these limitations.

Standard computer programs are very good at processing large amounts of predictable data in a very quick and deterministic way. While this is useful in many cases, there is a broad class of problems or tasks that humans solve or execute on a daily basis – sometimes apparently without conscious thinking – but which remain unsolved up to a certain point by standard programmed algorithms. Tasks such as listening, language processing, reasoning or moving around in new environments are mastered by most people yet no machine or device can do this with nearly the same level of accuracy or flexibility. It is the type of task that, loosely speaking, requires intelligence.

Intelligence (from the Latin *intellegere* – to understand) is difficult to define, but regardless of the definition it is quite clear that computers are not intelligent – sometimes to the frustration of their users. For instance, when a user executes a certain task a couple of times, the computer will not recognize this pattern and learn to do this task itself. Also, when a computer is presented with input it was not explicitly programmed to handle, it cannot think of a reasonable response by itself. Finally, because the amount of available information (think of the internet) and its complexity (think of robots navigating in unknown environments) is increasing, traditional algorithms can no longer provide the computational power needed to process this information.

The search for systems that do have these properties is what powers research in artificial intelligence (AI) and machine learning (ML). These closely linked research fields are quite young (even more recent than computer science itself), but have seen an explosive growth in the last decade(s). This is due to two factors: first of all, many of the methods that are developed and studied require large amounts of computing time and the technological advance in hardware has only very recently reached the point where machines are fast and powerful enough to simulate these methods in a usable manner. Secondly – and equally importantly – the traditional, algorithmic way to build computation devices and program-

ming them is reaching its limitations and novel paradigms for creating powerful information processing systems are needed.

1.2 Artificial Intelligence / Machine Learning

It is difficult to define which systems can be considered intelligent – this is probably more a philosophical than a technical question. The famous but controversial Turing test (Turing, 1950) tries to provide a test for artificial intelligence by letting a human judge the intelligence of the system through conversation from an isolated location. So far no machine has passed the test, and is has been argued that this test is not able to detect all aspects of intelligence.

One of the properties of intelligent systems is the fact that they are robustly able to give a meaningful response when they are faced with inputs they have not seen before. They are able to do this because they 'understand' how their (sometimes abstract) environment is structured, and how their desired output is fundamentally related to their inputs. While this mapping between inputs and desired outputs could be programmed explicitly if it was known, this is not the case for many interesting real world problems.

The focus of this doctoral thesis is on systems or methods whose behaviour is controlled and adjusted in a very specific way, namely by *learning*. Traditional rule- or heuristics-based AI has its merits and is used in a wide range of applications, but usually needs quite extensive intervention or design by a human expert. The techniques that fall under the name 'machine learning', on the other hand, usually require principally much less human intervention. These methods share the common property that they are not programmed or designed in the conventional sense, but that they learn by generalizing from training examples – hence the name.

There exist many techniques that can be placed in the general category of machine learning. After an on-and-off period from the sixties to the eighties in the previous century, fundamental work on statistical learning theory (Vapnik, 1995) and neural networks (Rumelhart et al., 1986) has made the field grow explosively into the well established and respected research area it is now. Partly driven by the exponential increase in raw computing power but also by theoretical progress in understanding the nature of learning, many insights into machine learning have been gained over the past decades – both abstractly and from an implementation point of view. It is beyond the scope of this thesis to present an overview of the

field of machine learning so I will limit myself to a brief discussion of the techniques that are relevant to this work, either because a direct extension of them is presented in this thesis or because they share interesting characteristics with the methods used in this thesis.

Many different ways to classify the many ML methods exist. One very high-level way to do this is according to the principal way the systems learn. One can discern three main classes:

- **Unsupervised learning methods** These methods learn only from the examples presented to them, without any clues as to what behaviour is required from the system. The main goal of these types of algorithms is to discover regularities or properties in the data without having them explicitly pointed out by the user. Examples are Self Organizing Maps (SOM) (Kohonen, 2001) or K-means clustering (Hartigan, 1975), and applications include data clustering and information retrieval.

- **Reinforcement learning methods** In this case, the system receives some clue about the desired behaviour, but the nature of the information is limited. When presented with an example input, the response of the system is evaluated and scored (rather good response vs. rather bad response), but in neither of the cases the actual correct behaviour is given. In other words, the system learns through a system of rewards and/or penalties. These algorithms are popular in the robotics community, since it is usually easier to define a reward signal (how well the robot is doing) than an explicit training signal (what the robot should do) in this case. Aside from robotics, this technique is also popular for learning complex games such as Go (Schraudolph et al., 1994), because here too it is difficult to define the desired response of the system for every situation.

- **Supervised learning methods** This type of learning mechanism will be the main focus of this thesis. In this case, for every input example, the desired output (a discrete class label or continuous system output) is known and is used for training the learning method. In many real world situations this information is not available, but when it is, it can greatly accelerate the learning process and lead to superior accuracy.

Research in ML has focused on creating methods for solving complex tasks, and it has looked for inspiration on how to construct these methods in quite different areas. For instance, the field of statistics has been a rich source of novel ideas for many mathematically inspired methods such as kernel machines (Cristianini and Shawe-Taylor, 2000). On the other hand,

neuroscience has inspired the use of more biologically plausible models which are loosely based on the operation of the brain. This has led to a quite extensive research area, and the models used here are generally called Artificial Neural Networks (ANN).

1.3 Artificial Neural networks

Artificial neural networks (ANN) are models of the brain structure. Depending on the research goal or application there is a wide variety of models in literature. The adjective 'artificial' is used to discern between the abstract mathematical model, and biological ('wet') neural networks, but it is usually clear from the context wether the real biological systems or their models are concerned so the term is abbreviated to NN.

NNs consist of input/output processing nodes (the *neurons*) that are connected into a network using weighted connections. In the case of analog neurons, every neuron receives weighted values from the incoming nodes to which it is connected, sums these values, computes its own output by applying some (usually nonlinear) function and transmits these output values via the outgoing weighted connections to the other nodes. One can describe these systems in both the continuous time and discrete (i.e., sampled) time domain, but there is no strict one-to-one mapping between systems described in discrete time and systems described in continuous time. For instance, one can already have chaos in one-dimensional discrete time systems, whereas in continuous time at least three dimensions are needed for chaotic behaviour. The transition between both domains is a research field in itself (very related to communication and sampling theory). Section 3.1 of this doctoral thesis addresses this issue in the context of Reservoir Computing. For this introduction, however, we will focus on the discrete time domain.

Formally, the input-output behaviour of a standard artificial neuron with index j is described by:

$$y_j = f\left(\sum_{i \in S_j} w_{ij} y_i\right),$$

where w_{ij} is the weight of the connection between neuron i and neuron j, y_i is the output or *activation level* of the ith neuron, S_j denotes the set of indices of neurons with connections leading to neuron j and f is a (usually nonlinear) transfer or *activation* function. A pictorial representation of a neuron inside a network is shown in Fig. 1.1.

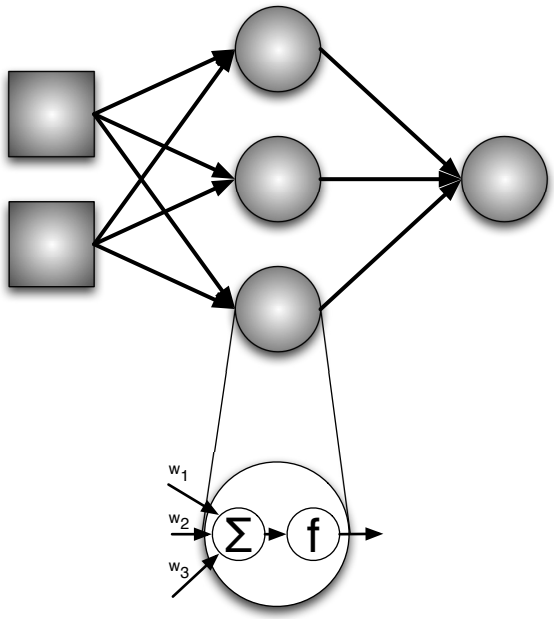


Figure 1.1: A simple network of neurons, with a close-up of the internal operation of a neuron: the weighted sum of the inputs is fed through a nonlinearity f.

Table 1.1: Overview of the main properties for classifying neural network models.

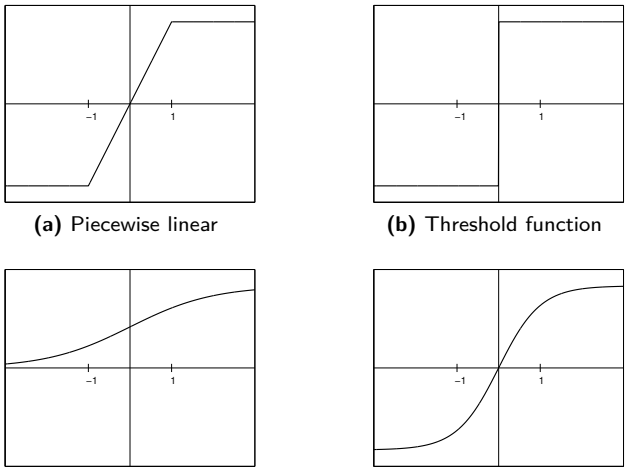
Time domain	continuous	discrete
Neuron communication	spiking	analog
Activation function	(piecewise) linear	nonlinear
Node memory	no memory	internal memory
Network topology	feedforward	recurrent

Usually the weight structure of the network is represented conveniently as a weight matrix W, where the element $W[i, j] = w_{ij}$. The behaviour of a standard ANN is completely determined by the interconnection topology and the transfer function f. It is beyond the scope of this thesis to present a complete taxonomy of neural networks, but some of the main properties (most of which are mutually orthogonal) of different NN models are shown in table 1.1.

Research on neural networks spans a whole spectrum, ranging from neuroscientists who try to model the operation of single biological neurons, to theoretical statisticians that use neural networks for datamining applications. The goals of these research lines span an equally broad spectrum: at the one end, NN models are constructed and simulated to enhance the understanding of the brain, and at the other end NNs are used to solve complex engineering problems or develop novel techniques for learning machines. The research presented in this thesis is situated on the engineering side of the spectrum.

1.3.1 Applications of neural networks

Neural networks are applied in various areas, perhaps more widely than most people realise (Jain and Vemuri, 1999). Some application fields include optical character recognition (neural networks are part of the state-of-the-art in this field) (LeCun et al., 1989; Simard et al., 2003), the autonomous flying of aircrafts in case of dramatic failures (such as loss of a wing) (Anon, 1999), a multitude of applications in the field of medical diagnosis and analysis (Ster et al., 1996), and fault detection or quality assessment (QA) for industrial processes (Bishop, 1995). A full description of the total range of tasks for which neural networks can be or are being used is beyond the scope of this work.



(a) Piecewise linear **(b)** Threshold function

(c) Fermi function **(d)** Tanh function

Figure 1.2: Some common activation functions used in neural networks.

1.3.2 Activation functions

The transfer function f of the neuron which is applied to the weighted sum of its inputs also determines the behaviour of the network. The most common transfer function is a sigmoid-type function – the name is due to its similarity to the letter S. Two common sigmoid functions are the tanh (Figure 1.2 (d)) and the logistic or fermi function (Figure 1.2 (c)). The fermi activation function is given by

$$\text{fermi}(x) = \frac{1}{1 + \exp(-x)},$$

and is related to the tanh activation function through :

$$\tanh(x) = 2\,\text{fermi}\,(2x) - 1.$$

Other common transfer functions include the piecewise linear function (Figure 1.2 (a)), the threshold function (Figure 1.2 (b)) and the identity function (in this case the node is sometimes referred to as a linear neuron). Because the transfer function is nonlinear in most cases, it is also commonly referred to as the nonlinearity of the neuron.

Most transfer functions share some common properties:

- They have a so-called squashing effect, referring to the boundedness of the output range of the neuron. This means that regardless of

the input values, the network's internal activation values will always remain limited.

- They are usually differentiable (but sometimes not continuously differentiable, e.g., in the case of the threshold function). This need for differentiability comes from the fact that many learning rules for neural networks try to compute the gradient of the error w.r.t the weights of the incoming connections to the neuron. Using this gradient, the weights can be adjusted using a technique called *stochastic gradient descent* (this method will be used in Chapter 4 of this work).

It should be obvious that the shape of the activation function has a large impact on the behaviour of the neurons and the complete network. Depending on the precise requirements of the task and the implementation, a trade-off can be made between the computational expressiveness and the complexity (and accompanying computational requirements) of the activation function. In Chapter 2, a study of this trade-off is presented in the context of Reservoir Computing.

1.3.3 Spiking neural networks

A biologically more realistic but more complex family of neuron models are the so-called *spiking neurons* (Maass and Bishop, 2001; Schrauwen, 2008), which can be used to create a *spiking neural network* (SNN) with weighted connections, similarly to the analog neural networks discussed above. These models are a more accurate representation of the way 'wet' neurons behave, but are significantly different from the analog neurons described in the section above. First, they communicate through isolated spikes instead of continuous values. These spikes are identical, so the information they convey is entirely contained in the precise timing or firing rate of the spikes. Secondly, due to the way these neurons are modelled, they posess some internal memory which means they can process temporal signals. Finally, it was shown theoretically that they are capable of performing more complex operations than analog neurons (Maass, 1997).

The actual behaviour of the neurons depends strongly on the model in use (there are a wide variety of models described in literature (Gerstner and Kistler, 2002; Izhikevich, 2007)), but we will limit this discussion to the model considered in this work, namely the leaky integrate and fire (LIF) neuron (Adrian, 1928). This elementary model describes the operation of a spiking neuron as a leaky capacitor which is charged by incoming spikes. The charge stored in the capacitor leaks away exponentially if no spikes are received at a rate determined by the so-called *membrane time*

constant. If enough spikes are received in a certain time window, the potential measured over the capacitor reaches a threshold. This triggers an outgoing spike from the neuron and the membrane potential in the capacitor is reset to a reset potential. Another behaviour that is observed in biological neurons and that is often incorporated in spiking models is a so-called refractory period. This is a brief (in the order of milliseconds) period after a neuron has fired during which it is not sensitive to incoming pulses. A side-effect of this refractory period is that the maximal firing rate of the neuron is bounded.

In biological neural networks, the neurons are connected through *synapses* – small gaps between the axons (outgoing fibers) and dendrites (incoming fibers) of the communicating neurons. The spikes are transmitted across these gaps through *neurotransmitters*, molecules that travel across the so-called synaptic cleft. The bridging of this synaptic gap does not happen instantaneously but is stretched over time, which means that the current that enters the incoming neuron is not a pure spike. This behaviour is modeled through synapse models, of which again many types exist. In this work only exponential synapse models are considered.

The use of spiking neural networks for engineering applications was researched extensively in (Schrauwen, 2008). One of the considerations that needs to be made when solving a problem with SNN is the transition between the analog and the spiking domain. Since information in the real world is of an analog (non-discrete) nature and spiking neurons communicate with spikes (isolated events in time), some attention should be payed to the way the analog inputs are transformed to so-called *spike trains*. There are a number of ways to encode analog information in spike trains and every method has certain advantages and disadvantages. I refer to (Schrauwen, 2008) for more information on this topic.

1.3.4 Network topologies

Another important property of NNs is the topology of the networks. The topology is fully determined by the weight matrix W, which determines both the connectivity, i.e., which neuron is connected to which, and the weights of those connections – a zero value simply indicating the absence of a connection. In the field of neural networks, many network topologies have been defined. There is one property of the topology that has a big impact on the behaviour and training of the network, namely the presence or absence of recurrent connections.

FEEDFORWARD NETWORKS In the most common case, there are no recurrent connections and the network has a so-called *feedforward* structure.

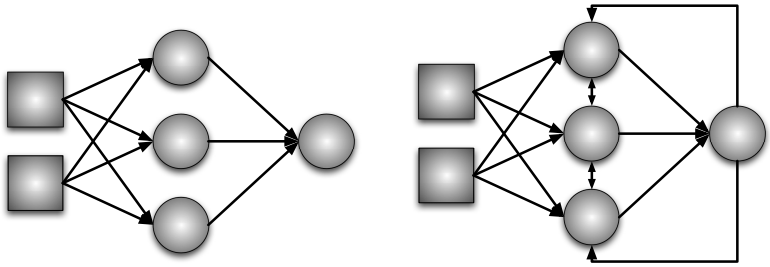
The nodes of the network are divided into layers, with information flowing only to consecutive layers and not backwards. The inputs to the network are fed in through the input layer, the outputs are read out at the output layer and the intermediate layers are called *hidden* layers because their activation is usually not directly observed. A schematic view of this network topology is shown in Figure 1.3a. These networks are usually called Multi-Layer Perceptrons (MLP).

The values of the internal and output neurons of these networks are fully determined by the values of the inputs. In an electronic analogy, these networks form a combinational circuit. There exist many learning rules for these feedforward networks, the most famous and widely used being doubtlessly the error-backpropagation rule (Rumelhart et al., 1986), while more powerful and sophisticated extensions include second-order versions such as the Levenberg-Marquardt method (Hagan and Menhaj, 1994) and fast but well performing extensions such as resilient propagation (RPROP) (Riedmiller and Braun, 1993).

Due to the layered architecture and lack of memory of the feed-forward networks, these types of NN are not capable of processing temporal information – i.e., in the cases where there is also information contained in the order in which the inputs are presented to the network such as for speech or robotics tasks. One way to compensate for this shortcoming is to implement a tapped delay line into which the samples of the inputs are fed chronologically, and use all taps as inputs to the network. This architecture is called a Time Delay Neural Network (TDNN) (first introduced in (Waibel et al., 1989)). The analogon to these networks in filter theory is the finite impulse response (FIR) filter.

The architecture is motivated by the famous Takens theorem (Takens et al., 1981), which states that the (hidden) state of a dynamical system can be reconstructed using an adequate delayed embedding of the observable variables. This explicit embedding effectively converts the temporal problem into a spatial one. While this topology enables the use of feedforward networks for temporal tasks, the disadvantages of this approach are the artificially introduced time horizon, the need for many parameters (i.e., weights) when a long delay is introduced, the artificial way in which time is represented in the spatial domain and the fact that there is no obvious biological analogon.

Recurrent networks The other (less common) topology type is the so-called recurrent network. This network topology will be the main focus of this thesis. In this case, there do exist connections projecting backwards through layers. Every connection is also characterized by a delay, so that the presence of these connections introduces a form of memory into the



(a) Feedforward network **(b)** Recurrent network

Figure 1.3: Different network topologies

network due to the fact that information does not flow in one direction through the network but remains circulating inside and is integrated with information of previous timesteps. Much in the same way Time Delay Neural Networks are analogous to FIR filters, there exists some analogy between recurrent networks and infinite impulse response (IIR) filters, which also feature internal feedback loops.

The fact that memory exists within the network is due to the fact that the network is a *dynamic system*. Indeed: the activation values (the states) of the neurons are not only determined by the current input but also by the previous state of the network (and thus, recursively, by all previous inputs). This property makes these networks ideally suited to solve inherently temporal problems from fields such as speech recognition, machine control or dynamic system identification. A schematic view of this topology is shown in Figure 1.3b. In this case, sometimes the layered structure is abandoned for an (equivalent) architecture with a single hidden layer with internal recurrent connections[1].

In principle, RNNs are very powerful tools for solving complex temporal machine learning tasks. They have the advantages of feedforward networks, which include robustness to noise, learning by example and the ability to model highly nonlinear systems, and add to that an inherent temporal processing capability. Possible – and actual – applications are manifold and include the learning of context free and context sensitive languages (Rodriguez, 2001; Gers and Schmidhuber, 2001), control and modelling of complex dynamical systems (Suykens et al., 1996) and speech recognition (Robinson, 1994; Graves et al., 2004). RNNs have been shown to be Turing equivalent (Kilian and Siegelmann, 1996) for common

[1] In terms of the weight matrix W, both topology types can be easily discerned. If the neurons indices i are chosen so that neurons belonging to a higher layer have a higher index, then the weight matrix for a feedforward topology will be upper-triangular. In the case of a recurrent network, this is not the case.

activation functions and can approximate arbitary finite state automata (Omlin and Giles, 1994).

Nonetheless, several factors still hinder the large scale deployment of RNNs in practical applications. So far, not many learning rules exist (Haykin, 1999; Jaeger, 2002; Suykens and Vandewalle, 1998) and most suffer from slow convergence rates (Hammer and Steil, 2002). This is partly due to the problem of vanishing gradients: first-order gradient descent methods such as BackPropagation Through Time (BPTT) (Werbos, 1974, 1990) (which was later rediscovered in (Rumelhart et al., 1986)) or RealTime Recurrent Learning (RTRL) (Williams and Zipser, 1989) use the gradient of the error to update the network parameters, and these gradients become very small after even a few timesteps back into the past – a problem that also exists for deep (i.e., many-layered) feed-forward networks. Another problem is the existence of bifurcations – sudden changes in the qualitative behaviour of the system due to only small changes in the parameters. These behavioural changes make the training more difficult and time-consuming.

Extensions using second-order curvature information of the error surface using Kalman filtering have been proposed and currently form the state-of-the-art of the field (Puskorius and Feldkamp, 1994; Prokhorov, 2007), but their use is very involved and is often reserved for experts in the field. One possible solution to this is a specially constructed Long Short Term Memory (LSTM) architecture (Schmidhuber and Hochreiter, 1997), which nonetheless does not always outperform time delayed neural networks.

Part of the difficulty when training an RNN arises from its dynamical nature, which means that the training process has to do two highly complex tasks simultaneously: change the network parameters so that the RNN operates in the correct dynamical regime, and enforce the desired input/output behaviour. This task is comparable to chasing a moving target. Since the RNN is a highly nonlinear system, it is clear that this task is not trivial to say the least. Reservoir Computing offers a solution to this problem.

1.4 The origins of Reservoir Computing

1.4.1 A brief history

Many good ideas have been inventend several times independently, such as the telephone or the fundamentals of calculus. One could argue that this is not a coincidence: if the settings are right for an interesting idea to

be born, it should not be suprising that more than one person discovers it. In 2001 and 2002, two seminal publications marked the birth of the research field of Reservoir Computing. The technical report "The 'echo state' approach to analysing and training recurrent neural networks" by Jaeger (Jaeger, 2001a) and the Neural Computation letter "Real-time computing without stable states: A new framework for neural computation based on perturbations" by Maass et al. (Maass et al., 2002b) both introduced a novel way of training and using complex networks of neural nodes. Later, in 2004, another publication presented similar ideas, which were however derived from an entirely different background. The BackPropagation DeCorrelation (BPDC) learning rule for recurrent neural networks that was introduced there shares some fundamental similarities with the previous two ideas. These contributions sparked considerable interest in the community, and the similarities between both approaches were immediately noticed (also by the authors themselves).

The ease of use and excellent performance of these methods were quickly picked up by others, and the research started to gain momentum. Additionally, several individual research groups – including the UGent Reservoir Lab – began collaborating and streamlining the research for this quickly growing research community. This has fueled the growth of the field and has caused an almost exponential increase in the number of Reservoir Computing related publications. For an overview of the research field we refer to (Lukosevicius and Jaeger, 2007; Schrauwen et al., 2007b; Lukosevicius and Jaeger, 2009). We will now briefly discuss these original RC incarnations (and others) in more detail.

1.4.2 Echo State Networks

The Echo State Network (ESN) concept describes an engineering approach to training and using recurrent neural networks. The fundamental issues associated with training RNNs (as discussed previously) were conveniently side-stepped by not training the network at all. The basic recipe for constructing an ESN is both simple and elegant: a recurrent neural network with a random topology and random weights is constructed. The weight matrix is globally scaled to get the dynamics of the network in a desirable regime. The network is driven by the external input, and the response of the network is then used to train a simple linear regression or classification function. Because the only training that is done is linear, this can be done using very simple one-shot methods. We will elaborate on the methods for constructing, training and using ESNs in chapter 2.

The ESN method is appealing not only because of its simplicity, which allows to reason about these systems in an intuitive manner. Much of its

appeal is caused by the good to excellent performance results on a variety of difficult benchmark tasks.

While ESNs are simple yet powerful information processing systems, not everything is known about their functionality. ESNs are determined by a few global parameters that determine, e.g., the global scaling of the weights from the input to the recurrent network and of the internal weights of the network. However, even for ESNs with identical global parameters there is still considerable variation on the performance on the same task. This is due to the randomness with which these networks are constructed, and the fact that there are still some unknown factors that determine the performance of a specific ESN besides these global scaling parameters.

The name Echo State Networks is based on the fact that, in order to be useful, these networks should have the so-called Echo State Property (ESP). This property states – informally – that the network should asymptotically forget its initial state when it is driven by an external signal. Due to the recurrent connections in the network, information about past inputs is stored in the network. According to the ESP, the network should forget this information eventually, so that the network has in effect a *fading memory*. Thus, the network contains a rich set of nonlinear transformations and mixings of the input signals of the current and past timesteps (these are called the *echos*).

The ESP is not the only desirable property one requires from a good ESN. The set of echos contained in the reservoir should also be dynamically rich enough to boost the computational power of the linear readout. On the other hand, if the network is too excitable, it moves to a different dynamical regime where it no longer asymptotically forgets its inital states. Thus, an optimal dynamical regime needs to be found for the problem at hand, where the dynamical transformation of the inputs provided by the network is rich enough, and where the fading memory property is still preserved. Unfortunately, this optimization at this point can only be done through manual tweaking of the parameters or brute-force searching of the optimal parameter values.

1.4.3 Liquid State Machines

The Liquid State Machine (LSM) is conceptually very similar to the ESN, but it originated from a rather different background. It was introduced by a research lab active in robotics and neuroscience.

In an abstract or theoretical sense, an LSM consists of two parts: a high-dimensional 'filter' or mapping that maps the current and past inputs $u[k]$ onto a state vector $x[k]$, and a (usually memoryless) readout

function that maps the state vector onto an output vector $y[k]$. In order to be computationally useful, the filter should have the so-called point-wise separation property, which states that different input signals should be mapped to different state vectors. Moreover, the readout function should have the universal approximation property. This property states that the readout function should be able to approximate any function on a closed and bounded domain with arbitrary precision. It was shown in (Maass et al., 2002b) that an LSM which satisfies these two properties, can be trained to approximate any stationary mapping on time-varying inputs with fading memory.

While the LSM is described as an abstract computational framework with a mathematical foundation, in practice it usually consists of a recurrent network of spiking neurons and a separate linear readout layer – very similar to the ESN. While the LSM can be used to solve engineering tasks, this was not its original intention. The LSM was invented by researchers whose main interest is neuroscience and cognitive modeling. In other words, they want to know how the brain works.

When neuroscientists model certain parts of the brain they aim to evaluate how accurately the model represents the real biological systems. It is however quite difficult to estimate how these models function from a cognitive point of view. The LSM offers a simple solution for this: by reading out the information contained in the network and feeding this to a linear readout function, the information processing capabilities of the model can be evaluated very easily. Because of this, most early descriptions of LSMs use a network of biologically plausible spiking neurons, which is excited by external inputs. Partly because of the superior computational power of spiking neurons these LSMs have also been used for engineering (Verstraeten et al., 2005) and robotics applications (Joshi and Maass, 2005).

1.4.4 BackPropagation DeCorrelation

The underlying notion of constructing a random recurrent neural network and training only the output layer was also formulated based on an entirely different reasoning. In (Steil, 2004), a learning rule is introduced for recurrent neural networks which is derived from the Atiya-Parlos (Atiya and Parlos, 2000) (APRL) learning rule. When studying the weight dynamics of the APRL learning rule, it was noticed that only the weights of the connections to the output nodes were substantially changed, and the 'internal' weights were only scaled up and down in a global fashion (Schiller and Steil, 2005). Based on this observation, a simplified version of the APRL rule was derived which is only applied to the output layer,

while the internal weights are chosen with a suitable initial global scaling. This rule is called Backpropagation-Decorrelation (BPDC), and expresses the same fundamental idea of training only the output layer based on a suitably chosen dynamic reservoir. The main differences between BPDC networks and Echo State Networks are the learning rule (online, epoch-based BPDC and off-line one-shot linear regression, respectively), the node type (fermi nonlinearities vs. tanh nonlinearities) and the network topology (in the case of BPDC networks, feedback inside the output layer is sometimes used).

1.4.5 Early descriptions of neural RC systems

Even earlier, in the nineties, some architectures were described that fall under the RC framework, but at that time the contributions somehow failed to attain the necessary critical mass to stimulate further research. In (Dominey, 1995) a random but fixed recurrent network structure of biologically plausible neurons is described, that is connected to a linear layer which is trained with a very simple update rule. Similary, (Buonomano and Merzenich, 1995) describes a randomly connected network of so-called integrate and fire neurons, which are fed with an external stimulus. This network is constructed and left unchanged afterwards. A readout layer is trained using a very simple adaptation rule – this system shares considerable similarities to the Liquid State Machine. The use of this system is demonstrated on a phoneme recognition task. Be it the circumstances of the times, the presentation of the ideas or simple coincidence, both publications did not, at the time, spark the interest of the community enough to build further upon these concepts.

1.4.6 Applications of Reservoir Computing

Several successful applications of reservoir computing to both synthetic data and real world engineering applications have been reported in the literature. The former include dynamic pattern classification (Jaeger, 2001b), autonomous sine generation (Jaeger, 2001a), grammar modelling (Tong et al., 2007) or the computation of highly nonlinear functions on the instantaneous rates of spike trains (Maass et al., 2004a). In robotics, RC systems have been used to control a simulated robot arm (Joshi and Maass, 2004), to model an existing robot controller (Burgsteiner, 2005b), to perform object tracking and motion prediction (Burgsteiner, 2005a; Maass et al., 2002a), event detection (Jaeger, 2005; Hertzberg et al., 2002) or several applications in the Robocup competitions (mostly motor control) (Oubbati et al., 2005; Plöger et al., 2004; Salmen and Plöger, 2005).

At our own lab, several applications in the field of autonomous robotics have been studied, including robot localization and event detection (Antonelo et al., 2008a), behaviour switching (Antonelo et al., 2008b) and autonomous place cell discovery (Antonelo et al., 2009). RC systems have been used in the context of reinforcement learning (Bush and Anderson, 2005).

Also, applications in the field of Digital Signal Processing (DSP) have been quite successful, such as speech recognition (Maass et al., 2003; Verstraeten et al., 2005; Skowronski and Harris, 2006; Ghani et al., 2008; Jaeger et al., 2007) or noise modeling (Jaeger and Haas, 2004). In (Rao et al., 2005), an application in Brain-Machine interfacing is presented and (Venayagamoorthy, 2007) discusses an RC-based monitor for a power system.

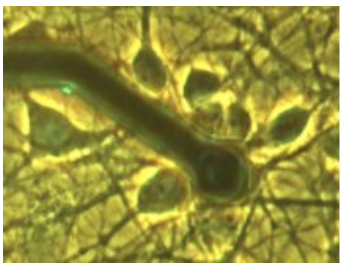
And finally, the use of reservoirs for chaotic time series generation and prediction have been reported in (Jaeger, 2001b, 2003; Steil, 2006, 2005a; wyffels et al., 2008b; Crone et al., 2008). In many areas such as chaotic time series prediction and isolated digit recognition, RC techniques already outperform state-of-the-art approaches. A striking case is demonstrated in (Jaeger and Haas, 2004) where it is possible to predict the Mackey-Glass chaotic time series with several orders of magnitude better accuracy than with classical techniques.

1.4.7 Towards generic Reservoir Computing

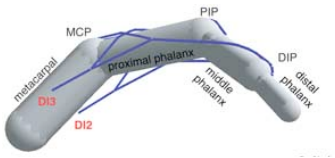
The systems and methods described above are all well grounded in the fields of neural networks and neuroscience. The fact that all these concepts were introduced into the same research field has enabled the similarities between them to be discovered, which has definitely increased the acceptance rate of the ideas. However, we argue that the true power of these concepts lies precisely in extending them to other domains. Indeed: the central idea that this doctoral thesis is centered around is the fact that the mechanisms of Reservoir Computing are not bounded to neural implementations (although they have proven themselves very useful there already). Transposing these ideas onto other, non-neural systems opens up a vast potential for using nonlinear dynamical media for computation.

As will be explained in more detail in later sections, the role of the reservoir can be seen as a complex nonlinear multidimensional filter that projects the input signals into a high-dimensional space, where the classification can be done much more accurately. Complex nonlinear filters are used in many research fields, and RC can provide an elegant and powerful mechanism for using these systems for computation. Examples (see Fig. 1.4) of such nonlinear media include actual *in vivo* neural networks grown

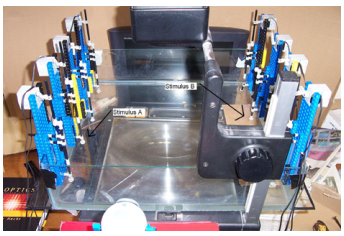
(a) The gene regulatory network in the bacterium E. Coli.



(b) Cultured neurons grown on an array of micro-electrodes.



(c) The tendons of a finger do computations (Figure from (Valero-Cuevas et al., 2007)).



(d) An actual reservoir of water (Figure from (Fernando and Sojakka, 2003)).

Figure 1.4: Several nonlinear dynamical systems that do computation, and have been or could be used for Reservoir Computing.

on chips (Dockendorf et al., 2009), chaotic attractors (Goh and Crook, 2007), biological networks such as the gene regulatory network of a bacterium (Jones et al., 2007) or the actual physical properties of a tendon in a finger in response to external forces (Valero-Cuevas et al., 2007). The idea of a reservoir has even been taken literally: a bucket of water disturbed by speech signals has already been used to do speech recognition (Fernando and Sojakka, 2003). Taking this idea to its extremes, there are even scientists who have the view that the whole universe is one big computation device (Lloyd, 2002; Fredkin, 2003), which is an interesting theoretic construct but has little practical use.

While the examples above show that a whole variety of non-neural substrates can be used for computation in the Reservoir Computing framework, the majority of the research still takes place in the neural networks and neuroscience communities. This is likely due to the historic origins of the field, and also because the concepts of learning systems are closely linked to neural network research. Nonetheless, applying the RC concept to other (physical) systems could unlock a whole new range of applications. Moreover, there is a wide range of technologies or physical systems that have properties which are required for RC (fading memory and a

tunable nonlinear mapping), but which have so far not yet been fitted into the RC framework.

The transition from neural reservoirs to novel implementations is however far from trivial. Many questions arise, such as: what dynamical systems are useful reservoirs? How should their dynamics be tuned for optimal performance? What other parameters or properties influence their use as a reservoir? To answer these questions, ideally a well-defined theoretical basis for RC should be available, but at this point it does not exist yet.

1.5 Contributions and structure

Reservoir Computing provides a simple but powerful framework for harnessing and using the computational capabilities of nonlinear dynamical systems. This enables the application of a wide variety of existing technology in an entirely new context, namely for learning complex temporal tasks. However, in order to use these dynamical systems for actual engineering problems, several hurdles still need to be overcome. This doctoral thesis aims to enhance the understanding of the principles underlying RC and its use in engineering applications.

The fundamental idea underlying Reservoir Computing is that the reservoir does a suitable nonlinear mapping with a fading memory of the input signals into a higher dimensional space. This enables the use of relatively simple but computationally undemanding linear classification or regression algorithms. However, this does not explain which mapping is most suitable, 'how' nonlinear this mapping should be, how much fading memory the reservoir should have etc. When an engineer is presented with a problem he wishes to solve using RC, the following issues need to be addressed:

- Which reservoir node type is most suitable for this application? Many different 'neural' nodetypes have been described and used in literature with very different computational requirements and characteristics. Depending on the specifications of the problem, sometimes a trade-off needs to be made between the computational power of the node types and the available processing power or time - think of real-time applications. In many engineering applications, the time or processing force needed for simulating a full-featured reservoir is simply not available. What impact does this have on the performance?

- Can we use preprocessing techniques to optimize the signal repre-

sentation before feeding it to the reservoir? It is very common in machine learning to incorporate *a priori* knowledge about the nature of the problem and the input signals into the system to enhance the representation. For instance, in the field of speech recognition it is well known that there is much redundancy in the temporal domain which can be reduced drastically through appropriate use of resampling and frequency domain techniques. How can an engineer transpose these techniques to other, less known problem areas?

- Reservoir Computing is a temporal processing mechanism, which means that it operates on signals where information is also encoded in the precise timing of the values. This automatically leads to the issue of how to represent time inside the different components of the system, namely the input signal space, the reservoir state space and the space of the readout. Every transition between these spaces offers the possibility of tuning the timescales for optimal performance.

- How should the dynamics of a reservoir be tuned so that it operates in the optimal regime? One of the tasks of the reservoir is to 'dynamically transform' the input signals so that the performance of the linear readout is enhanced. There are at present no tools to determine or quantify this dynamic regime accurately. There are some approximate measures and guidelines that work well for standard reservoirs, but which break down completely when going to more advanced node types. A more accurate and realistic method for measuring the way the dynamic regime of the reservoir evolves as it is driven by the input would be very helpful.

- Even if an engineer is able to tune the reservoir to the optimal settings for a given nodetype and set of input signals, this knowledge is often impossible to apply to a different reservoir type. Also, if the dynamic properties of the input signals change (for instance, if the dynamic range increases or decreases), this affects the operation of the reservoir and can lead to sub-optimal performance. Ideally, the reservoir should be able to adjust to these changes and try to automatically self-adapt to the input so that the dynamic regime is more or less optimal for the node type and input signals.

1.5.1 Main contributions of this thesis

In this section an overview is presented of the main research contributions of this doctoral thesis.

We start with an overview of several academic and engineering applications that were tackled using traditional RC solutions. We focus not only on the RC system itself, but also discuss the design choices made for the 'peripheral' processing such as pre- and postprocessing steps. The research on speech front-ends for digit recognition with spiking reservoirs and the Ford automotive signal classification are personal efforts. The work on epilepsy detection was initiated as a Master's thesis by Pieter Buteneers (currently pursuing a PhD on this topic) under my supervision.

Next, we introduce the notion of time scales in Reservoir Computing and show how this can be linked to memory properties of the reservoir. We further demonstrate the importance of tuning these time scales to a given task by evaluating their effect on a spoken digit recognition task. This work was done in close collaboration with dr. Benjamin Schrauwen, with contributions by the Master's thesis student Jeroen Defour.

Then, the possibility of using non-neural substrates in the RC framework is demonstrated by presenting and experimentally validating different reservoir implementations that range from standard to advanced and novel. The usefulness of these different reservoir types is shown experimentally by applying them to a set of benchmark tasks with differing requirements. The implementation and evaluation of CNN-based reservoir in simulation and on the hardware was done in close collaboration with dr. S. Xavier De Souza (KULeuven), for which I kindly acknowledge his support. The work on photonic Reservoir Computing was mainly done at the INTEC research department, in close collaboration with dr. Benjamin Schrauwen, prof. Joni Dambre and myself.

We introduce and derive techniques for measuring the dynamical regime that a reservoir operates in. We look at dynamical systems theory for inspiration, borrowing several insights from there and applying them to the RC framework. This yields a novel measure of dynamics that is universally applicable and that offers a more accurate quantification of the reservoir excitability than the standard stationary measures. This work represents largely a personal effort, with support from dr. Benjamin Schrauwen.

The next contribution is the derivation and experimental validation of a generalized version of Intrinsic Plasticity, an unsupervised, bio-plausible adaptation rule that has a mathematical foundation inspired by information-theoretical principles. We generalize this rule beyond the traditional reservoir activation functions and demonstrate that this rule can be used to automatically tune the dynamical regime of the reservoir, irrespective of the initialization or the input signals. The derivation of the generalized rule is a personal contribution, and the experimental validation of the

rule was done in close collaboration with colleague Marion Wardermann, Prof. Jochen Steil and dr. Benjamin Schrauwen.

Finally, we discuss the Reservoir Computing Toolbox (RCT), a complete set of Matlab scripts and functions that allows both rapid 'prototyping' experiments and thorough exploration of design parameter spaces. A wide range of datasets, reservoir types, adaptation rules and readout and training mechanisms are built in the toolbox. Moreover, the toolbox allows the creation and use of more complex, hierarchical reservoir structures with multiple interconnected reservoirs in a single system. The RCT is also built to allow parallelization of experiments on a computing grid of potentially heterogeneous nodes. The original implementation of the RCT was the result of a close collaboration between dr. Benjamin Schrauwen and myself. The revised design of the datastructures and general setup of the toolbox described in this thesis was a personal effort, and for the actual code writing I want to kindly acknowledge the collaborative efforts from colleagues Marion Wardermann, prof. Joni Dambre and Francis wyffels.

1.5.2 Structure of this thesis

The outline of this thesis is as follows.

Chapter 2 covers the construction and use of standard RC systems. I start with a basic recipe on how to create reservoirs of the ESN type, and an outline of the experimental settings that are used throughout this thesis. Next, I discuss some simple but essential tools for maximizing the performance of an RC system and correctly evaluating the results. Finally, the different academic and real-world engineering applications that will be used in the rest of this work are introduced and discussed. This chapter not only shows the broad applicability of the RC framework, but it can also be used as a source of inspiration when confronted with a novel problem.

In Chapter 3, I make a transition from neural reservoirs to more general other dynamical systems. I start with a discussion of the use of different timescale domains in RC systems, and show that tuning the transition between the domains is crucial for performance. Next, the universal applicability of the RC concept is shown by studying several non-standard reservoir implementations, that gradually drift further from the original neural RC incarnations.

As we have discussed, the use of novel dynamical systems as reservoirs requires some tools to facilitate the search for optimal performance. In Chapter 4, I first investigate more sophisticated measures for quantifying the dynamical regime of the reservoirs, using concepts from dynamical

system theory. The ability to quantify dynamics in a reservoir is useful, but ideally, one would like to have an automatic adaptation of the reservoir dynamics. I therefore present a generalized version of an adaptation rule that tunes the dynamics of the reservoir in an autonomous and unsupervised way, based on a criterion that tries to maximize the information transmission inside the reservoir.

Chapter 5 gives a summary and overview of the main conclusions that were reached in each of the individual chapters, and draws an overall conclusion of the research presented in this thesis.

In the Appendix, finally, I describe the Reservoir Computing Toolbox and its main features and design principles. The toolbox was built in collaboration with several colleagues throughout my PhD work, and is now used intensively both inside our research group and by other researchers around the world.

1.6 List of publications

Journal publications

1. K. Vandoorne, W. Dierckx, B. Schrauwen, D. Verstraeten, R. Baets, P. Bienstman and J. Van Campenhout. Toward optical signal processing using Photonic Reservoir Computing. *Optics Express*, Vol. 16(15), pp. 11182-11192 (2008).

2. B. Schrauwen, M. D'Haene, D. Verstraeten and D. Stroobandt. Compact hardware Liquid State Machines on FPGA for real-time speech recognition. *Neural Networks*(21), pp. 511-523 (2008).

3. B. Schrauwen, M. Wardermann, D. Verstraeten, Jochen J. Steil and D. Stroobandt. Improving reservoirs using intrinsic plasticity. *Neurocomputing* (71), pp. 1159-1171 (2008).

4. D. Verstraeten, B. Schrauwen, Michiel D'Haene and D. Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks*, Vol. 20(3), pp. 391-403 (2007).

5. D. Verstraeten, B. Schrauwen, D. Stroobandt and J. Van Campenhout. Isolated word recognition with the Liquid State Machine: a case study. *Information Processing Letters*, Vol. 95(6), pp. 521-528 (2005).

Conference publications

1. D. Verstraeten and B. Schrauwen. On quantifying dynamics in Reservoir Computing. *International Conference on Artificial Neural Networks (ICANN),* pp. 985-994 (2009).

2. P. Buteneers, B. Schrauwen, D. Verstraeten and D. Stroobandt. Real-time Epileptic Seizure Detection on Intra-cranial Rat Data using Reservoir Computing. *15th International Conference on Neural Information Processing of the Asia-Pacific Neural Network Assembly (APNNA),* pp. 56-63 (2009).

3. P. Buteneers, B. Schrauwen, D. Verstraeten and D. Stroobandt. Epileptic seizure detection using Reservoir Computing. *Proceedings of the 19th Annual Workshop on Circuits, Systems and Signal Processing*, on CD (2008).

4. D. Verstraeten, S. Xavier-de-Souza, B. Schrauwen, J. Suykens, D. Stroobandt and J. Vandewalle. Pattern classification with CNNs as reservoirs. *Proceedings of the International Symposium on Nonlinear Theory and its Applications (NOLTA)*, on CD (2008).

5. F. wyffels, B. Schrauwen, D. Verstraeten and D. Stroobandt. Band-pass reservoir computing. *Proceedings of the International Joint Conference on Neural Networks*, pp. 3203-3208 (2008).

6. B. Schrauwen, D. Verstraeten and J. Van Campenhout. An overview of reservoir computing: theory, applications and implementations. *Proceedings of the 15th European Symposium on Artificial Neural Networks*, pp. 471-482 (2007).

7. D. Verstraeten, B. Schrauwen and J. Van Campenhout. Adapting reservoirs to get Gaussian distributions. *Proceedings of the 15th European Symposium on Artificial Neural Networks*, pp. 495-500 (2007).

8. B. Schrauwen, M. D'Haene, D. Verstraeten and J. Van Campenhout. Compact hardware for real-time speech recognition using a Liquid State Machine. *Proceedings of the 20th International Joint Conference on Neural Networks*, on CD (2007).

9. B. Schrauwen, J. Defour, D. Verstraeten and J. Van Campenhout. The Introduction of Time-Scales in Reservoir Computing, Applied to Isolated Digits Recognition. *Proceedings of the International Conference on Artificial Neural Networks*, Vol. 1(4668), pp. 471-479 (2007).

10. D. Verstraeten, B. Schrauwen and D. Stroobandt. Reservoir-based techniques for speech recognition. *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, On CD (2006).

11. D. Verstraeten, B. Schrauwen, M. D'Haene and D. Stroobandt. Reservoir Computing and its digital hardware implementations. *Proceedings of the Annual Machine Learning Conference of Belgium and The Netherlands (Benelearn),* pp. 171-172 (2006).

12. D. Verstraeten, B. Schrauwen, M. D'Haene and D. Stroobandt. The unified Reservoir Computing concept and its digital hardware implementations. *Proceedings of the 2006 EPFL LATSIS Symposium*, pp. 139-140 (2006).

13. D. Verstraeten, B. Schrauwen and D. Stroobandt. Reservoir Computing with Stochastic Bitstream Neurons. *Proceedings of the 16th Annual ProRISC Workshop*, pp. 454-459 (2005).

14. D. Verstraeten. Stochastic Bitstream-based Reservoir Computing with Feedback. *Fifth FirW PhD Symposium*, on CD (2005).

15. D. Verstraeten, B. Schrauwen and D. Stroobandt. Isolated word recognition using a Liquid State Machine. *Proceedings of the 13th European Symposium on Artificial Neural Networks (ESANN)*, pp. 435-440 (2005).

16. D. Verstraeten, B. Schrauwen and J. Van Campenhout. Recognition of Isolated Digits using a Liquid State Machine. *Proceedings of SPS-DARTS 2005*, pp. 135-138 (2005).

Others

1. B. Schrauwen, D. Verstraeten. Hardware speech recognition using Reservoir Computing. *NIPS 2006 demo session.*

2. D. Verstraeten. An experimental comparison of reservoir computing methods. *Invited talk at NIPS 2007 Workshop on Liquid State Machines and Echo State Networks (2006).*

3. D. Verstraeten. Speech recognition using reservoir computing. *Invited talk : Paris workshop on New Ideas in Hearing (2006).*

2 Standard Reservoir Computing: methods and applications

This chapter focuses on the 'traditional', neural implementations of Reservoir Computing, i.e., Echo State Networks and Liquid State Machines. We first describe different ways of constructing, training and using neural RC systems. We continue by discussing three methodological techniques that are essential to the correct evaluation of the performance of a given RC system. Following this operational view, we take a step back and give three different high-level functional explanations of the functionality of a reservoir. Finally, we devote a section to a thorough specification and discussion of the academic and real-world learning problems that are used in this thesis. The real-world tasks are discussed in more detail by presenting an experimental exploration of the different design choices.

2.1 Operational and functional aspects of reservoirs: the basics

Reservoir Computing is an umbrella term for a set of learning systems. As was mentioned in the introduction to this work, these techniques were independently introduced. In (Verstraeten et al., 2007), I introduce the term Reservoir Computing and propose to unify the existing techniques under this term. Since then, the term has been adopted in literature as a general name for learning systems that consist of a dynamical recurrent network of simple computational nodes combined with a simple (usually linear) readout function.

Because of the broadness of the term Reservoir Computing, it is dif-

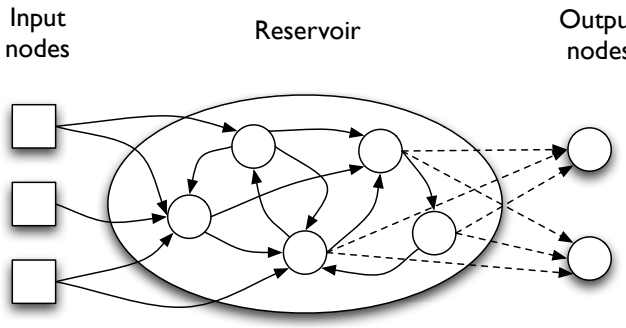


Figure 2.1: Schematic representation of an RC system. Fixed, random connections are indicated with a solid line and trained connections with a dashed line.

ficult to accurately define it: to my knowledge, no exact definition exists in literature. However, to fix the thoughts I will start by giving a qualitative description of the common properties that all the RC systems in this thesis have.

A Reservoir Computing system consists of two parts: the reservoir and the readout function (see Figure 2.1). The reservoir is a (usually) nonlinear dynamical system, consisting of a recurrently coupled network of relatively simple computational nodes. The connections between the nodes are randomly created and globally rescaled so that a suitable dynamical regime is reached. The readout function (or simply readout) is a linear classification or regression algorithm which is trained by example, using simple training mechanisms such as linear regression.

From an engineering point of view, Reservoir Computing systems are attractive learning machines because of the following properties:

- They are easy to construct and optimize. Most RC systems can be globally described using only a few parameters, which greatly reduces the computational requirements for finding optimal settings for these parameters.

- When trained properly, they are relatively robust to input and state noise. This property is inherited from conventional neural networks.

- Because reservoirs are created randomly, they tolerate some variation in their internal parameters. This is of particular interest when using analog electronics or physical hardware to implement the RC systems. In this case, manufacturing and processing variations can lead to differences between different reservoirs with the

same parameters. The RC training method is robust against these variations.

- Only the readout layer is trained, so it is possible to use the same reservoir for solving different tasks simultaneously based on the same input, with minimal additional computational requirements once the layers are trained.
- RC systems show competitive performance in a variety of temporal signal processing tasks. RC has been applied to many industrial and academic problems, and in most cases excellent performance can be reached.

We will now give a more detailed description of how to construct and train a basic Reservoir Computing system.

2.1.1 Creating and using reservoirs

2.1.1.1 The standard architecture

In the following text, we assume that the RC system consists of N reservoir nodes, M inputs and P outputs. Most descriptions of RC systems in literature use the ESN-style of creating the reservoir network, so we will focus on that.

Creating the input and reservoir connections

For the ESN-style RC systems, reservoirs are usually constructed as follows:

- Construct an $M \times N$ input to reservoir weight matrix $\mathbf{W_{in}}$. The weights are drawn from a random distribution or discrete set, and are globally scaled with the *input scale factor*. The input scaling is an important parameter for the performance because it determines how strongly the reservoir is driven by the input. Depending on whether the input signals should be fed to all the reservoir nodes, this matrix can be full (all elements non-zero) or have a sparsity defined by the *input connection fraction*.
- Construct an $N \times N$ reservoir interconnection weight matrix $\mathbf{W_{res}}$. The values for the weights are again drawn from a distribution (e.g., a gaussian distribution) or a discrete set of values (e.g.,$\{-1, 1\}$). Here, too, only a fraction (the *reservoir connection fraction*) of the weights are non-zero. In the original contribution by Jaeger, it is noted that the connection fraction should be small (around .1),

thus creating a sparsely connected network. The rationale behind this is that the network should create a rich 'reservoir' of different nonlinear transformations (called *echos*) of the current and past input values - and a sparse network leads to non-correlated echos. However, later research has shown that this line of reasoning is not always justified (see e.g. (Verstraeten et al., 2006), (Schrauwen et al., 2008a)).

- Rescale the weight matrix globally, such that the reservoir has a suitable dynamic excitability. The most common way to do this is to tune the spectral radius of $\mathbf{W_{res}}$. The spectral radius of a matrix is its largest absolute eigenvalue. It is a static measure[1] of the dynamic excitability of the network - a value close to 1 is usually proposed as a good starting point for optimizations of ESNs. The precise rationale behind the rescaling is explained in detail in Subsection 2.1.2.3 below and in Chapter 4.

Simulating the reservoir and training and testing the read-out

- Construct a dataset D consisting of $|D|$ samples.[2] The number of timesteps in the dth sample is written as T_d. The dth sample consists of a $M \times T_d$ input signal matrix $\mathbf{u}$ and a corresponding $P \times T_d$ desired output signal matrix $\mathbf{y}$ (we denote the actual output of the reservoir system as $\hat{\mathbf{y}}$ to discern between the desired and actual output). Split this dataset in a set D_{train} of training samples and D_{test} of testing samples, with $|D| = |D_{train}| + |D_{test}|$.

- Simulate the network using the training set D_{train} as follows: the network state at time k is denoted as $\mathbf{x}[k]$. For every sample, we initialize $\mathbf{x}[0] = \mathbf{0}$. Next, the network is simulated recursively, in a timestep[3] based way, as follows (see Figure 2.2):

$$\mathbf{x}[k+1] = f(\mathbf{W}_{res}\mathbf{x}[k] + \mathbf{W}_{in}\mathbf{u}[k]).$$

  Note that in (Jaeger, 2001a), the term $\mathbf{u}[k+1]$ is used instead of $\mathbf{u}[k]$ in the state update equation.

[1] I.e., it only takes stationary (unchanging) properties of the network into account.
[2] The notation $|A|$ denotes the cardinality of the set A, i.e., the number of elements it contains.
[3] Throughout this thesis, the notation $[k]$ with square brackets is used for timesteps in discrete time, and (t) with round brackets is used for coninuous time.

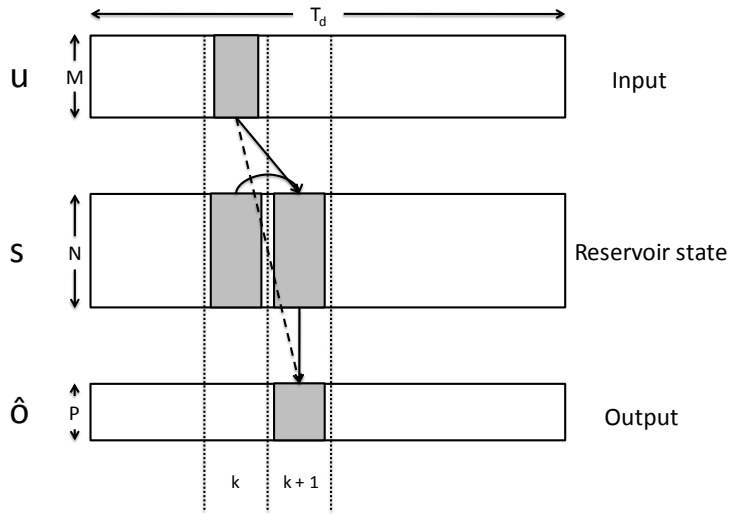


Figure 2.2: Schematic representation of the simulation of an RC system.

- After every sample is simulated, the $|D_{train}|$ reservoir state matrices are concatenated into a large state matrix $\mathbf{A}$ of dimension $(\sum_{d=1}^{|D|} T_d) \times M$.
- Compute the output weights by least squares regression on the matrix $\mathbf{A}$, using the desired output matrix $\mathbf{y}$ as the right-hand side. I.e., compute the matrix $\mathbf{W}_{out}$ that satisfies the following equation

$$\mathbf{W}_{out} = \min_{\mathbf{W}} \|\mathbf{A} \times \mathbf{W} - \mathbf{y}\|^2 .$$

  In practice, this can be done in one step by using the Moore-Penrose generalized matrix inverse (Penrose, 1955), or *pseudo-inverse* $\mathbf{A}^\dagger$ of the matrix $\mathbf{A}$, which is defined as : $\mathbf{A}^\dagger = (\mathbf{A}^\mathbf{T}\mathbf{A})^{-1}\mathbf{A}^\mathbf{T}$, as follows:

$$\mathbf{W}_{out} = \mathbf{A}^\dagger \mathbf{y} = (\mathbf{A}^\mathbf{T}\mathbf{A})^{-1}\mathbf{A}^\mathbf{T}\mathbf{y}.$$

- Simulate the network on the test set D_{test} in the same way as above, and compute the output as follows:

$$\hat{\mathbf{y}}[k] = \mathbf{W}_{out}\mathbf{x}[k].$$

- Evaluate the performance on the test set using an appropriate error measure. For instance, a commonly used error measure for one-

dimensional output signals is the Normalized Root Mean Square Error (NRMSE), defined as:

$$NRMSE = \frac{1}{|D_{test}|} \sum_{d=1}^{|D_{test}|} \frac{1}{T_d} \sum_{k=1}^{T_d} \sqrt{\frac{(\hat{y_d}[k] - y_d[k])^2}{\sigma_{d,y}^2}},$$

where $\sigma_{p,y}$ denotes the variance of the desired output signal y in example d.

2.1.1.2 Variations on the basic architecture

Many architectural and experimental variations on this basic setup have been used and described in literature. We list the main possibilities of extending this setup, but this list is by no means exhaustive:

- Using different activation functions. Many node types have already been described in literature, including threshold functions, hyperbolic tangent, Fermi functions, or spiking neurons.

- Using direct input-to-output connections, i.e. concatenating the reservoir states $\mathbf{x}$ and the inputs $\mathbf{u}$ to train and test the readout layer. Often, this leads to a slight increase of performance at the expense of a longer training time.

- Using output feedback, i.e. feeding the values of the readout layer back into the reservoir or even to the readout layer itself. This is useful in the case of signal generation tasks where the reservoir system is used to generate a (possibly multidimensional) signal autonomously or where longer memory is required (Maass et al., 2007).

- Using an online learning method such as Least Mean Squares (LMS) or Recursive Least Squares (RLS) to train the output weight matrix $\mathbf{W}_{out}$. This is useful in case the statistical properties of the inputs, outputs or their relation change - e.g. in the case of channel equalisation (Jaeger and Haas, 2004).

- Using online adaptation rules for adjusting certain parameters of the reservoir. This possibility will be discussed extensively in chapter 4.

- Using so-called *leaky integrator* neurons in the reservoir. In this case, the update equation for the reservoir can be written as

$$\mathbf{x}[k+1] = (1-\lambda) \cdot \mathbf{x}[k] + \lambda \cdot \tanh(\mathbf{u}[k]\mathbf{W}_{in} + \mathbf{x}[k]\mathbf{W}_{res}).$$

In this equation, λ is a time-constant that determines the slowness of the dynamics of the reservoir. In effect, this is simply a first-order low-pass filtered version of the actual sigmoid neuron output. The use of this neuron model for Echo State Networks was first described in (Jaeger, 2002), but has since been extended to more advanced bandpass versions in (Siewert and Wustlich, 2007; wyffels et al., 2008c). We will discuss both leaky integrator neurons and bandpass neurons in more depth in Sections 3.1 and 3.2.

2.1.2 Three views on reservoir functionality

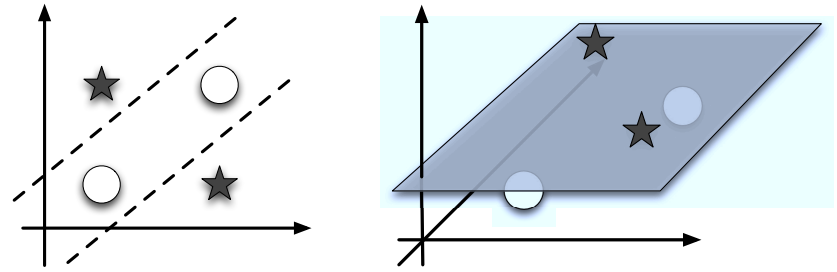
One of the attractive features of Reservoir Computing is the combination of its simplicity of use and its good performance on many tasks. At this point however, it is not yet clear why these systems perform well. In fact, the reservoirs are usually created randomly which is not associated with optimal performance.

An elaborate theoretical foundation of the operations of reservoirs could shed some light on why reservoirs work well, and more importantly: which ones work best. Such a theory is presently still lacking, but there is some knowledge to be gained when looking at the similarities between reservoirs and other computational systems. In this section, we will discuss three different angles from which to explain the functionality of reservoirs in RC systems, namely kernel machines, linear regressors or classifiers and dynamical systems. These explanations are not mathematically strict but do help to form an intuitive view on the properties of reservoirs that make them suited for signal processing.

2.1.2.1 The reservoir as a temporal kernel

The first concept with which RC shares some common ideas is that of kernel methods (Scholkopf and Smola, 2002), and particularly Support Vector Machines (SVMs) (Vapnik, 1999; Steinwart and Christmann, 2008). SVMs are a technique in machine learning that attains state-of-the-art performance results in many application fields (Cristianini and Shawe-Taylor, 2000). The theory underlying kernel methods is founded in statistics rather than biology or neuroscience.

Explained very briefly, kernel methods work by transforming input vectors using a nonlinear map into a high-dimensional space. In this so-called feature space, any standard (usually linear) technique from machine learning or statistical analysis can be applied, such as linear or nonlinear regression, principal component analysis (PCA) (Scholkopf et al., 1997) or canonical correlation analysis (Van Gestel et al., 2001). By projecting



(a) The 2D case: not linearly separable

(b) The 3D case: linearly separable

Figure 2.3: Projection of the input into a higher dimensional space can make a classification problem linearly separable.

the input into a higher-dimensional space using the right kernel, the computational performance of these methods can be boosted considerably.

This principle is illustrated in a simplified setting in Figure 2.3. The problem shown here is known as the XOR problem and is one of the most basic examples of a classification task that is not linearly separable[4]. In the two-dimensional case on the left, there is no single line that separates the circles from the stars. However, by choosing the right projection into a three-dimensional space (shown on the right), it becomes trivial to find a separating hyperplane (the equivalent of a line in higher-dimensional spaces). This is due to the fact that, loosely speaking, if labeled data is represented in a space with more dimensions, the probability of the data being linearly separable increases (Cover, 1965).

Arguably one of the most important characteristics of kernel methods is the fact that the projection into feature space is not explicitly computed. This is due to a mathematical nicety called the *kernel trick* (Aizerman et al., 1964), which follows from Mercer's theorem that states that any continuous symmetric positive semi-definite kernel $K(x, z)$ can be written as an inner product in the high-dimensional feature space. More formally, for a kernel K that has the properties cited above, there exists a mapping ϕ for which

$$K(x, z) =< \phi(x), \phi(z) > .$$

Many techniques in statistical analysis and machine learning use inner products between data samples as a means of defining some sort of simi-

[4]This task was a.o. used to show the limitations of the single layer perceptron, which can only solve linearly separable tasks (Minsky and Papert, 1969). This limitation was later overcome by adding hidden layers to the perceptron, forming a Multi-Layer Perceptron (MLP).

larity measure and use this in their computations, and due to the kernel trick this inner product can be conveniently replaced by a kernel function, without having to actually compute the high-dimensional mapping $\phi(\cdot)$. This enables for instance the use of infinite-dimensional feature spaces.

This projection of the input into a higher-dimensional space, followed by the application of a simple linear algorithm also occurs in the case of Reservoir Computing. The functionality performed by the reservoir can be seen as a spatio-temporal, nonlinear transformation into a higher-dimensional space, since the states of the neurons all represent a random but fixed nonlinear mapping of the current and previous inputs to the network. There are, however, two significant differences between kernel methods and RC: in the case of reservoirs, the mapping does have to be computed explicitly, and secondly, the mapping into feature space (i.e., the reservoir state space) is explicitly temporal because the reservoir has a fading memory of previous inputs[5]. However, this explicit computation of the kernel mapping does enable the use of, e.g., physical systems as reservoirs.

A simplified illustration of this temporal trajectory of the reservoir in the reservoir state space is shown with a separating hyperplane in Figure 2.4. Here, the trajectory of a reservoir in response to two different spoken digits is shown (see Section 2.2.2 below for a specification of this task). Because the reservoir has 100 nodes, the actual trajectory is 100-dimensional which is difficult to represent pictorially - to say the least. Because of this, I reduced the dimensionality of the trajectory by plotting only the three main components resulting from dimensionality reduction with Independent Component Analysis (ICA) (Karhunen et al., 2004). Also shown is a hyperplane that separates most of the points in the trajectory belonging to different classes. While the trajectories are actual dimension-reduced trajectories by a reservoir, the hyperplane was drawn manually. The figure is mainly intended to form some intuition about the kernel functionality of a reservoir.

2.1.2.2 The reservoir as a complex preprocessing filter for linear methods

Because the training and application of Reservoir Computing systems (as with kernel methods) is in practice reduced to the use of simple linear methods, nearly the whole body of theory from this well-researched field is also applicable. Least squares regression is a tried-and-true method for

[5]This temporal mapping can be added to kernel methods as well using, e.g., delay lines or specialised string-based kernels, but as discussed in subsection 1.3.4, this introduces several disadvantages.

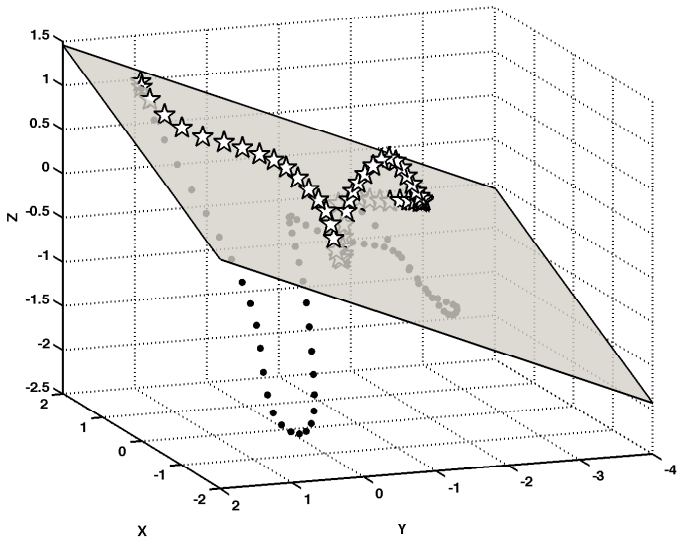


Figure 2.4: Three-dimensional image of a trajectory of the reservoir in response to two different spoken digits, one drawn as stars and the other as circles. Also shown is a classifying hyperplane that separates (most of) the points.

fitting linear models to data, and was first described by Gauss already in the nineteenth century. Many extensions on this basic algorithm have since been described such as weighted linear least squares (Ryan, 1997), LASSO regression (Tibshirani, 1996) , ridge regression (Tikhonov and Arsenin, 1977) or online variants from adaptive filter theory such as Least Mean Squares (LMS) (Haykin and Widrow, 2003) or Recursive Least Squares (RLS) (Haykin, 1991). In this respect, the reservoir can be seen as a random nonlinear preprocessing mechanism with fading memory that boosts the power of linear algorithms.

2.1.2.3 The reservoir as a dynamical system: computation at the edge of chaos

Finally, RC shares some research questions with the fields of nonlinear dynamics and control theory. Both the literature on LSMs and ESNs has put forth the notion that reservoirs should operate at the so-called *edge of chaos* (Langton, 1990; Legenstein and Maass, 2007). However, this term is a bit misleading since these systems, as well as most reservoirs, actually operate in a dynamically rich but stable regime. The *edge of stability* would therefore perhaps be a more appropriate term.

The reason behind the importance of this type of dynamic regime is best understood for classification problems: the reservoir should react dynamically enough to input signals from different classes so that the problem becomes linearly separable, which enables the linear readout to do the classification. This view is related to the function of a reservoir as a temporal kernel, described above. On the other hand, if the reservoir is too dynamic (i.e. chaotic), its sensitivity will be too large and the information about the inputs will be washed out by the wild dynamics. More importantly: a chaotic reservoir means a high sensitivity to noise, i.e., changes in the input signal that do not contain relevant information for the problem at hand. Hence, the edge of stability is the optimal region for the reservoir to operate in.

As was mentioned before, the most common parameter that tunes the dynamics of the reservoir is the spectral radius of the reservoir weight matrix. A value close to one is usually chosen. The choice of this value is due to the so-called echo state property (ESP). This property, introduced in (Jaeger, 2001a), states informally speaking that in the long term a reservoir 'forgets' its initial state. This is highly related to the fact that a network should possess 'fading memory' in LSM terminology (Maass et al., 2004c) in order to be useful. The ESP is proven to depend on the scaling of the reservoir weight matrix: for tanh reservoirs, if the largest singular value (LSV) of the matrix is smaller than one the ESP is present. However, if the spectral radius is larger than one, the ESP is not present *for zero input*. This condition can be explained from a systems theory point of view: for zero input the reservoir behaves as a linear system. The eigenvalues of the weight matrix are then essentially the poles of the system (see Figure 2.5). It is a well known result that a linear system is asymptotically stable if the magnitude of all poles is smaller than zero. Asymptotic stability means, informally speaking, that the system will eventually return to a zero state if the input is switched off.

The fact that the spectral radius is only a strict criterion for zero input is quite important and is often misinterpreted. The spectral radius is a static measure (it only depends on the weight matrix), and does not take the input to the network into account. However, once the network is driven by an external input, the operating point of each of the nodes shifts along the nonlinearity and the 'effective' spectral radius decreases. This means that for reservoirs that are driven by a sufficiently large input, the ESP can hold for weight matrices with spectral radius larger than one.

The spectral radius and other measures of dynamic behaviour in reservoirs will be revisited in Chapter 4.

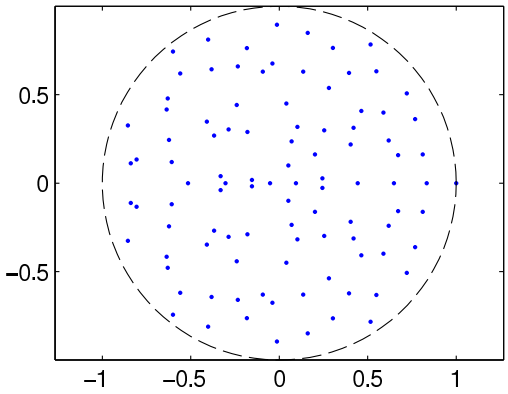


Figure 2.5: Figure showing the eigenvalues of a random matrix with gaussian distributed elements plotted in the complex plane. The spectral radius (largest absolute eigenvalue) is one, which is indicated by the single eigenvalue lying on the unit circle (on the far right).

2.1.3 Performance evaluation

In this section we briefly discuss three mechanisms which are more or less standard in the context of machine learning, but which are not rigourously applied in the context of Reservoir Computing. We argue that these simple extensions of the classical training/testing methodology can and in fact do increase not only the performance on tasks, but also the robustness of the trained systems and the scientific validity of the conclusions that can be drawn from experimental evidence. As such we argue that these methods should be applied to RC systems, in the cases where it is useful. Obviously, these techniques are also applied in the experiments throughout this thesis whenever it is appropriate (for instance, the Fisher relabeling discussed in 2.1.3.3 is not always useful).

2.1.3.1 Regularization

Consider the following series of numbers:

$$1, 1, 2, 3, 5, 8, ...$$

When asked to give the next number, most people will respond with 13, because they recognize the first digits of the Fibonacci sequence - and even if they don't know the exact name of the sequence, most will recognize the fundamental rule that generated this series of numbers: namely that every number (apart from the first two) is the sum of the two pre-

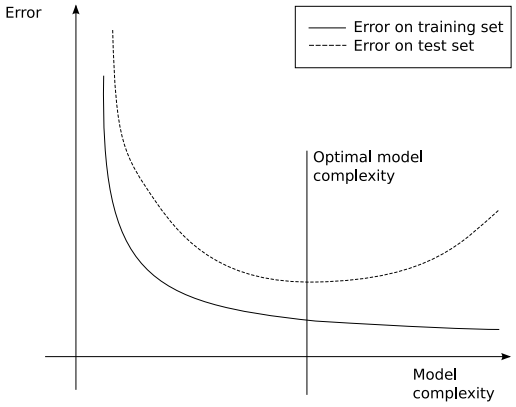


Figure 2.6: Overfitting: the error on the training set keeps decreasing while the error on the (unseen) test set increases. The model is learning the noise on the data instead of generalizing (learning the statistical properties of the data).

vious numbers. This type of sequence-completion tasks is quite common in some psychometric tests such as IQ-tests, but after some consideration one could argue that these tests are fundamentally flawed because there is no single right answer to the question 'What is the next number in this sequence?'. Even worse: there is an infinite number of possible answers. Given a finite sequence of numbers, one can think of any number of rules that have generated this sequence and that will come up with different values for the next number in the sequence. Why is it, then, that the majority of people still gets the answer that is intended, and that researchers even count on this general consensus about the 'right' answer to draw conclusions about someone's level of intelligence?

The reason for this is related to the principle of Ockham's razor. William of Ockham stated that 'entities must not be multiplied beyond necessity', which can be rephrased as: if multiple explanations of a phenomenon are available, the simplest one is preferable. This principle is used very often (sometimes implicitly) in science and research, and it is so embedded in our thinking that many people would characterize this as 'plain common sense'. This is why - even though there are infinitely many possible rules that could have generated the sequence above - people tend to choose the simplest rule: the Fibonacci rule.

Why are simple rules the best answer in this case? In other words: why does an overly complex model not generalize well to unseen data? This is because complex models or rules have a greater chance of incorporating properties specific to the data (such as noise), as opposed to the properties of the system that generated the data. One way to view this problem is

to consider the well-known concept of *overfitting*. Naively put, a model has overfitted to the data when it has 'learnt the data by heart' instead of capturing the underlying properties of the data. These models have failed to generalize to unseen data. Overfitted models will typically exhibit very good performance on the training set but bad performance on the test set (see Figure 2.6). In the context of machine learning, the assumption is sometimes used that a training set can be interpreted as a set of class prototypes, corrupted by some form of noise. A good model should be able to extract the fundamental underlying properties of the data while ignoring the irrelevant information contained in the noise. In case we use a model that is too complex (or: too computationally powerful) it will also learn the noise on the training set. This means that, when presented with unseen data samples, the model will be sensitive to the noise on these inputs, and will produce non-accurate results.

So, we can say in very general terms that a trade-off should be made between model complexity and generalization capability. The model should be complex enough to accurately model the underlying system, but not so complex that it becomes sensitive to the noise on the samples. This design methodology is known as *model selection*. One way to do this is by using regularization, which is a rather general concept of imposing some constraints on the model (usually some form of *smoothness constraint*) in order to control the trade-off between model complexity and overfitting avoidance. We will illustrate the concept with a simple introductory example.

In Figure 2.7, some datapoints where generated from the system described by the following fifth-degree polynomial:

$$y = 3 - 2x + 3x^2 - x^3 - 3x^4 + .04x^5$$

We take some values for x in the interval $[-1, 2]$, and compute the corresponding y values. Next, Gaussian noise is added to these y values with a standard deviation of 0.9. This set of noisy (x, y) datapoints was used as a training set to fit[6] several polynomials of different degrees. Here, the model complexity is determined by the degree of the polynomial (more accurately: the number of free parameters). The actual polynomial is plotted in the dashed lines, and in Figure 2.7 fitted polynomials of degree one, five and eight are plotted. Also, the original polynomial and the fitted curves were plotted in the interval $[2, 3]$ to evaluate the performance on unseen data (the test set).

It is clear that for the case of a polynomial of degree one (a simple line), the model is too simple and none of the properties of the data is

[6] Using Matlab's `polyfit` function, which uses least-squares fitting.

captured. In the case of a polynomial of degree eight, however, the model has become too complex and has overfitted, which is visible through the good performance on the training data, but very bad performance on the test data. The middle figure seems to strike the correct balance between the complexity and generalization and is (not coincidentally) of the same degree as the original polynomial.

How do we apply this to Reservoir Computing setups? In the case of polynomials, the model complexity is easily controlled with the degree of the highest term. For linear discriminant systems however, the degree of the variables is always one. How does one control the model complexity there? There exist multiple ways to do this, depending on how one chooses to characterise the regularization. Usually, linear models are regularized by adding an additional penalty term - related to the norm of the weights - to the prediction on the training set. This can be expressed formally as:

$$w_{opt} = \arg\min_{w} \|Aw - B\|_k + \lambda \|w\|_l , \tag{2.1}$$

where $\|\cdot\|_p$ denotes the p norm, and λ is a parameter that controls the tradeoff between the error on the training set and the weight norm. Depending on the values of k and l, different forms of regularized linear models can be obtained. For instance, if $k = 2$ and $l = 1$, we get what is called LASSO-regression (Tibshirani, 1996), which leads to sparse models (meaning that many of the weights are zero), which in turn could be useful in e.g. hardware implementations because of the limited number of connections. Recent results from our lab also show that for certain tasks LASSO-regression can achieve better results or more stable generators than classic linear regression (Dutoit et al., 2009).

The more common case, and the case we will discuss here and use in all the experiments described in this work, is the case where $k = l = 2$. Here, both norms are the standard Euclidian norms and we obtain what is known as *ridge regression* or (less commonly) Tikhonov regression (Tikhonov and Arsenin, 1977). Here, as with standard least-squares regression, the solution to the equation 2.1 can be obtained in one step as follows:

$$w_{opt} = (A^T A + \lambda I)^{-1} A^T B$$

In this case, there exists a convenient relation between the regularization parameter and the model complexity (which is related to its computational expressive power). One can state:

$$\gamma(\lambda) = \sum_{i=1}^{p} \frac{\sigma_i}{\sigma_i + \lambda},$$

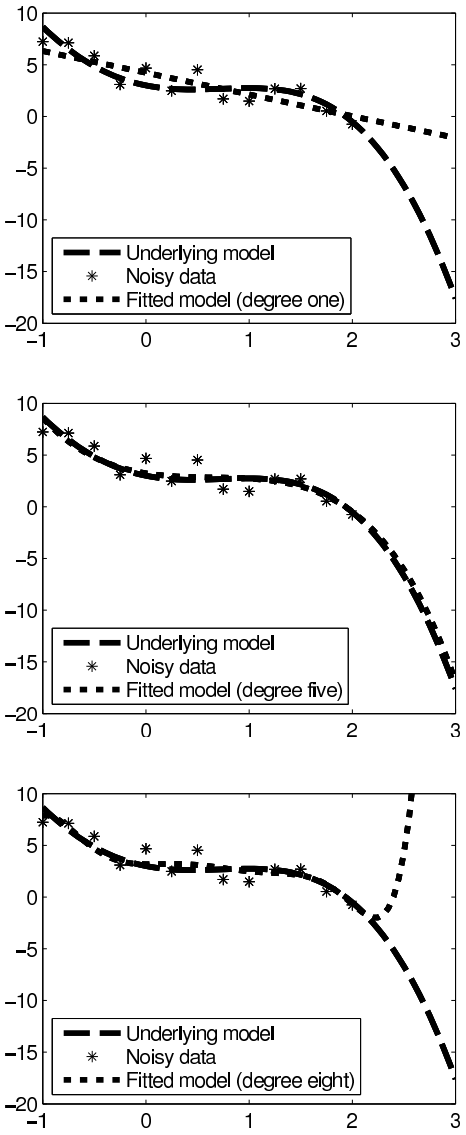


Figure 2.7: Illustration of overfitting and model complexity. In the top figure, the model is too simple to fit the data. In the middle figure the model complexity is just right, and in the bottom plot the model is too complex and overfitting occurs.

where λ is the value of the regularization parameter, $\gamma(\lambda)$ is the *number of effective parameters,* and σ_i is the ith eigenvalue of the matrix $A^T A$. The effective number of parameters quantifies the number of free parameters of the model (the readout weights, in the case of reservoirs) are actually used, and as such gives an indication of the complexity of the model (Moody, 1992).

Another way of performing regularization is by adding noise to the data. The noise on the training set will reduce the absolute magnitude of the parameters, which in turn constrains the model complexity. In (wyffels et al., 2008a), an experimental evaluation of both regularization methods is done using a time series prediction and a system identification task. The conclusion is that both methods yield similar optimal results. However, ridge regression is preferable for pragmatic reasons: using noise as regularization mechanism is inherently non-deterministic (which means that results are not generally reproducible), and in cases where state noise is added to a reservoir with output feedback, the whole reservoir needs to be resimulated for every noise level, which dramatically increases the computational requirements.

While ridge regression offers good control over the smoothness of the obtained regression/readout function through a single parameter λ, the best value to use still needs to be determined. The optimal value for this regularization parameter would be the one that gives the best performance on unseen data. Determining this optimal value is done using grid-searching of the parameter with cross-validation, which will be discussed in the next subsection.

2.1.3.2 Cross-validation

Cross-validation is a technique used in statistical analysis and machine learning for evaluating the performance of a model while eliminating as much as possible misleading results due to (accidental) poor choice of the training and test sets. These anomalies can occur (especially in the case of small datasets) when for instance in a classification task the training set does not cover the full space of classes. If the test set contains an instance of a class that was not present in the training set, the model will likely misclassify this sample which results in an unrepresentatively bad performance. Or, in a more general case, it is possible that the training set does not cover a representative sampling of the full population of possible input patterns, and as such the performance of the trained model will be suboptimal - not due to a lack of expressive power of the model but due to a problem with the dataset. These problems can be avoided by using cross-validation.

Cross-validation involves dividing the dataset, containing D samples, into a number K of subsets, each containing the same number of samples (which means that K should ideally be a divisor of D - if this is not the case, the subsets should be as similar in size as possible). $K-1$ subsets are selected for training, and the remaining subset is used for testing of the trained model. This process is repeated K times (hence the rather archaic term *folds*), every time with a different subset for testing. After this process, the performance is averaged out across all test performances. This way, every subset is used exactly once for testing, which reduces the chance of statistical anomalies considerably. In the special case where $K = D$, the term *leave-one-out* cross-validation is used, because for every fold a single sample is left out of the training set and used for testing. The number of folds K is a parameter that offers a trade-off between accurate evaluation of the performance of a model and computational requirements: the extreme case of leave-one-out will offer the most accurate estimate of the performance of the model (and will also maximize the use of the data for training), but will also require the most time to evaluate. However, the combination of a least squares model and leave-one-out offers an interesting extension: the impact on the error of removing one datasample from the training set can be computed without retraining the model - a process known as *virtual leave-one-out* (Dreyfus, 2005). In this way, an accurate evaluation of the performance of a linear classifier can be obtained in a computationally very efficient way.

Cross-validation can also be used to select the optimal regularization parameter. This is done by doing cross-validation on two levels in a nested manner (this is also implemented in the RC Toolbox, described in the Appendix). In this case, there are three different datasets: the training set (used for training), the *validation set* (used for evaluating performance in the inner cross-validation loop) and the test set (used for evaluating performance in the outer cross-validation loop). The following simple example illustrates this principle: suppose our dataset consists of 4 samples. We first split the dataset using an outer 4-fold cross-validation scheme, using every sample in turn for testing. For every fold in this outer cross-validation loop, the regularization parameter is set to different values. The performance of every value of the regularization parameter is then evaluated using inner three-fold cross-validation scheme, which splits the remaining three samples into a training set containing two samples, and a validation set containing one sample. While this training scheme can be computationally costly, especially for a large number of folds, it does ensure a more accurate evaluation of the performance of a classifier given a certain dataset, and also guarantees that the regularization is near-optimal.

Table 2.1: Illustration of how a dataset of 4 samples can be subdivided in a training set, validation set and test set using cross-validation.

Training set	Validation set	Test set
1, 2	3	
1, 3	2	4
2, 3	1	
1, 2	4	
1, 4	2	3
2, 4	1	
1, 3	4	
1, 4	3	2
3, 4	1	
2, 3	4	
2, 4	3	1
3, 4	2	

2.1.3.3 Unbalanced datasets and Fisher relabeling

A final method that is beneficial for achieving optimal results for practical classification applications in Reservoir Computing (which is as yet - to our knowlege - undocumented), is the use of so-called Fisher-relabeling. In many cases, the dataset being used is unbalanced, which means that the number of examples of each class is not representative of the true underlying distribution. This can occur for instance in a binary (i.e. two-class) classification task, where more examples of class one are present in the dataset than class two. Another example that occurs commonly is the case of a multi-class problem (such as the speech recognition task which is featured throughout this thesis). For multi-class tasks, with M classes, the designer of the system always needs to choose between one-versus-one classifiers or one-versus-all classifiers. The distinction is simple: in the former case, a classifier is trained to discern every class from every other class (resulting in $M(M-1)/2$ classifiers), in the latter case only M classifiers are trained - each one will discern a class from *all* other classes jointly (Duda et al., 2001). In the case of a one-versus-all classifier, even if the original dataset is balanced (i.e. there is an equal number of examples of every class), the dataset is unbalanced from the point of view of any one of the single classifiers since there are $M-1$ times as many negative examples as positive examples.

This unbalance will have an effect on the generalization capabilities of the classifiers. Since the readout is trained using least-squares regression,

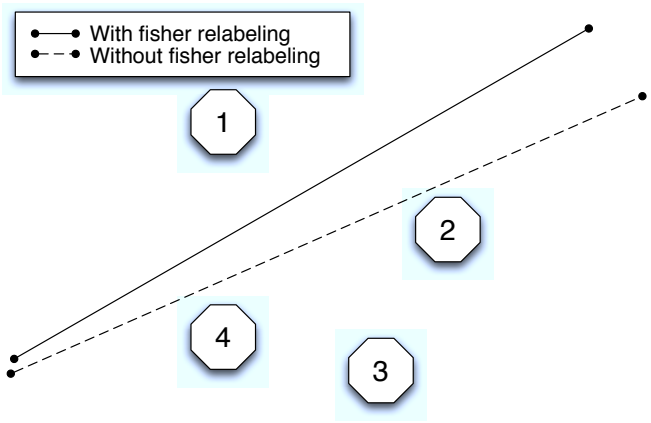


Figure 2.8: Illustration of the effect of fisher relabeling. Without relabeling, the separating hyperplane between the classes is shifted towards the classes occuring most in the dataset (dashed line), but if the class labels are reweighted, this effect can be countered (solid line). Clearly, the margin of error is larger in the latter case.

the separating hyperplane will shift towards the class centers that are most present in the dataset (this is illustrated in Figure 2.8). This effect is undesirable: in the case of a one-versus-all classifier one wants the hyperplane to lie 'in the middle' between the class in question and the other classes. This can be achieved e.g. in the two-class case[7] with n_1 examples of class 1 and n_2 examples of class 2 by relabeling the classes from the usual $[1, -1]$ for positive and negative examples respectively, to $[\frac{n_1+n_2}{n_1}, \frac{n_1+n_2}{n_2}]$. In this way, the class labels reflect the unbalance of the number of examples in each class, and the shifting of the hyperplane is undone. It is shown in (Duda et al., 2001) that the least-squares classifier that is obtained after relabeling is actually equivalent to the so-called Fisher discriminant (hence the name). This discriminant function builds a linear hyperplane that aims to maximize the separation between two classes (the *between-class scatter*) while at the same time minimizing the variance of the samples in the same class (the *within-class scatter).*

Finally, we note that in the case of RC and from the point of view of the readout layer, the reservoir state at every timestep should be considered an example of a given class (since the readout operates on the reservoir states). This means that if the examples of a class do not consist of the same number of timesteps (which is the case in e.g. digit recognition, where some digits are uttered faster than others), the dataset is unbalanced from the point of view of the linear readout, and it is beneficial

[7] This method can easily be extended to more than two classes.

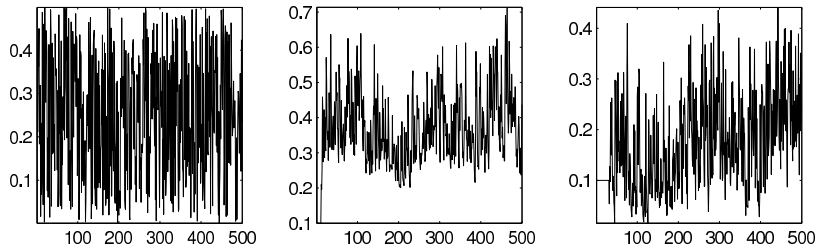


Figure 2.9: Example input (a) and output timeseries for the 10th order (b) and 30th order (c) NARMA task.

to apply fisher relabeling. While this operation is computationally cheap (the relabeling of the classes is a very simple operation), the effects on the performance can be substantial. As an illustratory example, we trained a simple reservoir of 100 nodes on the digit recognition task (see Section 2.2.2 below for a specification of this task) and attained a word error rate (WER) of 7.6% without relabeling, and a WER for the *same* reservoir of 6.6% after training on the relabeled data - meaning a full percent decrease in error.

2.2 Applications

In this section, I present a set of both academic and real-world applications that have been tackled using RC during my research. This section serves a double goal. The tasks presented here will be featured throughout this thesis and they are introduced and specified in detail here. Additionally, we present a more extensive experimental illustration of the use of RC in three different real-world engineering tasks. As such, this section can be read as a presentation of the broad scope of tasks that RC can be applied to, and can serve as a source of inspiration when trying to tackle a novel engineering problem.

2.2.1 Academic tasks

2.2.1.1 NARMA

The Non-Linear Auto-Regressive Moving Average (NARMA) task consists of modeling the output of a SISO (Single Input Single Output) system. Two versions are commonly used in literature, an 'easy' tenth order system

defined by the equation:

$$y[k+1] = 0.3y[k] + 0.05y[k](\sum_{i=0}^{9} y[k-i]) + 1.5u[k-9]u[k] + 0.1,$$

and the more difficult thirtieth order system defined by:

$$y[k+1] = 0.2y[k] + 0.04y[k](\sum_{i=0}^{29} y[k-i]) + 1.5u[k-29]u[k] + 0.001.$$

Here, $u[k]$ is a uniform random signal in $[0, .5]$, which serves as the input to the NARMA system. The first 10 resp. 30 timesteps of $y[k]$ are left constant until the system is warmed up. The modelling of the NARMA system is considered a quite difficult task that requires a relatively long memory (an example input and output timeseries for both systems is shown in Figure 2.9). The readout is trained to reproduce the signal $y[k+1]$. A closer inspection of the system's equations reveals that for instance in the 30th order case, the reservoir needs to retain not only the memory of the input signal of 30 timesteps ago (because of the term $u[k-29]$), but actually an exponentially fading memory of all past outputs through the term $0.2y[k]$. The error is measured as the normalized root mean square error (NRMSE), defined in Subsection 2.1.1 above. An important note is that the 10th order equation can sometimes become unstable and run away to very large values for certain random input timeseries. A check for this should therefore be built in when generating timeseries based on this system.

2.2.1.2 Memory capacity

This task was introduced in (Jaeger, 2002). It is not so much a learning problem as a way of characterising the amount of memory that a network has: the RC system simply has to reproduce successive delayed versions of the random input as accurately as possible. The input of the reservoir consists of a temporal signal $u[k]$ which is drawn from a uniform distribution in the interval $[-.8, .8]$. The outputs consist of an infinite number of outputs $\hat{y}_i[k]$ which try to reconstruct the delayed input $u[k-i]$ for $i = 0...\infty$. In practice, we take the number of outputs twice as high as the size of the reservoir ($i = 2N$) which is a good approximation since the recall performance will drop steadily whenever we try to recall more time steps back in time than there are nodes in the network. The "performance" here is the memory capacity (MC), defined in (Jaeger, 2002)

as $MC = \sum_{i=1}^{\infty} MC_i$, with the i-delay memory capacity MC_i defined as:

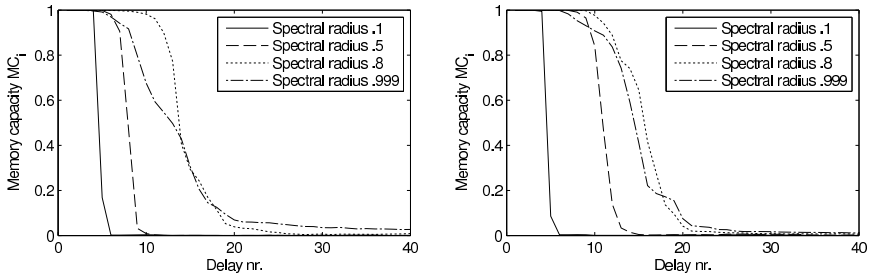
$$\begin{aligned} MC_i &= \max_{W_i} d[W_i](u_i[k], y_i[k]) \\ &= \max_{W_i} \frac{\langle \hat{y}_i[k] u[k-i] \rangle_k^2}{\left\langle (\hat{y}_i[k] - \langle \hat{y}_i[k] \rangle_k)^2 \right\rangle_k \left\langle (u[k-i] - \langle u[k-i] \rangle_k)^2 \right\rangle_k} \\ &= \max_{W_i} \frac{\mathrm{cov}^2(u_i[k], y_i[k])}{\sigma^2(y_i[k]) \sigma^2(u[k-i])}, \end{aligned}$$

with $\hat{y}_i[k]$ the output of the kth regressor and $y_k[k]$ the input signal delayed over k timesteps. Here, $d[W_k]$ denotes the determination-coefficient for the kth readout weight matrix, which is a measure of the variance in one signal caused by the other. The third form of the equation shows that the MC is defined as the normalized correlation between the outputs and their associated delayed input. The maximum determination coefficient for all possible readout weight matrices for the ith delay is taken, but this is automatically achieved by training the readout weights using the pseudo-inverse or Moore-Penrose inverse method, so in effect, by training the RC system in a standard manner this maximum is always achieved (Jaeger, 2002).

The amount of memory in a network is of vital importance for many tasks. However, intuitively one can see that there is a tradeoff to be made: the storage capacity of a network of a given size is limited, and the further a network needs to remember in the past, the less "capacity" it will have left to remember more recent inputs accurately. This can be seen quite clearly on memory curves: these plots detail the individual MC_i terms versus i (see (Jaeger, 2002; White et al., 2002)). Memory curves for reservoirs with both linear and tanh nodes are shown in Figure 2.10. These memory curves typically show a decreasing profile. It was shown theoretically in (Jaeger, 2002) that linear reservoirs (i.e., with the identity function as transfer function) have the highest memory capacity for a given reservoir size, and the MC is bounded by N, the number of neurons in the network. Adding nonlinearities to the nodes decreases the memory capacity, but for some tasks some nonlinear behaviour is also needed.

2.2.1.3 Signal template classification task

The problem here is to discern between two 'template' timeseries that are corrupted by noise. The two templates are a square wave and a sawtooth. Without noise, this task is relatively simple in the sense that conventional, explicitly programmed solutions can easily be devised. With noise, it



(a) Tanh reservoir **(b)** Linear reservoir

Figure 2.10: Memory curve for tanh reservoirs (a) and linear reservoirs (b), for different values of the spectral radius.

becomes less trivial. This task should be considered as a proof-of-concept problem that could demonstrate the potential usefulness of novel reservoir architectures (as was the case for the photonic reservoirs of Section 3.4).

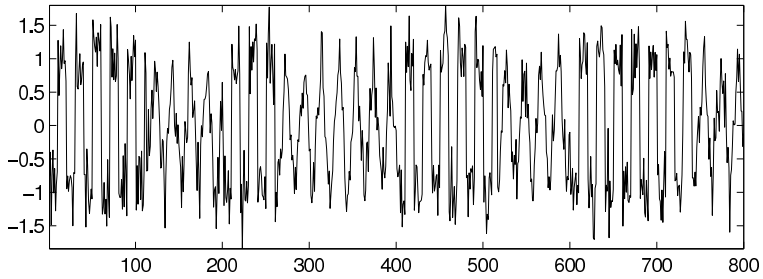
An example input timeseries and corresponding desired output is shown in Figure 2.11. The main difficulty of this task from an RC point of view lies in the transition points between the signal templates: the reservoir needs to be able to react quickly to a change in the signal class, while still being robust enough to retain the current classification output when needed. The error for this task is measured in terms of the zero-one loss (i.e., zero if correct, one if incorrect) on a timestep basis, which simply means the fraction of timesteps that the output of the system is incorrect.

2.2.1.4 Signal generation tasks

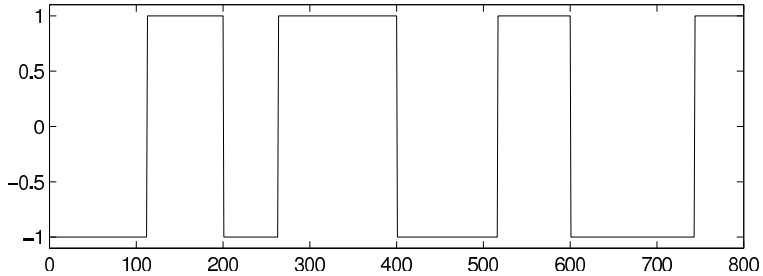
This task differs from the previous ones because this is a signal generation task: the RC system needs to generate the output signal by itself, i.e., without external inputs. An example of a signal generation task is the Mackey Glass timeseries prediction. The goal is to make the reservoir autonomously generate the signal described by the following time delay differential equation:

$$\dot{x} = \beta \frac{x_\tau}{1 - x_\tau^n} - \gamma x.$$

Here, x_τ represents the value of x at time $t-\tau$. Its behaviour is determined by four parameters, which we set to standard values: $\beta = .2$, $\gamma = .1$,$n = 10$. The fourth parameter τ (the delay in the feedback loop) can be used to tune the dynamics of the system from mild to moderate chaos (see Figure 2.12). The values of the timeseries are generated through Euler integration of the differential equation with stepsize 1.

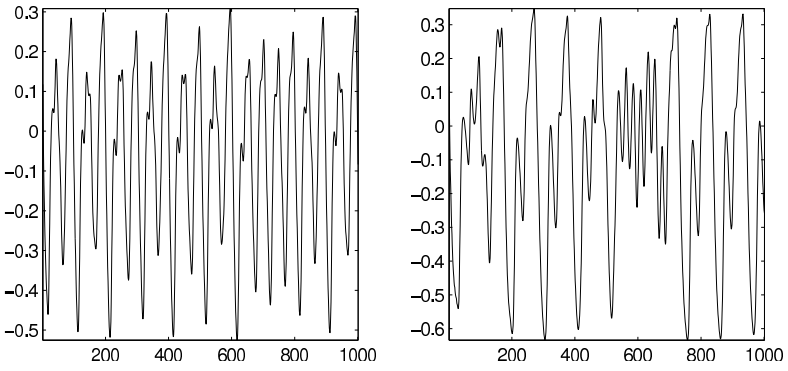


(a) Input signal: random switching between a noisy sawtooth and square signal.



(b) Output signal : signal template class.

Figure 2.11: Input signal (a) and corresponding desired output signal (b) for the signal classification task.



(a) $\tau = 17$ **(b)** $\tau = 30$

Figure 2.12: The Mackey Glass timeseries for two different values of the delay parameter τ.

The training for signal generation tasks is done by first feeding the teacher signal into the reservoir, and training the readout to do one-step ahead prediction of this teacher signal. This process is called *teacher forcing*. Once the readout is trained, the system can be run in signal generation mode, by feeding the predicted signal instead of the teacher signal back into the reservoir. Since the system was trained to do one-step ahead prediction of this signal, it will continue to generate this signal.

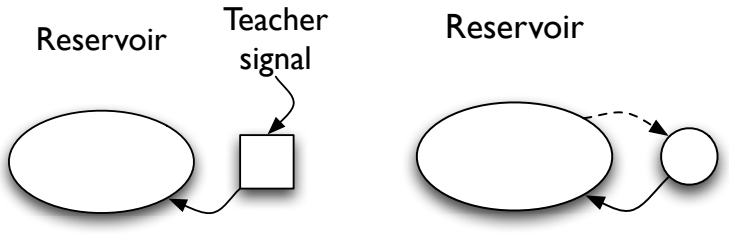
For signal generation tasks like this one the RC system generates its own input and there is no external signal that drives the reservoir. Because of this, any errors that are made in the signal prediction can possibly accumulate and cause the output to drift ever further from the original signal. This drifting can occur in three ways: the signal can die out because the gain of the feedback loop is not high enough, the output signal can diverge from the desired teacher signal but still remain qualitatively correct or the generated signal can become unstable and eventually run away to very large values.

The evaluation of performance for signal generation tasks is not trivial and depends on the task. For applications where the prediction horizon[8] is short, usually a high precision of the predicted signal is a priority. In this case, the NRMSE defined above is a suitable error measure. For long term prediction applications on the other hand, the stability of the predicted signal is more important than the precision, so here the deviation timestep error (the first timestep where the deviation between the generated and required signal reaches a certain value) is more important. Finally, for some applications the goal is to generate qualitatively similar signals to the teacher signal. An example of this is Central Pattern Generator (CPG) tasks: the objective is to generate complex periodic signals robustly, but slight phase or frequency deviations from the teacher signals can be tolerated. In this case, instantaneous error measures such as the NRMSE do not capture the real requirements of the problem, and more sophisticated error measures should be used based on methods that try to fit the generated signal to the teacher signal, for example with matching pursuit algorithms (Mallat and Zhang, 1993).

2.2.2 Spoken digit recognition with a Liquid State Machine

In (Hopfield and Brody, 2000) J. J. Hopfield and C. D. Brody describe the results of an experiment they conducted on the speech cortex of a fictional species of mouse, the *Mus Silicium*. In reality they constructed

[8] This is the time interval that the system is expected to predict autonomously



(a) Training mode: teacher forcing

(b) Testing mode: autonomous signal generation

Figure 2.13: Schematic image of training (a) and testing (b) for signal generation tasks. The dashed connection is trained.

a type of spiking neural network and trained it to perform a simple speech recognition task: the identification of the digit *one* out of other spoken digits. The dataset[9] they used is a subset of the TI46 speech corpus (Doddington and Schalk, 1981), and consists of ten digits, zero to nine, each uttered ten times by five different female speakers, resulting in 500 speech fragments sampled at 12 kHz.

In this section, we will use Liquid State Machine-type RC systems (i.e., reservoirs built with spiking neurons) to tackle this recognition task. The majority of the existing research on automated speech recognition (ASR) uses so-called Hidden Markov Models (HMMs), where the speech signal is modeled as a Markov chain. Markov chains are statistical models that are mathematically well understood and use relatively simple assumptions. The task of the ASR system is then to learn the structure of the underlying Markov chain, based only on indirect observations of the underlying state - hence the term 'hidden' in the name HMM.

The spiking reservoirs used here were built from Leaky Integrate and Fire (LIF) neurons, with a refractory period of 10 ms and a membrane time constant of 120 ms. This is biologically not very realistic (a more realistic value would be 20 ms), but it does give the neurons enough internal memory to solve this task. The reservoir contains no synapse model, which means that emitted spikes are transmitted directly to the receiving neurons. Ten different linear classifiers are trained, each sensitive to a different digit in the vocabulary. These classifiers are trained to output the correct value at each timestep (+1 for the correct class, −1 for the other classes). During testing a final classification is reached by taking the temporal mean of the output of every classifier, and applying winner-take-all

[9] Available for download at the website for the RC Toolbox, `http://snn.elis.ugent.be/rctoolbox`.

to the resulting vector. The performance for the speech recognition task is expressed as a Word Error Rate (WER): the fraction of incorrectly classified words as a percentage of the total number of presented words: $WER = 100 \cdot \frac{N_{nc}}{N_{tot}}$, with N_{nc} the number of incorrectly classified samples, and N_{tot} the total number of samples presented.

In (Schrauwen et al., 2008a), it was shown that for neurons with a binary (on-off) output, the two crucial parameters are the number of incoming connections each neuron has (the fan-in or in-degree) and the global weight scaling. In the same article it was shown that this effect is no longer present in analog reservoirs. A simple synthetic task was used where the delayed XOR-function of two random bit series was trained. In Section 2.2.2.1 below, we will do a sweep of the same parameters, but for a real-world engineering task of digit recognition. Additionally, we will compare the results of different preprocessing techniques for the raw speech files. For all the experiments in this section, a reservoir of 100 neurons was used. This is relatively small, but sufficient to allow for a fair comparison between the different speech preprocessing methods. All parameter points were evaluated with 25 different reservoir instantiations and the reported errors are averaged over these instantiations.

2.2.2.1 Preprocessing

It is common practice in the field of speech recognition to transform the sound to enhance the speech-specific features before applying a classification algorithm. The majority of the ASR systems use MFCC (Mel Frequency Cepstral Coefficients), which we will discuss in detail. In addition, we will also investigate the performance of more biologically realistic preprocessing models, which are all based on models of the first stages of the human auditory processing system (specifically the cochlea). These models vary in complexity and realism. The result of this preprocessing step is either directly fed into the reservoir in the case of non-spiking nodes, or else transformed into spike trains using a method called BSA (Ben's Spiker Algorithm) (Schrauwen and Van Campenhout, 2003). This algorithm takes a temporal signal and a filter as input, and yields the spiketrain that allows optimal reconstruction of the original signal when decoded using this filter. The algorithm works by scanning across the signal, and at each timestep computing a heuristic error measure. When this error measure exceeds a threshold, a spike is generated and the filter impulse response is subtracted from the signal. In this case, the filter is exponentially decaying with a time constant of 30 sample timesteps. For a spiking reservoir (a LSM), the spike trains generated by the reservoir are transformed back into the analog domain by filtering them with

an exponential filter, which mimicks the internal operation of the membrane potential of a simple spiking neuron. These analog signals are then subsampled by a factor of 20 before being fed to the readout.

PATTERSON-HOLDSWORTH-MEDDIS MODEL This model consists (as all cochlear models under consideration here) of a filter bank introduced in (Patterson et al., 1992), which models the selectivity of the human ear to certain frequencies (shown in 2.14) followed by a stage that models the nonlinear response of the hair cells in the cochlea (Meddis, 1986). The filter bank in this case consists of logarithmically spaced gammatone filters. The impulse response of these filters is a sinusoid modulated by a gamma distribution function - hence the name. The center frequencies of this filter bank were determined through psycho-acoustic experiments, where the detection threshold of a pure sine tone masked by a block of noise was determined. Based on this, the sensitivity to certain frequencies was determined.

An example set of input spike trains generated by this ear model is shown in 2.14, along with the performance results when varying the node fan-in and weight scaling as described above. The overal performance is not very good, with a best result of 45.2%.

SENEFF COCHLEAR MODEL The Seneff cochlear model (Seneff, 1988) is again a biologically realistic model of the human auditory system. It converts a sound stimulus into a series of waveshapes that represent the firing probabilities of the different hair cells in the human cochlea. It consists of two stages: a critical filter bank that performs an initial spectral analysis based on psycho-acoustic properties of the human ear. This stage is followed by a model of the nonlinear transduction stage implemented by the hair cells, and which models saturation, adaptation and forward masking. The parameters of the model were fitted to experimental data.

An example set of spike trains, the filter bank used by this model and the performance results are plotted in Figure 2.15. This front end performs considerably better than the Meddis model, with a smallest error of 18.15%. This is, however, still not good enough to be of practical use.

LYON PASSIVE EAR MODEL The Lyon Passive Ear model (Lyon, 1982) is a model of the human inner ear or cochlea, which describes the way acoustic energy is transformed and converted to neural representations. The model consists of a filter bank that mimics the selectivity of the human ear to certain frequencies, followed by a series of half-wave rectifiers (HWRs) and adaptive gain controllers (AGCs) both modeling the hair cell response. Figure 2.16 shows a schematic representation of the different steps of the model. We note that the Lyon model is relatively simple

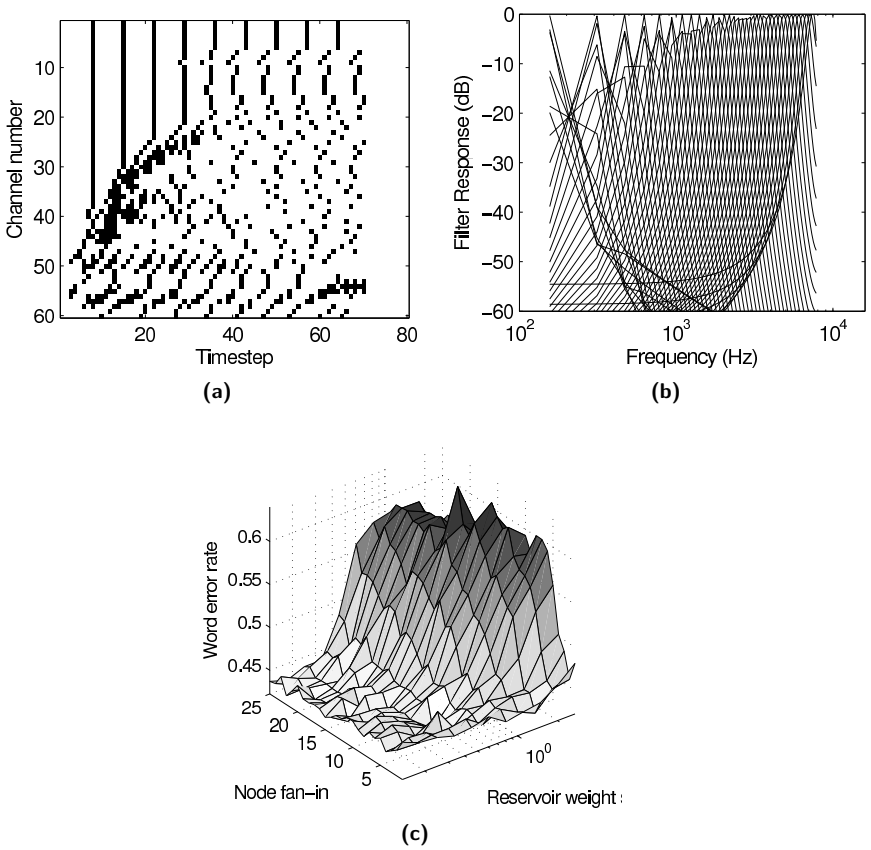


Figure 2.14: Illustration of the spike trains generated by the Meddis ear model (a). The filter bank used in the Meddis ear model (b). Experimental results for this preprocessing method (c).

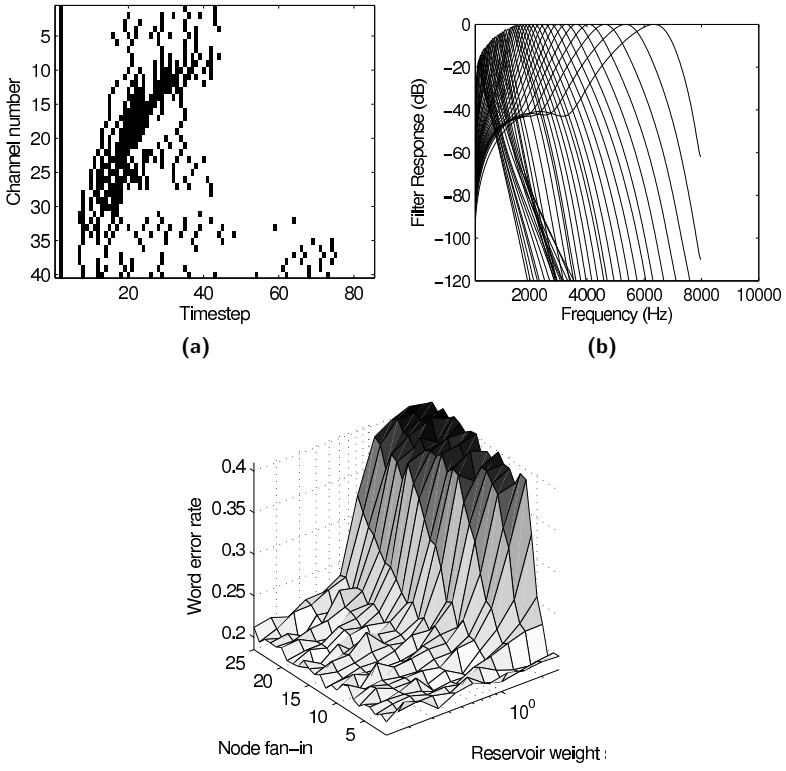


Figure 2.15: Illustration of the spike trains generated by the Seneff cochlear model (a). The filter bank used in the Seneff ear model (b). Experimental results for this preprocessing method (c).

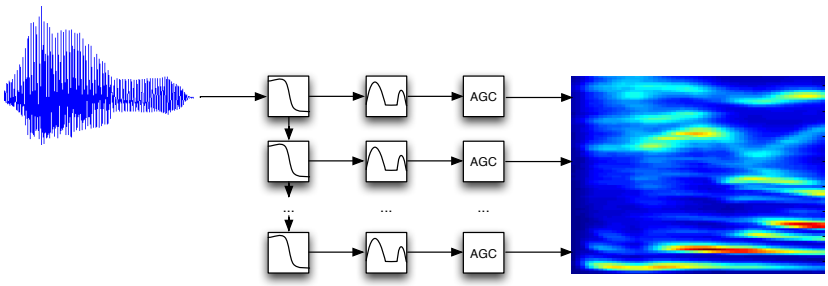


Figure 2.16: Schematic overview of the elements of the Lyon cochlear model: a cascading filter bank followed by half-wave rectifiers and adaptive gain controllers (AGC).

and that more complicated ear models also exist (Van Immerseel and Martens, 1993). Moreover, this form of preprocessing is computationally more intensive than the use of a MFCC front end, taking about three to five times as long to compute on a conventional processor.

The full time-sequence of the outputs of the last stage define a so-called *cochleagram* (Figure 2.17), a collection of coefficients which indicate a firing probability of a cochlear hair cell in response to the sound. The performance using this biologically realistic coding is shown in Figure 2.17. It appears that this preprocessing method is a good match for the recognition capabilities of the LSM. The smallest error is 8.6%.

MFCC MFCC stands for Mel-Frequency Cepstral Coefficients (Davis and Mermelstein, 1980). MFCC is the de facto standard technique for preprocessing speech before feeding it to a recognition system. The coefficients are calculated as follows:

- The sample data is windowed using a hamming window.
- A FFT is computed.
- Its magnitude is run through a so-called mel-scale[10] filter bank.
- The log_{10} of these values is computed.
- A cosine transform is applied to reduce the dimensionality and to enhance the speech-specific features of the input.

The result is the so-called *cepstrum*.

[10] A mel-scale is a non-linear transformation of the frequency domain to model the human selectivity to certain frequency bands.

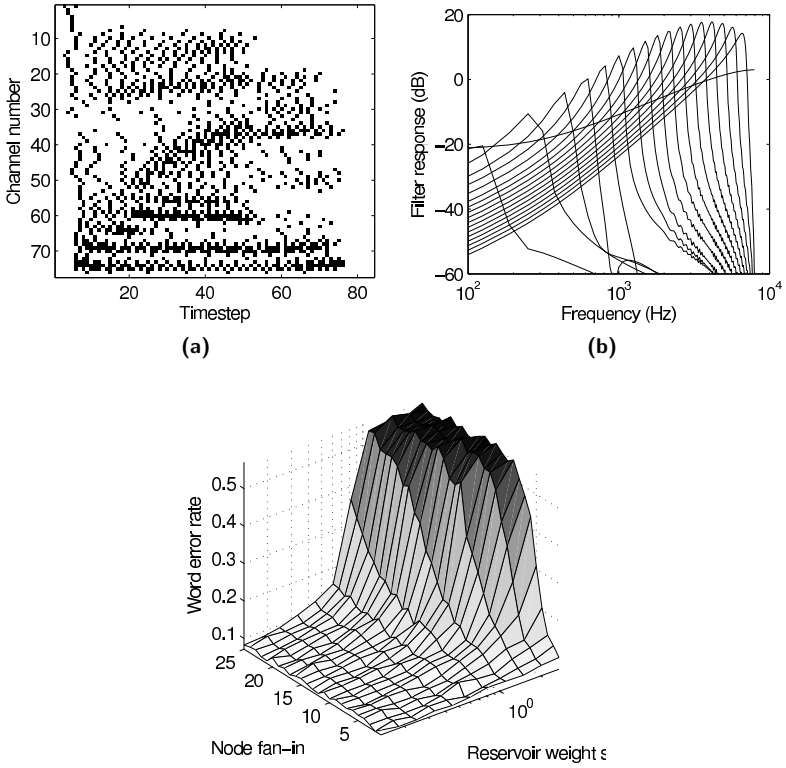


Figure 2.17: Illustration of the spike trains generated by the Lyon cochlear model (a). The filter bank used in the Lyon ear model (b). Experimental results for this preprocessing method (c).

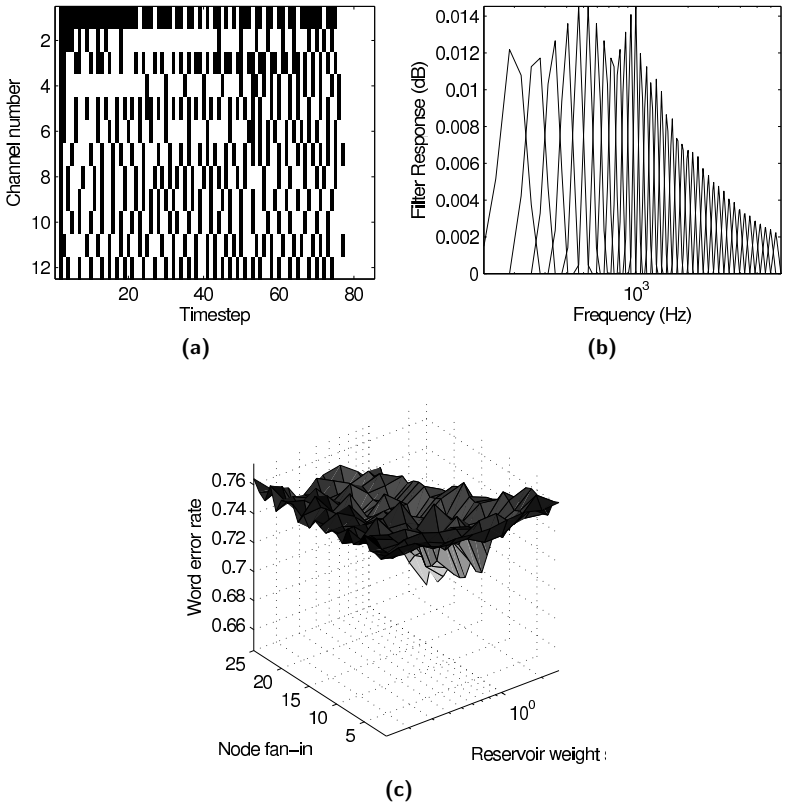


Figure 2.18: Illustration of the spike trains generated by the MFCC preprocessing frontend (a). The mel-scale filter bank used in MFCC (b). Experimental results for this preprocessing method (c).

In addition to the thirteen cepstral coefficients, the approximated first and second order time-derivatives of these coefficients (the so-called delta and delta-delta coefficients) were calculated - a common practice for speech processing. This means that in total 39 spike trains are fed into the liquid.

Best results for all models In table 2.2, I summarize the best performances of each of the four speech front-ends we discussed. Clearly, the Lyon model is the best match for this recognition architecture. This is also the model that will be used in the remained of this thesis. Surprisingly, the MFCC front-end does not combine well at all with our recognition setup. This is probably due to the fact that MFCC tries to incorporate temporal information into a single feature vector since it was explicitly designed for use with HMMs (which have no fading memory).

Table 2.2: Optimal results for the four speech front-ends we considered.

	Meddis	Seneff	Lyon	MFCC
Optimal WER	45.2%	18.15%	**8.6%**	64.5%

The differences between the other, biologically inspired speech front-ends are likely due to the different filter banks.

2.2.2.2 Noisy inputs

In (Verstraeten et al., 2005), I studied the robustness of the reservoir to noisy environments. Specifically, I added noise from the NOISEX[11] database to the test set. Different types of noise were added with different Signal to Noise Ratios (SNRs): speech babble (B), white-noise (W) and car interior noise (C) with SNR levels of 30, 20 and 10 dB. For these experiments we used a larger reservoir of 1232 LIF neurons. The data was preprocessed using Lyon's cochlear model.

For comparison, we also give the best results from (Deng et al., 2004), where a specific speech front end (Log Auditory Model or LAM) designed for noise robustness is followed by an HMM. We note that in this case, the dataset consisted of isolated digits from the TIDIGITS database, which is not identical but nonetheless comparable to the one we used. The results shown in Table 2.3 - expressed as recognition scores for easy comparison - are thus indicative and not quantitative.

Table 2.3: The robustness of the LSM against different types of noise.

		Clean	30 dB	20 dB	10 dB
C	**LSM**	**97.5%**	**91.5%**	**89.5%**	**88.5%**
	LAM	98.8%	98.6%	98.8%	98.6%
B	**LSM**	"	**94.5%**	**93.5%**	**89%**
	LAM	"	98.4%	93.2%	72.5%
W	**LSM**	"	**85%**	**84%**	**79.5%**
	LAM	"	98.4%	95.7%	72.7%

It appears that the LSM is very robust to different types of noise being added to the inputs. The HMM with the noise-robust front end performs better for low noise levels and in case of car noise, but the general decay

[11] Available online at *http://spib.rice.edu/spib/select_noise.html*.

in performance with regard to the noise level is more gradual for the LSM for the babble and white noise.

2.2.2.3 Comparison with the state of the art

In this subsection I present two relevant speech recognition systems as a comparison for our presented technique. Sphinx4 is a recent speech recognition system developed by Sun Microsystems (Walker et al., 2004), using HMMs and an MFCC front end. When it is applied to the TI46 database, a word error rate (WER) of 0.168% is achieved. The best RC system from our experiments with a reservoir size of 1232 neurons achieved a WER of 0.5% (Verstraeten et al., 2005). While slightly worse than the state-of-the-art, we point out that the LSM offers a number of advantages over HMMs. HMMs tend to be sensitive to noisy inputs, are usually biased towards a certain speech database, do not offer a way to perform additional tasks (like speaker identification (Verstraeten, 2004) or word separation) on the same inputs without a substantial increase in computational requirements and are originally designed to process discretely sampled discrete data while a lot of real-world information is continuous in nature (in time as well as magnitude).

An additional comparison can be made by looking at the results described in (Graves et al., 2004). There, a recurrent SNN is used with so-called Long Short-Term Memory (LSTM). It is trained for exactly the same dataset as used in this chapter. A WER of 2% was achieved. We therefore conclude that RC passes the test of isolated digit recognition very well and rivals the performance of standard HMM based techniques and other kinds of SNN solutions.

2.2.3 The Ford dataset competition

The next real world problem I will discuss is also a temporal signal classification task. At the 2008 World Conference on Computational Intelligence (WCCI), a series of machine learning competitions was organised, ranging from AI in games to robot car racing. One of the competitions was organized in collaboration with the Ford Motor Company, and I participated with an RC system. The problem is set in an automotive industrial context. It consists of classifying short, one-dimensional timeseries of 500 timesteps each. The short timeseries are said to be indicative of the maximum time window on which a classification decision typically needed to be made. There are only two classes, one where a certain (unknown) symptom is present and another class that is symptom-free. No additional information is given about the nature of the signals or the symptoms

that occurred, or about how these signals were measured (e.g., the sample rate), but presumably it is an audio measurement of some industrial automotive process. Also, visual inspection shows no discernable difference between positive and negative examples (see Figure 2.19). For this problem the lack of information that is usually available in other cases - and that can be used by the designer of the system to make some initial choices for e.g. feature selection - presents an additional challenge.

For the competition, two different datasets are presented, A and B. Dataset A consists of a clean training and test set and set B consists of a clean training but noisy test set. obviously, while the competition was running the test sets were not made public. There is also no indication about the nature of the noise in the B dataset. In total there are 3601 examples available in the A training and validation sets (whose labels were publicly available), and 1320 examples in the test set. For the B dataset, the training and validation sets contain 3636 examples and the test set consists of 810 samples. It is noted by the organizers that the two datasets are unrelated and should not be mixed. The competition rules allow two different classifiers to be trained/applied to the two datasets, but the principal approach has to be the same for both cases.

The problem presented here - classification of finite timeseries - is one that is traditionally solved by extracting feature vectors that capture the important temporal characteristics as accurately as possible, where common domains for these feature vectors include the frequency domain or the wavelet domain. Next, one of (very) many possible stationary classifiers is applied either to a feature vector at every timestep or to the ensemble of feature vectors (which would be possible in this case given the short length of the timeseries) and a decision can be made. For this competition, however, we use RC which is a fundamentally different approach because of the way time is handled: the reservoir is a dynamical system that runs along in the same time domain as the input signal, and the classification decision is only taken at the very end of the sample. This is done in a similar way to the speech recognition task described earlier: the mean of the classifier output is taken over the whole sample, and a threshold function is applied which gives the final classification answer.

2.2.3.1 Experimental setup

The requirements of the task do not ask for a substantial change to the classical RC setup. However, due to the similarity of the signals and the apparent absence of discerning characteristics in the time domain between the positive and negative examples, feeding the signals directly into the reservoir without any form of preprocessing is not desirable. This

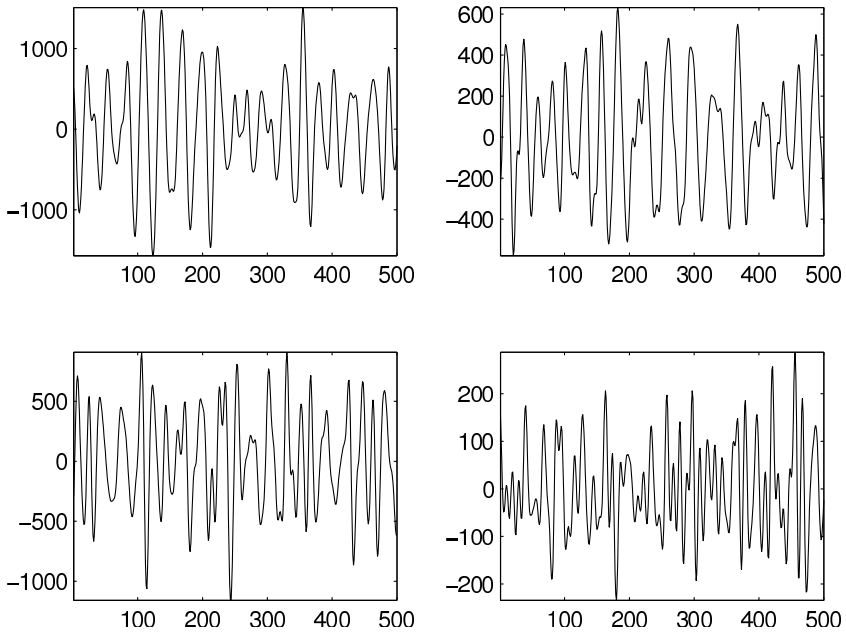


Figure 2.19: Example timeseries from the clean (top row) and noisy (bottom row) Ford dataset. On the left, positive examples are shown (with the symptom), on the right symptomless examples are plotted.

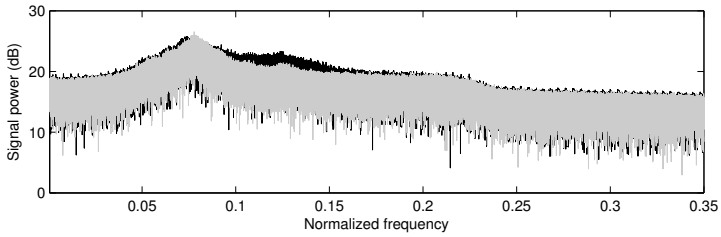
is confirmed by some of our preliminary experiments where the signal is fed into the reservoir directly: the classification accuracy is only a couple of percent higher than the 50% baseline achieved by simple guessing.

Due to the pseudo-periodic nature of the input signals, a natural next step is to look at the image of these timeseries in the frequency domain, to see if there is any discernable difference between positive and negative examples. To do this, we simply plot the periodogram of a concatenation of all positive and negative examples. As can be seen in Figure 2.20, there is no really dramatic difference in the spectral image of the positive versus the negative examples, but there is at least some small distinction possible. In particular: the maximal spectral power lies at slightly different frequencies for positive and negative examples - approximately 0.072 and 0.078 respectively, and for frequencies higher than the maximum-power frequency, there is a region where the positive examples contain considerably more power than the negative ones, in the normalized frequency region [0.09 .2].

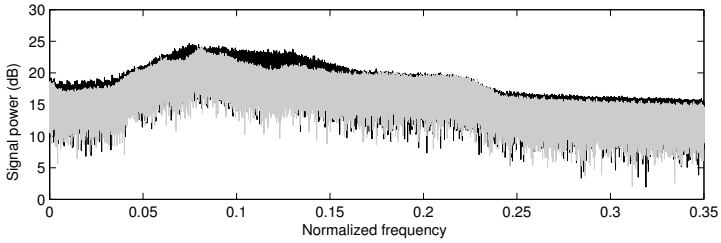
Motivated by the good results in the speech recognition task (which is in some ways quite similar to this) we have constructed a feature extraction scheme based on a filter bank decomposition. When decomposing a signal into different frequency bands, one has the choice to either apply a FFT transform and work on the spectrographic image of the signal to extract the necessary features, or to directly apply filters to the signal to get the features. Here, we chose the latter option because this gives more accurate control over the filter characteristics, at the expense of a higher computational cost.

We normalize the samples to a maximal absolute amplitude of 1 and apply a bank of second-order butterworth-type bandpass filters, resulting in a number of filtered signal channels. Then, half-wave rectification is applied followed by a downsampling and a transformation to a dB-scale[12]. The downsampling rate was set to 5, a value which was chosen through experimental optimization. Similarly, the values for the center and cutoff-frequencies were chosen initially by studying the relevant differences between the periodograms of the positive and negative examples, and later fine-tuned through experimentation. In total, we selected 14 bandpass filters, whose frequency responses are shown in Figure 2.20c. Notice how the pass-bands coincide with the main areas of difference between the positive and negative examples in Figure 2.20. An example of the feature vectors that are produced by this filter bank preprocessor is shown in Figure 2.21.

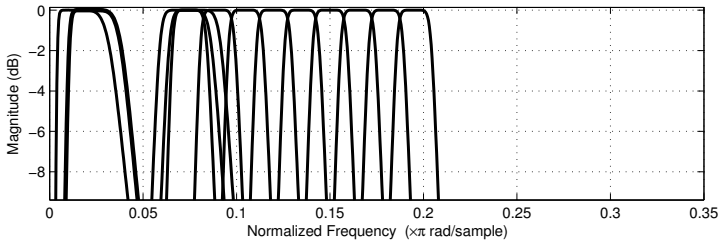
[12]Note the similarity with the cochlear ear model of the speech recognition task - the AGC is not used here because the amplitude of the signals is quite constant.



(a) Clean dataset



(b) Noisy dataset



(c) Frequency characteristics of the bandpass filters used in the preprocessing of the Ford signals.

Figure 2.20: Periodogram of all positive (black) and negative (grey) examples from the clean (a) and noisy (b) dataset. Frequency characteristics of the bandpass filters used for preprocessing are shown in plot (c). Note that the bandpass filters coincide with the main frequency regions where the positive and negative examples differ.

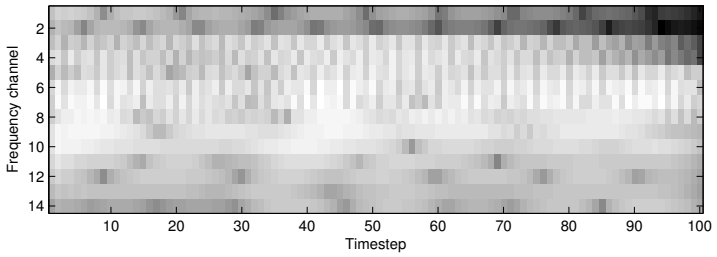


Figure 2.21: Example of the feature vectors extracted from the signals.

2.2.3.2 Results of the competition

The classifiers in the challenge were evaluated primarily based on the accuracy : $\frac{N_{tp}+N_{tn}}{N_{tot}}$, with N_{tp} the number of true positives, N_{tn} the number of true negatives and N_{tot} the total number of samples. In case of a tie, the false positive rate was used as a tie-breaker. Before the competition, it was obviously difficult to assess the performance of the method on the complete dataset. For testing purposes, we reserved a part of the available data for testing, but to construct the final classifier we trained our RC system on all the available data. The final rankings were made public at the conference itself, and according to the organisers of the competition a wide variety of techniques were submitted including neural networks, support vector machines and ensemble methods. A ranking list of the first twenty contestants was presented, for the A dataset, the B dataset and for a combined result. The best two methods for the A dataset (scoring an accuracy of 100% and 99.6%) ranked 11th and 18th respectively on the B dataset (scoring 68.1% and 59.3%), and the best two methods for the B dataset (scoring 86.2% and 84.3%) ranked 9th and 10th on the A dataset (94.9% and 94.5%). It appears therefore that the winners in the A and B subcompetitions were rather specifically tuned (either on purpose or by accident) for either the noisy or the non-noisy data.

Our proposed RC-based method ranked 12th on the A dataset, with an accuracy of 92.5%, and 7th on the B dataset with an accuracy of 82.5%. On both datasets together, our method attained a 9th place, with an accuracy of 88.7% - compare this with the winner on both datasets who attained 92.2%, so 3.5% better. In conclusion, we have shown that with only a very limited amount of time (only a couple of days work, vs. several personweeks for other competitors[13]) we were able to construct a classification method based on RC that achieves competitive results on a difficult real-world signal classification task.

[13] This was communicated during talks at the competition session at the conference.

2.2.4 Epilepsy detection

Recently, a line of research was started at our lab that focuses on a very concrete problem in a biomedical context, namely the detection of epileptic seizures. This problem is formulated in a very broad way: for instance the sensor signals on the basis of which the detection is done will be varied in nature, ranging from Electro-EncephaloGraphic (EEG) measurements over blood pressure to motion and accelerometry sensors. So far, two different seizure types were considered: Spike Wave Discharges (SWD) and tonic-clonic seizures. The former type is characterised as an 'absence' or 'petit mal' seizure, meaning that the subject is temporarily unresponsive to external stimuli, while the latter used to be called a 'grand mal' seizure and consists of the tonic phase - where the subject tenses up - and the clonic phase - where the subject exhibits jerking movements and convulsions. Both seizure types are potentially dangerous and an early detection is therefore highly desirable. Small animal models (rats) of both seizure types are available and close collaboration with the University Hospital of Ghent has provided us with access to extensive datasets.

In (Buteneers et al., 2009), the initial promising results of the application of RC to the problem of seizure detection on intracranially (inside the scalp) measured EEG data in rats are presented. Two evaluation criteria are used to assess the performance: classification speed and accuracy. The former is measured as the delay between the start of the actual seizure and the moment the RC system first detects the seizure correctly. The latter is slightly more intricate: the readout function of the RC system generates a continuous output at every timestep, and a threshold is applied to this continuous value to determine the actual classification (seizure/no seizure). This threshold determines the trade-off between false positives and false negatives: in case a high value indicates a seizure, a high treshold will minimize the false positives but reduce the fraction of true positive, and vice versa. Therefore, usually for binary detection systems the threshold is varied across the whole range, and the sensitivity[14] is plotted versus the specificity[15], resulting in a so-called Receiver Operating Characteristic (ROC) curve. The area under this curve (AUC) is then a good indication of the global performance of the detection system, regardless of the value of the threshold. For real world implementations in actual detection systems, the actual value of the threshold is usually determined based on the requirements of the problem, which is more costly: a false positive or a false negative response.

[14] $TP/(TP + FN)$, with TP the number of true positives and FN the number of false negatives

[15] $TN/(TN + FP)$, with TN the number of true negatives and FP the number of false positives

The reservoir used (Buteneers et al., 2009) uses leaky integrator nodes (see Section 3.1), where the leak rate was optimized experimentally. The AUC of the RC setup was compared with six other state-of-the-art methods for seizure detection, and RC improved on the performance of all other methods with an AUC of 0.987 for the SWD seizures and 0.993 for tonic-clonic seizures. However, recent (unpublished) results indicate that a more elaborate tuning of the method by Van Hese et al (Van Hese et al., 2003) can achieve similar results as the RC setup. Regarding classification speed, it was found that the RC system detects the seizure with a delay of on average 0.3 seconds for the SWD seizures and 3 seconds for the tonic-clonic seizures, again outperforming the other methods - although it should be noted that the other methods are not explicitly designed for quick online detection.

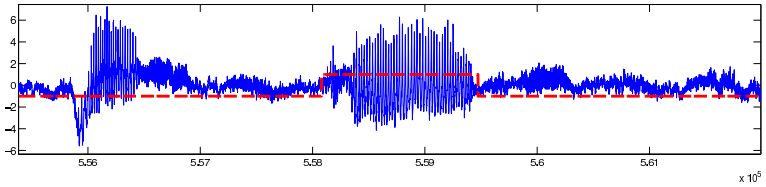


Figure 2.22: Figure showing a single channel of an intracranial EEG recording in a rat. The dashed line indicates where an SWD seizure occurs.

3

Towards generic Reservoir Computing: time scales and novel reservoirs

The previous chapter was focused on neural implementations of Reservoir Computing, using either analog sigmoidal or spiking neurons. In this chapter, the initial steps in the transition towards more general reservoirs will be taken.

As mentioned before, the idea of Reservoir Computing can be extended from recurrent neural networks to other nonlinear dynamical networks and even actual physical media. To make this transition possible, the first issue that needs to be addressed is the representation of time. I will tackle this topic by starting from a continuous time description of the system and evaluating what the effects of discretizing time are in the input, reservoir and output time domain. These effects will be linked to the memory the system has. Next, I will present and discuss three reservoir types that were studied by me or in collaboration with other research groups. The implementations that are discussed here will drift gradually further away from the original "neural" ESN or LSM incarnations: we will study reservoirs made of bandpass neurons, Cellular Nonlinear Networks and finally nonlinear photonic components.

3.1 Time scales and memory

So far, the RC setups that were discussed have been used in discrete time. Any continuous input signals (such as speech) were tacitly assumed to be sampled. However, if we want to apply the Reservoir Computing framework to more general reservoir types, such as actual physical systems, one of the questions that surfaces is how to deal with time, and more specifically different time scales. Moreover, even for standard, discrete-time reservoirs I will show that it can be useful to adjust the time scales of the

different components (input signals, reservoir states and output signals) of the system.

3.1.1 Three different time steps

Since we will only consider simulations of reservoirs here, we will still use discretized time. However, the signals at the input, inside the reservoir and at the output can be sampled at different sample rates. For the systems we will consider here, we can discern three different time domains, each with its own time step:

- the input time step, indicated with δ_1
- the reservoir time step, indicated with δ_2, and
- the output time step, indicated with δ_3.

Until now I have implicitly assumed that these time scales are identical, but they do not need to be. In fact, it will be shown that tuning these time scales can have a quite substantial effect on performance.

Because there is no absolute notion of time scale in these systems, they are only defined relative to each other. Thus, two possible relations can be investigated: the relation between δ_1 and δ_2 when going from the input time domain to the reservoir time domain, and the relation between δ_2 and δ_3 when going from the reservoir time domain to the output time domain. In the following sections, we will discuss these timescales and the transitions between them in more detail. While we will focus exclusively on downsampling in the transitions between the input, reservoir and output time domains, it may be useful to do upsampling for some tasks. For instance, in case a very rapid response is needed from the reservoir, increasing the sample rate of the input can help reduce the time needed for the reservoir to 'warm up' at the expense of a shorter memory.

3.1.1.1 The reservoir timescale δ_2: leaky integrator nodes

The first possibility to tune time scales in an RC system is located in the transition between the input and the reservoir. One way to do this is to consider first the following differential equation that describes a reservoir in continuous time with activation function f, state vector $\mathbf{x}$ and input signal $\mathbf{u}$:

$$\dot{\mathbf{x}} = \frac{1}{c}\left(-a\mathbf{x} + f\left(\mathbf{W}_{\text{in}}\mathbf{u} + \mathbf{W}_{\text{res}}\mathbf{x}\right)\right),$$

where a denotes the leaking rate of the neuron (the rate at which the activation decreases with time) and c is a scaling factor for the temporal dynamics. The discretization of this equation using the Euler method with a timestep δ_2 gives:

$$\mathbf{x}((t+1)\delta_2) = (1 - \frac{a\delta_2}{c})\mathbf{x}(t\delta_2) + \frac{\delta_2}{c} f\left(\mathbf{W}_{\text{in}}\mathbf{u}(t\delta_2) + \mathbf{W}_{\text{res}}\mathbf{x}(t\delta_2)\right),$$

or, when we go over to discrete time, using the notation $\mathbf{x}((t+1)\delta_2) = \mathbf{x}[k+1]$:

$$\mathbf{x}[k+1] = (1 - \frac{a\delta_2}{c})\mathbf{x}[k] + \frac{\delta_2}{c} f\left(\mathbf{W}_{\text{in}}\mathbf{u}[k] + \mathbf{W}_{\text{res}}\mathbf{x}[k]\right).$$

It is shown in (Jaeger et al., 2007) that the term $\frac{\delta_2}{c}$ can be lumped without loss of generality into a common factor λ called the *leak rate*. Although the retainment rate a was the major research topic of (Jaeger et al., 2007) and λ was ignored, the focus will now be on λ and I will set the retainment rate a to 1. The reason for this is the fact that the parameter λ was previously not thoroughly investigated, and changing λ does not affect the effective spectral radius of the reservoir while changing a does. This fact was first noted in (Schrauwen et al., 2007a) and later also in (Siewert and Wustlich, 2007). This coupling between time-scale settings and dynamical regime of the reservoir is not desired because one would like to be able to tune both separately and independently.

The assumptions introduced above ultimately yield:

$$\mathbf{x}[k+1] = (1 - \lambda)\mathbf{x}[k] + \lambda f\left(\mathbf{W}_{\text{in}}\mathbf{u}[k] + \mathbf{W}_{\text{res}}\mathbf{x}[k]\right).$$

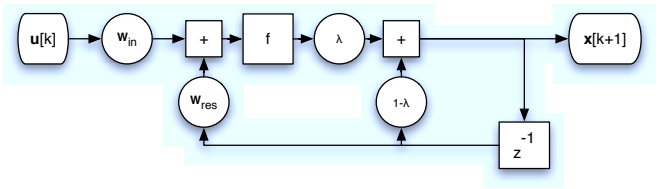
Closer inspection of this equation reveals that this is in fact a form of first-order recursive low-pass filter (an infinite impulse response or IIR filter). A general expression for this type of first order filter in discrete time is:

$$\mathbf{x}[k+1] = (1 - \lambda)\mathbf{x}[k] + \lambda\mathbf{u}[k+1].$$

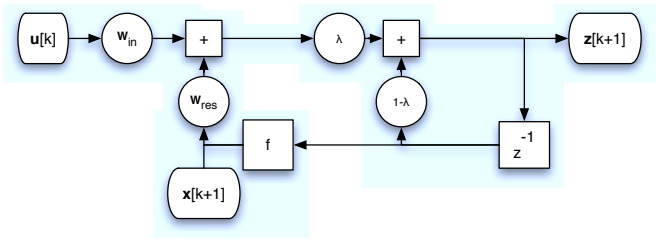
Comparison of this expression with the one above shows that the former is in fact a regular neuron with the integration mechanism placed behind the nonlinearity (shown schematically in Figure 3.1a). Jaeger introduced this model as leaky integrator nodes in (Jaeger, 2002), but for easy comparison with the other models under consideration here, I call this model 1.

This equivalence between leaky integrator neurons and low-pass filtered regular neurons introduces an interesting interpretation. Filter theory tells us that the cut-off frequency of such a filter[1] is given by:

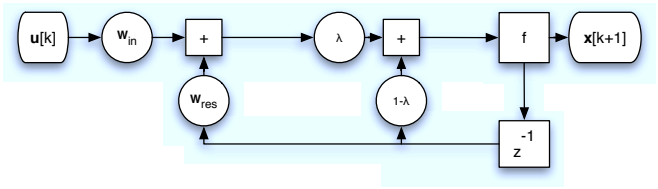
[1] The cut-off frequency is the frequency where the power of the filtered signal is 1/2



(a) Model 1: integrator behind the nonlinearity



(b) Model 2: integrator in front of the nonlinearity



(c) Model 3: integrator across the nonlinearity

Figure 3.1: Diagrams of the three ways of constructing leaky integrator nodes. Variables are represented in rounded boxes, multiplication factors in circles and operators are shown in squares. The z^{-1} operator represents a delay of one time step.

$f_c = \lambda/2\pi$. Thus, by tuning the time constant λ, one has an easy and intuitive way to filter out any frequencies of the signal transmitted by the neuron that are irrelevant to the problem. Such frequency components could be due to oscillators created inside the reservoir, or because these frequencies simply contain noise.

Model 1 is not the only way an integrator can be added to a neuron. One can also add the integrator in front of nonlinearity (shown in Figure 3.1b), which is commonly done when discretizing continuous time RNN (CTRNN), as is done in case of BackPropagation Decorrelation (BPDC)

of the power of the filtered signal at a frequency in the pass-band.

learning of reservoirs (Steil, 2004, 2005b). This yields the following model:

$$\begin{aligned}\mathbf{z}[k+1] &= (1-\lambda)\mathbf{z}[k] + \lambda\left(\mathbf{W}_{\text{in}}\mathbf{u}[k] + \mathbf{W}_{\text{res}}\mathbf{x}[k]\right)\\ \mathbf{x}[k+1] &= f(\mathbf{z}[k+1])\end{aligned}$$

I call this model 2.

Finally, one can place the integrator across the activation function, as shown in Figure 3.1c. This model was introduced in (Schrauwen et al., 2007a). The state update equation now reads:

$$\mathbf{x}[k+1] = f\left((1-\lambda)\mathbf{x}[k] + \lambda\left(\mathbf{W}_{\text{in}}\mathbf{u}[k] + \mathbf{W}_{\text{res}}\mathbf{x}[k]\right)\right).$$

This is model 3. Model 3 can be rewritten using $\widetilde{\mathbf{W}}_{res} = (1-\lambda)\mathbf{I}+\lambda\mathbf{W}_{res}$ and $\widetilde{\mathbf{W}}_{in} = \lambda\mathbf{W}_{in}$ as:

$$\mathbf{x}[k+1] = f\left(\widetilde{\mathbf{W}}_{res}\mathbf{x}[k] + \widetilde{\mathbf{W}}_{in}\mathbf{u}[k]\right),$$

which shows that this is model is in fact equivalent with a standard reservoir where the off-diagonal elements of the reservoir matrix and the input matrix have been scaled down with a factor λ. Note that this changes the spectral radius, so the rescaling of the spectral radius of $\mathbf{W}_{res}$ should be done after the rescaling due to λ. The advantage of this model is that the integrator state can never 'run away' because it is bounded through the nonlinearity. An additional advantage is the absence of computational overhead since the integrators are integrated in $\mathbf{W}_{res}$. However, a drawback is that the integrator state does leak away even with $a = 1$. The leak is due to the contracting property of the non-linear mapping of the hyperbolic tangent upon itself. This has as a consequence that the overall amplitude of the reservoir dynamics scales down when λ goes to 0.

An interesting correspondence exists between model 1 and model 2. We will show that these models can be transformed into one another, provided the input is filtered and rescaled appropriately. We start with the state update equation of model 1:

$$\mathbf{x}[k+1] = (1-\lambda)\mathbf{x}[k] + \lambda f\left(\mathbf{W}_{in}\mathbf{u}[k] + \mathbf{W}_{res}\mathbf{x}[k]\right).$$

Multiplying both sides with $\mathbf{W}_{res}$ gives:

$$\mathbf{W}_{res}\mathbf{x}[k+1] = (1-\lambda)\mathbf{W}_{res}\mathbf{x}[k] + \lambda\mathbf{W}_{res}f\left(\mathbf{W}_{in}\mathbf{u}[k] + \mathbf{W}_{res}\mathbf{x}[k]\right).$$

We introduce a new state vector $\hat{\mathbf{x}}[k] = \mathbf{W}_{in}\mathbf{u}[k] + \mathbf{W}_{res}\mathbf{x}[k]$:

$$\hat{\mathbf{x}}[k+1] - \mathbf{W}_{in}\mathbf{u}[k+1] = (1-\lambda)\left(\hat{\mathbf{x}}[k+1] - \mathbf{W}_{in}\mathbf{u}[k+1]\right) + \lambda\mathbf{W}_{res}f\left(\hat{\mathbf{x}}[k]\right)$$
$$\hat{\mathbf{x}}[k+1] = (1-\lambda)\hat{\mathbf{x}}[k] + \lambda\mathbf{W}_{res}f\left(\hat{\mathbf{x}}[k]\right) + \mathbf{W}_{in}\left(\mathbf{u}[k+1] - (1-\lambda)\mathbf{u}[k]\right)$$

We introduce a rescaled and filtered input signal

$$\hat{\mathbf{u}}[k] = \frac{1}{\lambda}\left(\mathbf{u}[k+1] - (1-\lambda)\mathbf{u}[k]\right),$$

which ultimately yields

$$\hat{\mathbf{x}}[k+1] = (1-\lambda)\hat{\mathbf{x}}[k] + \lambda\mathbf{W}_{res}f\left(\hat{\mathbf{x}}[k]\right) + \lambda\mathbf{W}_{in}\hat{\mathbf{u}}[k].$$

This is precisely the state update equation of model 2. Note however that we have had to filter the input signal and that the state vectors between the two models are not equivalent (the state vector of model 2 includes the input!), so for all practical purposes model 1 and model 2 are not trivially interchangeable. Theoretically there are some strong and weak points concerning all three models, but we will have to experimentally investigate which of these is somehow 'optimal'.

3.1.1.2 The output time scale δ_3

In addition to the possible transformation between input and reservoir time scales δ_1 and δ_2 (or λ), there is also the possibility to transform the time scale when going from the reservoir to the output. We will therefore study the output time-scale δ_3, used as follows:

$$\widehat{\mathbf{y}}((t+1)\delta_3) = W_{\text{res}}\mathbf{x}((t+1)\delta_3) + W_{\text{inp}}\mathbf{u}(t\delta_3).$$

The reservoir states and inputs are thus intepreted in the output time domain before training and applying the linear readout. At a first glance this time scale does not seem to be very important, but as we will show in the experiments, changing this time-scale can have a drastic effect on performance.

3.1.1.3 Transitions between time domains through resampling

The transitions between the input and reservoir time domain on the one hand and the reservoir and output time domains on the other is done through resampling. In the former case, the resampling ratio is δ_1/δ_2, in

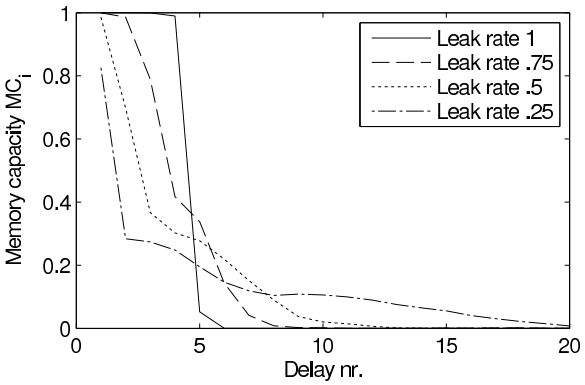


Figure 3.2: The memory curve for different leak rates.

the latter it is δ_2/δ_3. The process of resampling should ideally be done so that the Shannon-Nyquist sampling theorem is obeyed, i.e., that the highest frequency f_{max} of the signal before resampling is not higher than half the sampling rate f_s of the resampled signal (the so-called Nyquist frequency $f_s/2$). In this way, aliasing of the resampled signal is avoided. This is done by first applying a low-pass anti-aliasing filter to the signal with an appropriate cut-off frequency before doing the resampling. For the experiments described here, the Matlab `resample` function was used.

3.1.2 Node memory vs. reservoir memory

In the section above, I have mainly focused on interpreting the integration mechanism in the reservoir nodes as a low-pass filter. This is very useful when thinking about reservoirs in the frequency domain. However, another interpretation is possible and in some cases equally useful. When the leak rate of a reservoir is changed, in effect one changes the memory function (see Subsection 2.2.1.2) of the reservoir. This is shown in Figure 3.2. In this plot, the memory curve for a reservoir of 20 tanh nodes is shown with different leak rates. The plot shows that decreasing leak rates cause the memory curve to become flatter and wider. This means that the reservoir has worse memory of recent inputs, but better memory of inputs further in the past. Thus, by decreasing the leak rate we increase the longer-term memory of the nodes at the expense of precision in recalling the more recent inputs.

At this point, we have two types of memory in a reservoir with leaky integrator nodes. First of all there is the memory of the reservoir itself, caused by the recurrent connections that retain information inside the network. This type of memory is tuned by the dynamical regime of the

reservoir, but most importantly simply by the number of nodes in the network. Second, each node now has internal memory because of the integration mechanism. This node memory is changed by adjusting the leak rate. We can now investigate the interplay between the two types of memory.

Figure 3.3 shows plots for the performance on three tasks (30th order NARMA, speech recognition and memory capacity) as a function of the reservoir size and the leak rate. For the NARMA and memory task, we slightly tweaked the task to show the effects more clearly. The input used for these two tasks is normally white uniform noise, but in this case we filtered this noise signal with a low-pass filter with normalized cut-off frequency of 0.1 using a fourth order Butterworth filter. For the speech recognition task, we downsampled the speech samples by a factor of 128 before applying the Lyon cochlear model, to reduce the computing time.

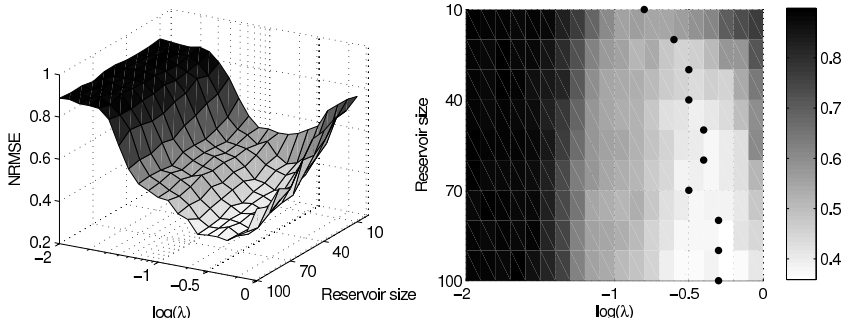
The plots for the NARMA task show that for a given reservoir size, there is an optimal leak rate. These optimal leak rates for each reservoir size are indicated with a dot in the figure on the right. The figure shows clearly that the optimal leak rate decreases for increasing reservoir sizes. This means that a trade-off is being made: for small reservoirs, the nodes need more internal memory (lower leak rates) to perform well, while for larger reservoirs the leak rate can become bigger because the network itself has enough memory for the task. In the latter case, the bigger leak rates are beneficial because they allow the higher frequencies contained in the input signal to pass. Still, lowering the leak rate does not fully compensate for the decrease in reservoir size.

A similar, yet less pronounced effect can be observed for the memory task. Here, too, the optimal leak rate decreases for increasing reservoir size. The effect is probably less pronounced in this case due to the simpler mapping from the input to the outputs (a simple delay). This means that the performance is likely less sensitive to the amount of information contained in the reservoir.

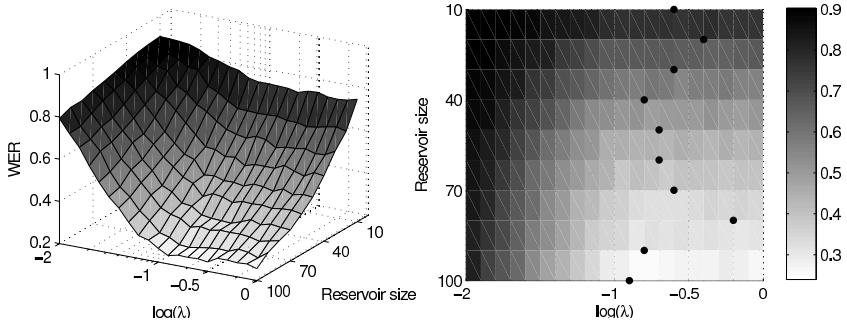
For the speech recognition task, finally, the trade-off is less clear. In fact, we observe that for every reservoir size, the performance levels off for leak rates above around 0.1. This could mean that the relevant information for solving this task is only contained in this frequency range.

3.1.3 The impact of time scales on spoken digit recognition

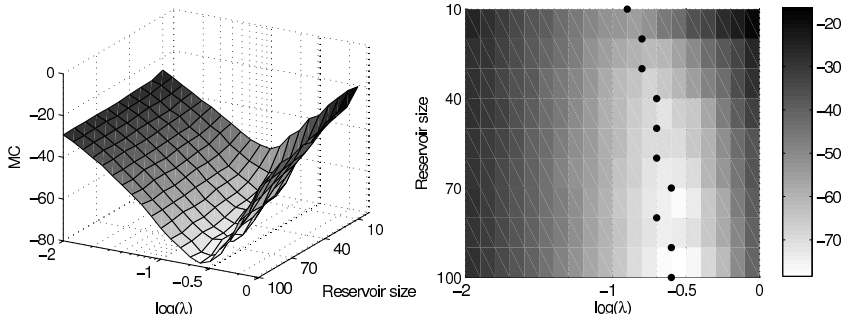
In the previous section we focused only on the internal reservoir timescales, defined by the leak rate λ. In this section, we will elaborate on this and



(a) 30th order NARMA



(b) Speech recognition



(c) Memory capacity

Figure 3.3: Plots showing the trade-off between reservoir memory (determined by the number of nodes) and the node memory (determined by the leak rate) for three different tasks. On the left, a 3D plot is shown, on the right the same plot in 2D, with the best leak rate for every reservoir size marked. The Y-axis on the right hand figures is reversed for better correspondence with the 3D plots.

Table 3.1: Minimal classification errors for all cases

	Input resampling	Reservoir resampling
Model 1	2.15%	0.48%
Model 2	2.32%	0.48%
Model 3	**1.36%**	**0.40%**

study the effects of the interplay of all three timescales, by experimentally evaluating their effects on the digit recognition task. As was mentioned before, the time steps δ_1, δ_2 (or λ) and δ_3 are only defined relatively to each other, so we will study the interplay between δ_1 and λ, and between δ_3 and λ.

3.1.3.1 Input Resampling vs. integration

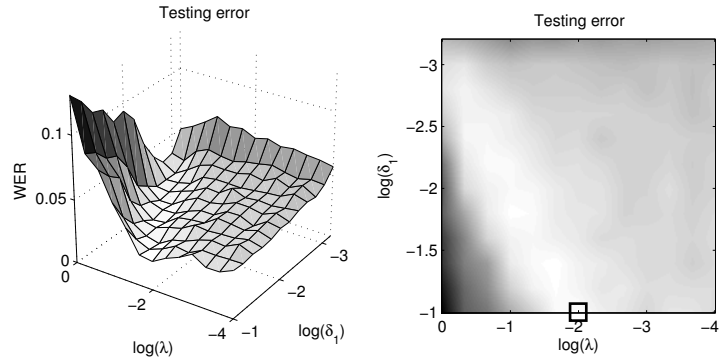
First we will study the influence on the performance of input downsampling versus integration for the three models introduced in 3.1.1.1. To be able to handle a broad range of parameters, we vary both these parameters in a logarithmic way, with base 10. Figure 3.4 shows the results of this experiment. We can observe a diagonal area on Figure 3.4 which corresponds to an optimal performance. For all three models we see that the optimal performance is attained with the least resampling (bottom of the plots). However, if we resample more, the error only slightly increases. This creates a trade-off between computational complexity and performance.

The optimal performance for the three different models is given in Table 3.1. We see that model 3, with the integrator across the nonlinearity, performs optimally, which is nice because this model introduces no extra computational requirements compared to standard tanh reservoirs.

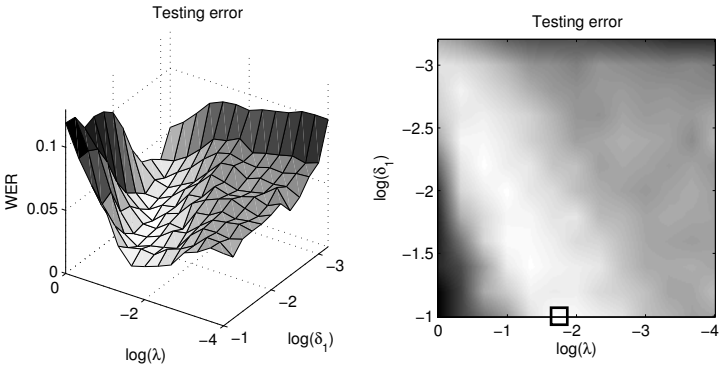
3.1.3.2 Reservoir resampling vs. integration

The second experiment studies the output time-scale compared to the internal integrator. The results of this experiment are shown in Figure 3.5. For these experiments, the input time-scale is set to $\log_{10}(\delta_1) = -2$. The setting of the input time-scale is not critical since the conclusions of the results also apply to other input time-scale settings.

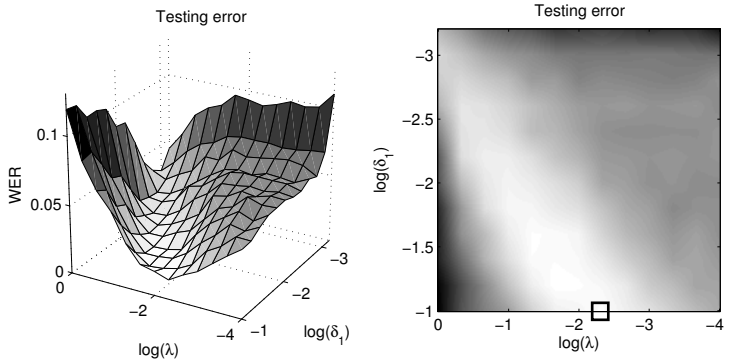
These figures are a bit more complex to interpret. The upper part, with $\log_{10}(\delta_3) = 0$, has no reservoir resampling and is thus equal to a slice of Figure 3.4 where $\log_{10}(\delta_1) = -2$. When increasing the resampling of the reservoir states, we see that for the region of low integration (close to 0) there is a significant decrease of the error. But in the region where the integration is optimal there is initially no improvement.



(a) Model 1

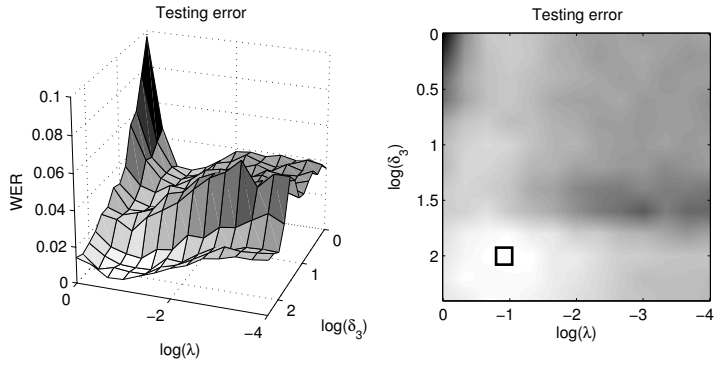


(b) Model 2

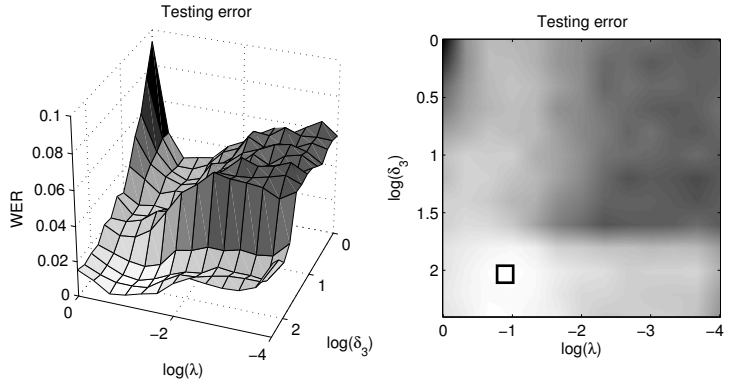


(c) Model 3

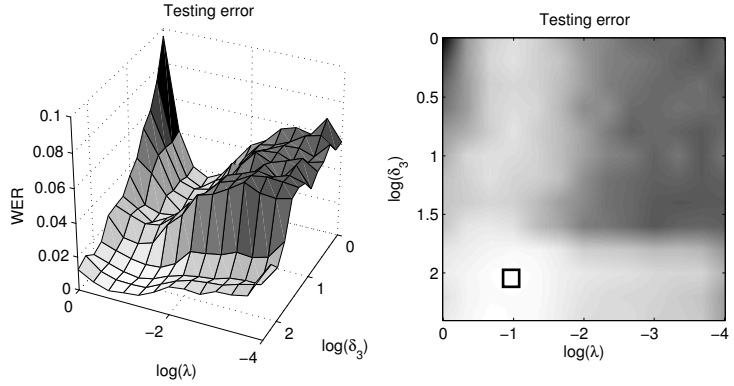
Figure 3.4: Results for input resampling versus leak rate for the three models. The small squares denote minimal error.



(a) Model 1



(b) Model 2



(c) Model 3

Figure 3.5: Results for leak rate versus output resampling for the three models. The small squares denote minimal error.

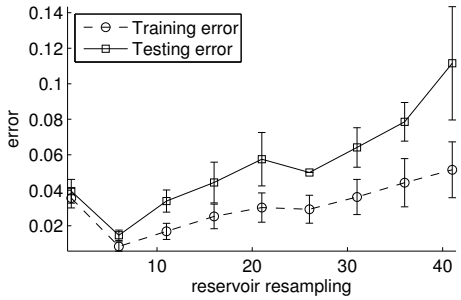


Figure 3.6: Classification error for a signal classification task with respect to the output time-scale. Note that the time-scale is not logarithmic in this figure.

The bottom part of the figures show a drastic increase in performance when $\log_{10}(\delta_3)$ is larger than 1.5. With such a high input and reservoir resampling, there is actually just one time step left! For this task we thus have optimal performance when reducing all the reservoir's dynamics to a single point in state space: the centroid of the dynamics in state space. This is not entirely unexpected since the post-processing of the linear classifier's output is done by taking its temporal mean before selecting the class output using winner-take-all. In fact, what we have done here is swap the post-processing mean operator with the linear readout. This is possible because both are linear operations.

Here, we can see one of the functional views we discussed in Section 2.1.2 on the RC network at work: when taking the centroid of the dynamics of the reservoir as input to the linear readout, we are in fact using the reservoir as a complex nonlinear kernel that is explicitly computed. In a way, the method demonstrated here bridges the gap between Reservoir Computing and kernel methods and shows that both are quite related. A drawback of this drastic reservoir resampling is that all temporal information is lost. With less reservoir resampling, the reservoir is able to already give a prediction of the uttered word even if it is for example only partially uttered. We thus trade-off performance to the ability of doing on-line computation.

One might think that when averaging out all the reservoir's dynamics, it has no real purpose. But when training a linear classifier to operate on the temporal average of the frequency-transformed input, so without using a reservoir, we end up with an error of 3%. This is quite good, but still an order of magnitude worse than when using a reservoir of only 200 neurons.

The conclusions drawn here are, however, partly due to the fact that

here the desired class output remains constant during the whole input signal. This is not generally the case. Let us consider for instance the signal classification task. As Figure 3.6 shows, in this case there is a trade-off between the amount of resampling of the reservoir responses and the amount of information available to the linear readout to do the classification. Only a small amount of reservoir resampling is needed to attain optimal performance. In state space this can be seen as taking the centroid of a small temporal region of the trajectory. This temporal averaging out of the dynamics seems to significantly increase the classification performance of the RC system.

In the first part of this chapter we have investigated the effects of timescales in RC. While we used standard tanh reservoirs for our experiments, the main message about the importance of tuning the timescales to the problem at hand is applicable to any reservoir type. In the next sections, we will introduce and discuss some specific, more advanced reservoir types that will gradually drift further from the standard 'neural' reservoirs.

3.2 Bandpass reservoirs

In the previous section, we discussed and investigated the properties of leaky integrator neurons. As was mentioned there, a closer look at the state update equation for this neuron model reveals that it can be seen as a standard sigmoidal neuron whose output is fed through a first-order low-pass filter. The use of leaky integrator neurons has the effect of slowing down the dynamics of the reservoir, as a result of the fact that the time constant present in the equation is actually the timestep with which the continuous-time model is sampled using an Euler approximation. However, from an engineering point of view, the filter interpretation is more useful since this allows a more intuitive way of thinking about these reservoirs. It was this interpretation that led to a more advanced extension on these leaky integrator neurons, called bandpass neurons.

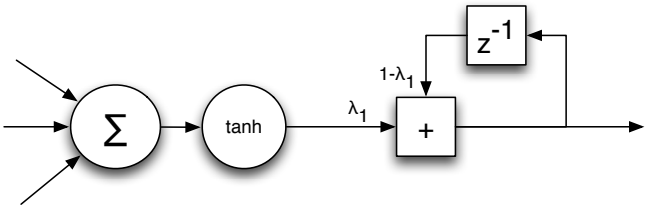
Bandpass neurons are described in the extensive technical report (Siewert and Wustlich, 2007). It first makes a thorough mathematical analysis of the different parameters that are used in the leaky integrator model and reduces the number of time constants from three to two without sacrificing generality. Next, a thorough investigation of the implications of the low-pass filter interpretation on the dynamics of the reservoir is made. This investigation leads the authors to a simple but elegant extension, first to high-pass filtered neurons (which are constructed by subtracting a low-pass filtered version from the original signal), and then to more

powerful bandpass neurons by combining both a low- and high-pass filter applied to the neuron output. This process is represented schematically in Figure 3.7.

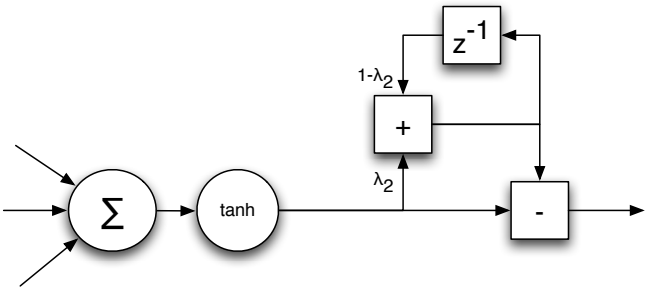
In the original technical report, it was already mentioned that an extension to more advanced higher order filters should be investigated, and this is precisely what was done in (wyffels et al., 2008c). Here, the use of second-order Butterworth filters for constructing bandpass neurons is investigated. Second order Butterworth filters have a sharper cut-off than the first-order filters that were originally used, and thus allow a finer-grained control over the frequency bands that are filtered out, and also cause less overlap between the bands of the individual neurons.

Since every neuron has its own bandpass filter, the bandwidths and center frequencies for every neuron can be tuned. In case the task at hand already suggests some prior information about which frequency bands are important, it obviously makes sense to choose the bands of the neurons around these important frequencies. This is for instance the case in the tasks presented in (wyffels et al., 2008c). Three signal generation tasks (see Subsection 2.2.1.4 for information on this type of task) were considered: a product of two sines with frequencies 23 Hz and 127 Hz, a product of three sines with frequencies 23 Hz, 127 Hz and 521 Hz, and the Mackey-Glass timeseries. For the first two tasks, it was shown by wyffels et al. that band-pass filters give a considerable improvement over standard leaky integrator neurons. Moreover, band-pass neurons with finely tuned center frequencies that match the signals that are to be generated yield the best results. For the Mackey-Glass task, the center frequencies for the bands were chosen based on visual inspection of the frequency spectrum of the Mackey-Glass signal itself. In this way, a reservoir was generated that could predict the Mackey-Glass series with an NRMSE of 0.0065 after 84 timesteps of autonomous generation. While this result is better than a previously published result using Long-Short Term Memory (LSTM) networks (Gers et al., 2001), it was not as good as the performance reported in (Jaeger and Haas, 2004) which was attained after thorough manual optimization.

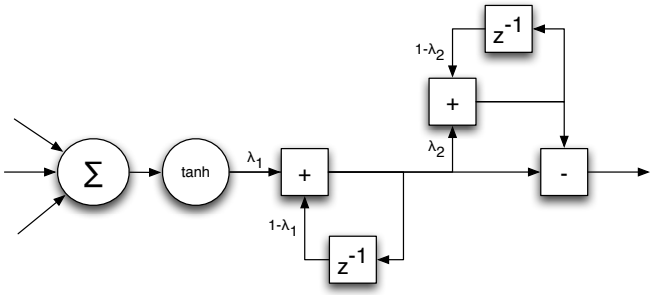
In most real-world cases, however, no such hints about the important frequencies contained in the input are available, or the input signals for the task span the whole frequency range in a more or less uniform way. In those cases, it is advisable to have the reservoir represent a well-balanced mix of frequency bands. This can be done by dividing the network into pools of neurons which share the same center frequencies and bandwidths. In (Siewert and Wustlich, 2007), one strategy that is suggested for choosing the center frequencies is an 'octave-style tuning', whereby the cut-off frequencies for a given band are a multiple of cut-off frequencies of the



(a) A low-pass (leaky integrator) neuron



(b) A high-pass neuron



(c) A bandpass neuron

Figure 3.7: Schematic representations of different filtered neurons. In figure (c), a bandpass neuron is shown as a composition of a low-pass filter and high-pass (but this can be swapped).

previous band, which results in an exponentially decreasing bandwidth.

In many of the studied cases, bandpass neurons can introduce significant improvement in performance over standard reservoirs or reservoirs built from leaky integrator neurons. Since time scales play an important role in the dynamics of reservoirs (see Section 3.1 of this work, and also (Lukosevicius et al., 2006; Schrauwen et al., 2007a; Holzmann, 2008)), bandpass neurons offer a finer and more usable way of tuning the time scales that are present in the reservoir. Indeed: the connection topology, nonlinearities and global weight scaling all introduce many additional frequencies in the reservoir signals that weren't present in the original input. Bandpass neurons offer an elegant way of filtering out these superfluous frequencies and retaining only information that is useful for the task at hand (of course, determining which frequencies are in fact useful is a different problem entirely).

3.3 Cellular Nonlinear reservoirs

Cellular Nonlinear Networks - sometimes also called Cellular Neural Networks (CNN) - are a computational concept introduced by L. Chua (Chua and Yang, 1988b,a). Their architecture bears some similarity to other parallel processing networks such as neural networks or cellular automata (CA). Many different CNN models are described in the research literature, but it is usually possible to define some common characteristics: CNNs are networks of simple nonlinear processing nodes that are discrete, and coupled in a local, nearest-neighbour manner. The precise interconnection topology can vary. Square, triangular, hexagonal and other interconnection topologies exist. However, this topology is always fixed and space-invariant which means that the weights of the connections between a node and its nearest neighbours are defined by a fixed template that is the same for each node. Also, the interconnection topology can always be mapped onto a 2D plane or a 3D toroid. CNN models can be continuous-time (CT-CNN) and discrete time models (DT-CNN).

Due to the 2D localized connections, one of the main application fields for these CNNs is image and video processing (Xavier-de Souza et al., 2006; Karacs and Roska, 2006; Török and Zarándy, 2002; Venetianter and Roska, 1998), since many algorithms in this field operate locally. However, many other applications exist, such as biological modelling (Nagy et al., 2005), physical modelling (Korbel and Slot, 2006) or signal processing (Chua et al., 1991; Bucolo et al., 2004). It has even been shown that universal computation in the Turing sense is possible with CNNs (Chua et al., 1993), resulting in what is known as a CNN Universal Machine

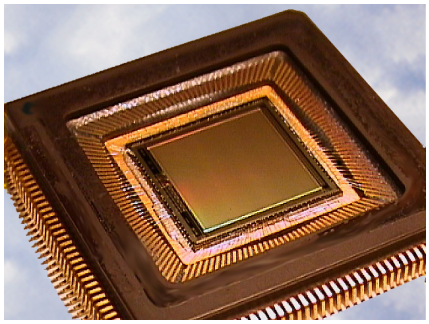


Figure 3.8: Image of the ACE16k chip.

(Roska and Chua, 1993). Much theoretical work has been done on the dynamical behaviour of these networks(Chua, 1998; Yalcin et al., 2005), in particular into the region at the edge of stability (Yang and Chua, 2001). This is perhaps not surprising given their similarities with cellular automata.

To our knowledge, all work on CNNs has focused on achieving the desired behaviour of the system based on explicit tuning of the weight template, either manually or through some learning rule. By concatenating several operations defined by weight templates, complex (image processing) algorithms can be executed. However, the use of a CNN (by definition a nonlinear dynamical system) as a reservoir has not yet been described in literature. In this section we explore this idea by a first proof-of-concept application to the spoken digit recognition task.

3.3.1 CNNs as reservoirs

Many CNN models exist, but here we use a space-invariant model where every cell is connected to its eight neighbours using the same weight template (see Figure 3.9). The cells are further characterized by a piecewise linear output function, and the network operates in a discrete time simulation mode. With these assumptions, the differences between the CNN and the traditional reservoir setup are twofold:

- instead of a randomly connected network, the cells are connected in a regular 2D lattice, with a space-invariant weight template;
- the output nonlinearity is a piecewise linear function instead of the traditional smooth tanh function.

These two restrictions can easily be incorporated into a simulation model, and we can thus simulate a CNN as reservoir using only adjust-

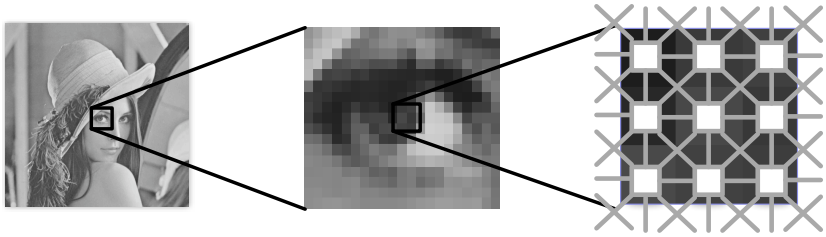


Figure 3.9: Schematic overview of the application of a CNN to an image and its topology.

ments to the network topology and nonlinearity. The input signal is connected to the CNN cells using a random input connection matrix - as with traditional reservoirs - where here the connection weights are randomly set to 0.1 or -0.1.

We also validated the simulation results on actual hardware. For this, we used an ACE16k chip (Rodriguez-Vazquez et al., 2004) with 128x128 cells and a weight precision of 8 bits - Figure 3.8 shows an image of the chip. However, for computational reasons we only use a center 8x8 grid of nodes leaving the other cells inactive. When making the transition from the software simulation to actual hardware, additional differences need to be noted:

- the limited precision of the template weights : the weights are represented internally as 8 bit digital values;
- the chip is built on analog VLSI technology, which means that the nodes have internal dynamical behaviour. It also introduces additional noise and other sources of variability such as thermal drift.

3.3.2 Sweeping the parameter space

In order to reduce the parameter space, we opted to use a symmetric template with only three distinct weight values: diagonal, lateral (horizontal and vertical) and self-recurrent. Thus, we were able to do a full sweep of the interesting part of the three dimensional parameter space.

Figure 3.10 on the left hand side shows simulation results for the signal classification task (see Subsection 2.2.1.3) as the diagonal, lateral and self-recurrent weights are varied. On the left, the error is shown, and on the right the variance of the error due to the variation of the remaining weight. The top two figures show that positive diagonal weights are beneficial, but that only the magnitude and not the sign of the lateral weights matters.

Additionally, the middle row shows that positive diagonal weights require small negative self-recurrent weights and vice versa.

Figure 3.11 shows the same plots for the speech recognition task. The aspect of the figures is less noisy, indicating a smoother error surface. Also, in all three plots a symmetry is apparent which means that only the absolute value instead of the sign of the weights is important. The top plot shows that the value of the lateral weights have a bigger impact on performance than the diagonal weights. Interestingly, the middle plot shows that the values of the self-recurrent and diagonal weights are related performance-wise, as indicated by the diagonal ridge of optimal performance. Finally, the variance of the performance is almost everywhere inversely related to the error, except in the middle area of the bottom plot which is therefore the optimal region with regards to performance and robustness.

3.3.3 Template optimization with Coupled Simulated Annealing

Next, we used a global optimization technique called Coupled Simulated Annealing (CSA) (Xavier de Souza et al., 2006) for the optimization of the CNN template, both in simulation and on the actual chip.

Traditionally, reservoirs are randomly constructed recurrent networks of nonlinear nodes, where the interconnection weights are drawn from a certain distribution - usually Gaussian. In the case of CNNs however, the interconnection structure is quite specific and the parameter space is far less-dimensional than for general reservoirs. This allows for the use of search-based optimization techniques. Here, we use an extension of the well known technique of Simulated Annealing (SA), which uses several coupled SA processes running in parallel. CSA couples these parallel classifiers by their acceptance probabilities, which results in a better overall performance and less sensitivity to the initial conditions. For our experiments, the initial temperature was set to 1, the number of parallel probes to 5 and the maximal number of iterations to 1000. Figure 3.12 shows a schematic view of the different components involved in the experiments. In case the experiments are done in software, the whole simulation, evaluation and optimization process is done using Matlab. For experiments on chip, the input timeseries are transformed into avi files with a single frame per timestep, and the resulting dynamical response from the chip is also saved as an avi file, which is then transformed back to the internal representation necessary for training and evaluation. In both cases, the result of the evaluation step, i.e., the error on the test set, is used by the

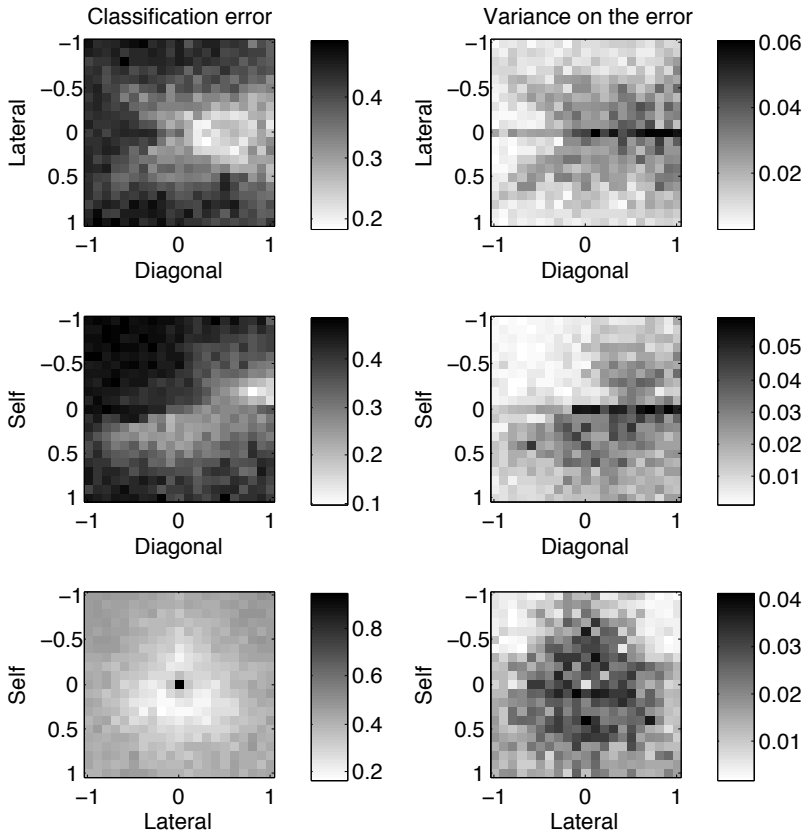


Figure 3.10: Simulation results for the signal classification task as a function of the template parameters. The template is symmetric - with the lateral (horizontal and vertical), diagonal and self-recurrent weights being three separate parameters. In the left column the mean error on the testset is shown, in the right column the variance on the error due to changes in the remaining weight.

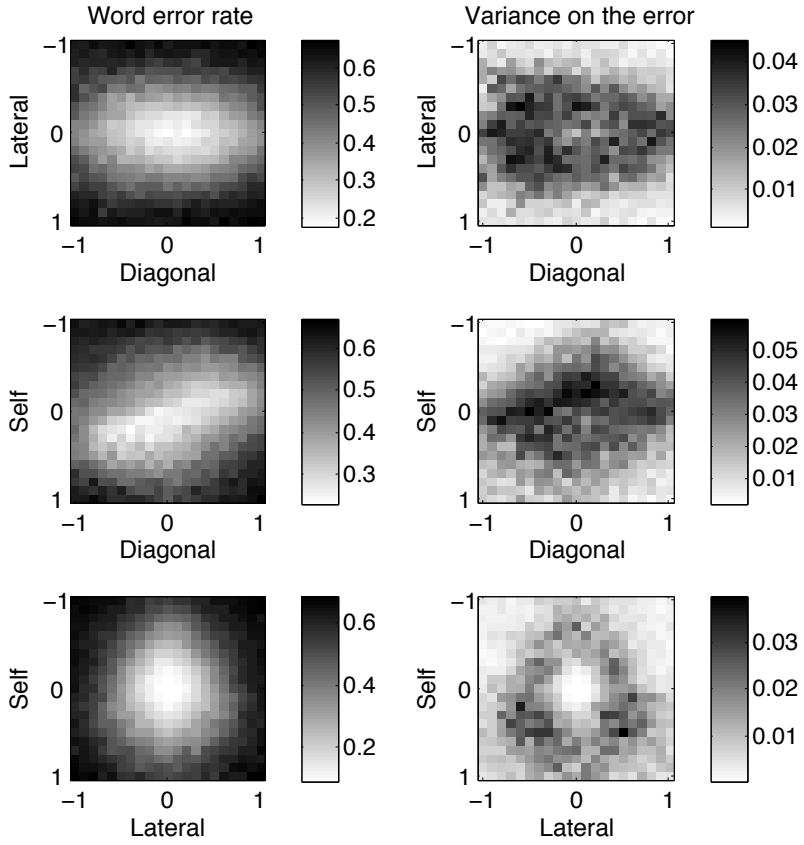


Figure 3.11: The same plots as in Figure 3.10 but for the speech recognition task. The error metric is Word Error Rate (WER) - the fraction of misclassified words. Error is plotted in the left column, variance on the error is plotted in the right column.

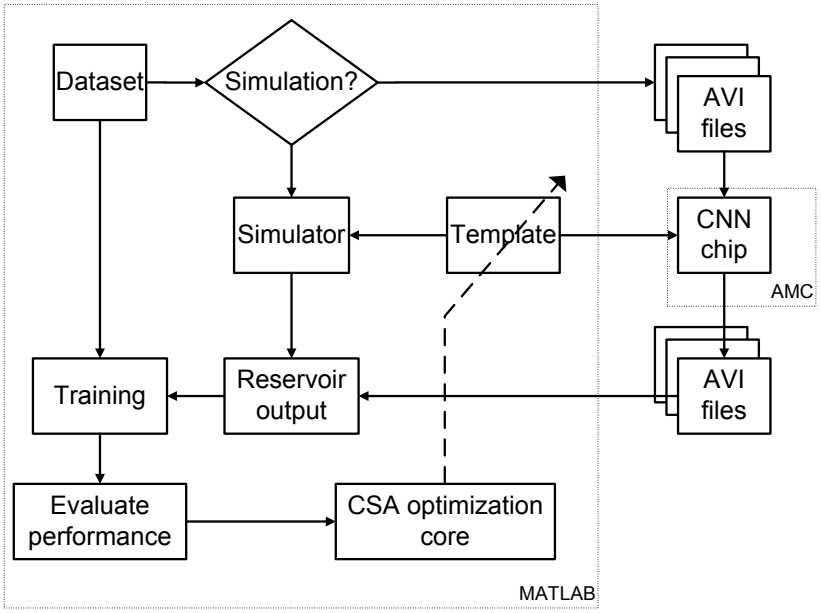


Figure 3.12: Overview of the experimental setup.

Coupled Simulated Annealing module to adjust the template.

Since CSA is a probabilistic algorithm, the parameter space is sampled in a non-uniform manner. Additionally, our experiments show that for random templates, the error surface is quite complex. This makes it difficult to visualize the results for the optimization process. We therefore only mention the optimal performances here.

On the speech recognition task, running the CSA optimization on our simulation model yielded a minimal error of 3.6%, and the same algorithm on the chip achieved a minimal error of 6%. We did notice a larger occurrence of badly performing templates on the chip (i.e., with large error rates), which is probably caused by the fact that those templates are less robust to the on-chip noise, rendering the task more difficult. As a means of comparison: the average error of a standard reservoir of the same size is 2%, and the direct application of a linear readout layer to the feature vectors, i.e., without a reservoir in between, yields an error of 11%.

On the signal classification task, the optimal template found by the CSA optimization core attained an error of 1% in simulation, and an even lower error of 0.1% on chip. Here, standard reservoirs obtain a similar error to the CNN simulation model of 1%, and the direct application of the linear readout to the input is not able to solve the task at all since it

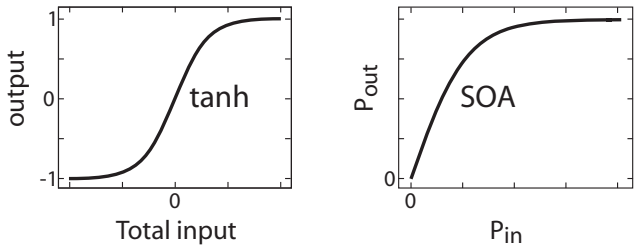


Figure 3.13: Comparison of the standard tanh nonlinearity with the steady-state input-output characteristic of an SOA. Note that the SOA can only have a positive output because here power is used as state variable. Figure taken from (Vandoorne et al., 2008).

requires at least some short-term memory. The fact that the speech task yields higher errors on chip than in simulation as opposed to the signal classification task, is possibly due to the fact that the on-chip noise has a negative effect on the speech recognition task whereas for the simple signal classification task it actually helps discerning between the signals. However, further work is needed to make this conclusive.

3.4 Photonic reservoirs

Arguably the most technologically exotic implementation of Reservoir Computing has been described in (Vandoorne et al., 2008). In that contribution, a proof of concept study is presented of a simulated setup of a reservoir constructed of so-called Semiconductor Optical Amplifiers (SOA). SOA's are optical gain elements made from semiconductor material such as GaAs/AlGaAs or InP/InGaAs. Their steady-state power input-output characteristic has a saturating shape, which bears strong resemblance to the positive part of a tanh nonlinearity (see Figure 3.13). However, it should be noted that these elements are dynamical in nature, and as such can exhibit additional temporal effects that are not visible in this steady-state curve. These SOA's can be coupled to form an excitable network of simple nonlinear nodes which can be used as a reservoir. This section discusses first results of the application of these photonic reservoirs, which were collected in collaboration with Kristof Vandoorne of the INformation TEChnology (INTEC) department of Ghent University.

The innovative character of this work lies in two fields: first of all, Vandoorne's article describes a model for a novel implementation medium for reservoirs in nanophotonic technology, offering several advantages over

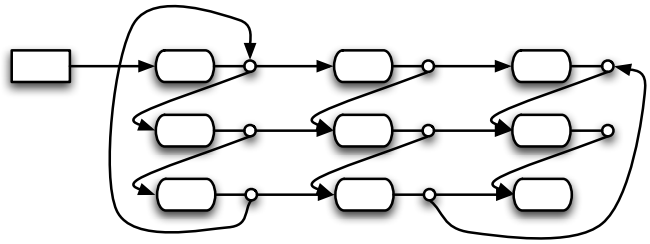


Figure 3.14: The 2D grid topology used for the SOA photonic reservoir. The input is fed into the top left node and propagates through the network in a waterfall manner. The long-term memory is provided by the feedback connections.

purely electronic silicon implementations such as processing speed and power efficiency. Second, the components used as nodes in the reservoir are rather complex devices with an inherently dynamical behaviour due to the build-up of carriers and the fact that the components are spatially extended (i.e., their spatial dimensions have an effect on the behaviour). This means that - contrary to traditional sigmoidal neurons - the SOA nodes themselves have an internal state and act as nonlinear dynamical components themselves. This also means that part of the memory of the reservoir will be stored inside the nodes.

A 2D interconnected lattice of SOAs used as reservoir is described in (Vandoorne et al., 2008). A 2D grid is used since the topology of a physical implementation would also be constrained by the achievable fan-in and fan-out of the neurons. The input signal is fed into the top-left node as an analog optical signal, and propagates in a waterfall manner through the network (shown in Figure 3.14). More long term memory is assured through the presence of a small number of longer feedback connections which feed the signals of downstream nodes back up to upstream nodes in the network.

The validity of the concept and the setup is demonstrated in simulation on the relatively simple square/sawtooth recognition task described in Subsection 2.2.1.3. Plots for the input and output signals, and for internal reservoir signals are shown in Figure 3.15.

Very recent experimental results show that after thorough optimization of the parameters of the SOA nodes, these photonic reservoirs can achieve comparable performance to standard tanh reservoirs on a 10th order NARMA task and for the memory capacity (Vandoorne, 2009). In this work, some of the techniques of resampling and adjusting the timescales discussed in Section 3.1 of this chapter were used to optimize performance. Specifically, the input signals were resampled to account

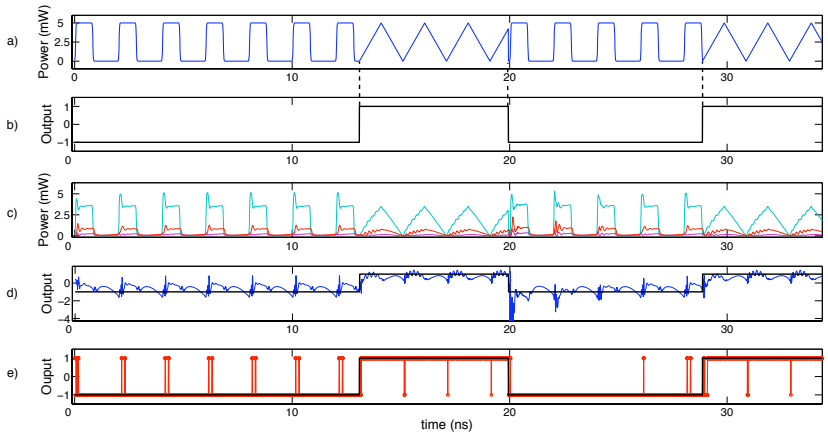


Figure 3.15: This figure shows the different signals generated during the signal classification task with a photonic reservoir. The top plot shows the input signal, the second plot shows the desired output signal. The third plot shows some selected responses of SOA nodes in the reservoir, the fourth plot shows the output of the readout layer, and the bottom plot shows the output after post-processing with a sign function. Figure taken from (Vandoorne et al., 2008).

for delays present in the SOA nodes.

The research into nanophotonic reservoirs has only started but has already kindled interest in many groups in the photonics community. This is due to the fact that Reservoir Computing offers a novel way of using photonic components for computing by not merely trying to mimic existing silicon components in photonic technology, but actually using the fundamental properties of the nano-photonic components and optical signals. Research on photonic RC is ongoing and is very active, with research groups from multiple Belgian universities collaborating on the topic.

3.5 Conclusions

In this chapter the initial steps towards generic reservoir types have been taken. RC can enable the use of a wide range of excitable nonlinear media for computation, many of which are actual physical systems that operate in continuous time. Usually, however, for engineering applications the transition to discrete sampled time has to be made at some point. I have introduced three different temporal domains in an RC system, one for the input, the reservoir and the output, and shown that adjusting these different timescales is crucial for optimizing performance in several tasks.

Additionally, I have shown that by adjusting these timescales, a trade-off can be made between memory that is present inside the network, due to the recurrent connections, and memory that is present inside the nodes, due to the integration mechanism.

Moreover, my experimental results have shown that in the case the reservoir states are resampled, the optimal performance for the spoken digit classification task we considered is actually reached for very drastic resampling, i.e., when the signals are reduced to a single timestep. This vector represents the centroid of the reservoir dynamics in the reservoir state space. Interestingly, this was what was being done all along, except it was done on the output signals instead of the reservoir signals. When swapping the linear readout operation and the mean operation, we are effectively transforming the reservoir into a kernel mapping that is explicitly computed. This demonstrates an interesting link between RC and kernel methods.

An interesting interpretation of the reservoir timescale as a low-pass filter is possible, which enables the designer of an RC system to think in the frequency domain. According to this viewpoint, each node is a standard neuron with a low-pass filter attached to it, which filters out any high-frequency components from the reservoir states that are not relevant for the task at hand. This idea was built upon by Siewert et al. to introduce first-order band-pass neurons, that allow a finer control over the filtering characteristics of the nodes and also allow to filter out low frequency components. Our lab has studied this node type and extended it to higher-order filters, where we have shown that these reservoir types can lead to better performance when the parameters are tuned properly.

Next, I have investigated a reservoir type that is yet one step further from standard ESN reservoirs. The usefulness of CNNs in the context of RC was evaluated by applying them to the isolated spoken digit recognition task. I used a hybrid analog/digital VLSI hardware implementation of such a CNN, where the nodes of the reservoir have actual internal dynamics because they are built from analog electronics. I have shown that these CNN reservoirs can be used very well as reservoirs, and in the case of the signal classification task the on-chip CNN reservoir even beats standard tanh reservoirs. Moreover, I have demonstrated that CNN chips (or models) can be used in an entirely different way than they normally are. Whereas CNNs are mostly used as complex, fast filtering devices for high speed image and video processing, we have shown that by using them as excitable nonlinear media a whole range of novel applications becomes available.

Finally, photonic reservoir computing was discussed. Here, the link with what we know about neural networks becomes very weak. The nodes

are nonlinear photonic components with complex dynamical internal behaviour and that communicate with light. Nonetheless, experiments using standard benchmarks have shown that this reservoir type is equally useful for computational purposes, and can rival standard neural reservoirs. Thus, Reservoir Computing can provide a whole new framework to use this photonic technology, taking it well beyond its standard use in, e.g., high-power laser systems.

4 Quantifying and adapting reservoir dynamics

We have shown that the idea of Reservoir Computing is extendable to other dynamical systems than recurrent networks of analog or spiking neurons. In this chapter, we will focus on developing tools for evaluating and tuning the computational power of these novel reservoirs. While RC can be a powerful framework for using dynamical systems for computation through learning, there are still many questions to be answered. The first issue that needs to be addressed is: are there systematic ways to evaluate the suitability of a novel dynamical system as a reservoir, other than 'blindly' exploring the parameter space using simulation? Since the dynamical regime is very important for the computational power of these systems, we will look at existing theory on dynamical systems for inspiration and develop novel metrics for quantifying reservoir dynamics. The next step is then not only to measure the dynamics of the reservoir, but to actually tune them. This will be done using unsupervised local adaptation that is based on biologically plausible principles with a theoretical basis founded in information theory. This adaptation mechanism is also applicable to novel, non-neural reservoirs and can tune the actual reservoir dynamics in a task-dependent way.

4.1 Computation with generic dynamical systems

4.1.1 Characterizing dynamical systems

There are many types of dynamical systems, and the range of dynamical behaviour these systems show is equally broad. We will limit this brief

discussion to discrete-time systems, but similar properties can be defined for continuous time. Discrete-time dynamical systems can be classified based on a variety of properties, but we will discuss their dynamical behaviour here based on two important subclasses: linear and nonlinear systems.

For linear systems, a characterization of the system dynamics is usually done in terms of stability. The stability of a linear system is determined by the location of the eigenvalues of the system transfer function in the complex plane, and its stability can be characterized by, a.o., its response to a Dirac input pulse (the so-called *impulse response*). In particular, depending on the location of the largest eigenvalue of the transfer function, the system can be:

- exponentially stable, which means that when the system is driven by a Dirac pulse, it will exponentially go back to its initial state;
- marginally stable, which means that if the system is driven by a Dirac pulse, it will not go back to its initial state but go to a different fix-point;
- unstable, which means that the system will 'run away' when driven by a Dirac pulse.

For nonlinear systems, stability is more difficult to define. For these systems the stability theory originally developed by Aleksandr Lyapunov (Lyapunov, 1966) comes into play. Here, too, different classes of stability can be defined, always expressed w.r.t. an equilibrium point x^* (a point that the system maps onto itself) in the system's state space. Nonlinear systems can be:

- stable, meaning that trajectories close to x^* stay close to x^*;
- asymptotically stable, meaning that trajectories that start close to x^* asymptotically return to x^*, and
- exponentially stable, meaning that trajectories that start close to x^* return to x^* with an exponential rate of convergence.

These definitions are all related to a single equilibrium point, which is called an attractor in those cases. However, a nonlinear dynamical system can have anywhere from zero to an infinite number of equilibrium points. Moreover, such systems can exhibit more complex behaviours, such as periodic attractors (limit cycles) or strange attractors.

Usually, when studying nonlinear dynamical systems, one is interested in the qualitative changes in behaviour of the system related to the change of its parameters. What is often observed for these systems is that there

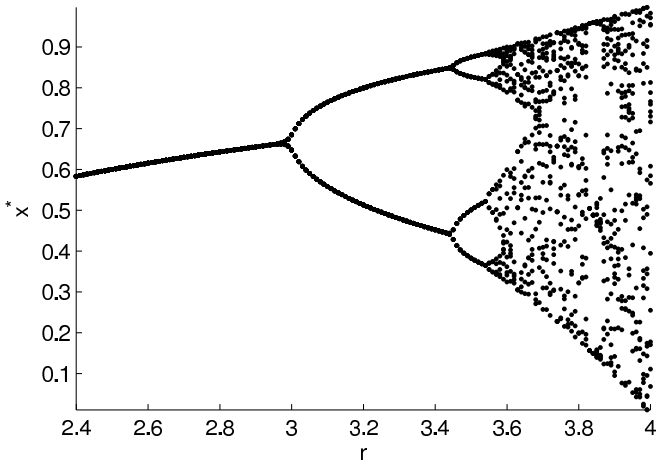


Figure 4.1: The period-doubling route to chaos for the logistic map.

is a whole range of possible behaviours ranging from a single equilibrium point all the way to chaotic behaviour. This is illustrated in, e.g., the very basic logistic map where the so-called period-doubling route to chaos can be observed (see Figure 4.1). The logistic map (which is actually a very simple model for population growth) is defined by:

$$x[k+1] = rx[k](1 - x[k]).$$

By varying the parameter r, the system goes through a whole range of dynamical regimes, going from a single equilibrium point over a series of period doublings to the chaotic regime for $r > 3.57$ (Sprott, 2003). If such a simple map can already produce such varied behaviour, it should come as no suprise that larger systems are even more complex to study (such as, e.g., recurrent neural networks (Doyon et al., 1992; Bauer and Martienssen, 1991)).

While most people have an intuitive notion of what chaos entails, some misunderstanding can arise if a good definition is lacking. For instance, chaos is sometimes confused with non-determinism. Non-deterministic systems are systems that have a stochastic (random) factor somewhere, either in their internal state variables (state noise) or in their parameters (parameter noise). These systems are inherently unpredictable and the same system starting from the same initial condition will almost certainly follow different trajectories. Chaos, on the other hand, is defined as 'sensitive dependence on initial conditions'. This means that the system itself can be both deterministic or not, but slight changes in the initial condition

(or slight perturbations of a trajectory) will result in a strong deviation from the original trajectory. This is related to Lyapunov's notion of exponential stability: whereas stable systems converge exponentially to a equilibrium point, chaotic systems do the opposite: they diverge exponentially when slightly disturbed. The importance of stability and chaos for computation, especially in the context of Reservoir Computing, will be elaborated upon in the following section.

4.1.2 Computation at the edge of stability

Reservoirs are generic, dynamical, excitable and generally nonlinear systems. The readout layer, on the other hand, is linear and memoryless, and - from a learning point of view - computationally not very powerful. The functionality of a reservoir in this context can thus be described as "offering a rich ensemble of nonlinear temporal transformations of the current and past inputs" to the readout. It follows that the dynamical regime that the reservoir operates in is crucial for the amount of information that the readout can extract from the reservoir states.

The term 'rich ensemble' is of course quite vague, and the question remains what kind of transformation should be done by the reservoir in order to be useful. The nature of this transformation is hinted at by the notion of *computation at the edge of chaos*. This idea was originally proposed in the context of cellular automata (Langton, 1990), where it was stated that certain cellular automata (Wolfram, 2002), when operating in a certain regime controlled by a single parameter, started exhibiting phase transitions, and precisely these automata were shown to be capable of universal computation in the Turing sense. While the general applicability of this idea was later questioned (Mitchell et al., 1994), the central message remains: if a system operates in a dynamical regime that lies on the edge between stability and chaos[1], its computational capabilities are maximal.

The link between the dynamical properties of a system with regard to information content and computation can be seen also in Pesin's theorem or formula (Pesin, 1977). This theorem relates the so-called Kolmogorov-Sinai (KS) entropy to the Lyapunov spectrum. More specifically, the theorem states that the KS entropy of a dynamical system is equal to the sum of the positive Lyapunov exponents. The KS entropy, also known as metric entropy, is a measure of the predictability of the trajectory of the dynamical system, and can be related to a better known information theoretic measure, namely Shannon's entropy (Billingsley, 1965).

[1] ... but still on the stable side. As was mentioned before, the *edge of stability* would be a more accurate but perhaps less appealing term.

In the context of RC, this concept can be explained in an intuitive way for classification tasks[2] as follows. The reservoir should be seen as a dynamical system that follows a certain trajectory in state space, driven by the external input signals. The readout layer - which is a linear discriminant function - draws hyperplanes through the state space of the reservoir, and as such creates regions which correspond to class decisions. The idea is then that, ideally, the reservoir responds in such a way to the external input signals that input signals of the same class drive the reservoir to the same region in state space, and inputs of different classes cause trajectories that are far enough apart that they lie on different sides of the separating hyperplanes. We have already briefly discussed this view on reservoirs in Subsection 2.1.2 and we refer again to Figure 2.4 for a visualization of this concept.

It follows that the reservoir should operate in a dynamical regime that lies in between highly stable behaviour (where the reservoir stays in the same region, largely uninfluenced by the external input) and chaotic behaviour, where every change in the input signal is amplified exponentially and most of the information about the input is lost. Moreover, for chaotic systems the echo state property no longer holds, since slight changes in the initial conditions are amplified exponentially. Thus, the reservoir should react just 'dynamically enough' to separate inputs of different classes. Of course, this view is only an abstraction because in reality there can be substantial differences between inputs of the same class, and because noise also determines the trajectory of the reservoir.

4.1.3 Static reservoir measures and their disadvantages

A different but related interpretation of the functionality of the reservoir can be found in the so-called echo-state property (ESP) which was defined by Jaeger in the seminal description of the Echo State Network (Jaeger, 2001a). The ESP states that the reservoir state observed after a sufficiently long time (to let initial transients die out) is uniquely determined by the input signal driving the reservoir. In the same document, three other properties are proven to be equivalent to the echo state property, namely: state-contracting, state-forgetting and input-forgetting. These properties in essence convey the idea that the reservoir asymptotically forgets inputs from the past - which is another way of saying that the reservoir has *fading memory*. The designer of a RC system should be

[2]The reasoning is likely also valid for regression tasks, but the image is less visual and intuitive.

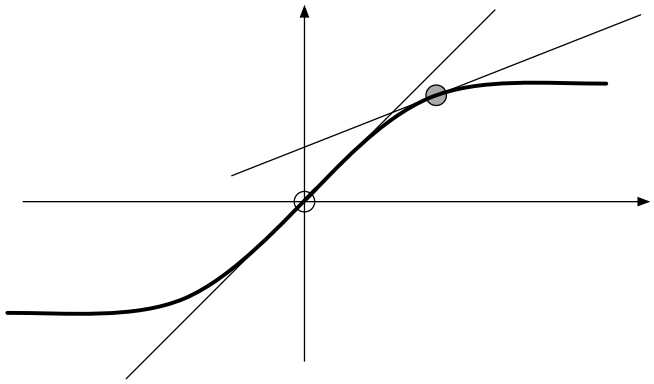


Figure 4.2: The gain of the sigmoid nonlinearity is largest around the origin. Once the neuron is driven by an external signal or a constant bias, the working point shifts up or downward and the gain decreases, resulting in a less dynamical reservoir.

able to control the memory properties of the reservoir. However, the ESP is not the only property one desires from good reservoirs. Ideally, the reservoir should provide also a suitable separation of the input signals in state space. This is controlled by the dynamical regime of the reservoir. One conceptually simple way (and with a mathematical justification) to control the dynamics while ensuring the ESP was proposed by Jaeger and is still widely used. It involves constructing a random reservoir with a chosen weight distribution, and then rescaling the global weight matrix to adjust the dynamical and memory properties of the reservoir.

As was mentioned in the brief description of the RC/ESN methodology, the spectral radius is an important parameter that controls the dynamical regime of the reservoir. It amounts to a global scaling of the eigenvalues of the connection matrix. From a system theoretic point of view, this can be interpreted as follows: for a small-signal approximation (i.e., the state of the reservoir remains near the zero fix-point), the reservoir can be approximated as a linear time-invariant, discrete-time system described by the following standard :

$$\begin{aligned} \mathbf{x}[k+1] &= \mathbf{A}\mathbf{x}[k] + \mathbf{B}\mathbf{u}[k] \\ \mathbf{y}[k] &= \mathbf{C}\mathbf{x}[k] + \mathbf{D}\mathbf{u}[k], \end{aligned}$$

where $\mathbf{x}[k]$ represents the state of the reservoir (the vector of neuron activations) at time k, and $\mathbf{u}[k]$ and $\mathbf{y}[k]$ represent the input and output to the system, respectively. The matrix $\mathbf{A}$ contains the internal weights of the reservoir ($\mathbf{W}_{res}$ from above), the $\mathbf{B}$ matrix contains the input-to-reservoir

weights ($\mathbf{W}_{in}$ from above), and $\mathbf{C}$ and $\mathbf{D}$ contain the (trained) reservoir-to-output ($\mathbf{W}_{out}$ from above) and input-to-output weights respectively (the latter is usually left zero).

It follows from linear system theory that if all singular values of the matrix $\mathbf{A}$ are smaller than 1[3], it is definitely stable, whereas if any absolute eigenvalue (i.e., spectral radius) is larger than 1, the system (i.e., the reservoir) will definitely be unstable in the sense that it will deviate unboundedly from the fixed point when started from a non-zero state. However, the reservoirs of the ESN type (and the reservoirs that we consider in this thesis), have a squashing tanh() nonlinearity applied to them, that counteracts this unbounded growth. This means that the norm of the state vector of the reservoir will always remain finite. Using a saturating nonlinearity also means that the spectral radius looses its significance as a stability measure when the system deviates from the zero state. Once the system is driven by an external input (especially with non-zero mean or large dynamical range) or if a constant bias is fed to the nodes, the operating point of all neurons shifts along the nonlinearity and the effective local gain (i.e., the slope of the tangent in the operating point) becomes smaller (see Figure 4.2).

In (Jaeger, 2003), the linear approximation described above is used to derive some mathematically founded guidelines for constructing weight matrices for reservoirs having the echo state property. Several bounds have been described for this ESP:

- A reservoir whose weight matrix $\mathbf{W}$ has a largest singular value (LSV) (denoted as $\sigma_M(\mathbf{W})$) smaller than one, is guaranteed to have the echo state property. However, in practice this guideline is of little use since the resulting reservoirs are not dynamically rich enough to perform well because this is a very strict stability bound.

- A reservoir whose weight matrix has a spectral radius (SR) $\rho(\mathbf{W})$ - i.e. a largest absolute eigenvalue - larger than one is guaranteed *not* to have the echo state property. So, $\rho(\mathbf{W}) < 1$ is a necessary condition for the echo state property. Note that this condition is only applicable *for zero input.* This last part is sometimes misunderstood or omitted in literature, causing the incorrect identification of a spectral radius smaller than one with the echo state property. In fact, it is possible that reservoirs with a spectral radius larger than one do possess the echo state property when driven by an external input - this was proven experimentally in, e.g., (Verstraeten et al., 2006). While the spectral radius criterium is not a sufficient

[3]This implies that the maximal gain in any direction in state space is smaller than one, and the system is always contracting.

condition, in practice it is used as a guideline for constructing good reservoirs for many problems.

- In (Buehner and Young, 2006), a tighter bound on the echo state property than $\sigma_M(\mathbf{W}) < 1$ was presented. A Euclidean weighted matrix norm $\|\mathbf{W}\|_{\mathbf{D}} = \left\|\mathbf{D}\mathbf{W}\mathbf{D}^{-1}\right\|_2 = \sigma_M(\mathbf{D}\mathbf{W}\mathbf{D}^{-1})$ was introduced, for which the relation $\rho(\mathbf{W}) < \inf_{\mathbf{D}_\delta} \left\|\mathbf{D}_\delta\mathbf{W}\mathbf{D}_\delta^{-1}\right\| < \sigma_M(\mathbf{W})$ holds. The infimum of the norm over all $\mathbf{D}$ that are structured in some way, called the *structured singular value (SSV)* μ_{SSV}, turns out to offer a bound on the echo state property that is less conservative than the standard $\sigma_M(\mathbf{W}) < 1$, namely $\inf_{\mathbf{D}_\delta} \left\|\mathbf{D}_\delta\mathbf{W}\mathbf{D}_\delta^{-1}\right\| < 1$. However, while this new bound is an improvement over the standard LSV, it is computationally quite demanding to evaluate (21 seconds for a reservoir of 500 nodes, versus .6 seconds to compute the spectral radius).

There are a couple of problems with using these measures as a tuning parameter for the dynamical regime of the reservoir. All the quantities described above are static measures that only take the internal reservoir weight matrix into account and disregard other factors such as input scaling, bias or dynamical range of the input signals - factors that are equally important in defining the dynamical properties of the system. Moreover, they are only strictly applicable to reservoirs with a nonlinearity which is linear around the origin (such as tanh or purely linear reservoirs): for the small-signal approximation (the system is driven by a very weak signal), all sigmoid nonlinearities behave more or less like identity functions. For more exotic nonlinearities, this approximation usually no longer holds.

Clearly, an accurate way of quantifying the dynamics of the reservoir, evaluated in the actual operating point of the reservoir, would be very useful. I will investigate this notion in the following section, by deriving and studying novel measures for quantifying the dynamic regime of the reservoir.

4.2 Quantifying reservoir dynamics

4.2.1 Linking different bounds for the echo state property to network dynamics

One possible way of measuring the stability (or chaoticity) of a trajectory through state space of a dynamical system is the Lyapunov exponent (Alligood et al., 1996). The Lyapunov exponent (LE) is a measure for the

exponential deviation in a certain direction of a trajectory, resulting from an infinitesimal disturbance in the state of the system. If the trajectory is near an attractor, the effect of the disturbance will disappear and the LE will be smaller than zero. In case the system drifts away from the trajectory exponentially, the LE is larger than zero and the trajectory is unstable. Note that the LE can be positive in one direction and negative in another.

A common definition of a *chaotic trajectory* is then that the trajectory

- is not asymptotically periodic,
- has no Lyapunov exponent that is exactly zero and
- has a largest Lyapunov exponent that is larger than zero.

A chaotic trajectory deviates exponentially in at least one dimension after a perturbation. The pth Lyapunov *number* L_p is related to the pth Lyapunov exponent h_p through $L_p = \exp(h_p)$. Thus, if a trajectory is chaotic, at least one Lyapunov number will be greater than one. For every trajectory of a dynamical system in state space described by the state vector $\mathbf{x}[k]$ at time k, we can calculate $p = 1, \ldots, n$ Lyapunov numbers (one for every dimension of the state) by considering a unit hypersphere in state space whose center follows the trajectory $\mathbf{x}[k]$, and considering how it is transformed by the update function of the system. The hypersphere will evolve to a hyperellipsoid as it travels through state space. The Lyapunov numbers for the trajectory are then given by:

$$L_p = \lim_{K \to \infty} \prod_{k=1}^{K} (r_p^k)^{1/k},$$

where r_p^k is the length of pth longest orthogonal axis of the hyper-ellipsoid at discrete timestep k. The value of these lengths can be calculated as follows. Let J_k denote the Jacobian of the kth iterative application of the map $\mathbf{f}$. The length of the axes of the hyper-ellipsoid is then given by the square root of the eigenvalues of $J_k J_k^{\mathrm{T}}$ (see (Alligood et al., 1996) for a more elaborate discussion). Thus, the Lyapunov number L_p expresses the exponential expansion ($L_p > 1$) or contraction ($L_p < 1$) of a unit sphere under the map $\mathbf{f}$. Note that this method assumes that the limit above converges.

In the case of sigmoidal reservoirs, we can calculate a measure analytically that is closely related to the LE. The Jacobian matrix of the

reservoir state $\mathbf{x} = [x_1, \ldots, x_n]$ at timestep k is given by:

$$\mathrm{D}\mathbf{f}(\mathbf{x}[k]) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}[k]) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}[k]) \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}[k]) & \cdots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}[k]) \end{pmatrix}.$$

Since a tanh reservoir is described by the following map:

$$\mathbf{f}(\mathbf{x}[k+1]) = \tanh(\mathbf{x}[k] * \mathbf{W}) = \left[\tanh(\textstyle\sum_{i=1}^{n} w_{1i}x_1) \cdots \tanh(\sum_{i=1}^{n} w_{ni}x_n)\right],$$

we obtain the Jacobian of the above map:

$$\begin{aligned} \mathrm{D}\mathbf{f}(\mathbf{x}[k]) &= \begin{bmatrix} \left[1 - \tanh^2(\sum_{i=1}^{n} w_{1i}x_1)\right] w_{11} & \cdots & \left[1 - \tanh^2(\sum_{i=1}^{n} w_{1i}x)\right] w_{1n} \\ \vdots & & \vdots \\ \left[1 - \tanh^2(\sum_{i=1}^{n} w_{ni}x_n)\right] w_{n1} & \cdots & \left[1 - \tanh^2(\sum_{i=1}^{n} w_{ni}x_n)\right] w_{nn} \end{bmatrix} \\ &= \begin{bmatrix} \left[1 - x_1^2(k)\right] w_{11} & \cdots & \left[1 - x_1^2(k)\right] w_{1n} \\ \vdots & & \vdots \\ \left[1 - x_n^2(k)\right] w_{n1} & \cdots & \left[1 - x_n^2(k)\right] w_{nn} \end{bmatrix} \\ &= diag[(1 - x_1^2) \ldots (1 - x_n^2)] \times \mathbf{W}. \end{aligned}$$

The above matrix offers an analytical expression for the Jacobian matrix J, which in turn allows us to approximate the Lyapunov numbers. The maximal estimated LE $\widetilde{h}_{max}$ can then be obtained by:

$$\widetilde{h}_{max} = \log\left(\max_p \prod_{k=1}^{K} (r_p^k)^{1/k}\right),$$

where $r_p^k = \sqrt{|\lambda_p|}$, λ_p being the pth eigenvalue of $J_k J_k^{\mathrm{T}}$.

Due to the analytical derivation of this rule, it can easily be calculated, and is exact. An extra speed-up can be attained by calculating the exponents in a sampled manner, e.g., every 10 timesteps. This proved to be an accurate approximation. We will demonstrate that this measure is a good indicator of reservoir dynamics and performance. The Lyapunov definition given above is in principle only suited for autonomous systems (or driven systems where the driving force can also be modeled as state variables, rendering the system autonomous). But in the case of reservoir computing we have systems that are constantly driven by complex, even random, input. The standard definition of a LE as the existence of a limit will therefore not be applicable because no steady state is reached due to the continuously changing input and state trajectory.

In (Legenstein and Maass, 2005), the LE is measured empirically for

a spiking reservoir by calculating the average euclidian distance of the reservoir states resulting from time-shifting a single input spike over .5 ms. For spiking neurons this experimental approach is probably the only way to formulate any conclusions about the stability of the system due to the complexity and time-dependence of the model. This method can be seen as the disturbance of an autonomous system where the exponential state deviation is an approximation of the LE. In (Schrauwen et al., 2008a), the maximal LE is studied for reservoirs with a quantized sigmoid nonlinearity, and the exponent is also computed numerically based on theory from branching processes (Athreya and Ney, 2004). A disadvantage of both methods is that they are very time-consuming.

To illustrate the relationship between these different bounds and reservoir performance for certain tasks, we evaluated the performance of different reservoirs on the NARMA (Figure 4.3), memory (Figure 4.4) and speech recognition benchmark (Figure 4.5) by constructing the weight matrix of the reservoirs and then rescaling it so that either the spectral radius, LSV or μ has a specified value[4]. For each of these reservoirs, the largest LE $\widetilde{h}_{max}$ was also estimated for every trajectory and averaged across all input timeseries.

For all three tasks, the optimal value for the μ_{SSV} parameter lies between that of the spectral radius and of the LSV (which was expected since the spectral radius and LSV are a lower and upper bound for the echo state property), but it is significantly higher than one – which indicates that the reservoir is not globally asymptotically stable. This is also confirmed when the corresponding LE is estimated, indicated by the dashed lines on the figure. It appears that the system is on average state-expanding for the trajectories caused by the inputs and the estimated LE for the optimal values of the different metrics is very similar for all three benchmarks and lies around 0.7, which indicates a dynamical regime: on average the trajectory locally shows exponential deviation in at least one direction. Note that the estimated $\widetilde{h}_{max}$ are very similar for the NARMA and memory capacity tasks, since in both cases the input to the reservoirs is a uniform random signal. Note also that a largest estimated Lyapunov exponent larger than zero in this case does not necessarily mean that a system is chaotic, because it is input driven and Lyapunov exponents are defined for $k \to \infty$.

[4]Note that these measures are not linearly related, meaning they are not simply a rescaled version of each other. For a given value of one measure, the other two vary quite substantially.

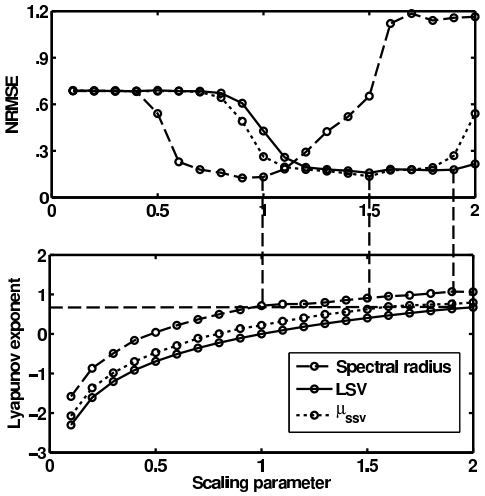


Figure 4.3: The top figure shows the performance on the NARMA task as a function of different reservoir metrics. The bottom figure shows the corresponding estimated Lyapunov exponents.

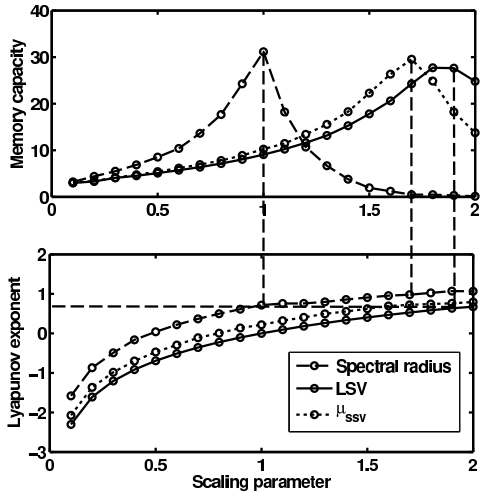


Figure 4.4: The top figure shows the performance on the memory capacity task as a function of different reservoir metrics. The bottom figure shows the corresponding estimated Lyapunov exponents.

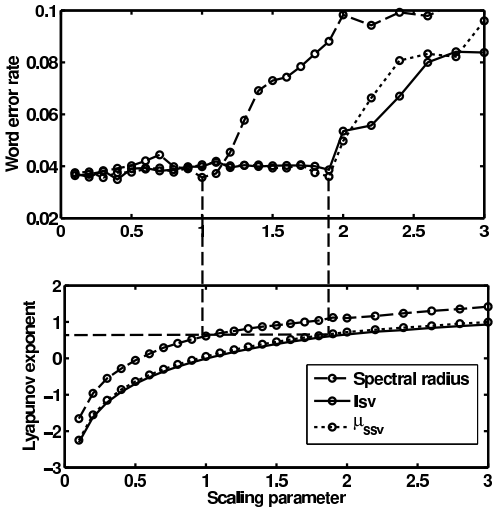


Figure 4.5: The top figure shows the performance on the speech recognition task as a function of different reservoir metrics. The bottom figure shows the corresponding estimated Lyapunov exponents.

4.2.2 Towards a more complete quantification of reservoir dynamics

In the section above, the relationship between the maximum of the Lyapunov spectrum and the performance of the reservoir was studied. It was found that for a given task, the optimal performance of a reservoir was attained for the same value of the maximal estimated Lyapunov exponent. While this finding was useful from a theoretical point of view because it offered a more refined measure of the reservoir dynamics than the stationary measures mentioned in the previous section, it does not supply a practical means for choosing the reservoir dynamics or offer insight into the meaning of this metric.

Closer inspection of the *complete* estimated Lyapunov spectrum reveals another, and in some ways more useful phenomenon. Figure 4.6 shows a plot of the mean over time of all Lyapunov exponents as the spectral radius of the reservoir is varied from .1 to 3 and the reservoir is driven by noise (which is the input for the NARMA task). The plot shows that the maximal exponent increases monotonically (as was shown previously in (Verstraeten et al., 2007)), but also that the *minimal* exponent reaches a maximum for a spectral radius of 1, and then decreases again. Thus, the spectrum of Lyapunov exponents becomes narrower and

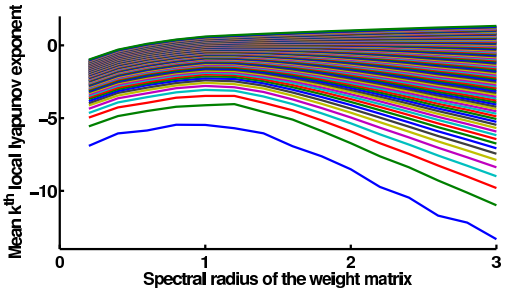


Figure 4.6: The full mean (over time) local Lyapunov spectrum for a reservoir of 100 nodes for the NARMA task.

then broader again as the spectral radius of the reservoir weight matrix is increased. More importantly, the maximum of the minimal LE coincides with the value of optimal performance of the spectral radius. In the next section, we will present some more elaborate experimental results and discuss the implications of this phenomenon.

The maximal Lyapunov exponent is - for autonomous systems - an indicator of chaotic behaviour: if it is larger than zero the system is said to be chaotic, meaning that perturbations from a trajectory are amplified exponentially in at least one direction. At first sight no such interpretation exists for the minimal LE - it simply quantifies the minimal direction of expansion of the system. However, closer inspection reveals that a more informative interpretation is possible by inspecting the Jacobian matrix itself.

We start with the following remark: when evaluating the Jacobian around the origin in state space (zero fixpoint, i.e. $\mathbf{x} = \mathbf{0}$), it reduces to the weight matrix $\mathbf{W}_{res}$ of the reservoir, and its largest eigenvalue is precisely the spectral radius of the reservoir. Therefore, the eigenvalue spectrum of the Jacobian can be seen as some form of dynamical extension of the static eigenvalue spectrum of the weight matrix (which was the subject of previous work on dynamics in reservoirs, e.g., (Ozturk et al., 2006)). Moreover, the so-called *local* Lyapunov spectrum (Wolff, 1992) at a single time step k, given by $\log\left(\text{eig}(J_k^T J_k)\right)$, is equal to the log of the squared singular value spectrum of the Jacobian itself[5] (Ziehmann et al., 1999). Following this line of reasoning, we measured the minimal singular value (SV) σ_m of the Jacobian[6] and computed its mean over time as we vary the spectral radius of the reservoir weight matrix, and the scaling

[5] In general, for any matrix M, the squares of its singular values are equal to the eigenvalues of $M^T M$ or MM^T.

[6] At every 50th timestep for computational reasons, but this provides sufficient accuracy.

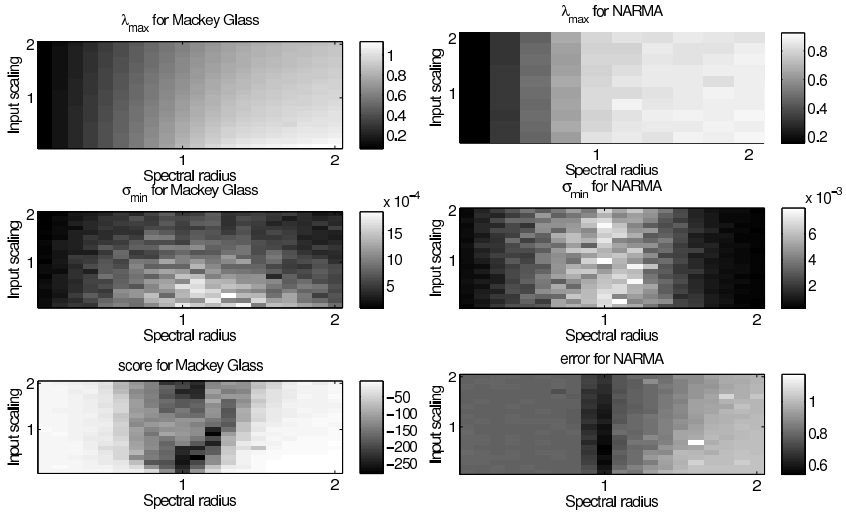


Figure 4.7: The top plots show the maximal LLE, the middle plots show the minimal SV and the bottom plots show the performance for the Mackey-Glass prediction (left) and NARMA (right) task. Note that for the Mackey-Glass performance plot, higher is better while for NARMA lower is better. The minimal SV attains a maximum in the same region as the performance, indicating that it is a good predictor of the suitability of a reservoir.

factor of the input matrix. We then compared this measure with the performance on two tasks: the Mackey Glass timeseries prediction with delay parameter $\tau = 17$ and the 30th order NARMA system identification task (see Subsection 2.2.1 for a specification of these problems).

Figure 4.7 shows the mean maximal LE, the mean minimal singular value of the Jacobian, and the score on both tasks, as the spectral radius and scaling factor of the input matrix are swept within plausible ranges (every point in the plots represents the average over twenty different reservoir instantiations). The top plots show the maximal LE that was introduced in the previous section. This measure clearly does not capture all necessary dynamical properties of the reservoir, since it increases monotonically with the spectral radius, and the input scaling has hardly any influence. The middle plots on the other hand - which show σ_m - offer a much more nuanced image. The minimal SV varies with both the spectral radius and the input scaling - which indicates that it captures the changing dynamical properties of the reservoir as a function of the scaling parameters quite well. Moreover, the area of optimal performance (bottom plots) coincides quite nicely with the areas where σ_m is maximal. Thus, the minimal SV seems to be a more accurate predictor

of performance than both the largest LE and the spectral radius for the tasks considered here.

The interpretation of the minimal SV of the Jacobian is at first sight not trivial: it simply qualifies the minimal gain of the system in any direction in state space. However, since $\sigma_m^{-1} = \left\| J_f^{-1} \right\|$ and $\kappa(J_f) = \left\| J_f^{-1} \right\| \|J_f\|$, σ_m can be written as the ratio between the norm $\|J_f\|$ and the condition number $\kappa(J_f)$ of the Jacobian:

$$\sigma_m = \frac{\|J_f\|}{\kappa(J_f)},$$

where $\|\cdot\|$ denotes the l_2 norm.

This relation yields an interesting interpretation. The norm of the Jacobian is a measure of the maximal gain of the system in any direction, while the condition number is used to quantify the degrees of freedom in the system. This is used in, e.g., the field of robotics (which borrows substantially from dynamical system theory), where both the condition number and the norm of the Jacobian are widely used measures for quantifying the dynamical behaviour of, e.g., robotic arms (Merlet, 2006). The condition number is used there to quantify the closeness to a singular position (where the robot looses a degree of freedom due to constraints on the joints) - large condition numbers are an indication of low dexterity. When we transpose this interpretation to the reservoir, we can see that the maximization of σ_m is in fact a joint optimization of :

- a high gain of the system, thus ensuring good separation of the input signals in state space, and
- a small condition number, which means that the dynamical system is far from singularity and has many degrees of freedom.

These two quantities are in opposition: if the gain of the reservoir is too high, the nodes will start to saturate and the expressive power of the nonlinearity decreases, which means that the reservoir is constrained to a lower-dimensional subspace of the state space. This trade-off is clearly present in the measure presented here.

A notable disadvantage of this measure of the reservoir dynamics, as well as the other empirical measures that were described in (Legenstein and Maass, 2005) and (Schrauwen et al., 2008a), is the fact that they can only be computed through actual simulation of the reservoir. While this is in a sense unavoidable (because the inputs determine the trajectory and as such also the dynamical regime the reservoir operates in), one would like to have a method that eliminates the computational cost of simulating the reservoir. One possibility would be to develop measures that are based on

statistical properties of the distribution of the input signals, rather than actual instances of the input signals. The development of such measures are a good candidate for future research directions.

4.2.3 The link between dynamics and state distributions

The measures introduced above quantify the dynamic regime of the reservoir in its current operating point. However, these dynamics can also be linked to the distribution of the reservoir states. This is illustrated in Figure 4.8 for the basic case of tanh reservoirs. On the right hand side we have plotted the state distribution of the reservoir after tuning the spectral radius and feeding it with uniform noise in $[0, 1]$. If we tune the dynamics by scaling the spectral radius to the values of .2, 1 and 2 (top, middle and bottom plots), we can see this reflected in the distribution of the neuron states: for small spectral radius, the dynamics is too stable and the state distributions are centered around the linear area of the nonlinearity. For a spectral radius setting which is too large , the state distributions are centered around the nonlinear saturating areas of the nonlinearity, which results in an unstable reservoir and loss of expressive power because of the saturation. In the middle plot, the dynamic regime is 'just right', which is reflected in the distribution of the states: most of the states lie in the linear area of the tanh function, but there is still a good amount of nonlinear states.

This link between the dynamics and state distributions can be used to our advantage. If we can develop a way to adapt the state distributions of the neurons to a desired distribution, we have a method for adaptin the dynamics of the reservoir. This idea is built upon in the next section.

4.3 Adapting reservoir dynamics

In the first sections of this chapter, we have introduced and investigated ways of quantifying the dynamical regime of reservoirs. These measures are, at least theoretically, applicable to generic reservoir types since they can be used for many types of activation functions. Indeed: RC systems are not limited to networks of sigmoidal neurons. While the spectral radius still bears some meaning in the case of Echo State Networks as a linearization of the underlying nonlinear system around the zero fixpoint, it quickly looses significance when we start to study more intricate reservoirs with nonlinearities that do not have such nice behaviour around the

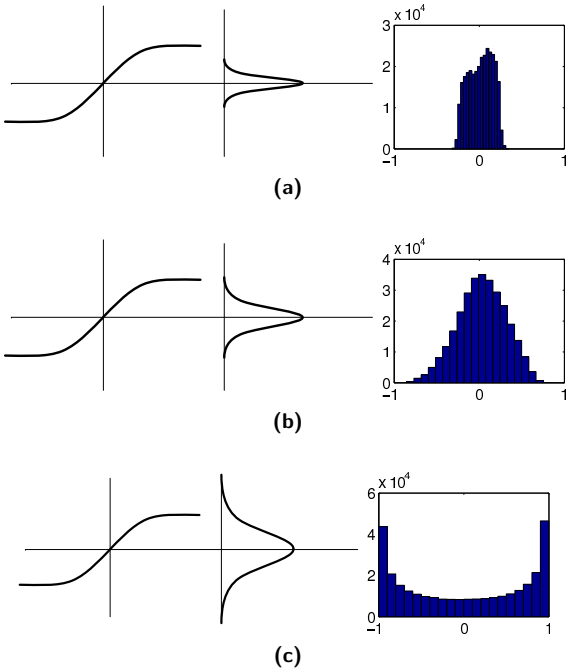


Figure 4.8: How dynamics affects the state distributions of the reservoir: in figure (a), the dynamics are too stable for most tasks and the distribution is quite narrow. In figure (b), the balance between linear and nonlinear components is just right, and in figure (c) the reservoir is too dynamic which causes the nodes to saturate and the reservoir to loose expressive power.

zero point. For these reservoirs, traditional tuning parameters such as spectral radius quickly become useless, and a new way of tuning the dynamics of these networks becomes necessary. Also, as mentioned before, the spectral radius is a network property while the actual dynamics of the reservoir are not only determined by the network weights but also by the dynamical range and evolution of the input.

In this Section, we move from quantifying the reservoir dynamics to actually controlling them. I will introduce a generalized version of an online learning rule that fine-tunes the dynamical properties of a reservoir in an unsupervised and biologically inspired way, but with a clear mathematical rationale underlying the rule, borrowing insights from information and communication theory. It is an improvement over the standard manual tuning of the spectral radius in two ways: it offers an automatic way of tuning the reservoir dynamics, and this tuning is input- or task-dependent, which means that the actual dynamical properties of the input signals are taken into account.

First, I will (very) briefly discuss some of the terminology that will be used in the chapter, and some concepts from information theory and their relationship to existing learning methods.

4.3.1 Information theory and learning

Information theory originated from the need for a mathematical foundation for quantifying properties of (digital) data transmission and communication channels. However, it has proven to be more broadly applicable. In particular, information theory has yielded some theorems and measures that provide insight into the fundamental properties of what information is, and more importantly - that quantify how much information data contains. The notion *information* is of course a rather vague term without a concrete definition - but this is precisely what information theory provides.

The seminal paper which started the field was written by Claude Shannon (Shannon, 1948). In it, the fundamental notion of *information entropy*, also known as Shannon entropy, was defined. The information entropy of a discrete stochastic variable X which can take n possible values is given by:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log(p(x_i)).$$

The information entropy $H(X)$ expresses the *uncertainty* associated with the random variable X. A variable for which all values are equiprobable

will have maximal information entropy - which matches our intuition: a variable for which all values are equally likely to occur has maximal uncertainty. The extension of entropy to continuous variables is given by the *differential entropy*, defined as :

$$h(x) = -\int_{-\infty}^{\infty} f(x)\log(f(x)),$$

but care should be taken when transposing interpretations of the discrete valued entropy to the differential entropy (for instance, differential entropy can become negative). These distinctions are however not important for the remainder of this discussion, since we will focus on the so-called *relative entropy* or *Kullback-Leibler divergence*.

The Kullback-Leibler divergence D_{KL} is a measure of the difference between two probability distributions. It is defined as:

$$D_{KL}(\tilde{p}, p) = \int_{-\infty}^{\infty} \tilde{p}(x) log \frac{\tilde{p}(x)}{p(x)} dx,$$

and is expressed in units of bits or nats, depending on the base of the logarithm used (2 or e, resp.). When expressed as bits, it can be interpreted as the number of additional bits needed to code instances from the distribution $\tilde{p}$ using instances of p, i.e. the amount of additional information needed to estimate $\tilde{p}$ based on samples from p. It is not a true metric in the mathematical sense, because it is not symmetric and it does not obey the triangle-inequality.

Using the entropy as an uncertainty measure, we can determine so-called *maximum entropy distributions.* These distributions are the distributions whose entropy is the largest of all distributions belonging to a class (where a class of distributions is defined, e.g., through constraints on the moments). Maximum entropy distributions are important because of two properties:

- Maximum entropy distributions minimize the amount of prior information incorporated in the distribution. This means that, given some instances of a stochastic variable or prior knowledge about that variable, a maximum entropy distribution taking this knowledge into account will make the least additional assumptions about that data. In a way this means that maximum entropy distributions are the most 'neutral' distributions.
- Maximum entropy distributions arise quite naturally in many physical - and thus also neurobiological - systems.

Constraints	Maximum entropy distribution	Expression	Entropy
$x \in [a, b]$	Uniform distribution over $[a, b]$	$f(x) = \begin{cases} \frac{1}{b-a} & \text{if } x \in [a, b] \\ 0 & \text{if } x \notin [a, b] \end{cases}$	$\ln(b - a)$
$\mu_x = \mu, \mu > 0$	Exponential distribution	$f(x) = \begin{cases} \frac{1}{\mu} e^{-\frac{x}{\mu}} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$	$1 - \ln(\frac{1}{\mu})$
$\mu_x = \mu$ and $\sigma_x = \sigma$	Gaussian distribution	$f(x) = \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}}$	$\ln \sqrt{2\pi\sigma^2 e}$

Table 4.1: The most common maximum entropy distributions corresponding to given constraints.

In Table 4.1 some examples of maximum entropy distributions are given for certain constraints on their moments.

These and related measures of information and entropy originated in information and communication theory, but have since also been used to develop various learning algorithms ranging from simple logistic regression, over an infomax-based algorithm for Independent Component Analysis (ICA) (Bell and Sejnowski, 1995) to even a whole family of rules captured under the term *Information theoretic learning* (Principe et al., 2000). These learning principles are all based on maximizing the mutual information between either the input to the system and the output of the system - resulting in an unsupervised rule - or between the output of the system and the desired output of the system - resulting in a supervised rule.

In this chapter, we will introduce, extend and study an unsupervised, bio-plausible learning rule called *Intrinsic Plasticity (IP)* for individual neurons in a reservoir, which is based on the information theoretic concepts introduced above.

4.3.2 An unsupervised adaptation rule for reservoirs

As was discussed in the introductory Chapter 1 of this thesis, various ways of constructing reservoir topologies and weight matrices have already been described in literature (Liebald, 2004; Hajnal and Lorincz, 2006; Maass et al., 2004b). As was already extensively argued above, the often assumed rule-of-thumb of setting the spectral radius close to 1 is not generally

applicable (Steil, 2006; Verstraeten et al., 2006) because the dynamical regime of the reservoir is also determined by the bias to the neurons and the dynamical range of the inputs. So, optimization of a reservoir for applications is typically based on experience and heuristics and partly on a brute-force search of the parameter space. Moreover, the variance of the performance across different reservoirs with the same spectral radius is still quite substantial, which is clearly undesirable. So, a computationally simple way to adapt the reservoirs to the task at hand without requiring a full parameter sweep or hand tuning based on experience would be welcome.

In this line, it was shown (Steil, 2007b; Wardermann and Steil, 2007; Steil, 2007a) that the performance of off-line and on-line learning for ESN networks with Fermi transfer functions[7] can be improved by using an unsupervised and local adaptation rule based on information maximization, called Intrinsic Plasticity (IP). This rule was first introduced in (Triesch, 2005) as a formal model of processes known in neurobiology as *homeostatic plasticity* - the term alludes to the self-regulatory behaviour of biological neurons (Turrigiano and Nelson, 2004). Since this kind of plasticity is present in almost all biological neurons, it seems natural to investigate its formal counterpart in combination with standard artificial network learning algorithms. For this intrinsic plasticity rule, Triesch has also shown that in combination with Hebbian learning IP can drastically change the behavior of Hebbian networks towards finding heavy-tail directions in arbitrary input distributions (Triesch, 2007). The interplay between these two unsupervised adaptation rules was further investigated in (Lazar et al., 2007). There, it was shown that the combination of IP and STDP - which is a Hebb-like rule for spiking neurons - enhances the robustness of the network against small perturbations and aids the networks in discovering temporal patterns present in the input signals. The results from that work suggest that a fundamental underlying link between IP, STDP and unsupervised information maximization exists, and that these rules operating strictly on the local neuron-level succeed in steering the dynamics of the entire network towards a computationally desirable regime.

In this Section, we investigate different versions of IP learning further, extending and generalizing the idea. Whereas previous work on IP has focused on the Fermi transfer function and an exponential target distribution, we derive a more general formalism here that is independent of the neuron's activation function or the desired output distribution. This allows the formalism to be applied to more general reservoir types. We

[7] As mentioned in subsection 1.3.2, the Fermi or logistic function is given by $y = \frac{1}{1+\exp(-x)}$.

use this generalized IP rule to derive a version for tanh neurons with a Gaussian output distribution. Simulations show that in practice these targets are reached surprisingly well despite the interference between neurons introduced by the recurrency in the network.

The simple, local IP rule effectively makes reservoirs significantly more robust: it gives the reservoirs the ability to autonomously and robustly adapt their internal dynamics, irrespective of initial weight setting, input scaling or topology, to a dynamical regime which is suited for the given task. The rule is purely input driven, and adapts the reservoir in an unsupervised way. Moreover, since it can be used for many activation functions (not only standard sigmoids), this opens the way for an unsupervised adaptation mechanism for novel reservoir types.

4.4 Towards *generalized* Intrinsic Plasticity

Intrinsic Plasticity (IP), as introduced in (Triesch, 2005), models a well-known phenomenon called *homeostatic plasticity,* which is observed in a variety of biological neurons*:* these *wet* neurons tend to autonomously adapt to a fixed average firing rate for physiological reasons. In (Triesch, 2005) it was shown that such a mechanism, when the neurons have an exponential output firing rate distribution, are effectively maximizing information transmission when the neurons are interpreted as communication channels. While the biological mechanisms are not yet known precisely, it is very plausible that every single neuron tries to balance the conflicting requirements of maximizing its information transmission while at the same time obeying constraints on its energy expenditure. IP formalizes these hypotheses by incorporating the following three principles:

1. *information maximization*: the output of the neuron should contain as much information on the input as possible. This is achieved by maximizing the entropy of the output firing rates;

2. *constraints on the output distributions*: these are first of all the limited output range of the neuron (Atick, 1992), but can be also the limited energy available (Baddeley et al., 1997);

3. *local adaptation*: a biological neuron is only able to adjust its internal excitability, and not the individual synapses (see (Zhang and Linden, 2003), (Destexhe and Marder, 2004) for a discussion of this common effect in biological neurons).

Following Triesch' original line of argument, the information maximization principle corresponds to a maximization of the entropy of the output

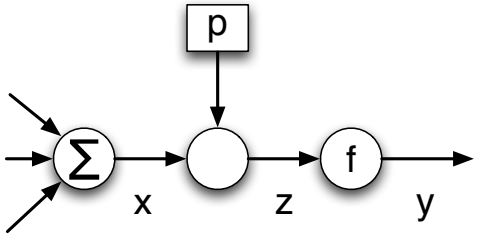


Figure 4.9: Schematic view of the generalized neuron model used for deriving IP. The nonlinearity is indicated with f, and p is an additional neuron parameter that will be tuned by the IP rule.

distribution of each neuron. In combination with the second principle this leads to maximum entropy (ME) distributions with certain fixed moments. It is known that the ME distribution for a given mean (first moment) and support in the interval $[0, \infty]$ is the exponential distribution. Likewise, the ME distribution for a given mean and standard deviation with support in $[-\infty, \infty]$ is the Gaussian.

We focus here on distributions of the exponential family because these tend to have much of the probability density centered around zero. This means that there is a high probability that many of the neuron activations will be close to zero. This is very related to the biologically important concept of sparse codes. Sparse codes seem to be present throughout the mammalian brain, and represent a compromise between a local code – which enables fast learning but has low generalization and error tolerance – and a dense code – where learning is slow but which has a good representational capacity and is robust against errors (Foldiak and Young, 1995).

4.4.1 Derivation of the generalized rule

The target of the Intrinsic Plasticity learning rule will be to drive the distribution of the output (activation) values of a single neuron to a certain ME distribution. This will be done by minimizing the Kullback-Leibler divergence D_{KL} of the actual output distribution of the neuron, $h_y(y)$, w.r.t. a certain target distribution $h(y)$. Here, we will derive a general expression for the derivative of D_{KL} w.r.t. a generic parameter p of the activation function, independent of the activation function of the neuron or the target distribution. The parameter p can be, e.g., the gain or bias of a neuron. This structure is shown schematically in Figure 4.9.

From this general expression, it is then quite easy to instantiate IP

rules for specific activation functions and/or target distributions. We will use a neuron with activation function f and input z, such that its activation value is given by $y = f(z)$. To avoid clutter, we will denote $\frac{df}{dz} = f'$ and $\frac{d^2f}{dz^2} = f''$. We will further assume that z is a parametrized function of the actual neuron input x.

We start from the definition of D_{KL}:

$$\begin{aligned} D_{KL}(h_y||h_t) &= \int h_y(y) \log\left(\frac{h_y}{h_t}\right) dy \\ &= \int h_y(y) \log\left(h_y(y)\right) dy - \int h_y(y) \log\left(h_t(y)\right) dy \\ &= E\left(\log\left(h_y(y)\right) - \log\left(h_t(y)\right)\right), \end{aligned}$$

where $E(\cdot)$ denotes the expected value. Derivation of D_{KL} w.r.t. p gives:

$$\frac{\partial D_{KL}}{\partial p} = E\left(\frac{\partial}{\partial p}\log\left(h_y(y)\right) - \frac{\partial}{\partial p}\log\left(h_t(y)\right)\right).$$

If f is strictly monotonous and increasing[8], we have that $h_y(y) = \frac{h_z(z)}{f'(z)}$, with $h_z(z)$ the probability density function (pdf) of z, so this gives:

$$\begin{aligned} \frac{\partial D_{KL}}{\partial p} &= E\left(\frac{\partial}{\partial p}\log\left(\frac{h_z(z)}{f'(z)}\right) - \frac{\partial}{\partial p}\log\left(h_t(y)\right)\right) \\ &= E\left(\frac{\partial}{\partial p}\log\left(h_z(z)\right) - \frac{\partial}{\partial p}\log\left(f'(z)\right) - \frac{\partial}{\partial p}\log\left(h_t(y)\right)\right) \\ &= E\left(\frac{\partial}{\partial p}\log\left(h_x(x)\right) - \frac{1}{\frac{\partial z}{\partial x}}\frac{\partial^2 z}{\partial p \partial x} - \frac{f''(z)}{f'(z)}\frac{\partial x}{\partial p} - \frac{h_t'(y)}{h_t(y)}f'(z)\frac{\partial z}{\partial p}\right) \\ &= -E\left(\frac{1}{\frac{\partial z}{\partial x}}\frac{\partial^2 z}{\partial p \partial x} + \frac{f''(z)}{f'(z)}\frac{\partial z}{\partial p} + \frac{h_t'(y)}{h_t(y)}f'(z)\frac{\partial z}{\partial p}\right). \end{aligned}$$

We now have a general expression for the derivative of D_{KL} w.r.t. the neuron parameter p, expressed in function of first and second derivatives of the activation function f and the target pdf h_t. This allows us to derive a learning rule that minimizes this quantity using stochastic gradient descent:

$$\Delta p = -\eta \frac{\partial D_{KL}}{\partial p},$$

whereby we approximate the expected values by the instantaneous values.

[8] If f is not monotonous, the input domain can be split in intervals where f *is* monotonous and the rule can be derived for each of these intervals. In case f is decreasing, $h_y(y) = -\frac{h_z(z)}{f'(z)}$ should be used.

4.4.2 Specific IP rules for Fermi and tanh neurons

Based on the result obtained above, we can now construct individual learning rules by approximating the expectation operator using the instantaneous values, and applying stochastic gradient descent as done by Triesch et al. For instance, in the case of the original IP rule introduced by Triesch et al in (Triesch, 2005), the target pdf is the exponential distribution: $h_t = \frac{1}{\mu} e^{-\frac{y}{\mu}}$, so $\frac{h_{t'}}{h_t} = -\frac{1}{\mu}$. The activation function is the Fermi function, given by: $y = f(x) = 1/(1 + e^{-z})$, so $f' = y(1 - y)$ and $\frac{f''}{f'} = 1 - 2y$. There are update rules for the neuron gain a and bias b, defined as: $z = ax + b$, with x the weighted sum of the neuron inputs. So, we have: $\frac{\partial z}{\partial a} = x$,$\frac{\partial z}{\partial b} = 1$ and $\frac{\partial z}{\partial x} = a$. When we fill in a and b as parameters p in the general equation derived above, this gives:

$$\frac{\partial D_{KL}}{\partial a} = -E\left(\frac{1}{a} + (1 - 2y)x - \frac{1}{\mu} y(1 - y)x\right)$$
$$\frac{\partial D_{KL}}{\partial b} = -E\left((1 - 2y) - \frac{1}{\mu} y(1 - y)\right),$$

which yields precisely the update rules described by Triesch:

$$\Delta a = \eta\left(\frac{1}{a} + x - \left(2 + \frac{1}{\mu}\right) xy + \frac{xy^2}{\mu}\right)$$
$$\Delta b = \eta\left(1 - \left(2 + \frac{1}{\mu}\right) y + \frac{y^2}{\mu}\right).$$

Similarly, we can derive an IP rule for tanh neurons with a Gaussian target distribution. Here, $h_t(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right)$, so $\frac{h_{t'}}{h_t} = -\frac{(y-\mu)}{\sigma^2}$. For the activation function we have $y = \tanh(z)$, so $f' = 1 - y^2$ and $\frac{f''}{f'} = -2y$. Filling this in gives:

$$\frac{\partial D_{KL}}{\partial a} = -E\left(\frac{1}{a} - 2xy - \frac{y - \mu}{\sigma^2}(1 - y^2)x\right)$$
$$\frac{\partial D_{KL}}{\partial b} = -E\left(-2y - \frac{y - \mu}{\sigma^2}(1 - y^2)\right),$$

which gives the following update rule:

$$\Delta a = \eta \left(\frac{1}{a} + \frac{\mu x}{\sigma^2} - \frac{xy}{\sigma^2}(2\sigma^2 + 1 - y^2 + \mu y) \right)$$
$$\Delta b = \eta \left(\frac{\mu}{\sigma^2} - \frac{y}{\sigma^2}(2\sigma^2 + 1 - y^2 + \mu y) \right).$$

Note that for both learning rules $\Delta a = \frac{\eta}{a} + \Delta b x$. This relation between gain and bias update steps is independent of the activation function or output distribution and gives the opportunity for some computational optimization.

4.4.3 The effects of IP on the neuron parameters and weight distributions

In this subsection, we will discuss the effects of IP on the distribution of the adjusted parameters and weights of the reservoir. For illustratory purposes, we will study the case of a tanh reservoir, initially scaled to a (generally suboptimal) spectral radius of 1.5, with IP adjustment of the gain a and bias b in the case of a gaussian target distribution with a target standard deviation of 0.2 and zero mean (which is the optimal setting for the experiments discussed below).

Since the gain a of each neuron is changed by IP, this in effect changes the weight matrix of the reservoir by scaling the input weights to each neuron with the corresponding gain term. We can therefore easily compute the resulting effective weight matrix due to the IP adaptation by multiplying each row of the reservoir weight matrix with the corresponding neuron gain. The eigenvalue distribution of the resulting effective weight matrix before and after IP are shown in Figure 4.10. This shows that the effective weight matrix is tuned to a spectral radius of close to 1, which shows that tuning the output distributions also tunes the dynamical regime of the reservoir.

Figure 4.11 shows the evolution of four reservoir properties during the application of IP: the mean and standard deviation of the neuron states is shown in the top plots, and the minimal singular value of the reservoir Jacobian and the spectral radius of the effective weight matrix (taking the changed gain into account) is shown in the bottom plots. The state mean is not affected by the IP since we require a zero mean state distribution. However, the standard deviation is brought down from an initial value of 0.5 to close to the desired target standard deviation of 0.2. There is a slight undershoot of the actual standard deviation which is due to the fact that the state distributions are truncated in the interval [-1,1]. We will discuss this phenomenon further below.

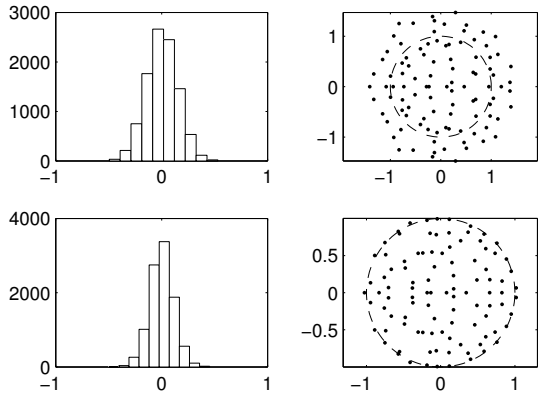


Figure 4.10: The left hand side shows a histogram of the reservoir states without (top plots) and with (bottom plots) IP adaptation. On the right hand side, the corresponding eigenvalue distribution of the resulting weight matrix is shown.

The bottom plots show that the minimal singular value of the Jacobian – which we discussed above as a measure of the reservoir dynamics – is effectively increasing due to the IP rule. This confirms the claim that IP tunes not only the state distributions but also the effective dynamic regime of the reservoir. This is also shown in the bottom right, showing the spectral radius of the effective weight matrix: it drops from the initial value of 1.5 to a much more suitable value of slightly above 1. Of course, the resulting effective spectral radius depends on the setting of the desired standard deviation. We will discuss the link between the standard deviation of the target distribution and the effective spectral radius in more detail in the section on experimental results below.

4.4.4 Limitations of the assumptions

During the derivation of the general IP rule above, there are two implicit assumptions being made that are not applicable without certain cautionary remarks. The first assumption is the fact that the output distribution of the neuron's activation can be unbounded. This is necessary because the desired distributions considered here (the exponential and Gaussian) have an infinite support. However, in practice this can never be the case due to the boundedness of the activation functions: their absolute value is never larger than one. This of course has ramifications for the accuracy with which the target distributions can be approximated. In particular, the actual mean and standard devations of the output distributions

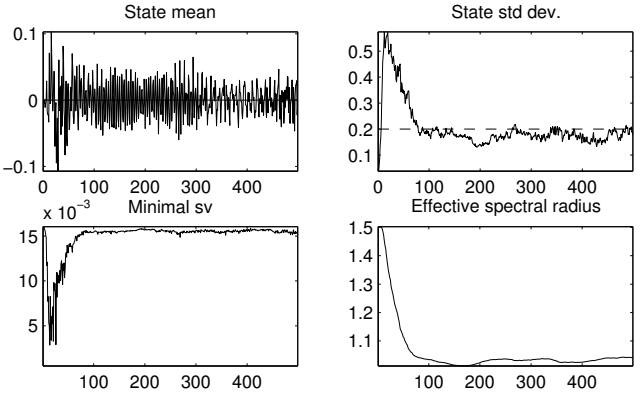


Figure 4.11: The effects of IP on the mean and standard deviation of the reservoir states, the minimal singular value of the jacobian and the effective spectral radius. On all the plots, the horizontal axis represents time.

will differ from the desired ones. However, it turns out to be possible to compute this difference, based on truncated versions of the target distributions. This discrepancy is further studied in (Schrauwen et al., 2008b).

The second assumption that does not hold when applying IP to reservoirs is the fact that all the derived rules are based on assuming independence of the neurons' input distributions. It is however not obvious that this assumption holds in recurrent networks (in fact, quite probably it doesn't), where neurons are coupled and the output of a neuron indirectly influences its input. Also, the limited number of parameters with which to tune the transfer function, might hinder the convergence of the actual output distribution to the desired one. Still, for a single neuron it was shown in (Triesch, 2005) that an exponential distribution can be approximated very well by only adjusting the bias and gain of the neuron. Similarly it was shown in (Schrauwen et al., 2008b) that for recurrently coupled neurons the actual distribution approximates the desired distribution very well, given the edge effects discussed earlier. This is illustrated in Figure 4.12.

It has already been shown qualitatively in a previous publication by Steil (such as (Steil, 2006)) that even though the original IP learning rule actually only tunes the temporal distributions of single neurons, when they are coupled together in a recurrent network, the same distribution can be perceived spatially (even after a single time step if the reservoir is large enough). A sparse temporal distribution thus results in a sparse spatial code. However, it has not yet been proven theoretically that this ergodic property always holds.

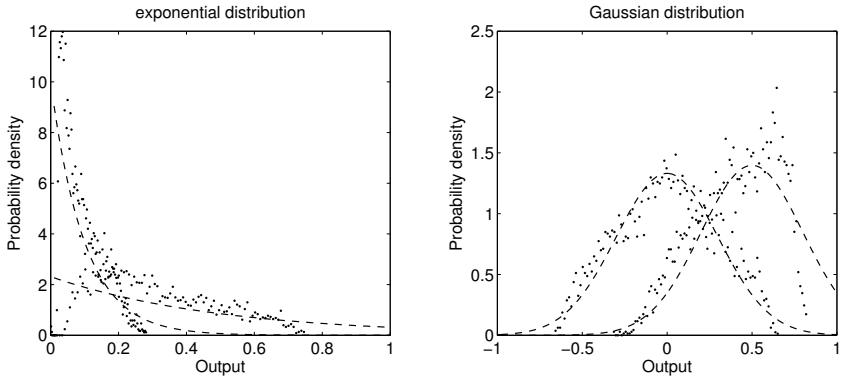


Figure 4.12: Comparison between the desired distributions (dashed lines) and the actual distributions of the neurons after IP. Note the edge effects that prevent the activations to reach high values.

4.5 Experiments

In this section, we will study the impact of both exponential and Gaussian IP on the performance of RC on several benchmark applications. We have already discussed the limited usefulness of the spectral radius for non-tanh reservoirs. There is furthermore a quite large variance on the performance when creating several random reservoirs with the same spectral radius. And finally, the spectral radius stability measure cannot be used on generic reservoir nodes or very constrained topologies. We will show that IP can help with regard to all three issues.

4.5.1 Preliminaries

The three benchmarks we use in this section to evaluate the performance of the different IP rules are chosen to span a wide range of desired features we expect from reservoirs: the 30th order NARMA system identification task, the memory capacity task and the isolated digit recognition task. For the experiments we will use the following extension of a standard ESN setup (see Section 2.1.1), which takes the additional gain parameter a and per-neuron bias b into account. The reservoir states at time step k, $\mathbf{x}[k] \in \mathbb{R}^{N\times 1}$, are calculated by

$$\begin{aligned}\mathbf{x}[0] &= \mathbf{0}\\ \mathbf{x}[k+1] &= \mathbf{f}(\mathbf{W}^{res}_{res}\mathbf{x}[k] + \mathbf{W}^{res}_{inp}\mathbf{u}[k], \mathbf{a}, \mathbf{b}),\end{aligned}$$

where N is the number of network nodes, $\mathbf{W}_{res}^{res} \in \mathbb{R}^{N \times N}$ is the matrix of network weights, $\mathbf{f}(.)$ is the vector-valued version of the generalized transfer function $f(.)$, $\mathbf{a}, \mathbf{b} \in \mathbb{R}^N$ are the vectors of gain and bias parameters in $\mathbf{f}(.)$, M is the number of external inputs to the network at each time step, $\mathbf{u}(t) \in \mathbb{R}^{M \times 1}$ the external input at time step t and the weights connecting these to the reservoir nodes $\mathbf{W}_{inp}^{res} \in \mathbb{R}^{N \times M}$. Note that above equation can be rewritten as

$$\begin{aligned} \mathbf{x}[k+1] &= \mathbf{f}(\hat{\mathbf{W}}_{res}^{res}\mathbf{x}[k] + \hat{\mathbf{W}}_{res}^{inp}\mathbf{u}[k] + \mathbf{b}, \mathbf{1}, \mathbf{0}) \\ \hat{\mathbf{W}}_{res}^{res} &= \mathrm{diag}(\mathbf{a})\mathbf{W}_{res}^{res} \\ \hat{\mathbf{W}}_{inp}^{res} &= \mathrm{diag}(\mathbf{a})\mathbf{W}_{inp}^{res}. \end{aligned}$$

This technique allows us to compare reservoirs that have been adapted by IP directly with networks which have not been adapted. We will call the spectral radius of $\hat{\mathbf{W}}_{res}^{res}$ after applying IP the *effective spectral radius* ρ_{eff}.

While it is possible to add linear terms from the input to the output, we chose not to do this because we want to evaluate the influence of IP on the reservoir, and adding the linear terms can cloud the effects of IP. The training of the readout layer is done using ridge regression (see Subsection 2.1.3.1), and the optimal regularization constant λ was determined through linear searching using five-fold cross-validation.

Some parameters of the network are kept fixed over all experiments. Network size is always 100 - this could have been optimized, but we care here just for comparison between the two techniques. The $\mathbf{W}_{res}^{res}$ matrix is always created by initializing all the weights uniformly distributed between -1 and 1, which is then scaled to a given spectral radius, whereas the input weights are set with equal probability to -.1 or .1. The IP parameters $\mathbf{a}$ and $\mathbf{b}$ are initialized to $\mathbf{1}$ and $\mathbf{0}$, respectively.

All results in plots and tables are averages and standard deviations over 30 different reservoir instantiations with the same parameter settings. For training the linear readout, the first 100 time steps of the reservoir dynamics for every presented example were disregarded to allow the network to 'warm up' (forget its initial state) sufficiently.

For pre-training a reservoir, the IP rule is applied with a learning rate η_{IP} of 0.0005 for 100000 time steps. To check whether IP has had sufficient time to adapt after this time, we verified that $\mathbf{a}$ and $\mathbf{b}$ had converged to small regions and compared the expected probability density with the one estimated from the reservoir's output.

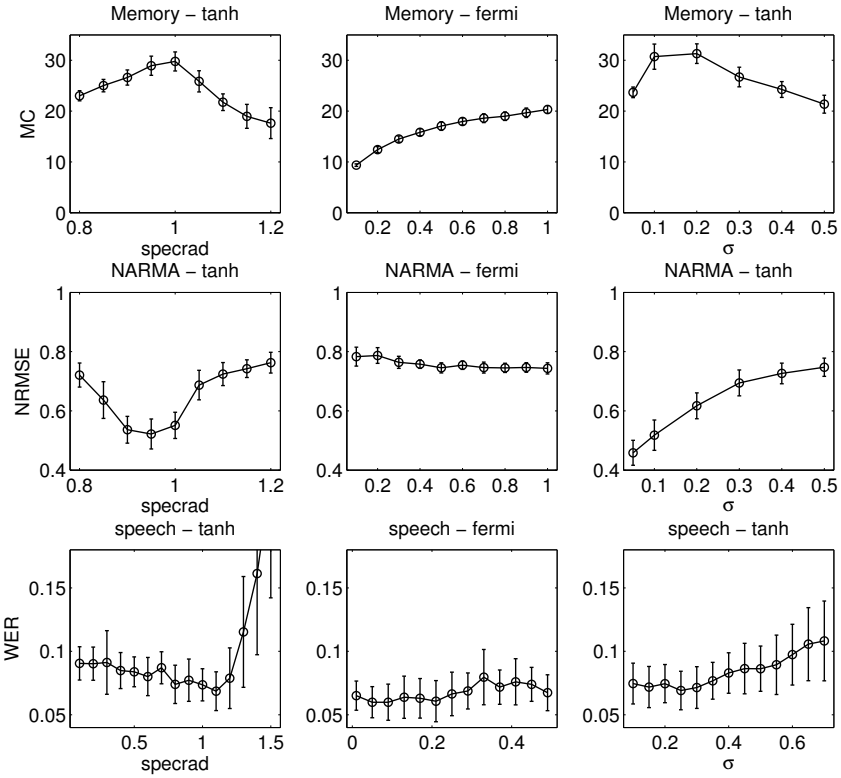


Figure 4.13: Results for all three benchmarks for tanh with spectral radius ranging (left column), exponential IP for Fermi nodes (middle column) and Gaussian IP for tanh nodes (right column).

4.5.2 Results

The three benchmark experiments are now evaluated using different experimental setups. We first use the standard way of scaling reservoirs by ranging over the spectral radius. We do this for both Fermi and tanh nodes. Next, we evaluate the use of IP for Fermi nodes with an exponential distribution (ranging over the mean μ), and tanh nodes with a Gaussian distribution (ranging over the variance σ). For the Gaussian distribution we only use $\mu = 0$ since for other values, experiments show a considerable performance drop.

The results are shown in Figure 4.13, where the first row shows the results for the memory task, the second row shows the results for the NARMA task, and the bottom row shows the result for the isolated speech. The columns show the effects of ranging the spectral radius for

Table 4.2: Best results for the three different benchmarks. IP is slightly better than ranging the spectral radius, both in average performance as in standard deviation (except for one case), denoted between brackets. Some tasks perform better with a Gaussian distribution, others with an exponential distribution.

	Fermi-specrad	tanh-specrad	Fermi-exp. IP	tanh-Gauss. IP
MC	17.41 (1.48)	29.78 (1.87)	20.32 (0.77)	**31.31 (1.93)**
NARMA	0.77 (0.012)	0.52 (0.050)	0.74 (0.019)	**0.46 (0.042)**
Speech	0.070 (0.018)	0.069 (0.015)	**0.060 (0.012)**	0.069 (0.015)

the tanh node type, exponential IP for Fermi nodes, and Gaussian IP for tanh nodes. The results for ranging the spectral radius for the Fermi nodes are not shown since the performance of this reservoir type is very poor. This is expected since the spectral radius is not a good measure for the dynamical regime when using Fermi nodes. The best values for all settings are summarized in Table 4.2 where the results of ranging the spectral radius for the Fermi nodes are added as a reference.

Memory The best achievable memory capacity of reservoirs without IP is quite different for networks using tanh nodes and those using Fermi nodes (see Table 4.2). Note that for memory capacity, higher is better. While tanh networks can have a capacity of up to 31, Fermi networks have little more than half that memory, 17. In tanh networks, the drop in performance for spectral radii smaller than optimal is not drastic. When using IP, the largest change in results can be found in Fermi networks. While the best achievable memory capacity increases by one-fourth, i.e. 4, the variance of the results for one and the same parameter setting, i.e. μ, gets very small, only 0.77, contrasting to setting the spectral radius, where this was 1.48. In tanh networks, the difference between using spectral radius scaling and IP pre-adaptation was not as pronounced. The best memory capacity achievable there, also increased by using IP from 29.78 to 31.31, but the variance is a little bit worse if using IP. Furthermore, the performance drop for suboptimal parameter settings is equally pronounced for σ as for spectral radii.

NARMA With tanh networks, the best error achieved was 0.52, at a spectral radius of 0.95. For spectral radii of smaller than 0.9 or larger than 1, performance drops considerably. When using Gaussian IP, the performance improves considerably with up to 10%. But, in contrast to the memory task, where the performance drop looks qualitatively similar for changes in spectral radius and σ, the

optimal value of σ is very small, and the performance decreases steadily for increasing values of σ. Fermi node networks perform very poorly on this task. When using exponential IP, the performance slightly increases, but is still considerably worse than tanh nodes. Interestingly, if μ was chosen larger than 0.3, its value did not have any notable impact on the performance, and the variance dropped to very small values of 0.019.

Speech Using IP slightly improves both the mean and standard deviation of the performance for both exponential and Gaussian distributions. For this task, the exponential distribution seems to perform best. Another effect is the extreme drop in performance when ranging the spectral radius to high values, where the reservoir becomes unstable. When using IP, this instability is never reached.

When using Fermi neurons, and imposing an exponential distribution, the optimal settings for μ differ for the three different tasks. Increasing μ increases the linear memory present in the network, i.e. the best performance could be observed for $\mu = 0.1$. In NARMA, the opposite was true: up to $\mu = 0.1$, the variance of results was too high to be usable.

But when using tanh neurons and imposing a Gaussian distribution, for σ, the optimal setting was almost the same for both tasks, besides the performance drop in memory capacity for $\sigma = 0.1$, smaller settings were better.

An interesting observation can be made from comparing optimal spectral radius and the effective spectral radius of the optimal σ: where the σ was optimal, the effective spectral radius was equal to the optimal spectral radius of a network without IP (see Figure 4.14). Notice the relatively small variance of the relation between moments and effective spectral radius shown in this figure. Imposing certain output distributions on the nodes of the reservoir is thus actually a precise way of controlling the dynamics in the reservoir. Note that the effective spectral radius for Fermi nodes can not at all be related to those of tanh neurons, which are normally used. For tanh neurons, we see that when varying σ over its complete range, we actually vary the effective spectral radius in its most important range: between 0.8 and 1.1.

Note that IP and spectral radius influence each other in two ways. Firstly, the initial scaling of the weight matrix can alter the learning behavior of IP, because the adjustment factors of the intrinsic parameters by the rule depend on the amount of activity present in the network. Secondly, changing the gain of the transfer function corresponds to scaling all incoming weights, and therefore changing the spectral radius of the weight matrix. Thus, the effective spectral radius after applying IP will

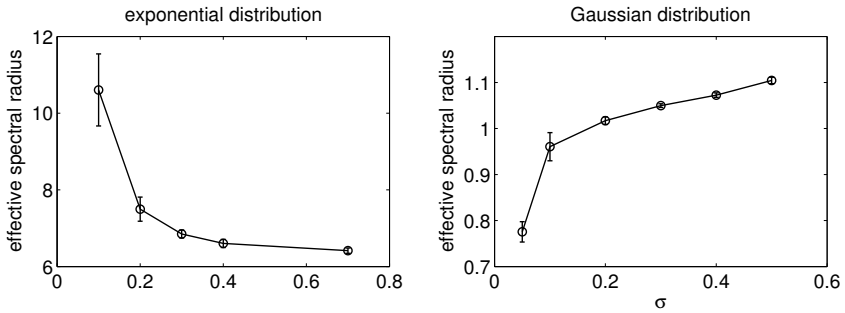


Figure 4.14: These plots show the relation between the mean and standard deviation of the exponential and Gaussian distribution respectively, and the effective spectral radius which is attained after pre-training the reservoir using IP. These plots are generated from the Memory Capacity task, but look almost identical for the other tasks. This shows that there is a clear relation between moments and effective spectral radius, which is task independent.

be different from the one the network was initialized to.

It is known that the memory capacity is highest for linear reservoirs that have a spectral radius close to one (Jaeger, 2001b). We see a similar effect when using IP: the best performance is attained for Gaussian distributions with small variance, where the bulk of the dynamics is thus in the linear part of the tanh nonlinearity. The optimal variance is $\sigma = 0.2$ which we can clearly relate to an effective spectral radius of 1 in Figure 4.14. The NARMA task also seems to prefer reservoirs that for the most part operate in their linear regime. This can be explained due to the relatively large memory that is needed to solve the 30th-order NARMA task. The speech task on the other hand appears to be a task that can take advantage of the non-linearities when operating the bulk of the dynamics in the non-linear part of the Fermi neurons due to the exponential distribution.

Note that for Fermi node networks, only IP with exponential target distribution is studied here. When not pre-adapting Fermi networks, the performance is always very bad. But, when using IP pre-adaptation, Fermi neurons can already get the best performance for the speech task. The bad performance on the memory and NARMA task might suggest that the Fermi neurons are intrinsically flawed. But when pre-adapting Fermi nodes with Gaussian IP, which is possible, but not thoroughly investigated in this work, the results seem to be qualitatively similar to the ones achieved with tanh node networks. The above results thus mainly relate to the distributions, and not to the node types.

In previous work of Steil (Steil, 2006, 2007a) a good performance was

achieved for Fermi nodes with exponential IP on a related NARMA task. This was however accomplished by additional features in the overall reservoir architecture that were deliberately left out in this work to be better able to discern the actual influence of the reservoir: in the cited work not only the current time step was fed to the reservoir, but also a time delayed version of the input, and both these inputs were also directly fed to the linear output. Since the NARMA task depends on a long input history, and has a large linear component, these two features allow the reservoir to perform better. The optimal value for μ in this setup is actually quite low, which suggests that the reservoir only has to solve the non-linear part of the NARMA task, while the linear, time delayed connection takes care of the linear part. When the reservoir has to solve all these parts by itself, we see that the exponential distribution is not the best option since it is not able to generate a long enough memory of its input, as is suggested by the results on the Memory Capacity task.

One important remark to be made is that the tasks considered here are tasks that require reservoirs that operate close to the edge of stability. This is confirmed when looking at the results for scaling the spectral radius in the case of tanh reservoirs: for all three tasks the optimal spectral radius lies close to 1. For these tasks, this means that IP tunes the reservoir into the desired dynamical regime. However, some tasks require reservoirs that operate in a different dynamical regime, far away from the edge of stability. This is for instance the case in the multi-stable switching task described in (Jaeger, 2001b). Still, the majority of the tasks where Reservoir Computing performs on par with or better than other techniques require reservoirs at the edge of stability.

4.6 Constrained topologies

When constructing reservoirs for RC, in most cases completely random connectivity matrices are used. These type of topologies have very good reservoir properties because the dynamical regime can be easily changed by globally scaling all the weights up or down. Due to this scaling, the dynamical regime can be precisely and progressively varied from very damped to highly unstable dynamics. The topologies are however not easy to be implemented on, e.g., a planar substrate such as a silicon chip.

When we look at very constrained topologies such as 1D and 2D lattices, which are the easiest topologies to implement on planar substrates, they behave very badly as reservoirs: for example a 1D lattice where the input is only fed to one of the nodes (see Figure 4.15) will have a very sudden transition from order (only the node which is fed with the input

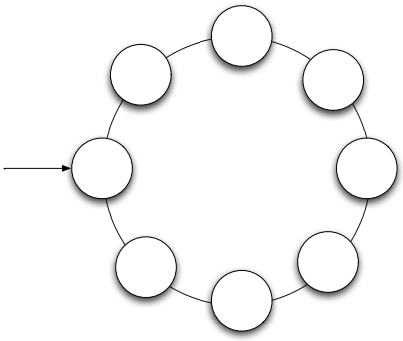


Figure 4.15: Example ring topology, where each node is connected to its nearest-neighbours, and the input is only fed to a single neuron.

is active, and the activity of all the other nodes is orders of magnitude smaller) to a wildly oscillating reservoir when the weights are scaled up. The boundary of stability is very narrow in this case. These topologies can thus not easily be used in the usual sense when just globally scaling the weights.

We will now demonstrate that imposing distributions on the reservoir dynamics is the key to using these constrained topologies as reservoirs in a very robust and reproducible way. We repeated the memory capacity and NARMA task with the same settings as in the previous section, but now using the constrained topology shown in Figure 4.15 which is a 1D ring lattice consisting of 50 neurons that are only connected to their nearest neighbors. For this experiment we only look at tanh nodes and IP with a Gaussian distribution. The results can be seen in Figure 4.16. We did not do this experiment for the speech task, since this task has multiple inputs (one input for each frequency component in the cochlear model), and in this experiment we only want to evaluate if IP is able to create usable dynamics in a ring topology where only a single neuron initially has some activity.

We first evaluate the performance of this ring topology when just scaling the spectral radius. For both the memory capacity and the NARMA task, scaling the spectral radius performs very poorly. Especially when scaling up the reservoir for the NARMA task, we see a drastic increase in error and variance of error, which is due to the unstable regime that is reached.

When using Gaussian IP to pre-adapt this ring topology, we see an increase in performance for both the memory capacity and NARMA task. This clearly demonstrates that IP is very capable of enforcing a desired

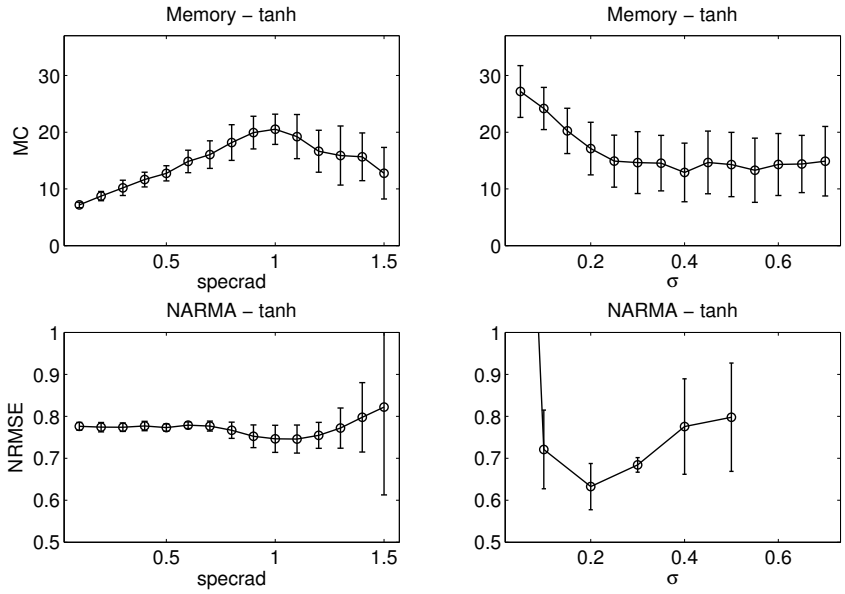


Figure 4.16: Results for a ring topology, on the left for ranging across the spectral radius without IP, on the right for adjusting the desired standard deviation with IP.

dynamical regime on a reservoir, even if its topology is very constrained. Using this IP rule thus allows to pre-adapt simple, constrained topologies prior to hardware implementation. Due to the pre-adaptation, these very sparsely and regularly connected reservoirs can be actually used to perform tasks in an efficient way.

4.7 Conclusions

The transition from neural implementations of reservoirs to more exotic excitable media is not trivial. We are dealing with nonlinear dynamical systems which are known to exhibit a wide variety of possible behaviours. The engineer or researcher who wants to apply RC to a novel reservoir implementation needs tools that offer at least some guidance as to which reservoir or which parameter setting is suitable for computational purposes. While guidelines based on stationary measures such as the spectral radius are useful for standard reservoirs, these methods break down when, e.g., non-standard activation functions are used or when the input signal drives the reservoir into a very different dynamical regime.

In this chapter, we have introduced and investigated metrics for mea-

suring the dynamical excitability of the reservoir. This excitability changes as the reservoir is driven by external inputs, and these metrics take this into account by looking at the reservoir at the current point in the trajectory. We have shown that for the tasks we considered, they are a good predictor of performance. Moreover, we have given an interpretation of the measure that offers more insight into the functionality of the reservoir, showing that a trade-off is made between the excitability of the reservoir and its 'degrees of freedom' in state space.

Next, we have presented and derived a generalization of a reservoir adaptation rule called IP that maximizes the entropy of the reservoir states w.r.t. certain constraints on the moments. We have shown for two instantiations that the adaptation is capable of driving the neurons in larger networks to the theoretically derived renormalized mean and variance of the desired truncated exponential and Gaussian output distributions. An important effect of IP is that it makes it possible to use node types and topologies which normally perform very poorly as reservoir. The very special ring-topology can, through the use of IP, also be used as a real reservoir. This has consequences for implementations of reservoirs in hardware: for special reservoir types such as delay coupled systems consisting of a delay line with nonlinear processing elements, application of IP can aid to tune the system into the desired regime.

The idea of autonomously regulated robustness of dynamics is a powerful concept. It was shown that a certain dynamic regime in reservoirs leads to good performance for a given task. The IP rule allows the reservoirs to autonomously perceive and adapt their dynamics to this specific regime, irrespective of disturbances, initial weights or input scaling. Reservoirs were already robust in the sense that the performance variance for random reservoirs is small. This robustness even improves when adding IP, since reservoirs can now autonomously evolve towards the correct dynamic properties. This sets the stage for automatic tuning of novel RC architectures with more complex or exotic nonlinearities than the traditional ESN implementations. Moreover, for some (e.g., hardware) reservoir types it may be useful to incorporate these adaptation mechanisms into the reservoir to allow online tuning of the reservoir dynamics.

5 Conclusions and perspectives

Here we will summarize the main contributions of this thesis. Next, we draw some global conclusions and also list some possible avenues for future research in this area.

5.1 Summary

In this work, I have presented experimental evidence to support the case that Reservoir Computing is a novel method for computing with time series. I started by showing the broad applicability and impressive performance of Reservoir Computing by presenting and discussing several difficult temporal processing tasks studied at our lab, both academic and real-world problems - such as speech recognition and epileptic seizure detection.

The claim that a reservoir can be seen as a generic nonlinear excitable medium, beyond the original neural and bio-inspired implementations, was illustrated by presenting an experimental overview of several standard and non-standard reservoir implementations, ranging from Echo State Networks and Liquid State Machines over bandpass reservoirs to implementations on an analog CNN VLSI chip and reservoirs built from nano-photonic nodes. We have shown that each of these reservoir types is suitable for doing nonlinear computation on timeseries. This establishes RC as a novel, non-Turing computational paradigm, whose fundamental principles can be applied to physical and biological systems, either by studying them from this point of view, or by actively constructing reservoir-based systems.

Starting from the idea that reservoirs are nonlinear dynamical systems, I then discussed several stationary metrics for quantifying the dynamical

regime of standard reservoirs, and extended these metrics to dynamical measures that take the actual trajectory of the reservoir through state space into account as it is driven by the external input. I showed that these dynamical measures offer an accurate prediction of performance on several tasks. Moreover, since these measures are taken at the actual operating point of the reservoir instead of a linearization around the zero fixpoint, they can be applied to other, more complex nonlinearities - which is necessary when studying more exotic reservoirs such as those that occur in photonic reservoir computing or other physical RC systems.

Next - keeping in mind that the dynamics are crucial for the performance of a reservoir - we have studied and extended an unsupervised, bioplausible local adaptation rule for the intrinsic parameters of the nodes in the reservoir. This generalized IP rule was then experimentally validated on a variety of different tasks, and it was shown that for each of these tasks, the rule adapts the nodes of the reservoir into a desirable dynamic regime. Moreover, we also used the rule to tune a ring reservoir topology, which is notoriously difficult to operate. We showed that here too, the IP rule adapts the parameters of the reservoir automatically so that the dynamics are in the right regime. We claim that this rule can be of use when going to more exotic reservoir implementations, where it is initially not clear at all how to set the parameters of the nodes.

5.2 Conclusions

The take-home message of this thesis is that Reservoir Computing is a promising and fundamental shift in the way computation with timeseries is done. Currently almost all computation on temporal signals - mono- or multidimensional - is done using conventional sequential programmed machines, which means that they execute certain human-made algorithms on their inputs. The main claim or message of this Ph.D. thesis is that Reservoir Computing can and should be seen not only as a convenient and computationally efficient way of training or using RNNs, but that it represents a novel way of computing. More specifically, RC as a computational paradigm should be contrasted with the well known Turing class of computing machines.

Turing machines were introduced by Alan Turing as a model for procedural or algorithmic computing, and are a very powerful but abstract logical construct. Their introduction has been crucial for the development of theoretical computer science, by defining in a strict logical and mathematical way which devices are useful for computation. The Turing machine is very important since it actually defines an equivalence class

of computational devices that have the same computational capabilities. In practice, it is impossible to construct a Turing machine due to the requirement of infinite storage capacity, but if these physical limitations are ignored, many (abstract and physical) computational devices have been proven to be Turing equivalent, including programming languages such as Java or Pascal, cellular automata, and all modern computers.

In addition to being a powerful construct for reasoning about the properties of computability, the Turing machine is also a model for how to *perform* these computations. As such, the TM has been a source of inspiration for Von Neumann when he introduced his architecture that serves as the basis for all modern digital computers. A TM is an essentially sequential machine, which actively requests its external input as it is needed and performs the computations on its own pace - meaning that computationally more complex operations take more time.

This computational paradigm can be contrasted with the RC way of doing computation. The main differences are:

- **Learning vs. explicit programming**. This difference is more generally applicable to the majority of methods in the field of machine learning, but it does represent a quite fundamental shift in the way computational devices are constructed. Learning machines learn by example, which means that a lot of difficult problems can be tackled without exact knowledge of the underlying system or properties of the problem, but it also means that there will always remain a certain impression of a black-box for these systems, with no strict proofs of them behaving in any circumstances. Moreover, learning systems will usually still make errors, although it is the task of the system designer to minimize these errors.

- **Transient dynamics vs. stable states for computing**. Reservoirs are dynamical systems that are continually driven by external inputs or that generate their own dynamics. In contrast with, e.g., Hopfield nets (Hopfield, 1982), a reservoir is not used in a stable state but is constantly in a transient dynamical regime. This is very different from the way electronic computing devices using digital logic operate: in those devices, care is taken that the transients have died out as much as possible and in synchronous systems this is even guaranteed by a common clock signal.

- **Generalization capabilities vs. predictable input domain**. When traditional computers are presented with inputs that they are not programmed to handle, they generally do not know what to do - depending on how the error handling is done this unknown input will either be ignored, generate erroneous output or even cause the

program to halt. RC systems on the other hand (and many other learning systems) are more robust to unseen inputs, and indeed they are capable of generalizing what they have seen in the training set to novel inputs. This generalization is a very important property in many applications and enables the robust deployment of systems in real world environments.

- **Implementation substrate**. Most physical implementations of Turing machines aim to model what is actually an abstract sequential processing machine, and try to minimize the unnecessary nonlinear physical effects. Several physical RC implementations however (modelled or actual), actively use the physical or dynamical properties of the implementation substrate, meaning that the medium in which the reservoir is implemented forms an integral part of the computational mechanism. While the application and study of these physical reservoirs is certainly not trivial, it does open a potential research and application area for many systems that were as yet not seen as computational devices.

We have shown that RC shows much promise, both theoretically and from an engineering point of view, as a framework for computation on timeseries. The initial steps towards understanding and tuning the operation of reservoirs to optimal performance have been taken in this thesis. Nonetheless, there is still a lot of open questions and much insight to be gained. Reservoirs are complex nonlinear dynamical systems, and a mathematical theory that fully explains their functionality would greatly enhance our understanding and enable a more directed search for media that are suitable in the context of Reservoir Computing.

5.3 Perspectives

Alternate reservoir implementations In this work, some Reservoir Computing implementations were already presented that deviate from the neural models that started this research area. However, there is still a vast range of possible reservoir implementations that can - and likely will - be explored in the future. The use of very fast analog VLSI chips was initiated through the study of CNN reservoirs, but there is still a large potential for further study here. For instance, currently the readout layer is simulated offline on the host computer which is both cumbersome and relatively slow. Modern CNN chips usually also incorporate an accompanying DSP chip or FPGA, on which the readout layer could be implemented. A working implementation of this CNN reservoir setup

would enable a vast array of applications, because these CNNs are often tightly integrated with other visual processing hardware such as high-speed Charge Coupled Devices (CCD). Reservoir Computing offers an entirely novel way of using these sophisticated hardware visual processors using learning mechanisms, instead of having to program them by hand.

The photonic Reservoir Computing research line is even more innovative, but perhaps also more promising. Currently, industrial applications of nanophotonic research are mostly focused on very high speed communication (because of the high speeds at which light moves), but the real breakthrough going from fast silicon hardware computing to ultrafast photonic hardware computing still has to happen. This is partly because the direct transposition of design principles from the silicon world to the photonic world causes problems: photonic memory is very difficult to impossible to construct, the components that are used are very bulky compared to their silicon counterparts and they often exhibit undesirable nonlinear or chaotic behaviour. Due to this, most photonic systems still need to make the transition to silicon at some point, which nullifies many of their advantages. Photonic Reservoir Computing has the potential to offer an entirely novel way of using photonic systems for computation, not by constructing standard computers using photonic components, but by actually using the complex nonlinear properties of light and nanophotonic devices. The transition to photonic reservoirs opens up a large amount of potential novel implementations that share no resemblance anymore to ESNs or LSMs, but that do enable the use of very powerful, full-optical high speed computation with light.

Dynamics measures We have presented the case in this work that a reservoir can be seen as a generic nonlinear dynamic medium that boosts the discriminative power of the simple linear readout function. Following this line of reasoning, we have used techniques from dynamical systems theory to measure and quantify the actual dynamic properties as the reservoir is driven by external inputs. Additionally, this quantification has yielded more insights into the operation of the reservoir, and what constitutes a good reservoir. More importantly, however, these techniques in many cases translate to novel, non-neural reservoir implementations with more advanced or complex nonlinear behaviour, because they evaluate the dynamics in the actual working point of the reservoir. The extension of these measures and their application to novel reservoir implementations is an interesting line of research that will surely be continued in the future. Moreover, since the link between RC and dynamic systems theory has been shown in this work, it seems obvious to borrow more extensively from the knowledge in the latter field. The study of nonlinear control

theory has yielded many results that could be applied to the design of suitable reservoirs. Finally, the development of methods for measuring useful dynamical reservoir properties do not only yield more insight into the operation of these systems, but can also form the basis for the development of adaptation or learning rules that optimize these dynamical measures.

Reservoir adaptation The first steps towards an automatic adaptation - rather than brute-force random search or tweaking by experts - towards a good reservoir have been taken in this thesis. The proposed adaptation rule is based on information theoretic principles and modeled after a biological phenomenon, but has been proven to also adapt the dynamics of the reservoir to a desirable regime in an unsupervised way. This type of adaptation will be crucial for constructing novel reservoir implementations, where traditional design principles such as spectral radius are no longer valid. Intrinsic plasticity is one possibility that has shown to be useful in the context of neural reservoirs, but it has yet to be extended to more exotic reservoir implementations and transfer functions. Moreover, the impact of IP on the reservoir dynamics and the interplay between the rule and the dynamic properties should be studied thoroughly from a system's point of view, possibly using the dynamic quantification methods presented in this work. In particular, since our proposed dynamics measure has increased the insight into the necessary criteria for a good reservoir, this can serve as a guideline for extending the current adaptation rule to a more advanced version that might require far less or even no intervention from the designer at all.

Novel reservoir architectures The classical two-layer (reservoir and readout) setup has been used extensively throughout this thesis. However, the viewpoint of the reservoir as a nonlinear complex transformation with memory can be extended beyond this basic architecture. The central message of (Bengio, 2009) is that truly powerful learning machines need to have a multi-layered or hierarchical structure, with each successive layer operating on a more abstract level than the previous one. The fact that these *deep networks* are capable of achieving - despite their complexity - very impressive and even state-of-the-art results, has been shown multiple times, for instance by Yann LeCun's LeNet (LeCun et al., 1989) or Hinton's Deep Belief Networks (Hinton et al., 2006). In this context, Reservoir Computing also constitutes an appealing paradigm for constructing deep architectures, due to the separation of the learning system into a complex, random or unsupervisedly tuned nonlinear layer, and a simple trainable linear layer. By stacking these layers, and ap-

plying suitable learning rules to the intermediate linear layers, powerful hierarchical structures like the ones cited above can be constructed.

A The Reservoir Computing Toolbox

All the experiments described in this thesis were done using the RC Toolbox (RCT). This toolbox was developed during my PhD work in collaboration with some of the colleagues at our lab. The toolbox is written in Matlab, is open source[1] and offers a user-friendly interface for the simulation of all common - and less common - RC implementations. The toolbox is essentially a collection of functions that are written around common data structures. The RCT can be used in several ways, ranging from a standard ESN setup applied to a user-defined dataset, to more flexible custom experimentation scripts. The toolbox offers functions that can help the user generate custom reservoir topologies, custom hierarchical architectures, custom nonlinearities or complex node types, and custom training functions.

This appendix describes the current version of the RCT. The old version (designed and developed by B. Schrauwen and myself) already incorporated much of the functionality described below. However, as research on reservoirs progressed, it became clear that a rewrite was necessary to enable the toolbox to cope with new developments. Specifically, the emerging of novel node types, adaptation rules and the investigation of hierarchical RC structures did not fit in the old version. This is why I, in collaboration with M. Wardermann, redesigned the underlying datastructures and code to allow the RCT to be used for researching these new topics.

[1] Available from `http://reslab.elis.ugent.be/rct`.

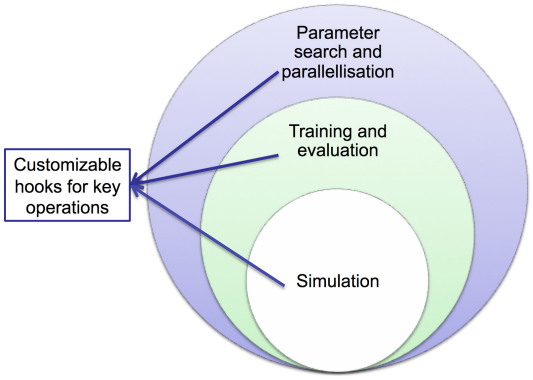


Figure A.1: A schematic overview of the three basic functionalities provided by the RCT.

A.1 A high-level description

From a user point of view, there are a few possible scenarios for which the RCT can be used. Many users who are new to the toolbox will want to start with a very basic experiment, or apply the RC framework to their own dataset. This can be easily done, and several examples on how to do this will be given below. However, more advanced use of the toolbox (i.e., custom architectures or learning rules) requires a more advanced knowledge of the Matlab language, in particular function pointers and anonymous functions.

The toolbox essentially consists of three layers of functionality - see Figure A.1. At the center, the core is formed by a simulation engine that simulates a given RC architecture based on the topology (which layers are connected to which), the node nonlinearities and the weight matrices. This core functionality is mainly provided by the function `generic_simulate` (see Section A.5). Around this, a framework is provided for training and evaluating an architecture on a given dataset. This functional layer contains functions that can train, e.g., the linear readouts, but also provides a flexible way of doing cross-validation (see Section A.6).

Finally, at the highest level there is a parallellization and optimization layer. The fundamental unit of work here is a single RC experiment (provided by the training/evaluation layer). The optimization functionality currently allows manual parameter sweeps and post-experiment gathering and processing of the data, contained in the function `rc_simulate.m` - but work is ongoing on extending this functionality to an automatic op-

timization framework, that finds a quasi-optimal parameter setting with far less intervention from the user. The parallelization functionality, finally, enables the deployment of large-scale batches of jobs (for instance parameter sweeps) across a computing grid of worker nodes with access to a shared network drive (see Section A.7). At every level, many function hooks are built-in that can be configured by the user, and replaced if necessary by custom functions.

A.2 Getting started

The root location of the RCT directory will be denoted by `$RCTROOT`. From this location, the user can execute the script `install.m`, which will automatically add the necessary directories to the Matlab path.

By default, the toolbox is configured to train and test a standard ESN reservoir of 100 tanh nodes on a tenth-order NARMA task. This experiment can be run immediately by executing `rc_simulate`, which should give an output similar to :

```
>> rc_simulate
Warning: Using default settings.
To use your own configuration, set the variable custom_configuration to ...
     the path of your own configuration.
Warning: Results of simulation will not be saved!
Job 1/1 (100 %). Time passed: 0.
Creating topology.
indices number 1/1.
Simulating network.
Resampling responses and outputs to highest sampling rate of any layer.
Training and evaluating readout using cross_validate. Elapsed time is ...
     12.327436 seconds.
train_result 1: 0.15933(0.00302113), test_result: 0.160691(0.00810677)
```

The train and test results are the normalized mean square error (NMSE) on the training and test set, respectively.

The `rc_simulate` script executes a standard workflow when doing experiments with RC. The precise parameters that define the experiment (such as the dataset to be used, the reservoir size and topology, training method, ...) are read from a configuration `.m` script. The user can set the configuration file to be used by setting the variable `custom_configuration` to the name of the configuration file (without the `.m` extension). If this variable is not set, the default settings are used (as in the example above). See `defaults.m` and the settings files

in `$RCTROOT/default_settings/` for how this default experiment and topology is defined exactly.

The `rc_simulate` and `rc_simulate_job` scripts do the following tasks:

- Generate a dataset, by calling the function stored in the variable `dataset_generation_function`. This is discussed in more detail in Section A.3.
- Construct a reservoir topology (all untrained weight matrices) based on the parameters settings stored in the variable `topology`. The weight matrices are also stored in the variable `topology`. This is discussed in detail in Section A.4.
- Simulate the reservoir given the topology and input signals. This is done by calling `generic_simulate.m`, which will be discussed in more detail in Section A.5.
- Train the readout function and evaluate it on the train and test data. More information on this is given in Section A.6.

A.2.1 Configuration files

As was already mentioned, the RCT is constructed to work with configuration files. These files are simple matlab scripts where parameters are set for the experiments. The default configuration file is `$RCTROOT/defaults.m`. This file gets called at the beginning of `rc_simulate`. If a variable `custom_configuration` exists, the contents of this variable are evaluated (using Matlab's `eval` function), if not the default settings are used. The variable `custom_configuration` can contain either the name of a Matlab script, or some Matlab code that sets parameters. As an example, there are two ways of changing the default reservoir size from 100 to 200 nodes:

- By writing `'topology.layer(3).size=200;'` in a configuration file (say, `myconf.m`), and setting `custom_configuration='myconf'`.
- By setting `custom_configuration='topology.layer(3).size=200;'` directly.

For this simple example, the two forms are equivalent, but when more parameters need to be changed it quickly becomes more convenient to store everything in a configuration file. For more advanced users, `custom_configuration` can be a cell array of strings that will be evaluated in order. This is for instance useful if you have small variations

(say `arch1.m, arch2.m, ...`) on a basic configuration (say `base.m`). You can then set, e.g., `custom_configuration={'base', 'arch1'};` or `custom_configuration={'base', 'arch2'};`.

A.2.2 Some use cases

Depending on what the user wants, the toolbox can be used for a few different scenarios. We will briefly discuss some of these scenarios and the options that are available, ranked in increasing order of difficulty, flexibility and required Matlab programming knowledge.

A.2.2.1 Own dataset

The user may wish to run the standard ESN setup on his/her own dataset. This is done by creating a Matlab script (e.g., `my_dataset.m`) that generates or loads the data, and returns the input and output signals. See Section A.3 for information on how to do this. Then the following code will run the experiment on the custom dataset.

```
>> custom_configuration='dataset_generation_function=@my_dataset';
>> rc_simulate
```

A.2.2.2 Parameter sweeps

The `rc_simulate` script also supports 'sweeping' certain parameters (running experiments for a range of values of that parameters). This can be specified through the variable `parameter_ranges`. It is a struct array with two fields, `.name` and `.range`. The name field is a string containing the name of the variable to be ranged over, and the range field is a vector containing the values that the parameter should take. For instance, if you want to range the spectral radius of the reservoir connection matrix (see Section A.4 for more information about the topology) from .1 to 1 in steps of .1, the following code will do that:

```
>> parameter_ranges = struct('name', {'topology.conn(3,3).scale_factor'}, ...
    'range', {.1:.1:1});
>> rc_simulate
```

You can also range over more than one parameter. For instance, if you want to range both the reservoir scaling and the input scaling over the same range, you can use:

```
>> parameter_ranges = struct('name', {'topology.conn(1,3).scale_factor', ...
     'topology.conn(3,3).scale_factor'}, 'range', {.1:.1:1, .1:.1:1});
>> rc_simulate
```

Finally, it is also possible to range over non-scalar values, such as function pointers. For instance:

```
>> parameter_ranges = struct('name', {'dataset_generation_function'}, ...
     'range', {{'@dataset_narma_10', '@dataset_narma_30'}});
>> rc_simulate
```

If a parameter sweep is done, the user will want to store the results of every parameter setting to file. This is done by setting `save_results` to `true`. By default, it is `false`, and a warning is printed to remind the user of this. If `save_results` is `true`, a directory will be created to store the results. This directory is named with a timestamp to avoid conflicts between experiments, and is created as a subdirectory of the location specified by `output_directory`, whose default value is `'results'`. In this directory, for every parameter setting selected by `parameter_ranges` a copy of the workspace is saved. If `save_data` is false, the `data` variable is erased before saving which will reduce the necessary diskspace.

A.2.2.3 Custom scripts

If a user wants to execute a custom script that does not fit into the standard experimental flow provided by `rc_simulate_job.m`, it suffices to specify the name of the script like so: `custom_script='my_script'`. In this way, all the functionality of `rc_simulate` - such as parallelization and parameter sweeping - to be used with user-defined scripts. Beware: the script should save all necessary variables itself.

A.3 Datasets

The RCT includes some of the main benchmark tests that have been described in RC literature and that were used in this thesis. The tasks include the Mackey-Glass timeseries prediction (Steil, 2005a), the isolated spoken digit recognition task from (Verstraeten et al., 2005), the NARMA task described in (Jaeger, 2003) and (Steil, 2005a) and others.

Two main categories of problems can be defined: input-output tasks and generation tasks. In the former case, the task consists of mapping a

(possibly multidimensional) input signal to a (also possibly multidimensional) output signal - an example is the NARMA task. In the latter case, there is only a single (input) timeseries that is to be autonomously generated by the reservoir - an example of this is the Mackey-Glass timeseries production. For these generation tasks, the output of the readout is typically fed back into the reservoir. The system is trained by teacher forcing the timeseries on the inputs, training the readout weights to do one-step-ahead prediction and during the testing phase turning off the teacher forced signal and feeding the network's own predicted signal back. This also means that in this case, there is no output layer, only an input layer whose values are either teacher-forced or generated by the reservoir itself. In the directory `$RCTROOT/default_settings/` there is a configuration file `settings_pattern_generation` that configures the topology for a generation task (e.g., by taking away the output layer and setting a connection from the reservoir to the input layer).

It is easy to create additional, custom datasets by defining your own dataset function. This function should return two cell arrays, containing the input and output signals. Each cell corresponds to an example in the dataset. For input-output tasks, the output signals corresponding to the inputs are returned by the dataset function, but for generation tasks, the output cell array will be empty.

The function `generate_datastruct`, which is called at the beginning of `rc_simulate_job`, will use the `dataset_generation_function` to generate the dataset and fills it into the `data` variable. This variable contains the field `layer`, which is a struct array with as much elements as there are layers in the architecture (see Section A.4 below for more on layers). For every layer, there are two fields: `layer.r` and `layer.s`. The `layer(:).r` field contains the required or teacher forced signals, and the `layer(:).s` field contains the signals or states that were actually simulated by running `generic_simulate`. For instance, in a simple input output classification task, the `data.layer(1).r` field would contain the inputs to the network, the `data.layer(3).s` field would contain the simulated reservoir states, and the `data.layer(4).r` and `data.layer(4).s` would contain the desired outputs and simulated outputs respectively.

A.4 Topology generation and layers

Research on RC is currently starting to move from the standard single reservoir setup to hierarchical approaches. Some innovative layered architectures have already been described in literature (Jaeger, 2007), and clearly more advanced setups will be studied in the future. That is why

the way the toolbox handles topologies was changed drastically compared to previous versions.

The RCT works with a datastructure based on layers, and connections between the layers. A layer consists of a set of nodes (ranging from very simple linear nodes to nodes with complex nonlinearities and learning rules). The connections between the layers are always defined by a weight matrix, connecting each node of the outgoing layer to each node of the incoming layer (obviously, if a weight is zero there is no connection). By default, a standard four layer setup is used, with the first layer being the input layer, the second being the bias layer, the third being the reservoir itself and the fourth layer is the output layer - this is the most common setup described in literature. However, the toolbox supports an arbitrary number of layers (only limited by memory constraints) and connections between the layers.

In the toolbox, all information about the topology and architecture of the system is stored in the variable `topology`. It is a struct with the fields `layer` and `conn` (for connections). Suppose there are four layers in the architecture[2], then `topology.layer` will in turn be a 1x4 struct array, and `topology.conn` will be a 4x4 struct array (one for every possible connection between the layers). A single `layer` struct contains the following fields:

- `nonlin_functions` : this is a cell array of function pointers, containing the nonlinearity functions to be applied to the nodes in the layer (e.g. `tanh()`). See Section A.5 for more information.

- `is_trained_offline` : a boolean field to indicate if a layer is trained offline (such as for the output layer in the standard setup).

- `is_trained_online` : a boolean field to indicate if a layer is trained online.

- `init_simulation`: a function pointer to a function that will be called at the beginning of `generic_simulate.m`. This can be used to, e.g., initialize the layer to a certain state. See Section A.5 for more.

- `finish_simulation` : similarly to `init_simulation`, this contains a function that will be called at the end of `generic_simulate.m`. This is useful, e.g., to compute some metrics on the states of the layer.

[2]This is the default setting: an input layer, a bias layer, a reservoir layer and an output layer.

- `no_delay` : a boolean field to indicate that the connections to this layer are instantaneous, i.e., with zero delay. This is needed for generator tasks, since there are two connections which would result in a delay of two timesteps.

- `is_teacher_forced`: a boolean field that indicates whether a layer is teacher-forced. This means that in `generic_simulate`, the states of the layer are not computed through simulation, but the signals contained in the `data.layer().r` field are used to drive the other layers. Note that in an input-output task, the input layer is also teacher-forced because the external inputs are used to drive the other layers!

- `dt`: scalar field that indicates the timestep on which the layer operates. This type of resampling was used for the experiments of Section 3.1.

- `node_type`. String field that indicates the node type. Currently only analog reservoirs are supported, but this will be extended to, e.g., spiking or photonic reservoirs in the future.

- `size` : scalar field that determines how many nodes are in the layer.

- `training_function` : function pointer to the training function for layers that are trained offline.

- `regul_param` : scalar field that sets the regularization parameter for the training function. This can be optimized using cross-validation - see Section A.6.

- `scoring` : function pointer to the scoring function. Default value is `score_nrmse`, but many others are available in the toolbox.

- `generic_scoring` : boolean field that indicates if the scoring should be done on a sample-by-sample basis. This is useful for large datasets.

The `topology.conn` struct contains all information about the connections between the layers. Usually, only a small number of all possible connections between the layers will be in use. The `conn` struct has the following fields:

- `is_active` : a boolean field that indicates if a connection is active or not, i.e if it is taken into account during simulation with `generic_simulate`. This can be used to quickly turn connections between on and off.

- `creation_pipeline` : a cell array of function pointers that will be called during the construction of the weight matrices. The functions are called in succession, and act as a pipeline - i.e., the results are passed on from function to function. This allows a lot of flexibility when generating connections. For instance, if the creation pipeline consists of : `{@gen_rand, @assign_randn, @scale_specrad}`, first a random connectivity matrix will be created with a given connection fraction (see below), next weights are assigned from a random normal distribution and finally the weight matrix is rescaled to a certain spectral radius given by `scale_factor` below.
- `scale_factor`: a scalar value that indicates the scaling factor used in the scale functions in the creation pipeline. This can be for instance the spectral radius, a constant scale factor or the largest singular value. The meaning of this scalar is determined by the scaling function that is used.
- `conn_frac` : the connection fraction used when creating the connectivity matrix (which node is connected to which). A fraction `conn_frac` of all possible connections will be set, with a value of 1 indicating that all nodes of the outgoing layer are connected to all nodes of the incoming layer.
- `discrete_set` : a vector of values from which the weights are randomly drawn by the function `assign_discrete`.

A.5 generic_simulate.m

The function `generic_simulate` is the core of the toolbox. Its signature is:

```
[data, topology] = generic_simulate(topology, data, simulated_connections, ...
    sample_len)
```

The function takes a topology and a dataset, and simulates the entire architecture for all samples in the dataset. The simulated states of the layers that are not teacher forced, are written into the `data.layer().s` fields. The function only simulates active connections (where `is_active` is `true`).

At every timestep, the state vector of every layer is updated by computing the weighted sum of all layers that are incoming to that layer.

Then, all nonlinearities are applied in the order given by `topology.layer(:).nonlin_functions`. There are two types of nonlinear functions: simple functions that only operate on the current timestep (such as `tanh()`), and complex nonlinearities that take - in addition to the current state vector - also the topology, data and other arguments (see `generic_simulate.m` for the exact argument list). The complex nonlinearities can be used for, e.g., online learning rules or adaptation rules such as Intrinsic Plasticity (see Section 4.3 of this thesis and (Schrauwen et al., 2008b)).

A.6 Training and cross-validation

This section deals with offline training and cross-validation methods. The difference with online training methods is that the adjustment of the weights is done in a single-shot fashion after all simulations have been done, whereas for online learning the weights are continually adjusted while the reservoir or other layers are simulated.

A layer `n` can be trained offline simply by setting `topology.layer(n).trained_offline` to true, and filling in an appropriate training function in `topology.layer(n).training_function`. The most popular linear training functions in the RC community are available in the toolbox (`train_pseudo_inverse` and `train_ridgre_regress`), but other less common training methods are also provided (such as robust linear training, `train_robust.m` (Ryan, 1997) or iteratively weighted least squares (IWLS), `train_iwls.m` (Ryan, 1997)).

These functions can be used to train a layer on data from other incoming layers that were already simulated on the dataset. Obviously, for a trained layer, the `data.layer(n).r` field of required states should be filled in, otherwise an error will occur. The trained weights for all incoming layers are automatically distributed and filled into the corresponding `topology.conn(i,n).w` fields.

For a more accurate evaluation of a certain architecture/parameter combination, cross-validation can be applied (see Subsection 2.1.3.2 of this work). Cross-validation is implemented through the function `cross_validate.m` and offers a function hook that allows to specify the precise way in which cross-validation occurs. This function hook, specified in `train_params.cross_val_set_function` returns a struct array that has two fields: `train` and `test`. Every element of the struct array represents a fold, and for each fold, the `train` and `test` fields contain the sample numbers (the indices for the `data` struct containing the dataset) that should be used for training and testing respectively. By

default this function hook is set to `random_cross_val`, which creates a number of folds equal to `train_params.folds`, and randomly distributes all examples across the training and test fields making sure that every sample is used for testing once. Other cross-validation-set functions include `no_cross_val` (only one fold, simple train and testset - the fraction of train samples is set using `train_params.train_frac`), `cross_val_only_training` (all examples in the training set, useful for maximizing data use) and `random_freerun_cross_val`.

This latter function is useful when using the freerun operation mode, where the reservoir is left to generate signals on its own instead of being driven by an external input. This mode is activated by setting `train_params.freerun` to true. If this is the case, the `cross_validate` function will split all samples from the testset in every fold, and split off the first fraction (specified by `train_params.freerun_split`). The portion of the data before this freerun splitpoint is then added to the training set (for maximizing the data use) and also used to warmup the reservoir during the testing. The testing is then done starting from the freerun splitpoint, using a reservoir warmed up with a teacher-forced training signal.

Many linear training methods use some form of regularization (see Subsection 2.1.3.1 of this work). This regularization parameter can be automatically optimized by using `cross_validate_grid` instead of `cross_validate`. In this case, a linear sweep is done of the regularization parameter, and for every value a cross-validation-set struct array is again created. The training examples of every fold are then passed to the regular `cross_validate` function as a *complete* dataset, which means that inside `cross_validate` the training samples will be further divided into a training and validation set. This results in the nested cross-validation setup that was described in Subsection 2.1.3.1. After all values of the regularization parameter were evaluated, the optimal value is selected and the performance on the testset is returned.

A.7 Parallellization

If a network of computers capable of running matlab and having access to the same shared network drive is available, the RCT supports parallelization of batch jobs. This is done in an 'embarrasingly parallel' way, meaning that there is no data dependency between the tasks running in parallel. The parallelization is being done on the level of jobs or experiments. This means that for, e.g., parameter sweeps with multiple runs (instantiations of reservoirs with the same parameter settings), every

experiment will be a single job in the batch. On a lower level, multicore/-multithreaded parallelization of, e.g., matrix operations is provided by (the more recent versions of) Matlab itself.

The parallel infrastructure is based on a client-server architecture. All necessary files and functions are contained in the folder `$RCTROOT/parallel`. The interface for submitting a job to the grid has been made the same as running experiments locally, i.e., with `rc_simulate`. This means that it suffices to define a `custom_configuration` variable and if necessary a `custom_script` variable and then the user can call `rc_simulate_par` instead of `rc_simulate`. This function will execute the default configuration file, all custom configuration files and then save the workspace to the shared network drive in a new job directory - which is named with a timestamp of the moment when the job is created. This saved workspace will then be loaded by the worker nodes when starting a job, to ensure that every worker node executes the job in the same environment. The script also saves a jobfile, which is a `.mat` file that contains the number of jobs, and which parameter combinations should be simulated.

The server or *dispatcher* is a matlab process that continually polls the shared directory for new job files. If a job file is detected, the dispatcher loads the jobfile, checks which job is next in the queue and waits for a message from one of the worker nodes. The worker nodes periodically contact the dispatcher through a TCP/IP connection and ask for new jobs. If a job is waiting in the waiting queue, the dispatcher sends the job number, job directory and job name to the worker node, which then starts the experiment, and the dispatcher moves the job number to the running queue. When a worker node is finished it also contacts the dispatcher to notify that the job is done, and the dispatcher moves the job number from the running queue to the done queue. If all waiting/running jobs are done, the filename of the jobfile is prepended with `done_` so that the dispatcher ignores it from then on.

All communication between the dispatcher and the worker nodes is done through two TCP/IP ports, which can be specified when starting the dispatcher and worker processes. For automating the management of the worker nodes, several batch-scripts have been written in `$RCTROOT/parallel/bash_scripts`, which read a file called `nodes.list`. All commands to the worker nodes (such as starting/stopping or checking the presence of a matlab process) are issued through an ssh connection to all network clients contained in the file `nodes.list`.

Bibliography

Adrian, E. (1928). *The Basis of Sensation: The Action of the Sense Organs*. W.W. Norton & Co, New York.

Aizerman, M., Braverman, E., and Rozonoer, L. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25(6):821–837.

Alligood, K., Sauer, T., and Yorke, J. (1996). *Chaos: An Introduction to Dynamical Systems*. Springer.

Anon (1999). Intelligent flight control: Advanced concept program, final report. Technical report, The Boeing Company.

Antonelo, E. A., Schrauwen, B., and Stroobandt, D. (2008a). Event detection and localization for small mobile robots using reservoir computing. *Neural Networks*, 21:862–871.

Antonelo, E. A., Schrauwen, B., and Stroobandt, D. (2008b). Modeling multiple autonomous robot behaviors and behavior switching with a single reservoir computing network. In *IEEE International Conference on Man, Systems and Cybernetics*.

Antonelo, E. A., Schrauwen, B., and Stroobandt, D. (2009). Unsupervised learning in reservoir computing: Modeling hippocampal place cells for small mobile robots. In *International Conference on Artificial Neural Networks (ICANN)*. (accepted).

Athreya, K. and Ney, P. (2004). *Branching processes*. Dover Publications.

Atick, J. J. (1992). Could information theory provide an ecological theory of sensory processing? *Network: Computation in Neural Systems*, 3(2):213–251.

Atiya, A. F. and Parlos, A. G. (2000). New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks*, 11:697.

Baddeley, R., Abbott, L. F., Booth, M. C. A., Sengpiel, F., Freeman, T., Wakeman, E. A., and Rolls, E. T. (1997). Responses of neurons in primary and inferior temporal visual cortices to natural scenes. In *Proceedings in Biological Sciences*, volume 264, pages 1775 – 1783.

Bauer, M. and Martienssen, W. (1991). Lyapunov exponents and dimensions of chaotic neural networks. *J. Phys. A: Math. Gen*, 24:4557–4566.

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

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations & Trends in Machine Learning, to appear.*

Billingsley, P. (1965). *Ergodic Theory and Information*. John Wiley and Sons Inc.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford.

Bucolo, M., Fortuna, L., Frasca, M., La Rosa, M., Virzi, M., and Shannahoff-Khalsa, D. (2004). A nonlinear circuit architecture for magnetoencephalographic signal analysis. *Methods of information in medicine*, 43(1):89–93.

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

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

Burgsteiner, H. (2005a). *On learning with recurrent spiking neural networks and their applications to robot control with real-world devices*. PhD thesis, Graz University of Technology.

Burgsteiner, H. (2005b). Training networks of biological realistic spiking neurons for real-time robot control. In *Proceedings of the 9th International Conference on Engineering Applications of Neural Networks*, pages 129–136, Lille, France.

Bush, K. and Anderson, C. (2005). Modeling reward functions for incomplete state representations via echo state networks. In *Proceedings of the International Joint Conference on Neural Networks*, Montreal, Quebec.

Buteneers, P., Schrauwen, B., Verstraeten, D., and Stroobandt, D. (2009). Real-time epileptic seizure detection on intra-cranial rat data using reservoir computing. In *Proceedings of the 15th International Conference on Neural Information Processing of the Asia-Pacific Neural Network Assembly, APNNA*.

Chua, L. (1998). *CNN: A paradigm for complexity*. World Scientific.

Chua, L., Roska, T., and Venetianer, P. (1993). The CNN is universal as the Turing machine. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 40(4):289–291.

Chua, L., Yang, L., and Krieg, K. (1991). Signal processing using cellular neural networks. *The Journal of VLSI Signal Processing*, 3(1):25–51.

Chua, L. O. and Yang, L. (1988a). Cellular neural networks: Applications. *IEEE Transactions on Circuits and Systems*, 35(10):1257–1272.

Chua, L. O. and Yang, L. (1988b). Cellular neural networks: Theory. *IEEE Transactions on Circuits and Systems*, 35(10):1273–1290.

Cover, T. (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE transactions on electronic computers*, 14(3):326–334.

Cristianini, N. and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press.

Crone, S. F., Nikolopoulos, K., and Hibon, M. (2008). Automatic modelling and forecasting with artificial neural networks - a forecasting competition evaluation. Technical report, Lancaster University Management School.

Davis, S. and Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 28(4):357–366.

Deng, Y., Chakrabartty, S., and Cauwenberghs, G. (2004). Analog auditory perception model for robust speech recognition. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 1705–1710.

Destexhe, A. and Marder, E. (2004). Plasticity in single neuron and circuit computations. *Nature*, 431:789–795.

Dockendorf, K. P., Park, I., He, P., PrÃncipe, J. C., and DeMarse, T. B. (2009). Liquid state machines and cultured cortical networks: The separation property. *Biosystems*, 95(2):90 – 97.

Doddington, G. and Schalk, T. (1981). Speech Recognition: Turning Theory to Practice. *IEEE Spectrum*, 18(9):26–32.

Dominey, P. (1995). Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning. *Biological Cybernetics*, 73(3):265–274.

Doyon, B., Cessac, C., Quoy, M., and Samuelides, M. (1992). Destabilization and Route to Chaos in Neural Networks with Random Connectivity. In *Advances in Neural Information Processing Systems 5,[NIPS Conference] table of contents*, pages 549–555. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA.

Dreyfus, G. (2005). *Neural Networks: Methodology And Applications*. Springer.

Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification - Second Edition*. John Wiley and Sons, Inc.

Dutoit, X., Schrauwen, B., Van Campenhout, J., Stroobandt, D., Van Brussel, H., and Nuttin, M. (2009). Pruning and regularization in reservoir computing. *Neurocomputing*, 72:1534–1546.

Fernando, C. and Sojakka, S. (2003). Pattern recognition in a bucket. In *Proceedings of the 7th European Conference on Artificial Life*, pages 588–597.

Foldiak, P. and Young, M. (1995). *The handbook of brain theory and neural networks*, chapter Sparse coding in the primate cortex, pages 895–898. Bradford Books.

Fredkin, E. (2003). An introduction to digital philosophy. *International Journal of Theoretical Physics*, 42(2):189–247.

Gers, F., Eck, D., and Schmidhuber, J. (2001). Applying LSTM to time series predictable through time-window approaches. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 669–676. Springer-Verlag London, UK.

Gers, F. and Schmidhuber, J. (2001). LSTM recurrent networks learn simple context free and context sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340.

Gerstner, W. and Kistler, W. (2002). *Spiking Neuron Models*. Cambridge University Press.

Ghani, A., McGinnity, T., Maguire, L., and Harkin, J. (2008). Neuro-inspired Speech Recognition with Recurrent Spiking Neurons. In *Proceedings of the 18th international conference on Artificial Neural Networks, Part I*, pages 513–522. Springer.

Goh, W. and Crook, N. (2007). Pattern recognition using chaotic transients. In *Submitted to 15th European Symposium on Artificial Neural Networks (ESANN'2007)*.

Graves, A., Eck, D., Beringer, N., and Schmidhuber, J. (2004). Biologically plausible speech recognition with LSTM neural nets. In *Proceedings of Bio-ADIT*, pages 127–136.

Hagan, M. and Menhaj, M. (1994). Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, 5(6):989–993.

Hajnal, M. and Lorincz, A. (2006). Critical Echo State Networks. *Lecture notes in Computer Science*, 4131:658.

Hammer, B. and Steil, J. J. (2002). Perspectives on learning with recurrent neural networks. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN)*.

Hartigan, J. (1975). *Clustering algorithms*. John Wiley & Sons, Inc. New York, NY, USA.

Haykin, S. (1991). *Adaptive filter theory*. Prentice-Hall, 2nd edition.

Haykin, S. (1999). *Neural Networks: a comprehensive foundation (Second edition)*. Prentice Hall.

Haykin, S. and Widrow, B. (2003). *Least-Mean-Square Adaptive Filters*. Wiley-Interscience.

Hertzberg, J., Jaeger, H., and Schönherr, F. (2002). Learning to ground fact symbols in behavior-based robots. In *Proceedings of the 15th European Conference on Artificial Intelligence*, pages 708–712.

Hinton, G., Osindero, S., and Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554.

Holzmann, G. (2008). Echo state networks with filter neurons and a delay&sum readout. Technical report, Institute for Theoretical Computer Science, Technische Universität Graz.

Hopfield, J. and Brody, C. D. (2000). What is a moment? "Cortical" sensory integration over a brief interval. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 97, pages 13919–13924.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Science USA*, 79(8):2554–2558.

Izhikevich, E. M. (2007). *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. The MIT Press.

Jaeger, H. (2001a). The "echo state" approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology.

Jaeger, H. (2001b). Short term memory in echo state networks. Technical Report GMD Report 152, German National Research Center for Information Technology.

Jaeger, H. (2002). 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.

Jaeger, H. (2003). Adaptive nonlinear system identification with echo state networks. In *Advances in Neural Information Processing Systems*, pages 593–600.

Jaeger, H. (2005). Reservoir riddles: Sugggestions for echo state network research (extended abstract). In *Proceedings of the International Joint Conference on Neural Networks*, pages 1460–1462.

Jaeger, H. (2007). Discovering multiscale dynamical features with hierarchical echo state networks. Technical report, Jacobs University.

Jaeger, H. and Haas, H. (2004). Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication. *Science*, 308:78–80.

Jaeger, H., Lukosevicius, M., and Popovici, D. (2007). Optimization and applications of echo state networks with leaky integrator neurons. *Neural Networks*, 20:335–352.

Jain, L. C. and Vemuri, V. (1999). *Industrial applications of neural networks*. CRC Press, Boca Raton.

Jones, B., Stekel, D., Rowe, J., and Fernando, C. (2007). Is there a liquid state machine in the bacterium escherichia coli? pages 187–191.

Joshi, P. and Maass, W. (2004). Movement generation and control with generic neural microcircuits. In *Proc. of BIO-AUDIT*, pages 16–31.

Joshi, P. and Maass, W. (2005). Movement generation with circuits of spiking neurons. *Neural Computation*, 17(8):1715–1738.

Karacs, K. and Roska, T. (2006). Route number recognition of public transport vehicles via the bionic eyeglass. In *Proceedings of the 10th International Workshop on Cellular Neural Networks and their Applications (CNNA'06)*, pages 28–30.

Karhunen, J., Hyvärinen, A., and Oja, E. (2004). *Independent component analysis*. Wiley-Interscience.

Kilian, J. and Siegelmann, H. (1996). The dynamic universality of sigmoidal neural networks. *Information and Computation*, 128:48–56.

Kohonen, T. (2001). *Self-organizing maps*. Springer.

Korbel, P. and Slot, K. (2006). Modeling of elastic inter-node bounds in Cellular Neural Network-based implementation of the deformable grid paradigm. In *10th International Workshop on Cellular Neural Networks and Their Applications, 2006. CNNA'06.*, pages 1–6.

Langton, C. G. (1990). Computation at the Edge of Chaos - Phase-transitions and Emergent Computation. *Physica D*, 42(1-3):12–37.

Lazar, A., Pipa, G., and Triesch, J. (2007). Fading memory and times series prediction in recurrent networks with different forms of plasticity. *Neural Networks*, 20(3):312–322.

LeCun, Y., Jackel, L. D., Boser, B., Denker, J. S., Graf, H. P., Guyon, I., Henderson, D., Howard, R. E., and Hubbard, W. (1989). Handwritten digit recognition: Applications of neural net chips and automatic learning. *IEEE Communication*, pages 41–46.

Legenstein, R. and Maass, W. (2005). *New Directions in Statistical Signal Processing: From Systems to Brain*, chapter What makes a dynamical system computationally powerful? MIT Press.

Legenstein, R. A. and Maass, W. (2007). Edge of chaos and prediction of computational performance for neural microcircuit models. *Neural Networks*, pages 323–333.

Liebald, B. (2004). Exploration of effects of different network topologies on the esn signal crosscorrelation matrix spectrum. Master's thesis, School of Engineering & Science at International University Bremen.

Lloyd, S. (2002). Computational capacity of the universe. *Physical Review Letters*, 88(23):237901–237901.

Lukosevicius, M. and Jaeger, H. (2007). Overview of Reservoir Recipes. Technical report, Jacobs University.

Lukosevicius, M. and Jaeger, H. (2009). Reservoir Computing Approaches to Recurrent Neural Network Training. *Computer Science Review*, 3(3):127–149.

Lukosevicius, M., Popovici, D., Jäger, H., and Siewert, U. (2006). Time-warping invariant echo state networks. *Jacobs University technical report*, 2:1–15.

Lyapunov, A. (1966). *Stability of motion*. Academic Press, New-York and London.

Lyon, R. (1982). A computational model of filtering, detection and compression in the cochlea. In *Proceedings of the IEEE ICASSP*, pages 1282–1285.

Maass, W. (1997). Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons. In Mozer, M., Jordan, M. I., and Petsche, T., editors, *Advances in Neural Information Processing Systems*, volume 9, pages 211–217. MIT Press (Cambridge).

Maass, W. and Bishop, C. (2001). *Pulsed Neural Networks*. Bradford Books/MIT Press, Cambridge, MA.

Maass, W., Joshi, P., and Sontag, E. D. (2007). Computational aspects of feedback in neural circuits. *PLOS Computational Biology*, 3(1):e165, 1–20.

Maass, W., Legenstein, R. A., and Markram, H. (2002a). A new approach towards vision suggested by biologically realistic neural microcircuit models. In *Proc. of the 2nd Workshop on Biologically Motivated Computer Vision*, Lecture Notes in Computer Science. Springer. in press.

Maass, W., Natschläger, T., and H., M. (2004a). Fading memory and kernel properties of generic cortical microcircuit models. *Journal of Physiology*, 98(4-6):315–330.

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

Maass, W., Natschläger, T., and Markram, H. (2003). A model for real-time computation in generic neural microcircuits. In *Proceedings of NIPS*, volume 15, pages 229–236. MIT Press.

Maass, W., Natschläger, T., and Markram, H. (2004b). Computational models for generic cortical microcircuits. In Feng, J., editor, *Computational Neuroscience: A Comprehensive Approach*, chapter 18, pages 575–605. Chapman & Hall/CRC, Boca Raton.

Maass, W., Natschläger, T., and Markram, H. (2004c). Fading memory and kernel properties of generic cortical microcircuit models. *Journal of Physiology – Paris*, 98(4–6):315–330.

Mallat, S. and Zhang, Z. (1993). Matching pursuit with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41:3397–3415.

Meddis, R. (1986). Simulation of mechanical to neural transduction in the auditory receptor. *The Journal of the Acoustical Society of America*, 79:702.

Merlet, J. (2006). Jacobian, manipulability, condition number, and accuracy of parallel robots. *Journal of Mechanical Design*, 128:199.

Minsky, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA.

Mitchell, M., Crutchfield, J., and Hraber, P. (1994). Dynamics, Computation, and the" Edge of Chaos": A Re-Examination. 19:497–497.

Moody, J. (1992). The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In *Proceedings of NIPS*, volume 4, pages 847–854.

Nagy, Z., Voroshazi, Z., and Szolgay, P. (2005). An emulated digital retina model implementation on FPGA. In *9th International Workshop on Cellular Neural Networks and Their Applications, 2005*, pages 278–281.

Omlin, C. W. and Giles, C. L. (1994). Constructing deterministic finite-state automata in sparse recurrent neural networks. In *IEEE International Conference on Neural Networks (ICNN'94)*, pages 1732–1737, Piscataway, NJ. IEEE Press.

Oubbati, M., Levi, P., Schanz, M., and Buchheim, T. (2005). Velocity control of an omnidirectional robocup player with recurrent neural networks. In *Proceeding of the Robocup Symposium*, pages 691–701.

Ozturk, M. C., Xu, D., and Principe, J. C. (2006). Analysis and design of echo state networks. *Neural Computation*, 19:111–138.

Patterson, R., Robinson, K., Holdsworth, J., McKeown, D., Zhang, C., and Allerhand, M. (1992). Complex sounds and auditory images. *Auditory physiology and perception*, pages 429–446.

Penrose, R. (1955). A generalized inverse for matrices. 51(1955):406–413.

Pesin, Y. (1977). Characteristic Lyapunov exponents and smooth ergodic theory. *Russian Mathematical Surveys*, 32(4):55–114.

Plöger, P. G., Arghir, A., Günther, T., and Hosseiny, R. (2004). Echo state networks for mobile robot modeling and control. In *RoboCup 2003: Robot Soccer World Cup VII*, pages 157–168.

Principe, J., Xu, D., and Fisher, J. (2000). Information theoretic learning. *Unsupervised adaptive filtering*, pages 265–319.

Prokhorov, D. (2007). Training recurrent neurocontrollers for real-time applications. *IEEE Transactions on Neural Networks*, 18:1003–1015.

Puskorius, G. V. and Feldkamp, L. A. (1994). Neurocontrol of nonlinear dynamical systems with kalman filter trained recurrent networks. *IEEE Transactions on Neural Networks*, 5:279–297.

Rao, Y., Kim, S.-P., Sanchez, J., Erdogmus, D., Principe, J., Carmena, J., Lebedev, M., and Nicolelis, M. (2005). Learning mappings in brain machine interfaces with echo state networks. In *Proceedings of the 2005 IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 233–236.

Riedmiller, M. and Braun, H. (1993). A direct adaptive method for fast backpropagation learning: The RPROP algorithm. In Ruspini, H., editor, *Proceedings of the IEEE International Conference on Neural Networks*, pages 586–591, San Francisco.

Robinson, A. J. (1994). An application of recurrent nets to phone probability estimation. *IEEE Transactions on Neural Networks*, 5:298–305.

Rodriguez, P. (2001). Simple recurrent networks learn context-free and context-sensitive languages by counting. *Neural Computation*, 13(9):2093–2118.

Rodriguez-Vazquez, A., Linan-Cembrano, G., Carranza, L., Roca-Moreno, E., Carmona-Galan, R., Jimenez-Garrido, F., Dominguez-Castro, R., and Meana, S. (2004). ACE16k: the third generation of mixed-signal SIMD-CNN ACE chips toward VSoCs. *IEEE Transactions on Circuits and Systems I*, 51(5):851–863.

Roska, T. and Chua, L. (1993). The CNN universal machine: an analogic array computer. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 40(3):163–173.

Rumelhart, D., Hinton, G., and Williams, R. (1986). *Parallel Distributed Processing*, chapter Learning internal representations by error propagation. MIT Press, Cambridge, MA.

Ryan, T. P. (1997). *Modern Regression Methods*. Wiley and sons.

Salmen, M. and Plöger, P. G. (2005). Echo state networks used for motor control. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1953–1958.

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

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

Scholkopf, B. and Smola, A. (2002). *Learning with kernels*. MIT press Cambridge, Mass.

Scholkopf, B., Smola, A., and Muller, K. (1997). Kernel principal component analysis. *Lecture notes in computer science*, 1327:583–588.

Schraudolph, N., Dayan, P., and Sejnowski, T. (1994). Temporal Difference Learning of Position Evaluation in the Game of Go. *Advances in Neural Information Processing*, pages 817–817.

Schrauwen, B. (2008). *Towards applicable spiking neural networks*. PhD thesis.

Schrauwen, B., Busing, L., and Legenstein, R. (2008a). On Computational Power and the Order-Chaos Phase Transition in Reservoir Computing. In *Proceedings of NIPS*.

Schrauwen, B., Defour, J., Verstraeten, D., and Van Campenhout, J. (2007a). The introduction of time-scales in reservoir computing, applied to isolated digits recognition. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*.

Schrauwen, B. and Van Campenhout, J. (2003). BSA, a fast and accurate spike train encoding scheme. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 2825–2830.

Schrauwen, B., Verstraeten, D., and Van Campenhout, J. (2007b). An overview of reservoir computing: theory, applications and implementations. In *Proceedings of the European Symposium on Artifical Neural Networks (ESANN)*.

Schrauwen, B., Warderman, M., Verstraeten, D., Steil, J. J., and Stroobandt, D. (2008b). Improving reservoirs using intrinsic plasticity. *Neurocomputing*, 71:1159–1171.

Seneff, S. (1988). A joint synchrony/mean-rate model of auditory speech processing. *Journal of Phonetics*, 16:1.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656.

Siewert, U. and Wustlich, W. (2007). Echo-state networks with band-pass neurons: Towards generic time-scale-independent reservoir structures. Technical report, Planet GmbH.

Simard, P., Steinkraus, D., and Platt, J. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, pages 958–963.

Skowronski, M. D. and Harris, J. G. (2006). Minimum mean squared error time series classification using an echo state network prediction model. In *IEEE International Symposium on Circuits and Systems*.

Sprott, J. C. (2003). *Chaos and Time–Series Analysis*. Oxford University Press.

Steil, J. J. (2004). Backpropagation-Decorrelation: Online recurrent learning with O(N) complexity. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 1, pages 843–848.

Steil, J. J. (2005a). Memory in backpropagation-decorrelation O(N) efficient online recurrent learning. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*.

Steil, J. J. (2005b). Stability of backpropagation-decorrelation efficient O(N) recurrent learning. In *Proceedings of ESANN'05*, Brugge.

Steil, J. J. (2006). Online stability of backpropagation-decorrelation recurrent learning. *Neurocomputing*, 69:642–650.

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

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

Steinwart, I. and Christmann, A. (2008). *Support Vector Machines*. Springer Verlag.

Ster, B., Dobnikar, A., et al. (1996). Neural networks in medical diagnosis: Comparison with other methods. In *Proceedings of the International Conference EANN*, volume 96, pages 427–430.

Suykens, J. and Vandewalle, J., editors (1998). *Nonlinear Modeling: Advanced Black-Box Techniques*, chapter Enhanced Multi-Stream Kalman Filter Training for Recurrent Networks, page 29?53. Kluwer Academic Publishers.

Suykens, J., Vandewalle, J., and De Moor, B. (1996). *Artificial Neural Networks for Modeling and Control of Non-Linear Systems*. Springer.

Takens, F., Rand, D. A., and L.-S., Y. (1981). Detecting strange attractors in turbulence. In *Dynamical Systems and Turbulence*, volume 898 of *Lecture Notes in Mathematics*, pages 366–381. Springer-Verlag.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *J. Royal. Statist. Soc B.*, 58(1):267–288.

Tikhonov, A. and Arsenin, V. Y. (1977). *Solutions of Ill-Posed Problems*. Winston and Sons.

Tong, M. H., Bickett, A. D., Christiansen, E. M., and Cottrell, G. W. (2007). Learning grammatical structure with echo state networks. *Neural Networks*, 20(3):424–432.

Török, L. and Zarándy, Á. (2002). CNN based color constancy algorithm. *Cellular Neural Networks and Their Applications*, page 452.

Triesch, J. (2005). A gradient rule for the plasticity of a neuron's intrinsic excitability. In *Proc. of the Int. Conf. on Artificial Neural Networks (ICANN)*.

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

Turing, A. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2):230–265.

Turing, A. (1950). Computing machinery and intelligence. *Mind*, 59(236):433–460.

Turrigiano, G. and Nelson, S. (2004). Homeostatic plasticity in the developing nervous system. *Nature Reviews Neuroscience*, 5(2):97–107.

Valero-Cuevas, F., Yi, J., Brown, D., McNamara, R., Paul, C., and Lipson, H. (2007). The tendon network of the fingers performs anatomical computation at a macroscopic scale. *IEEE Transactions on Biomedical Engineering*, 54(6 Part 2):1161–1166.

Van Gestel, T., Suykens, J., De Brabanter, J., De Moor, B., and Vandewalle, J. (2001). Kernel canonical correlation analysis and least squares support vector machines. *Lecture notes in computer science*, pages 384–389.

Van Hese, P., Martens, J., Boon, P., Dedeurwaerdere, S., Lemahieu, I., and Van de Walle, R. (2003). Detection of spike and wave discharges in the cortical EEG of genetic absence epilepsy rats from Strasbourg. *Physics in Medicine and Biology*, 48(12):1685–1700.

Van Immerseel, L. and Martens, J.-P. (1993). Pitch and voiced/unvoiced determination with an auditory model. *Journal of the Acoustical Society of America*, 91(6):3511–3526.

Vandoorne, K. (2009). Photonic Reservoir Computing with SOAs: Memory task and NARMA10 task. Technical report, INTEC, Ghent University.

Vandoorne, K., Dierckx, W., Schrauwen, B., Verstraeten, D., Baets, R., Bienstman, P., and Van Campenhout, J. (2008). Toward optical signal processing using photonic reservoir computing. *Optics Express*, 16(15):11182–11192.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag, New York.

Vapnik, V. N. (1999). *The nature of statistical learning theory*. Statistics for engineering and information science. Springer, second edition edition.

Venayagamoorthy, G. (2007). Online design of an echo state network based wide area monitor for a multimachine power system. *Neural Networks*, 20(3):404–413.

Venetianter, P. and Roska, T. (1998). Image compression by cellular neural networks. *IEEE transactions on circuits and systems I: fundamental theory and applications*, 45(3):205–215.

Verstraeten, D. (2004). Een studie van de Liquid State Machine: een woordherkenner. Master's thesis, Ghent University, ELIS department.

Verstraeten, D., Schrauwen, B., D'Haene, M., and Stroobandt, D. (2007). A unifying comparison of reservoir computing methods. *Neural Networks*, 20:391–403.

Verstraeten, D., Schrauwen, B., and Stroobandt, D. (2006). Reservoir-based techniques for speech recognition. In *Proceedings of the World Conference on Computational Intelligence*, pages 1050–1053.

Verstraeten, D., Schrauwen, B., Stroobandt, D., and Van Campenhout, J. (2005). Isolated word recognition with the liquid state machine: a case study. *Information Processing Letters*, 95(6):521–528.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions on acoustics, speech and signal processing*, 37(3):328–339.

Walker, W., Lamere, P., Kwok, P., Raj, B., Singh, R., Gouvea, E., Wolf, P., and Woelfel, J. (2004). Sphinx-4: A flexible open source framework for speech recognition. Technical report, Sun Microsystems Inc.

Wardermann, M. and Steil, J. J. (2007). Intrinsic plasticity for reservoir learning algorithms. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN)*.

Werbos, P. J. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Applied Mathematics, Harvard University, Boston, MA.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proc. IEEE*, 78(10):1550–1560.

White, O., Lee, D., and Sompolinsky, H. (2002). Short-term memory in orthogonal neural networks. *Neural Comput Phys Rev Lett*, 92:148102.

Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280.

Wolff, R. (1992). Local Lyapunov exponents: looking closely at chaos. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 353–371.

Wolfram, S. (2002). *A New Kind of Sciences*. Wolfram Media, Inc.

wyffels, F., Schrauwen, B., and Stroobandt, D. (2008a). Stable output feedback in reservoir computing using ridge regression. In *International Conference on Artificial Neural Networks*.

wyffels, F., Schrauwen, B., and Stroobandt, D. (2008b). Using reservoir computing in a decomposition approach for time series prediction. In Lendasse, A., editor, *Proceedings of the European Symposium on Time Series Prediction*, pages 149–158, Porvoo. Multiprint Oy / Otamedia.

wyffels, F., Schrauwen, B., Verstraeten, D., and Stroobandt, D. (2008c). Band-pass reservoir computing. In Hou, Z. and Zhang, N., editors, *Proceedings of the International Joint Conference on Neural Networks*, pages 3203–3208, Hong Kong.

Xavier de Souza, S., Suykens, J., Vandewalle, J., and D., B. (2006). Cooperative behavior in coupled simulated annealing processes with variance control. In *Proc. of the International Symposium on Nonlinear Theory and its Applications (NOLTA)*.

Xavier-de Souza, S., Van Dyck, M., Suykens, J., and Vandewalle, J. (2006). Fast and Robust Face Tracking for CNN chips: application to wheelchair driving. pages 200–205.

Yalcin, M., Suykens, J., and Vandewalle, J. (2005). *Cellular neural networks, multi-scroll chaos and synchronization*. World Scientific.

Yang, T. and Chua, L. (2001). Testing for local activity and edge of chaos. *Int. J. Bifurcation and Chaos*, 11:1495–1591.

Zhang, W. and Linden, D. J. (2003). The other side of the engram: Experience-driven changes in neuronal intrinsic excitability. *Nature Reviews Neuroscience*, 4:885–900.

Ziehmann, C., Smith, L., and Kurths, J. (1999). The bootstrap and Lyapunov exponents in deterministic chaos. *Physica D*, 126(1-2):49–59.