Echo State Networks with Filter Neurons and a Delay&Sum Readout with Applications in Audio Signal Processing

Georg Holzmann
grh@mur.at

Echo State Networks with Filter Neurons and a Delay&Sum Readout with Applications in Audio Signal Processing

Master's Thesis

at

Graz University of Technology

submitted by

Georg Holzmann

Institute for Theoretical Computer Science,
Graz University of Technology
A-8010 Graz, Austria

June 2008

Advisor: o.Univ.-Prof. Dr. Wolfgang Maass
Co-Advisor: DI Helmut Hauser
Co-Advisor: Dr. Dmitriy Shutin

Echo State Networks mit Filterneuronen und einem Delay&Sum Readout mit Anwendungen in der Audio-Signalverarbeitung

Diplomarbeit

an der

Technischen Universität Graz

vorgelegt von

Georg Holzmann

Institut für Grundlagen der Informationsverarbeitung,
Technische Universität Graz
A-8010 Graz

Juni 2008

Diese Arbeit ist in englischer Sprache verfasst.

Begutachter:	o.Univ.-Prof. Dr. Wolfgang Maass
Mitbetreuer:	DI Helmut Hauser
Mitbetreuer:	Dr. Dmitriy Shutin

Abstract

Echo state networks (ESNs) are a novel approach to recurrent neural network training with the advantage of a very simple and linear learning algorithm. They can in theory approximate arbitrary nonlinear dynamical system with arbitrary precision (universal approximation property), have an inherent temporal processing capability, and are therefore a very powerful enhancement of linear blackbox modeling techniques in nonlinear domain. It was demonstrated on a number of benchmark tasks, that echo state networks outperform other methods for nonlinear dynamical modeling.

This thesis suggests two enhancements of the original network model. First, the previously proposed idea of filters in neurons is extended to arbitrary infinite impulse response (IIR) filter neurons and the ability of such networks to learn multiple attractors is demonstrated. Second, a delay&sum readout is introduced, which adds trainable delays in the synaptic connections of output neurons and therefore vastly improves the memory capacity of echo state networks. It is shown in benchmark tasks that this new structure is able to outperform standard ESNs and other models, moreover no other comparable method for sparse nonlinear system identification with long-term dependencies could be found in literature.

Finally real-world applications in the context of audio signal processing are presented and compared to state-of-the-art alternative methods. The first example is a nonlinear system identification task of a tube amplifier and afterwards ESNs are trained for nonlinear audio prediction, as necessary in audio restoration or in the wireless transmission of audio where dropouts may occur. Furthermore an efficient and open source C++ library for echo state networks was developed and is briefly presented.

Kurzfassung

Echo State Networks (ESNs) sind eine neueartige Form von Rekurrenten Künstlichen Neuronalen Netzen und besitzen einen besonders einfachen und linearen Lernalgorithmus. Theoretisch sind sie fähig, jedes nichtlineare dynamische System mit beliebiger Genauigkeit zu approximieren (Universal Approximation Property) und sind somit eine sehr leistungsstarke Erweiterung von linearen Black-Box Techniken. In mehreren Benchmark-Tests wurde demonstriert, dass Echo State Networks anderen Methoden zur Modellierung von nichtlinearen dynamischen Systemen oft weit überlegen sind.

Diese Diplomarbeit schlägt zwei mögliche Erweiterungen des Netzwerkmodells vor. Erstens wird die bereits vorgestellte Idee von Filtern in den Neuronen erweitert zu allgemeineren Infinite Impulse Response (IIR) Filterneuronen und die Fähigkeit solcher Netzwerke mehrere Attraktoren gleichzeitig zu lernen wird demonstriert. Zweitens wird ein Delay&Sum Readout eingeführt, welcher lernbare Verzögerungen in den synaptischen Verbindungen der Ausgansneuronen hinzufügt und somit die Speicherkapazität von Echo State Networks stark verbessert. An einigen vielverwendeten Benchmark-Tests wird demonstriert, dass diese neue Struktur in der Lage ist Standard-ESNs und andere Modelle zu überbieten.

Schließlich werden praktische Anwendungen aus dem Bereich der Audio-Signalverarbeitung präsentiert und mit alternativen Methoden verglichen. Im ersten Beispiel handelt es sich dabei um eine nichtlineare Systemidentifikation von analogen Röhrenverstärkern und danach werden ESNs zur nichtlinearen Audioprädiktion verwenden, wie es bei Aussetzern in der Funkübertragung oder in der Restauration von alten Musikaufnahmen erforderlich ist. Zudem wurde eine effiziente und freie (open source) C++ Bibliothek für Echo State Networks implementiert, welche kurz vorgestellt wird.

Pledge of Integrity

I hereby certify that the work presented in this thesis is my own, that all work performed by others is appropriately cited, and that I have not submitted this thesis elsewhere.

Place:

Date:

Signature:

Eidesstattliche Erklärung

Ich versichere ehrenwörtlich, dass ich diese Arbeit selbständig verfasst habe, dass ich andere als die angegebenen Quellen nicht benutzt habe, dass jede Quelle gekennzeichnet ist und dass ich diese Arbeit an keiner anderen Stelle eingereicht habe.

Ort:

Datum:

Unterschrift:

Contents

List of Figures

List of Tables

Acknowledgements

Many thanks to Wolfgang Maass, who supported me and this thesis from the beginning, without him I probably would have made something completely different. I especially wish to thank Helmut Hauser for his fast answers to my questions and endless hours of correcting draft versions of this thesis, and Dmitriy Shutin for all his ideas and explanations.

I also like to thank Alois Sontacchi for his interest and aid, Friedrich Schäfer for many valuable discussions, Gernot Kubin for support and encouragement, Brigitte Bergner for assistance and of course Daniela Potzinger.

Furthermore special mention goes to Michael Lehn, people from the *music-dsp*, *scipy* and *numpy* mailing list, people form the reservoir computing mailing list, in especially Wolf Wüstlich for his comments, and all people from the Institute for Theoretical Computer Science, who always answered my questions.

Last but not least, without the support of my whole family, my study and this thesis would not have been possible. In the final and critical phase of this writing, where my computer died, my mother was even so kind to give me her own one.

Georg Holzmann
Graz, Austria, June 2008

Credits

I would like to thank the following individuals and organizations for permission to use their material:

- Herbert Jäger for some figures of his papers, which are used in chapter 2.
- Code for Volterra Systems by Dmitriy Shutin.
- Permission to use the original soundfiles from the thesis of Hubert Außerlechner about dropout concealment algorithms (Ausserlechner [2006]).
- All simulations were implemented with *NumPy* (Oliphant [2007]), an open source library for fast N-dimensional array manipulation in *python*, and *SciPy* (Jones et al. [2001–]), a library for scientific computation build on top of *NumPy*.
- Plots were generated with the *python* library *Matplotlib* (Hunter [2007]).
- Michael Lehn for FLENS (Flexible Library for Efficient Numerical Solutions - Lehn [2008]).
- All diagrams were created with the open source program *Dia*.
- This thesis was written in LaTeX using the skeleton from Andrews [2006].

Chapter 1

Introduction

Signal processing algorithms try to handle complex datastreams observed in the environment. Most of the classical signal processing methods, which resulted in numerous essential applications and are indispensable to life today, are founded on three basic assumptions: linearity, stationarity and second-order statistics with emphasis on Gaussianity (Haykin [1996]). These assumptions are made for the sake of mathematical tractability. Nevertheless are most or maybe all the physical signals that one has to deal with in real life generated by dynamic processes which are simultaneously nonlinear, nonstationary and non-Gaussian. If one wishes to simulate, predict, filter, classify or control nonlinear dynamical systems, one needs an executable system model and often it is infeasible to obtain analytical models, so one has to use blackbox modeling techniques.

The most powerful computational system we know of is the brain. It consists of billions of recurrently connected neurons and enables living beings to learn, memorize and navigate in a nonstationary, complex world. Inspired by biological neural networks McCulloch and Pitts [1943] introduced the idea of an artificial neural network with simple threshold units (perceptrons). Up to now numerous neuron types and network structures have been proposed and analyzed. One of the most common structure is the feedforward neural network, which consists of multiple neuron layers without recurrent connections. A more complex architecture is called recurrent neural network (RNN), which has at least one cyclic path of synaptic connections. Mathematically RNNs implement dynamical systems and can in theory approximate arbitrary nonlinear dynamical system with arbitrary precision (universal approximation property, Siegelmann and Sontag [1991]), they are also able to (re)produce temporal patterns. However, recurrent neural networks are very hard to train and a number of specialized learning algorithms exist in literature, but usually they are difficult to use and lead to suboptimal solutions. A new and surprisingly easy to use network structure for RNNs was discovered independently in Jaeger [2001], who called these RNNs echo state networks (ESN), and in Maass et al. [2002], who developed a similar approach for spiking neural networks and called the structure liquid state machine (LSM). Both methods are subsummed under the more general term reservoir computing (Verstraeten et al. [2007a]).

Classical approaches for nonlinear signal processing consists of designing specific algorithms for specific problems (for instance median and bilinear filters, special Volterra filter structures, polynomial filters, functional links, ... see Uncini [2003]). Therefore reservoir computing techniques could extend and generalize currently used linear methods in nonlinear domain, resulting in a tool which is not designed for only one specific application but can be trained and adapted to many practical problems.

1.1 Reservoir Computing

Reservoir computing is a term for recently emerged supervised learning techniques for recurrent neural networks. Its main representatives are echo state networks (Jaeger [2001]), liquid state machines (Maass et al. [2002]) and a few other models like backpropagation decorrelation (Steil [2004]). The common idea is that input signals are fed into a fixed, random dynamical system, called dynamic reservoir or liquid, which is composed of recurrently connected neurons, and only output connections, the readout, are trainable. Most implementations of the liquid state machine use a spiking neuron model called leaky integrate and fire (LIF) neuron (Maass and Bishop [2001]), whereas echo state networks are composed out of analog neurons, for instance linear, sigmoid or leaky integrator units. The function of the reservoir can be compared to that of the kernel in support vector machines (Cristianini and Shawe-Taylor [2000]): input signals drive the nonlinear reservoir and produce a high-dimensional dynamical "echo response", which is used as a non-orthogonal basis to reconstruct the desired outputs by a linear combination. This strategy has the advantage that the recurrent network is fixed and simple offline or online algorithms for linear regression can compute the output weights, therefore training cannot get stuck in local minima of the mean squared error function.

On a first view it might look that a fixed reservoir must not work satisfactorily. However, Schiller and Steil [2005] showed that also in traditional training methods for RNNs, where all weights are adapted, the dominant changes are in the output weights. It was also demonstrated on a number of benchmark tasks (for instance Jaeger and Hass [2004]), that reservoir computing methods outperform other methods for nonlinear dynamical modeling in many applications. Furthermore classical RNN learning algorithms adapt the connections by some sort of gradient descent, which renders them slow, is biologically implausible and convergence cannot be guaranteed.

Reservoir computing belongs to the family of computational methods with a clear biological footing (Jaeger et al. [2007a]) and a related mechanism has been investigated in cognitive neuroscience by Dominey [1995] in the context of modelling sequence processing in mammalian brains, especially speech recognition in humans. Moreover it was shown in Maass et al. [2002] and Maass et al. [2007] that reservoir computing techniques have a universal computation and approximation property and can realize every nonlinear filter with bounded memory arbitrarily well. Altogether they offer an attractive method for solving complicated engineering and especially signal processing tasks. In the rest of this thesis the discussion will be restricted to echo state networks, because applications with analog inputs and analog outputs will be considered.

1.2 Extensions to the Model

The analysis of two problems with echo state networks is the main contribution of this work. Reservoir neurons are connected to each other, therefore their states are coupled and it is not possible to learn multiple attractors or oscillators with one network. For instance a standard nonlinear ESN cannot be a generator for a sum of multiple sines at different frequencies. While additive superpositions of sines are easy to handle in a linear systems view, nonlinear systems do not lend themeselves to noninterfering additive couplings and a number of unwanted dynamic side effects can happen (Jaeger et al. [2007b]). If reservoir neurons have additional build in filters, "specialized" neurons tuned to different frequencies can emerge and more diverse echoes, which are able to model multiple attractors, will evolve because not all neurons are coupled. Wustlich and Siewert [2007] introduced the idea of filter neurons as an extension and combination of ESNs with leaky-integrator units, here this idea is extended to an arbitrary order infinite impulse

response (IIR) filter neuron (chapter 3), which allows to reuse commonly known structures from filter theory.

A second problem is the memory capacity of ESNs, how many data points from the past are actually relevant for the computation of the current outputs. It was shown in Jaeger [2002a] that the memory capacity is restricted by the reservoir size. Especially in signal processing applications, which often use very high sampling rates, this would imply that one has to deal with very big reservoirs (> 100000 neurons, consider for instance an acoustic echo cancellation task), which is not trivial to implement on current computers. In chapter 4 an alternative approach to handle long-term dependencies is introduced, which adds trainable delays in the synaptic connections of readout neurons. The method was inspired by the delay&sum beamformer as used in acoustical multiple-input multiple-output (MIMO) signal processing with microphone arrays (for instance Y.Huang et al. [2006]) and is therefore named delay&sum readout. Compared to standard echo state networks, where only weights of the readout are adjusted in training, a learning algorithm is introduced which modifies delays and weights of the delay&sum readout. Viewed from a biological perspective, also axons insert delays in connections between neurons. Although axonal transmission delays do not vary continually in the brain, a wide range of delay values have been observed (Paugam-Moisy et al. [2008]). Neuroscience experiments in Swadlow [1985] give also evidence to the variability of transmission delay values from 0.1 to 44 ms, see also Bringuier et al. [1999] for a more recent investigation.

The introduction of delays makes it possible to shift reservoir signals in time and is a computationally cheap method to vastly improve the memory capacity, furthermore it boosts also the performance of ESNs trained as a generator, where long-term dependencies are not important. Simulations in chapter 5 show that also in chaotic timeseries prediction (Mackey-Glass System), where ESNs vastly outperformed all other techniques for nonlinear system modeling (Jaeger and Hass [2004]), a delay&sum readout is still able to improve the results. Moreover no other method with comparable performance for sparse nonlinear system identification (section 5.3) of higher order systems could be found.

1.3 Audio Signal Processing Examples

Nonlinear signal processing with neural networks has already been investigated in Haykin [1996], Kung [1997], Feldkamp and Puskorius [1998] or Uncini [2003]. Most of them used feedforward structures, but some also tried to model more complex systems with recurrent neural networks or other very special structures. Echo state networks could be seen as an unification of those investigations, which are very powerful, can cope with all the studied tasks, and are most of the time much easier to use.
Two main signal processing problems are addressed with nonlinear systems: system identification tasks and time series forecasting. Chapter 6 analyzes echo state networks for both problems and compares the result to state-of-the-art alternative models in the context of audio signal processing. A third big area of applications would be classification (music information retrieval, speech recognition, etc.), which is not studied in this thesis. Especially for real-world problems the delay&sum readout and filter neurons vastly improved the performance.

With the introduction of digital methods in music reproduction the first major effort was to eliminate a number of technical artifacts produced by traditional analog audio systems so far (Schattschneider and Zoelzer [1999]). Today, decades later, it seems desirable to bring some nonlinearities of the old equipment back into the all digital processing systems, because of their perceptible characteristic distorsions. Tube amplifiers are one example of nonlinear systems,

which are still used today because of the "warmth" and "dirt" of its sound. Volterra series were utilized in literature to simulate tubes on digital computers (Helie [2006]), but usually only second or third-order systems with special methods for complexity reduction performed exceptable. Simulations in chapter 6 show that echo state networks are an easy to use alternative tool for blackbox modeling of nonlinear devices and that they are able to outperform plain Volterra systems. Furthermore it was demonstrated in Maass and Sontag [2000], that neural networks with time delays are able to approximate all filters that can be characterized by Volterra series.

Nonlinear audio prediction is the second example studied here, where one tries to forecast future samples out of a given history horizon. Such methods are necessary for instance in audio restoration, whenever a sequence of consecutive samples is missing, when impulsive noise appears, or in the wireless transmission of digital signals where short dropouts can occur. Common methods try to model missing data with an autoregressive process (Godsill and Rayner [1997]), with the main drawback of a big model order when filling long dropouts, or use several forms of pattern matching algorithms (see for instance Goodman et al. [1986], Wasem et al. [1988] or Niedzwiecki and Cisowski [2001]), where additional knowledge like pitch information or zero crossing rates are used to fill gaps. Echo state networks are compared to different pattern matching algorithms and a linear autoregressive model and were able to outperform all studied methods in a short-term prediction task.

1.4 Thesis Outline

The contents of this thesis is organized in seven chapters. In the current one a general introduction into the field of reservoir computing and its possible applications was given and afterwards chapter 2 will continue with a more detailed analysis of echo state networks. The basic online and offline learning algorithms, additional nonlinear transformations, short term memory effects and leaky integrator neuron models are discussed. In chapter 3 leaky integrator units are extended to simple band-pass neurons, afterwards to general IIR-filter neurons and possible filter structures are shown. Next the delay&sum readout is introduced in chapter 4 with on overview of general time delay estimation techniques and finally two possible delay learning algorithms are presented.

Three synthetic simulations, which demonstrate the usefulness of filter neurons and a delay&sum readout, are analyzed in chapter 5. The first is a multiple superimposed oscillations task, where filter neurons are mandatory, the second is a common benchmark task, the Mackey-Glass system, and finally the strength of a delay&sum readout is presented in a sparse nonlinear system identification task with long-term dependencies. Real-world audio signal processing examples in the area of system identification (tube amplifier) and audio prediction are discussed and compared to alternative methods in chapter 6. Finally chapter 7 gives a conclusion, some practical hints on when and how one should use the proposed model extensions and possible ideas for future work. A short overview of the implemented open source C++ library for echo state networks, called *aureservoir*, is presented in appendix A.

Chapter 2

Introduction to Echo State Networks

Echo state networks (ESNs), as introduced in Jaeger [2001], are a special kind of recurrent neural networks (RNNs) with a very simple training algorithm. Usually a large, sparsely connected RNN is used as a "dynamic reservoir", which can be excited by inputs and feedback of the outputs. The big advantage of ESNs is that the connection weights of the reservoir are not changed by training and only weights from the reservoir to the output units are adapted, therefore training becomes a simple linear regression task as shown in figure 2.1.

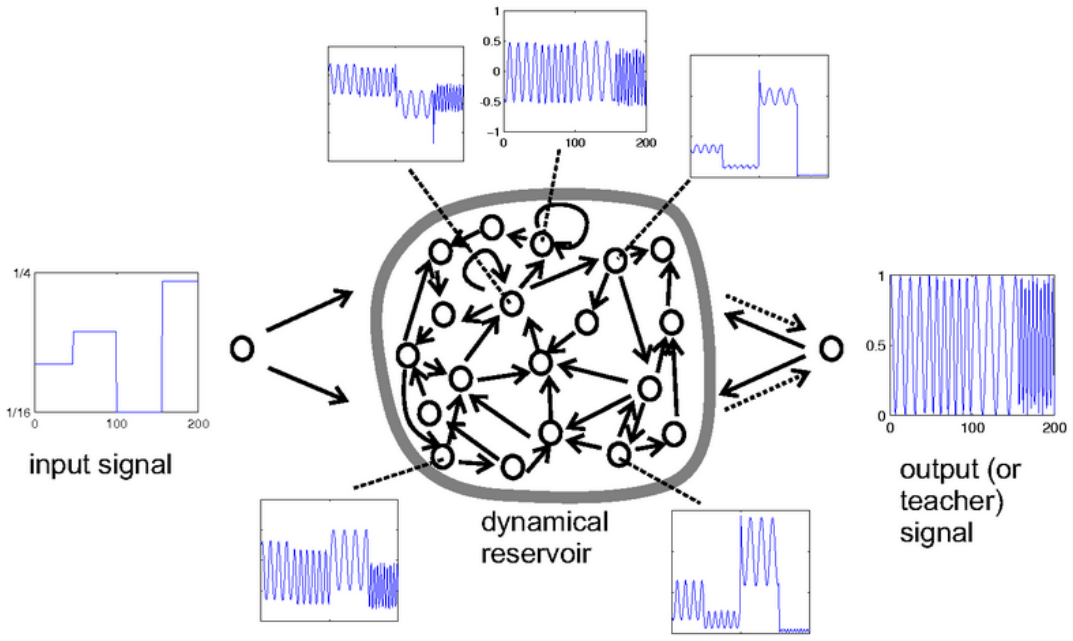

Figure 2.1: In this example an ESN is trained as a tuneable sinewave generator. Solid arrows indicate fixed, random connections and dotted arrows trainable connections. [Image extracted from Jaeger [2007a]]

Difficulties with existing algorithms have so far precluded supervised training techniques for RNNs from widespread use. Unlike with feedforward neural networks, several types of training algorithms are known for RNNs with no clear winner (for example backpropagation through time, real-time recurrent learning, extended kalman filtering approaches, ... see Jaeger [2002b]) and usually these algorithms have a slow convergence and can lead to suboptimal solutions.

Therefore echo state networks are a new, promising technique in supervised training of RNNs. A comparison between a traditional RNN approach and the echo state approach is shown in figure 2.2.

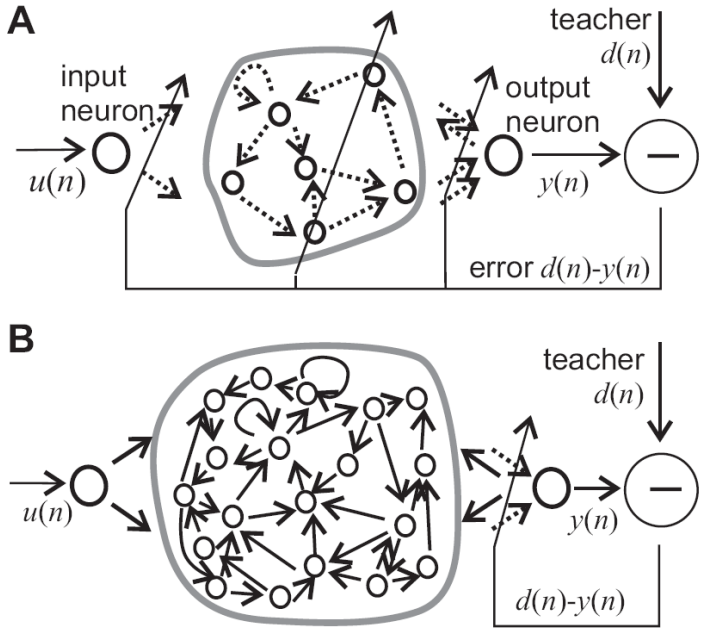

Figure 2.2: **A:** Schema of previous approaches to RNN training, all weights are adapted. **B:** ESN approach, where usually more neurons are used in the dynamic reservoir and only the output weights are adapted. [Image extracted from Jaeger and Hass [2004]]

This chapter will introduce in section 2.1 the notation and basic network state update equations, section 2.2 shows how to initialize echo state networks and which properties they must fulfill and section 2.3 presents online and offline training algorithms. Furthermore it is demonstrated in section 2.4 how to increase the modeling power for hard nonlinear tasks with additional nonlinear transformations, the short term memory of ESNs is analyzed in section 2.5 and finally an alternative, leaky integrator neuron type will be introduced in section 2.6.

2.1 Notation and Activation Update

Echo state networks are able to model nonlinear multiple-input, multiple-output (MIMO) systems. They can be used with output feedback, where the outputs of the network are fed back as additional inputs, which makes it possible to train ESNs as oscillators or generators. The term "generator mode" is used for ESNs which are trained to be a generator of a signal (here output feedback is mandatory) and the term "filter mode" for ESNs which are driven by input signals (output feedback is not necessary). Figure 2.3 shows a single-input, single-output (SISO) ESN without feedback connections, as it is quite common in signal processing, and a more general multiple-input, multiple-output (MIMO) ESN.

Consider an ESN with L inputs, M outputs and a dynamic reservoir with N internal network units. At time $n = 1, 2, ..., n_{max}$ the input vector is $\mathbf{u}(n)$ and the output $\mathbf{y}(n)$. The activations of internal units (neurons in the reservoir) are collected in a $N \times 1$ vector $\mathbf{x}(n) = (x_1(n), ..., x_N(n))$

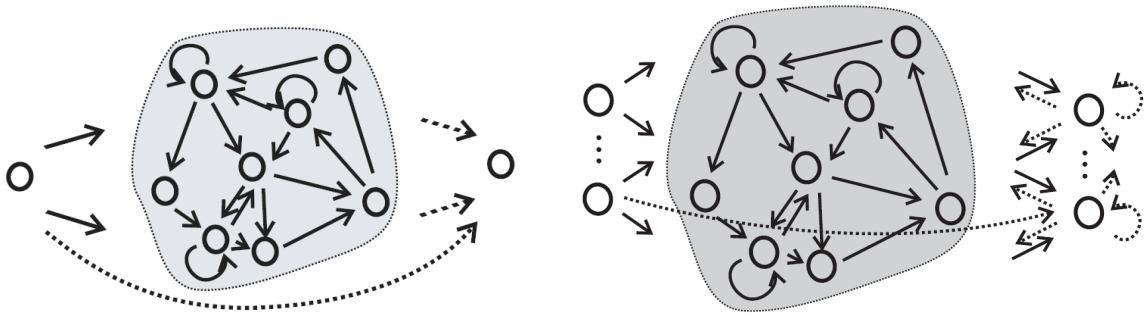

Figure 2.3: Left image is a single-input, single-output ESN without feedback connections, the trainable output weights are dashed arrows. The right image shows a more general multiple-input, multiple-output ESN, where dashed arrows indicate connections that are possible but not required (like output feedback). [Images extracted from Jaeger [2003] and Jaeger [2001]]

and internal connection weights in a $N \times N$ matrix $\mathbf{W}$, weights of input connections in a $L \times N$ matrix $\mathbf{W}^{in}$, feedback weights in a $M \times N$ matrix $\mathbf{W}^{fb}$ and output weights in a $(N + L) \times M$ matrix $\mathbf{W}^{out}$ (input and network to output connections, are the dotted lines in figure 2.3).

The activations of neurons in the dynamic reservoir are updated according to

$$\mathbf{x}(n+1) = f(\mathbf{W}\mathbf{x}(n) + \mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}^{fb}\mathbf{y}(n) + \upsilon(n+1)) \tag{2.1}$$

where $\upsilon(n+1)$ is a small noise term and f is the nonlinear activation function of the dynamic reservoir, for example tanh.

After the internal states, the outputs can be computed with

$$\mathbf{y}(n+1) = g(\mathbf{W}^{out}[\mathbf{x}(n+1); \mathbf{u}(n+1)]) \tag{2.2}$$

where g is again a linear or nonlinear activation function and $[\mathbf{x}(n+1); \mathbf{u}(n+1)]$ means a serial concatenation of the internal state and input vector.

2.2 Network Creation and Echo State Property

In order to be able to model nonlinear fading memory systems echo state networks must have specific properties. Under certain conditions the network states become asymptotically independent of initial conditions and depend only on input history, which is called the "echo state property" as defined in Jaeger [2001]. This means that if the network has been run for a long time, the current network state is uniquely determined by the history of input and output feedback signals and is independent of an initial state vector $\mathbf{x}(n)$.

The echo state property is solely determined by the reservoir weights $\mathbf{W}$, $\mathbf{W}^{in}$ and $\mathbf{W}^{fb}$ of the untrained network, with the requirement that an input sequence $\mathbf{u}(n)$ comes from a compact interval U and a desired output teacher signal $\mathbf{d}(n)$ from a compact interval D. Unfortunately there is no known necessary and sufficient algebraic condition for the echo state property, but there exists a sufficient condition for the non-existence of echo states in Jaeger [2002b]:

> "Assume an untrained network ($\mathbf{W}^{in}$, $\mathbf{W}$, $\mathbf{W}^{fb}$) with state updates according to equation 2.1 with transfer functions tanh. Let $\mathbf{W}$ have a spectral radius $|\lambda_{max}| > 1$,

where λ_{max} is the largest absolute value of an eigenvalue of $\mathbf{W}$. Then the network has no echo states with respect to any input/output interval $U \times D$ containing the zero input/output (0,0)."

However, in practice it was consistently found (Jaeger [2002b]), that if the previous proposition is NOT satisfied, one will have the echo state property. Therefore the following procedure can be used for initializing an echo state network:

- $\mathbf{W}$ is a sparse matrix and randomly generated from a uniform distribution over $[-1, 1]$.
- $\mathbf{W}$ is then normalized to the spectral radius $\alpha < 1$ by scaling $\mathbf{W}$ with $\alpha / |\lambda_{max}|$, where λ_{max} is the largest absolute value of an eigenvalue of $\mathbf{W}$.
- Now the untrained network ($\mathbf{W}^{in}$, $\mathbf{W}$, $\mathbf{W}^{fb}$) has the echo state property, regardless of how $\mathbf{W}^{in}$ and $\mathbf{W}^{fb}$ are chosen.

The spectral radius α is a crucial parameter for the eventual success of an ESN. A small α means that the echoes in the dynamic reservoir will die out early and is suited to model fast signals, whereas a large spectral radius results in a longer short term memory of the reservoir and is useful for slower signals. A more detailed analysis of the short term memory of echo state networks can be found in section 2.5.

2.3 Training Algorithms

A training algorithm collects the internal state vector $x(n)$ over time and tries to compute an optimal mapping from it to a desired output or teacher signal $\mathbf{y}_{teach}(n)$. Finding the right mapping is a linear regression task and if a mean square error is used the regression algorithm will find a global minimum of this error function, which is one of the main advantages of echo state networks. Commonly known online and offline linear regression algorithms can be used for this task.

2.3.1 Offline Training Algorithm

As demonstrated in Jaeger [2001] one tries to compute the output weights $\mathbf{W}^{out}$ such that the mean squared training error MSE_{train} is minimized.
The error $\mathbf{e}_{train}(n)$ at the current timestep is the difference between a target signal $\mathbf{y}_{teach}(n)$ and the network output

$$\mathbf{e}_{train}(n) = g^{-1}(\mathbf{y}_{teach}(n)) - \mathbf{W}^{out}[\mathbf{x}(n); \mathbf{u}_{teach}(n)]$$

where the effect of an output nonlinearity is undone by g^{-1}.

Now the output weights $\mathbf{W}^{out}$ are determined such that $\mathbf{e}_{train}(n)$ is minimized in a mean square error (MSE) sense

$$MSE_{train} = \frac{1}{n_{max} - n_{min}} \sum_{n=n_{min}}^{n_{max}} \mathbf{e}_{train}(n)^2$$

where n_{max} is the number of training examples and n_{min} is an initial washout time. Due to the echo state property the dynamic reservoir needs some time until the effects of initial transients are washed out and these initial samples should not be considered when calculating the output

weights.

In an offline training algorithm, based on the pseudo-inverse, the following steps calculate optimal output weights $\mathbf{W}^{out}$:

1. Consider an ESN with L inputs, M outputs and a dynamic reservoir with N internal network units.
2. Init reservoir matrix $\mathbf{W}$ as discussed in section 2.2 and run the ESN with a teaching input signal $\mathbf{u}_{teach}(n)$.
3. Dismiss data from initial transients where $n < n_{min}$ and collect the remaining input and network states $(\mathbf{u}_{teach}(n), \mathbf{x}(n))$ for each timestep row-wise into a $(n_{max} - n_{min}) \times (N + L)$ matrix $\mathbf{S}$.
4. Collect target signals $g^{-1}(\mathbf{y}_{teach}(n))$ for each output neuron and timestep into a $(n_{max} - n_{min}) \times M$ matrix $\mathbf{T}$.
5. Compute pseudo-inverse $\mathbf{S}^{\dagger}$ and put
$$\mathbf{W}^{out} = (\mathbf{S}^{\dagger}\mathbf{T})^{\intercal} \tag{2.3}$$
where $(.)^{\dagger}$ denotes the pseudo-inverse and $(.)^{\intercal}$ the transposition of a matrix.
6. The ESN is now trained and can be used.

Alternatively the following algorithms can be considered in step 5 instead of the pseudo-inverse:

- Wiener-Hopf solution (see Jaeger [2007a]):
$$\mathbf{W}^{out} = (\mathbf{R}^{-1}\mathbf{P})^{\intercal} \tag{2.4}$$
where $\mathbf{R} = \mathbf{S}^{\intercal}\mathbf{S}$ is the correlation matrix of the network states and $\mathbf{P} = \mathbf{S}^{\intercal}\mathbf{T}$ the cross-correlation matrix of states and desired outputs.
The Wiener-Hopf solution is in principle equivalent to the pseudo-inverse solution, but is faster to compute, especially if n_{max} is large. However, when $\mathbf{R}$ is ill-conditioned the solution can become numerically unstable while this is no problem with the pseudo-inverse method.

- Tikhonov regularization, ridge regression (see Jaeger et al. [2007b]):
$$\mathbf{W}^{out} = ((\mathbf{S}^{\intercal}\mathbf{S} + \alpha^2\mathbf{I})^{-1}\mathbf{S}^{\intercal}\mathbf{T})^{\intercal} \tag{2.5}$$
where $\mathbf{I}$ is the identity matrix and α an additional regularization parameter, called Tikhonov factor.
This algorithm tries to get the output weights as small as possible. If the regularization factor $\alpha = 0$ it will be equivalent to the standard Wiener-Hopf solution and the higher α, the stronger is the smoothing/regularization effect.

Additionally the noise term v in the state update equation 2.1 can be used as regularization during training. With a high noise term the output weights $\mathbf{W}^{out}$ will become smaller, resulting in a more stable network. Especially when an ESN is trained to be a generator problems can occur and regularization is necessary.

2.3.2 Online Training Algorithm

Standard algorithms for mean square error minimization known from adaptive linear signal processing, like the least mean squares (LMS) and recursive least squares (RLS) algorithm (Farhang-Boroujeny [1998]), can be applied to online ESN estimation. However, there are problems with the rate of convergence of an LMS algorithm. As noted in Jaeger [2007b] the convergence speed of an LMS algorithm depends on the ratio of largest and smallest absolute eigenvalue of the input signal correlation matrix, which is called spectral spread. In an ESN context this means that LMS is only useful if the correlation matrix of the reservoir signals $\mathbf{x}(n)$ have a low spectral spread, which is unfortunately not the case. With randomly created recurrent reservoirs the spectral spread is typically 10^{12} and higher, which precludes the use of an LMS algorithm for an online calculation of readout weights.

Therefore the RLS algorithm, which is widely used in adaptive signal processing when fast convergence is of prime importance, was used in Jaeger [2003]. The RLS convergence is independent of the spectral spread of input signals, but it has a quadratic update complexity (compared to reservoir size) and can become numerically instable. Various implementations of this algorithm (plain version, fast version, ...) exists and in Jaeger [2003] or Jaeger and Hass [2004] a plain version was used as summarized in table 12.1 of Farhang-Boroujeny [1998] or in the appendix of Jaeger and Hass [2004]. However, it would be also possible to utilize the fast RLS as introduced in chapter 13 of Farhang-Boroujeny [1998].

Given an open-ended, non-stationary training sequence, the training algorithm should determine an augmented vector $\mathbf{W}^{out}(n)$ at each timestep. Therefore the RLS algorithm minimizes the square error

$$\sum_{k=1}^{n} \lambda^{n-k} (g^{-1}(\mathbf{y}_{teach}(k)) - g^{-1}(\mathbf{y}_{[n]}(k)))^2$$

where $\lambda < 1$ is a forgetting factor, g^{-1} is the inverse of the output nonlinearity and $\mathbf{y}_{[n]}(k)$ is the model output that would be obtained at time k if the network with the current $\mathbf{W}^{out}(n)$ would have been used at all timesteps $k = 1, ..., n$.

Two parameters characterize the tracking performance of an RLS algorithm. The misadjustment M gives the ratio between the excess MSE incurred by the adaptation process and the optimal MSE that would be obtained with offline training. The time constant τ determines the exponent of the MSE convergence $e^{-n/\tau}$. For instance $\tau = 200$ would result in an excess MSE reduction by $1/e$ each 200 steps. These parameters are related to the forgetting factor λ and the length of the tap-vector N (= length of $\mathbf{W}^{out}$) with the equations

$$M = N \frac{1-\lambda}{1+\lambda}$$

$$\tau \approx \frac{1}{1-\lambda}$$

Important for the RLS algorithm stability is the noise term υ in the state update equation 2.1. When using no noise in training the output weights grow very high and the RLS algorithm enters a region where it becomes numerically unstable.

2.4 Additional Nonlinear Transformations

The modeling power of an ESN grows with network size. However, for hard nonlinear tasks a cheaper way to increase the power is to use additional nonlinear transformations of network

states $\mathbf{x}(n)$ as shown in Jaeger [2003]. This can be done by extending the internal state vector $\mathbf{x}(n)$ and inputs $\mathbf{u}(n)$ with additional squared states:

$$\mathbf{x}_{square}(n) = (x_1(n), ..., x_N(n), x_1^2(n), ..., x_N^2(n))$$

$$\mathbf{u}_{square}(n) = (u_1(n), ..., u_L(n), u_1^2(n), ..., u_L^2(n))$$

where N is the number of neurons in the reservoir and L the number of inputs. With M outputs the trainable output weight vector $\mathbf{W}^{out}_{square}$ must be of size $(2N + 2L) \times M$ and the network state update equations are similar to equation 2.1 and 2.2

$$\mathbf{x}(n+1) = f(\mathbf{W}\mathbf{x}(n) + \mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}^{fb}\mathbf{y}(n) + \upsilon(n+1)) \tag{2.6}$$

$$\mathbf{y}(n+1) = g(\mathbf{W}^{out}_{square}[\mathbf{x}_{square}(n+1); \mathbf{u}_{square}(n+1)]). \tag{2.7}$$

The learning procedure remains linear and can be done as in section 2.3:

1. Init reservoir matrix $\mathbf{W}$ with spectral radius $\alpha < 1$ and run the ESN with a teaching input signal $\mathbf{u}_{teach}(n)$.
2. Dismiss data from initial transients where $n < n_{min}$ and collect the remaining input and network squared states $(\mathbf{u}_{teach,square}(n), \mathbf{x}_{square}(n))$ for each timestep row-wise into a $(n_{max} - n_{min}) \times (2N + 2L)$ matrix $\mathbf{S}$.
3. Collect target signals $g^{-1}(\mathbf{y}_{teach}(n))$ for each output neuron and timestep into a $(n_{max} - n_{min}) \times M$ matrix $\mathbf{T}$.
4. Compute pseudo-inverse $\mathbf{S}^\dagger$ and put

   $$\mathbf{W}^{out}_{square} = (\mathbf{S}^\dagger\mathbf{T})^\intercal$$

   or alternatively use any other online or offline training algorithm from section 2.3.
5. The ESN is now trained and can be used.

2.5 Short Term Memory Effects

When processing time series the impact of data from the past is important, which is called short term memory in an ESN context. A limit of the short term memory can be found by analyzing how many of the previous input and output arguments $(\mathbf{u}(n-k), \mathbf{y}(n-k-1))$ are actually relevant for the computation of the current state vector $\mathbf{x}(n)$. Especially for many tasks in audio signal processing, where a high sampling rate is used, it is very important that the short term memory is as long as possible.

In Jaeger [2002a] and Jaeger [2002b] the short term memory capability of an ESN is measured with a simple delay learning task. The input $u(n)$ is a white noise signal and the output is the input signal delayed by k samples. Now $k = 1, 2, ...$ is increased as long as it is possible to train the target signal $y_{teach}(n) = u(n-k)$ with an echo state network without feedback connections $\mathbf{W}^{fb}$. To measure the correspondence of the correctly delayed target signal $y_{teach}(n) = u(n-k)$ and the network output $y_k(n)$ at delay k the squared correlation coefficient $r^2(u(n-k), y_k(n))$ is calculated. A value of $r^2(u(n-k), y_k(n)) = 1$ would mean perfect correlation and $r^2(u(n-k), y_k(n)) = 0$ a complete loss of correlation between $u(n-k)$ and $y_k(n)$.

By summing up the squared correlation coefficients r^2 over all delays $k = 1, 2, ..., \infty$ one gets the memory capacity MC of a network as defined in Jaeger [2002a] with

$$MC = \sum_{k=1}^{\infty} r^2(u(n-k), y_k(n)). \tag{2.8}$$

It was proven in that paper that for an ESN whose dynamic reservoir has N nodes, the maximal possible memory capacity MC is bounded by N

$$MC \leq N$$

and furthermore that networks with linear activation functions in the dynamic reservoir have a memory capacity

$$MC = N.$$

When plotting the squared correlation coefficient r^2 against the delay k the forgetting curve is obtained. Figure 2.4 shows forgetting curves for different ESN setups with 400 neurons in the reservoir and some interesting phenomenas can be observed:

- Linear networks should have a memory capacity $MC = N$. However, curve A in figure 2.4 shows a MC of about 120 only and not of 400. According to Jaeger [2002b] this is due to rounding errors in the network update.

- Curve B was created in the same way as A, but with tanh activation functions in the reservoir and therefore the memory capacity is lower. In general one can say that the more nonlinear a network is, the lower its memory capacity. Additionally the input weights are important in tanh networks, if they are very big the signals will be in the very nonlinear region of the tanh activation functions and therefore lowering the memory capacity.

- Curve C and D are like A and B, but with an additional noise term between $[-0.01, 0.01]$ in the state update equation. Surprisingly the effect of a small noise term is quite strong and the memory capacity is much lower.

In Jaeger [2002a] a more detailed analysis of the influence of noise and rounding errors on the memory capacity can be found, which comes to the conclusion that by using an "almost" unitary weight matrix $\mathbf{W}$ in the reservoir, these effects are not that dramatic. Such a matrix $\mathbf{W}$ can be created by replacing the singular value diagonal matrix in the SVD (singular value decomposition) of a randomly created weight matrix by the identity matrix. Afterwards this matrix must be scaled by a factor ϑ slightly smaller than 1, otherwise the echo state property won't be fulfilled.

Figure 2.5 shows plots of a 400 unit ESN with linear activation functions and an almost unitary weight matrix:

- The network of curve A uses a scaling factor $\vartheta = 0.98$ and one can see that the maximum possible memory capacity of 400 is almost obtained.

- By adding a noise term between $[-0.01, 0.01]$ in the state update equation in training and testing, as shown in curve B of figure 2.5, the memory capacity decreases again but by far not as dramatically as in figure 2.4.

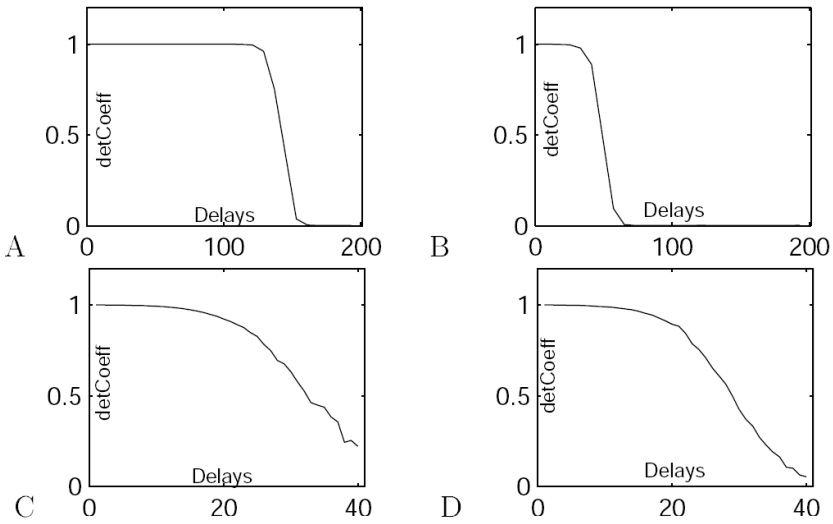

Figure 2.4: Forgetting curves for a 400 unit ESN: **A:** randomly created linear reservoir; **B:** randomly created reservoir with tanh activation functions; **C:** like A, but with noisy state update; **D:** like B, but with noisy state update; [Image extracted from Jaeger [2002a]]

- Finally curve C shows a network with a very high scaling factor $\vartheta = 0.999$. This leads to long-term reverberations of the input and a forgetting curve with a long tail far beyond the network size $N = 400$. On the other hand these strong echoes from the past influence also the recall of immediately past inputs and therefore the squared correlation coefficient is only about 0.5 at the beginning.

When needing ESNs with long short term memory effects one can resort to a combination of the following approaches:

- Use large dynamic reservoirs.
- Use small input weights or a reservoir with linear activation functions, which might conflict with nonlinear modeling tasks.
- Use an almost unitary weight matrix in the dynamic reservoir.
- Use a spectral radius α close to 1, which will not work if one wants to have fast oscillating dynamics.

In chapter 4 a delay&sum readout will be introduced, which improves the quite limited short term memory. This readout consists of output neurons, where additionally to a weight also a delay is learned.

2.6 Leaky Integrator Neurons

In the previous sections of this chapter only standard sigmoid neurons, with a weight on all inputs and finally a tanh activation function, were analyzed. The general idea of reservoir computing or echo state networks can also be applied to networks with different neuron types, for instance the

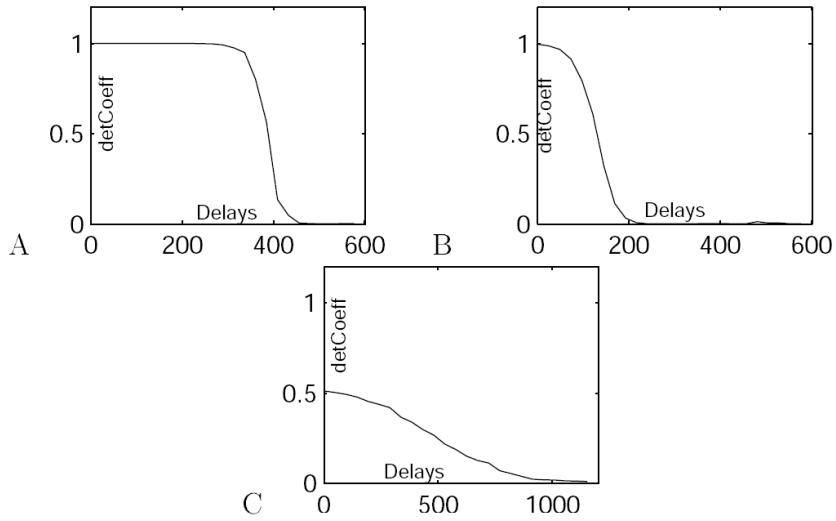

Figure 2.5: Forgetting curves for 400 unit ESNs with an almost unitary weight matrix: **A:** randomly created linear reservoir with scaling factor $\vartheta = 0.98$ without noise; **B:** like A, but with noisy state update; **C:** like A, but with a scaling factor $\vartheta = 0.999$; [Image extracted from Jaeger [2002a]]

liquid state machine from Maass et al. [2002] uses spiking neurons. Leaky integrator neurons, as shown in Jaeger [2001] and refined in Jaeger et al. [2007b], incorporate information of the network states from the previous time steps when calculating the current state $\mathbf{x}(n)$, therefore the dynamics in the reservoir are somehow slowed down and these neurons have a smoothing effect. For instance ESNs with standard sigmoid neurons cannot be trained to be a generator of a very slow sinewave, whereas it is possible with leaky integrator units.

The continuous-time dynamics of a leaky integrator ESN is given in Jaeger et al. [2007b] with

$$\dot{\mathbf{x}} = \frac{1}{c}\left(-a\mathbf{x} + f(\mathbf{W}\mathbf{x} + \mathbf{W}^{in}\mathbf{u} + \mathbf{W}^{fb}\mathbf{y})\right)$$

$$\mathbf{y} = g(\mathbf{W}^{out}[\mathbf{x};\mathbf{u}])$$

where $c > 0$ is a time constant, $a > 0$ the reservoir neurons leaking rate, f is a tanh activation function of the reservoir and g a tanh or linear output activation function. Using an Euler discretization with stepsize δ one gets the discrete time network update equations given a sampled input $\mathbf{u}(n\delta)$

$$\mathbf{x}(n+1) = \left(1 - \frac{a\delta}{c}\right)\mathbf{x}(n) + \frac{\delta}{c} f\left(\mathbf{W}\mathbf{x}(n) + \mathbf{W}^{in}\mathbf{u}((n+1)\delta) + \mathbf{W}^{fb}\mathbf{y}(n)\right)$$

$$\mathbf{y}(n+1) = g\left(\mathbf{W}^{out}[\mathbf{x}(n+1);\mathbf{u}((n+1)\delta)]\right)$$

By setting the stepsize $\delta = 1$, introducing a new variable $\gamma = \frac{\delta}{c}$ and assuming that $\mathbf{W}$ has unit spectral radius one obtains the state update equations

$$\mathbf{x}(n+1) = (1 - a\gamma)\mathbf{x}(n) + \gamma f(\rho\mathbf{W}\mathbf{x}(n) + \mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}^{fb}\mathbf{y}(n) + \upsilon(n+1)) \tag{2.9}$$

$$\mathbf{y}(n+1) = g(\mathbf{W}^{out}[\mathbf{x}(n+1);\mathbf{u}(n+1)]) \tag{2.10}$$

where ρ is now the effective spectral radius of the reservoir weight matrix and $\upsilon(n+1)$ an additional noise term as in equation 2.1. This model depends with the factor $(1-a\gamma)$ also on the previous state $\mathbf{x}(n)$ and fulfills the echo state property if $\mid \lambda \mid_{max} (\mathbf{W}) = 1$, $a > 0$, $\gamma > 0$, $0 \leq \rho < 1$ and $a\gamma \leq 1$.

Finally it was shown in Jaeger et al. [2007b] that the influence of parameter γ can be distributed over the other parameters without a loss of essence, resulting in the state update equations

$$\mathbf{x}(n+1) = (1-a)\mathbf{x}(n) + f(\rho\mathbf{W}\mathbf{x}(n) + \mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}^{fb}\mathbf{y}(n) + \upsilon(n+1)) \tag{2.11}$$

$$\mathbf{y}(n+1) = g(\mathbf{W}^{out}[\mathbf{x}(n+1); \mathbf{u}(n+1)]) \tag{2.12}$$

where the parameter a is called leaking rate. This simplifies the constraints for the echo state property to $a > 0$, $\rho < 1$ and $\rho < a$.

Chapter 3

Filter Neurons

In the following chapter one problem of echo state networks is addressed: because reservoir neurons are connected to each other, their states are coupled and it is not possible to learn multiple attractors or oscillators with one network. For instance a standard echo state network cannot be a generator for a sum of multiple sines at different frequencies. It is helpful for some applications that reservoir neurons have additional filters, where "specialized" units are tuned to different frequencies and more diverse echoes can evolve because not all neurons are coupled.

The idea of ESNs with low-pass, high-pass and band-pass neurons was introduced in Wustlich and Siewert [2007] as an extension of leaky-integrator ESNs and will be the topic of section 3.1. In section 3.2 this idea is extended to an arbitrary order infinite impulse response (IIR) filter neuron[1], which allows to reuse commonly known structures from filter theory, and finally some possible filter structures for reservoir neurons are explained in section 3.3.
Simulations and an evaluation of echo state network with filter neurons will be presented in chapter 5 and 6.

3.1 From Leaky Integrator to Band-Pass Neurons

The leaky-integrator ESN model was introduced in section 2.6. Equations 2.9 and 2.10 are the general discrete time network update equations depending on three parameters a, γ and ρ. For such a system the echo state property is fulfilled if $\mid \lambda \mid_{max} (\mathbf{W}) = 1$, $a > 0$, $\gamma > 0$, $0 < \rho < 1$ and $a\gamma \leq 1$. As already demonstrated in Jaeger et al. [2007b], the influence of one parameter, for instance γ, can be distributed over the others, which resulted in equation 2.11.

Wustlich and Siewert [2007] showed that the effect of any parameter a, γ or ρ can be distributed over the others without loss of essence. For instance by setting $a = 1$ this results in the state update equation

$$\mathbf{x}(n+1) = (1-\gamma)\mathbf{x}(n) + \gamma f\left\{\rho\mathbf{W}\mathbf{x}(n) + \mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}^{fb}\mathbf{y}(n) + \upsilon(n+1)\right\}. \tag{3.1}$$

Now one can compare this system equation to a discrete time low-pass filter with one pole in the transfer function. As analyzed in [Smith, 1997, Chapter 13] a single pole low-pass filter can be written as

$$\tilde{y}(n) = (1-\theta)\tilde{y}(n-1) + \theta\tilde{u}(n)$$

[1] Note that it was also demonstrated in Maass et al. [2007], that ESNs with trained feedbacks become more powerful. Although here the feedbacks of IIR neurons are not trained.

where $\tilde{u}(n)$ is the input, $\tilde{y}(n)$ the output, θ corresponds to the decay time of the system and must be between $0 < \theta \leq 1$. The relationship between θ and the -3dB cutoff frequency f_C of this digital filter is

$$\theta = 1 - e^{-2\pi f_C}.$$

When using a non-interacting single neuron reservoir with low input impacts, which means $\rho = 0$, $\mathbf{W}^{fb} = 0$, $\mathbf{W}^{in} = 1$, $u(n) \ll 1$ and therefore $tanh(u(n)) \approx u(n)$, the state update equation 3.1 becomes

$$x(n+1) = (1-\gamma)x(n) + \gamma u(n+1)$$

where $x(n+1)$ is the output of an one pole low-pass filter compared to the input $u(n+1)$ and γ determines the -3dB cutoff frequency f_C

$$\gamma = 1 - e^{-2\pi f_C}. \tag{3.2}$$

For such a system, the echo state property is fulfilled if $0 \leq \rho < 1$ and $0 < \gamma \leq 1$. Simulations in Wustlich and Siewert [2007] suggested that the low-pass filter characteristic is also approximately maintained when using multiple interacting neurons where $\rho > 0$ and with arbitrary inputs.

Having a low-pass neuron it is obvious to extend the idea to a high-pass neuron. One possible way to obtain a high-pass version is to subtract the lowpass filter output from the original output. For $a = 1$ this results in

$$\mathbf{x}_{LP}(n+1) = (1-\gamma)\mathbf{x}_{LP}(n) + \gamma f\left\{\rho\mathbf{W}\mathbf{x}(n) + \mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}^{fb}\mathbf{y}(n) + \upsilon(n+1)\right\} \tag{3.3}$$

$$\mathbf{x}(n+1) = f\left\{\rho\mathbf{W}\mathbf{x}(n) + \mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}^{fb}\mathbf{y}(n) + \upsilon(n+1)\right\} - \mathbf{x}_{LP}(n+1) \tag{3.4}$$

where the first equation is a low-pass filter as in 3.1, the subtraction from the original signal is done in the second equation and the cutoff frequency of the high-pass filter can be calculated with γ as in equation 3.2.

Finally, combining the high-pass and low-pass filter in one neuron results in a more general band-pass neuron

$$\mathbf{x}_{LP}(n+1) = (1-\gamma_1)x_{LP}(n) + \gamma_1 f\left\{\rho\mathbf{W}\mathbf{x}(n) + \mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}^{fb}\mathbf{y}(n) + \upsilon(n+1)\right\} \tag{3.5}$$

$$\mathbf{x}_{HP}(n+1) = (1-\gamma_2)\mathbf{x}_{HP}(n) + \gamma_2\mathbf{x}_{LP}(n+1) \tag{3.6}$$

$$\mathbf{x}(n+1) = \mathbf{x}_{LP}(n+1) - \mathbf{x}_{HP}(n+1) \tag{3.7}$$

where γ_1 determines the cutoff frequency of the low-pass and γ_2 of the high-pass filter. It is also possible to get the original high-pass and low-pass neurons out of these equations, for instance by setting $\gamma_2 = 0$ the system includes low-pass units and by setting $\gamma_1 = 1$ it results in a high-pass version.

As the maximum band-pass response cannot exceed the maximum response of a low-pass or high-pass version, the echo state property is fulfilled if

$$0 \leq \rho < 1 \text{ and } 0 < \gamma_1 \leq 1 \text{ and } 0 \leq \gamma_2 < 1.$$

In fact, the band-pass response will be usually weaker than the low-pass response, therefore the output can be renormalized with a gain M to not influence the echo state property (Wustlich and Siewert [2007]):

$$M = 1 + \frac{\gamma_2}{\gamma_1}$$

Such networks, including neurons with different band-pass cutoffs in the reservoir, can be used to process data at different timescales or frequencies and are able to model multiple attractors. As noted in Wustlich and Siewert [2007]:

> "Clever combinations of band-pass neurons may then result into richer and more diverse echoes, which are the crucial ingredients of the reservoir computing approach."

In the same report they were able to train an ESN as a generator for a sum of three sines and a sum of a slow sine and a Mackey-Glass signal.

3.2 General IIR-Filter Neurons

The band-pass neuron by Wustlich from section 3.1 consists of first order recursive filters. This idea was extended to an arbitrary order infinite impulse response (IIR) filter neuron, thus commonly known structures from filter theory can be recycled and one acquires a more fine-grained control over the filter's frequency and phase response.

As defined in Smith [2007a] a general IIR filter can be described with

$$\begin{aligned} a_0 y(n) = b_0 x(n) + b_1 x(n-1) + ... + b_M x(n-M) \\ -a_1 y(n-1) - ... - a_N y(n-N) \end{aligned} \tag{3.8}$$

where vector $\mathbf{b}$ contains weights of the feedforward and vector $\mathbf{a}$ of the feedback path. Usually this general structure is implemented in scientific computation software (like *signal.lfiter* in scipy or *filter* in matlab), because it's possible to compute any finite impulse response (feedback path is zero) or infinite impulse response structure with it. However, some attention should be payed on possible numerical instabilities. As explained in Smith [2007a] the IIR filters should be implemented in *transposed direct form II* to avoid possible instabilities of internal filter states and higher order filters should be realized as a cascade of second order systems (biquad filters).

Figure 3.1 shows an IIR filter neuron, where the filter is calculated after the nonlinearity. One other distinction to Wustlich's band-pass model from equation 3.7 is that this model is a filter compared to the reservoir neuron signals, whereas Wustlich's model is a filter compared to the input signal $\mathbf{u}(n)$ for non interacting neurons.

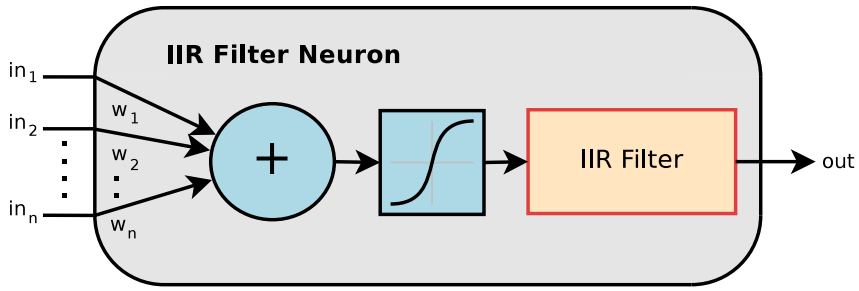

Figure 3.1: Structure of an analog neuron with an additional IIR filter.

For IIR-filter neurons in the reservoir the state update equation becomes

$$\mathbf{x}(n+1) = \mathbf{G}\left\{f(\rho\mathbf{W}\mathbf{x}(n) + \mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}^{fb}\mathbf{y}(n) + \upsilon(n+1))\right\} \tag{3.9}$$

where operator $\mathbf{G}$ is an array of IIR filters as defined in equation 3.8 with independent parameters for each neuron. To be sure that the echo state property is fulfilled, the maximum gain of each filter in $\mathbf{G}$ should be one, the spectral radius ρ within $0 \leq \rho < 1$ and the maximum absolute eigenvalue $\mid \lambda \mid_{max} (\mathbf{W}) = 1$. Filter structures with that requirement are developed in the next section.

3.3 Possible Filter Structures

Of course there are many possibilities how to choose the right filters for neurons in the reservoir. Two possible structures will be presented:

- band-pass filters with variable bandwidth, spaced logarithmically, linear or in groups over the frequency band of interest
- a variation of band-pass filters called peak filter, where one can additionally specify a stop-band attenuation

Figure 3.2 shows band-pass filters spaced logarithmically over a frequency range of interest, implemented as second order IIR systems (biquad filter). The additional parameter ϕ specifies the bandwidth in octaves between the -3dB cutoff frequencies.

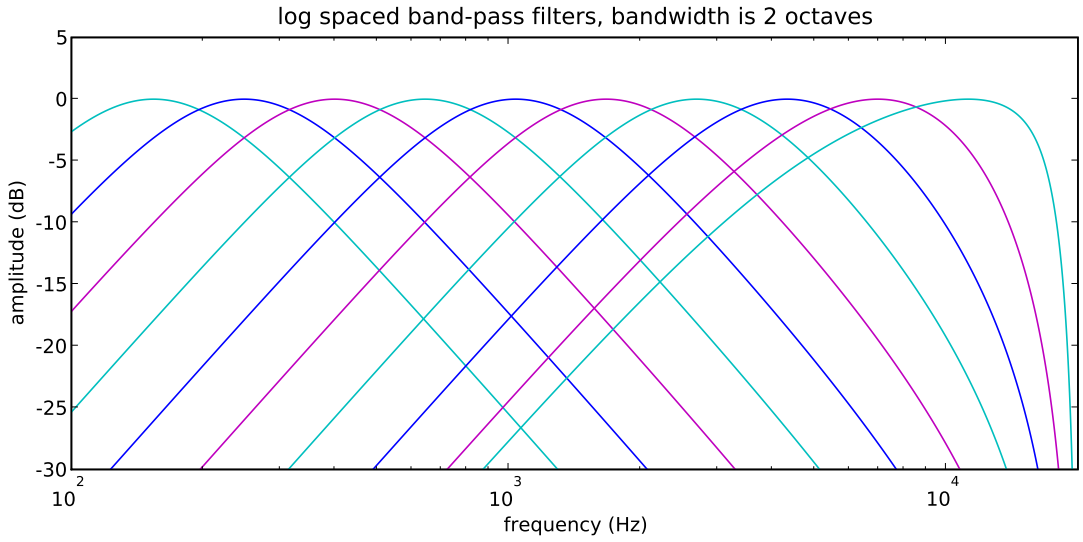

Figure 3.2: Band-pass filters spaced logarithmically from 170 Hz to 19000 Hz at a sampling rate of 44100 Hz (note that the frequency axis is also logarithmic), printed every 10th neuron of a reservoir with 100 neurons. Each filter has a bandwidth of 2 octaves.

The filter function is derived from analog prototypes and has been digitized using the bilinear transform (Bristow-Johnson [2008]) to finally get the following parameters for coefficients **a** and

b of equation 3.8:

$$\omega_0 = \frac{2\pi f_0}{f_s} \tag{3.10}$$

$$\alpha = sin(\omega_0) sinh\left(\frac{ln(2)}{2}\phi\frac{\omega_0}{sin(\omega_0)}\right) \tag{3.11}$$

$$\begin{aligned} b_0 = \alpha, b_1 = 0, b_2 = -\alpha, \\ a_0 = 1+\alpha, a_1 = -2cos(\omega_0), a_2 = 1-\alpha \end{aligned} \tag{3.12}$$

where f_0 is the center frequency of the band-pass filter, f_s the samplingrate and ϕ the bandwidth in octaves between -3dB cutoff frequencies.

Figure 3.3 shows a plot of peak filters, which are quite similar to band-pass filters but have an additional stop-band attenuation in decibel (dB). This can be used to guarantee that all filters (neurons) have at least a region of overlap at the stop-band level, which is for instance -25 dB in figure 3.3. For peak filters parameter ϕ is the bandwidth in octaves between midpoint (stop-band/2) gain frequencies.

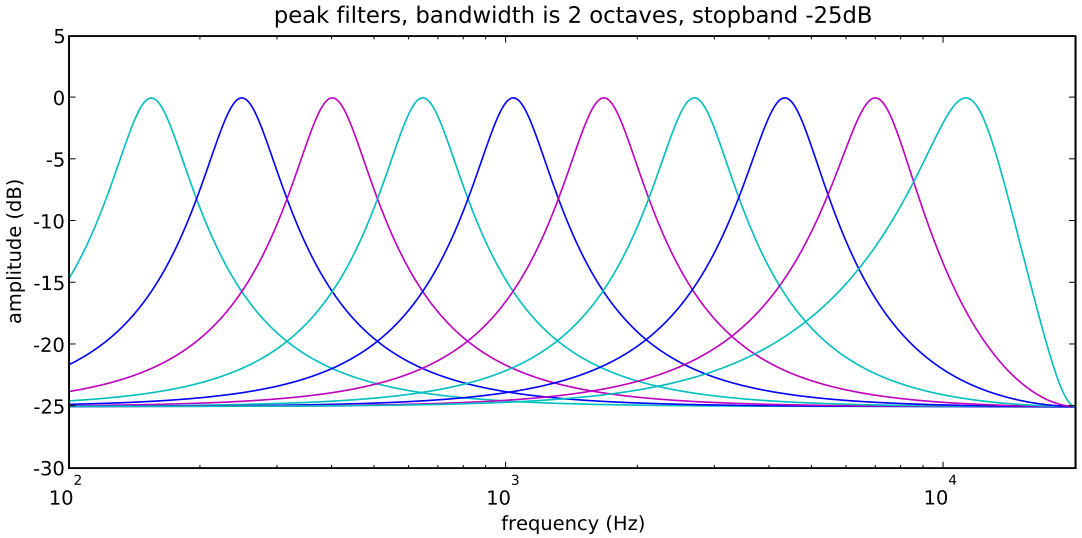

Figure 3.3: Peak filters spaced logarithmically from 170 Hz to 19000 Hz at a sampling rate of 44100 Hz, again every 10th neuron out of a reservoir with 100 neurons is shown. Each filter has a bandwidth of 2 octaves and a stop-band attenuation of -25 dB, therefore they overlap at least in the stop-band.

Peak filters were again implemented as second order IIR systems and the parameters as calculated in Bristow-Johnson [2008] were used. However, to not violate the echo state property these parameters had to be renormalized with a factor r to get a maximum gain of one and resulted in the equations

$$A = 10^{\frac{g}{40}} \tag{3.13}$$

$$r = 10^{\frac{-g}{20}} \tag{3.14}$$

$$\begin{aligned} b_0 = \frac{1+\alpha A}{r}, b_1 = \frac{-2cos(\omega_0)}{r}, b_2 = \frac{1-\alpha A}{r}, \\ a_0 = 1+\frac{\alpha}{A}, a_1 = -2cos(\omega_0), a_2 = 1-\frac{\alpha}{A} \end{aligned} \tag{3.15}$$

where g is the maximum stop-band attenuation in dB (for example -20 dB), ω_0 and α are from equations 3.10 and 3.11.

Chapter 4

Delay&Sum Readout

Section 2.5 already introduced the concept of short term memory in an ESN context. The main conclusion of this section was, that the short term memory is bounded by the reservoir size N and the maximal possible memory capacity MC from equation 2.8 is $MC \leq N$. One might think that this result is already quite good and by using large enough reservoirs the memory capacity should be sufficient. However, consider an ESN used as an audio signal processing system, where the signal is sampled at a high sampling rate, for instance at 44100 Hz like on audio CDs. To be able to incorporate information from only one second in the past the reservoir size would have to be at least 44100 or more, which is not trivial to simulate on current computers and maybe impossible for realtime applications.

This chapter introduces an alternative approach to handle long-term dependencies. The basic idea is that in the synaptic connections of readout neurons trainable delays are added. Compared to standard echo state networks, where only the weights of the readout are adjusted in training, delays and weights are modified by a learning algorithm in the delay&sum readout. This approach makes it possible to shift reservoir signals in time and is a computationally cheap method to vastly improve the short term memory, furthermore it is biologically not implausible because also biological axons insert delays in connections between neurons. Neuroscience experiments in Swadlow [1985] give also evidence to the variability of transmission delay values from 0.1 to 44 ms, see also Bringuier et al. [1999] for a more recent investigation.

A general overview of the delay&sum readout and state update equations is given in section 4.1, section 4.2 is an introduction to the problem of time delay estimation, then a simple learning algorithm will be presented in section 4.3 and a more advanced, EM-based method in section 4.4. Finally some general comments and ideas for further research are discussed in section 4.5. Simulations and an evaluation of the presented algorithms can be found in chapter 5 and 6.

4.1 Delay&Sum Readout Overview

The delay&sum readout has to learn in addition to a weight also a delay from each neuron signal of the reservoir to each output signal. Figure 4.1 shows an overview of the readout, which was inspired by the delay&sum beamformer as used in acoustical MIMO signal processing, see for instance Y.Huang et al. [2006].

Consider an echo state network with a delay&sum (D&S) readout, N internal units, L inputs and M outputs, where the state update $\mathbf{x}(n+1)$ is computed for standard ESNs as in

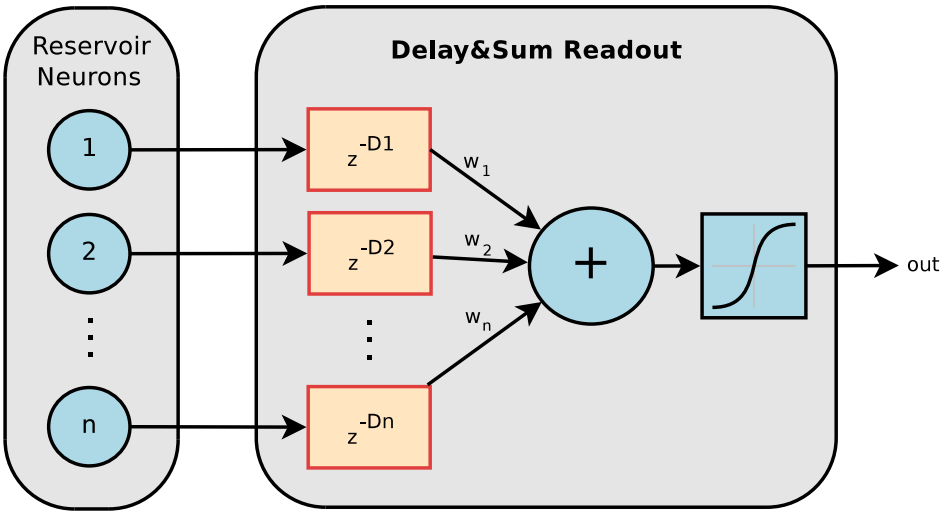

Figure 4.1: Illustration of the delay&sum readout for one output neuron, z^{-d} denotes a delay of a signal by d samples. Each synaptic connection from a reservoir neuron to the output neuron consists of a trainable weight and a trainable delay.

equation 2.1 or for reservoirs with IIR filter neurons as in equation 3.9. The readout neurons have connections to the reservoir units $\mathbf{x}(n)$ and inputs $\mathbf{u}(n)$, which can be combined into a vector $\mathbf{s}(n)$

$$\mathbf{s}(n) = (x_1(n), \ldots, x_N(n), u_1(n), \ldots, u_L(n)). \tag{4.1}$$

Afterwards the outputs $m = 1, \ldots, M$ can be calculated with

$$y_m(n) = g\left(\sum_{i=1}^{N+L} W_{i,m}^{out} s_i(n - D_{i,m})\right), \quad m = 1, \ldots, M \tag{4.2}$$

where $\mathbf{D}$ is a $(N + L) \times M$ matrix (same size as $\mathbf{W}_{out}$) with integer delays from each neuron and input signal to each output and g is the activation function of the output neuron. Note that for zero delays $\mathbf{D} = \mathbf{0}$ this is the same as equation 2.2, only written with a sum instead of the scalar product.

4.2 Time Delay Estimation

Estimating the time delay between two signals is a common task in signal processing and is usually known as the time difference of arrival (TDOA) problem (see Knapp and Carter [1976] or Azaria and Hertz [1986]). This section will discuss the use of a basic version, which just calculates the cross correlation between two signals and takes the highest peak as the delay, and the generalized cross correlation (GCC) method, which uses an additional prefilter to be more robust against artifacts.

Assume two signals

$$g_1(n) = \chi(n) + v_1(n)$$
$$g_2(n) = \alpha\chi(n - d_{g_1 g_2}) + v_2(n)$$

where $\chi(n)$ and noise terms $v_1(n)$, $v_2(n)$ are real baseband signals, α a scalar and $d_{g_1 g_2}$ the unknown delay. Furthermore it is assumed that $\chi(n)$ is uncorrelated with the noise signals

$v_1(n)$ and $v_2(n)$ and that the environment is only slowly time varying, which means that the characteristics of the signal and noise should remain stationary only for a finite observation time N_{max}. The delay $d_{g_1g_2}$ and scalar α may also change slowly.

The simple method of determining $d_{g_1g_2}$ is to compute as in Knapp and Carter [1976] the cross correlation function

$$r_{g_1g_2}(\tau) = E\left[g_1(n)g_2(n+\tau)\right]$$

where $E[.]$ denotes the expectation operator and the argument τ that maximizes $r_{g_1g_2}(\tau)$ provides an estimate of the time delay $d_{g_1g_2}$. Because of the finite observation time N_{max} the cross correlation function $r_{g_1g_2}(\tau)$ can only be estimated. For two real signals $g_1(n)$,$g_2(n)$ the estimate may be found as in Smith [2007b] by

$$\hat{r}_{g_1g_2}(\tau) = \frac{1}{N_{max}} \sum_{n=0}^{N_{max}-1} g_1(n)g_2(n+\tau), \quad \tau = 0, 1, \ldots, N_{max}-1 \tag{4.3}$$

and finally the time delay can be calculated with

$$d_{g_1g_2} = \arg\max_{\tau} \hat{r}_{g_1g_2}(\tau). \tag{4.4}$$

To improve the accuracy of the delay estimation it is often helpful to prefilter signals $g_1(n)$,$g_2(n)$ before calculating the cross correlation. Knapp and Carter [1976] show various ways how to properly select a prefilter $\Phi(\omega_k)$ and call this method the generalized cross correlation (GCC). By using a filter $\Phi(\omega_k) = 1$ the GCC results in a simple cross correlation method as described above.

Out of performance reasons the cross correlation and filtering operation should be computed in frequency domain. First the discrete fourier transforms (DFT, see Smith [2007b]) of signals g_1 and g_2 are calculated

$$G_i(\omega_k) = DFT\{g_i(n)\} = \sum_{n=0}^{N_{max}-1} g_i(n)e^{-\frac{2\pi j}{N_{max}}kn} \tag{4.5}$$

where N_{max} is the observation time and $k = 0, 1, \ldots, N_{max}-1$. Having $G_1(\omega_k)$ and $G_2(\omega_k)$ it is possible to estimate the cross power spectral density function

$$\hat{R}_{g_1g_2} = G_1(\omega_k)G_2^*(\omega_k) = DFT\{\hat{r}_{g_1g_2}\}$$

where $(.)^*$ denotes the complex conjugate. The prefiltering operation is now a simple multiplication in frequency domain

$$\hat{R}^{GCC}_{g_1g_2} = \Phi(\omega_k)G_1(\omega_k)G_2^*(\omega_k) \tag{4.6}$$

and finally the time delay can be estimated by retransforming $\hat{R}^{GCC}_{g_1g_2}$ into time domain using the inverse discrete fourier transform (IDFT)

$$\hat{r}^{GCC}_{g_1g_2}(\tau) = IDFT\{\Phi(\omega_k)G_1(\omega_k)G_2^*(\omega_k)\} = \frac{1}{N_{max}} \sum_{k=0}^{N_{max}-1} \Phi(\omega_k)G_1(\omega_k)G_2^*(\omega_k)e^{\frac{2\pi j}{N}kn} \tag{4.7}$$

with $n = 0, \ldots, N_{max}-1$ and $d_{g_1g_2}$ can be calculated as in equation 4.4 by

$$d_{g_1g_2} = \arg\max_{\tau} \hat{r}^{GCC}_{g_1g_2}(\tau).$$

There are a number of different GCC algorithms depending on the selection of $\Phi(\omega_k)$. A more detailed overview is given in Knapp and Carter [1976], Azaria and Hertz [1986] and Y.Huang et al. [2006]. For this work the phase transform (PHAT) method will be used, which performs a pre-whitening of the signals to be more robust against noise and disturbances. The PHAT prefilter $\Phi(\omega_k)$ is defined as

$$\Phi(\omega_k) = \frac{1}{\mid R_{g_1 g_2} \mid} \tag{4.8}$$

and can be estimated with

$$\Phi(\omega_k) = \frac{1}{\mid \hat{R}_{g_1 g_2} \mid} = \frac{1}{\mid G_1(\omega_k) G_2^*(\omega_k) \mid}. \tag{4.9}$$

Figure 4.2 shows a comparison of the simple cross correlation and the GCC with phase transform method calculated between an audio signal and the same signal delayed by 20 samples. One can see that the GCC method results in a much sharper and higher peak at the exact delay position, whereas in the cross-correlation method also other similar regions have quite high values.

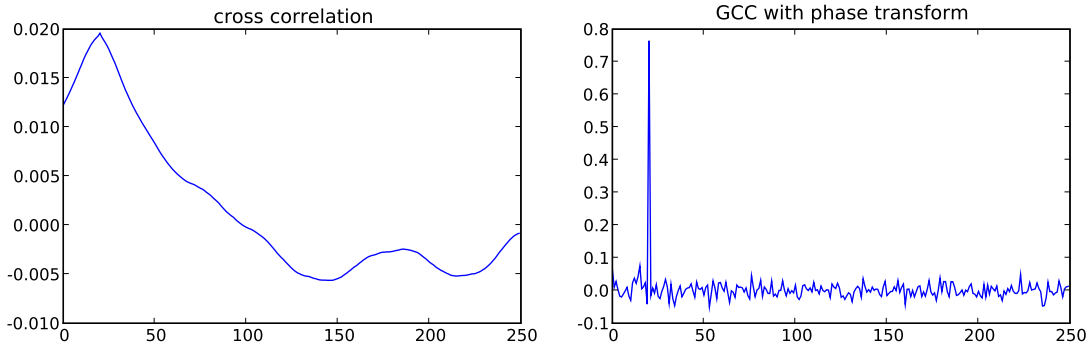

Figure 4.2: Cross correlation (left) and generalized cross correlation with phase transform (right) of a 20 sample delayed audio signal with itself. Note the different scalings of the y-axes.

4.3 Learning Algorithm A: Simple Method

A simple learning algorithm could calculate the cross correlation or GCC between all neuron/input signals and all output signals, taking always the highest peak as the trained delay. Afterwards the neuron and input signals are delayed by that value and weights can be computed offline with the correctly delayed signals as in section 2.3.1.

Consider for instance the linear system

$$y(n) = x(n) + x(n-1) + x(n-2) + \xi \left(x(n-200) + x(n-201) + x(n-202) \right).$$

If the scaler ξ is small this simple method will only find correct delays for $x(n)$, $x(n-1)$ and $x(n-2)$, but not for $x(n-200)$, $x(n-201)$ and $x(n-202)$, therefore this algorithm won't be optimal for all tasks[1]. However, for some reasons this method still works quite good, often even

[1] For example if $\xi < 0.1$ using an ESN with 100 neurons in the reservoir, then the learning algorithm won't find the delays for $x(n-200)$, $x(n-201)$ and $x(n-202)$.

better as the more advanced (and computationally more expensive) EM-based algorithm from section 4.4.

Learning algorithm A is summarized in the following steps:

1. Consider an ESN with N internal units, L inputs and M outputs. Calculate delays between all reservoir/input signals $s_j(n)$ as defined in equation 4.1 with $j = 1, \ldots, N+L$ and all target output signals $g^{-1}(y_j(n))$ with $m = 1, \ldots, M$ and collect them into a $(N+L) \times M$ delay matrix $\mathbf{D}$. For each reservoir/input to output connection

   - transform signals into frequency domain as in equation 4.5 and invert output activation function g of the ESN

   $$\begin{aligned} S_j(\omega_k) &= DFT\left\{s_j(n)\right\} \\ Y_m(\omega_k) &= DFT\left\{g^{-1}(y_m(n))\right\} \end{aligned} \tag{4.10}$$

   - estimate generalized cross correlation as in equation 4.7

   $$\hat{r}^{GCC}_{s_j y_m}(\tau) = IDFT\left\{\Phi(\omega_k) S_j(\omega_k) Y^*_m(\omega_k)\right\} \tag{4.11}$$

   and select a prefilter $\Phi(\omega_k) = 1$ for the simple cross correlation or $\Phi(\omega_k) = \frac{1}{|S_j(\omega_k) Y^*_m(\omega_k)|}$ for the phase transform (PHAT) method

   - finally obtain the time delay and collect it in delay matrix $\mathbf{D}$ at position (j, m)

   $$D_{j,m} = \arg\max_{\tau} \mid \hat{r}^{GCC}_{s_j y_m}(\tau) \mid \tag{4.12}$$

   where $\mid . \mid$ denotes the absolute value[2].

   Afterwards repeat the whole procedure for each target output $g^{-1}(y_m(n))$ where $m = 1, \ldots, M$.

2. Delay all reservoir and input signals according to the estimated values in delay matrix $\mathbf{D}$.

3. Dismiss data from an initial washout period n_{min} where $n < n_{min}$ and collect the remaining correctly delayed input and network states $(s_1(n - D_{1,m}), \ldots, s_{N+L}(n - D_{N+L,m}))$ for each output target signal $y_m(n)$ row-wise into a $(n_{max} - n_{min}) \times (N+L)$ matrix $\mathbf{S}_m$, where n_{max} is the number of training examples, and afterwards collect current target signal $g^{-1}(y_m(n))$ into a $(n_{max} - n_{min}) \times 1$ matrix $\mathbf{T}_m$.

4. For each output $y_m(n)$ compute pseudo-inverse $\mathbf{S}^{\dagger}_m$ and put

   $$\mathbf{W}^{out}_m = (\mathbf{S}^{\dagger}_m \mathbf{T}_m)^{\intercal}$$

   where $\mathbf{W}^{out}_m$ denotes the weights from all reservoir and input signals to output signal $y_m(n)$ (m-th column of $\mathbf{W}^{out}$). Alternatively one might use any other offline training algorithm from section 2.3.

[2] Note that the difference to equation 4.4 is that here one can use the absolute value of $\hat{r}^{GCC}_{s_j y_m}(\tau)$ because a weight in the readout can also be negative and therefore will invert the phase.

4.4 Learning Algorithm B: EM-based Method

The second version of the learning algorithm is based on an expectation-maximization (EM) method as applied in a similar problem with antenna array systems in Pedersen et al. [1997]. They used a special form of the EM algorithm, the space-alternating generalized expectation-maximization (SAGE) method from Fessler and Hero [1994], which allows to separate this multi-dimensional optimization problem into one dimensional optimization processes that can be performed sequentially, which means in an ESN context calculating one delay and afterwards one weight for each synaptic connection.

In contrast to learning algorithm A from section 4.3, which computes the correlation always between one reservoir/input signal compared to the whole output signal, this algorithm subtracts the influence of all other reservoir/input signals and estimates the delay and weight compared to this residuum. Thus it leads to an iterative procedure where single delays and weights are calculated sequentially.
The algorithm starts for instance by estimating the delay between the first reservoir signal $x_1(n)$ and the target output $y(n)$ and then the single weight of this synaptic connection. Afterwards the delayed and weighted signal $x_1(n)$ will be subtracted from the original output $y(n)$ to produce the new target signal $y'(n)$ for the second reservoir signal $x_2(n)$. In theory this procedure will be repeated for all reservoir and input signals until the algorithm converges, which means that delays and weights won't change anymore. However, unfortunately the weights converge very slowly when used with reservoir signals of echo state networks, but on the other hand the delays usually converge quite fast and won't change after a few iterations. For ESNs without output feedback they are usually fixed after 2-5 iterations and with feedback it very much depends on the specific task. Therefore a combined approach will be applied in learning algorithm B, which first uses the EM algorithm for a few iterations to estimate delays of the readout and afterwards the weights will be recalculated offline with the correctly delayed reservoir and input signals.

Learning algorithm B is illustrated in figure 4.3 and can be summarized in the following steps:

1. Consider an ESN with N internal units, L inputs and M outputs. The EM-algorithm estimates delays and weights between all reservoir/input signals $s_j(n)$ as defined in equation 4.1 with $j = 1, \ldots, N+L$ and all target output signals $g^{-1}(y_m(n))$ with $m = 1, \ldots, M$ and collects them into a $(N+L) \times M$ delay matrix $\mathbf{D}$ and weight matrix $\mathbf{W}^{out}$. One iteration for updating $D_{j,m}$ and $W^{out}_{j,m}$ for one target output $g^{-1}(y_m(n))$ reads

   - *E-Step*:
     Subtract influence of all other reservoir and input signals[3] to get residuum $\kappa_{j,m}(n)$

     $$\kappa_{j,m}(n) = \beta_{EM} \left(g^{-1}(y_m(n)) - \sum_{i=1}^{N+L} W^{out}_{i,m} s_i(n - D_{i,m}) \right) + W^{out}_{j,m} s_j(n - D_{j,m}) \quad (4.13)$$

     where g^{-1} is the inversion of the ESN output activation function and β_{EM} is an additional factor which influences the convergence speed, which was set to $\beta_{EM} = 1$ after some evaluations[4].
   - *M-Step*:
     Calculate (generalized) cross correlation $\hat{r}^{GCC}_{s_j \kappa_{j,m}}(\tau)$ between signals $s_j(n)$ and $\kappa_{j,m}(n)$ as in equation 4.11 and estimate the time delay

     $$D_{j,m} = \arg\max_{\tau} \mid \hat{r}^{GCC}_{s_j \kappa_{j,m}}(\tau) \mid \quad (4.14)$$

[3] Note that the E-step can be implemented recursive to save computation time !
[4] Another possibility, which sometimes works better, is to set $\beta_{EM} = \frac{1}{N+L}$.

afterwards compute weight $W_{j,m}^{out}$ with correctly delayed reservoir/input signal $\hat{s}_j(n) = s_j(n - D_{j,m})$

$$W_{j,m}^{out} = (\hat{\mathbf{s}}_j^\intercal \hat{\mathbf{s}}_j)^{-1} \hat{\mathbf{s}}_j^\intercal \kappa_{j,m} \tag{4.15}$$

Perform E- and M-steps for each element of $\mathbf{s}(n)$ for one complete iteration.

2. Iterate step 1 (EM algorithm) until the delays are converged and repeat the whole procedure for each target output $g^{-1}(y_m(n))$ where $m = 1, \ldots, M$.

3. Afterwards all delays are fixed and reservoir/input signals are delayed according to the estimated values in delay matrix $\mathbf{D}$.

4. Dismiss data from an initial washout period n_{min} where $n < n_{min}$ and collect the remaining correctly delayed input and network states $(s_1(n - D_{1,m}), \ldots, s_{N+L}(n - D_{N+L,m}))$ for each output target signal $y_m(n)$ row-wise into a $(n_{max} - n_{min}) \times (N + L)$ matrix $\mathbf{S}_m$, where n_{max} is the number of training examples, and afterwards collect current target signal $g^{-1}(y_m(n))$ into a $(n_{max} - n_{min}) \times 1$ matrix $\mathbf{T}_m$.

5. For each output $y_m(n)$ compute pseudo-inverse $\mathbf{S}_m^\dagger$ and put

$$\mathbf{W}_m^{out} = (\mathbf{S}_m^\dagger \mathbf{T}_m)^\intercal$$

or alternatively use any other offline training algorithm from section 2.3.

4.5 Discussion

Unfortunately no clear winner is found yet for the delay&sum readout learning algorithm, but it should be possible to find a more optimal method in further research, so that the user has no additional parameter for choosing an algorithm. In most simulations the EM-based method B outperformed simple method A for system identification tasks (ESNs without output feedback) and learning algorithm A was more suited for ESNs in generator mode. A more detailed comparison will be given in simulations from chapter 5 and 6.

One way to improve the algorithm could be to use more advanced multipath time delay estimation methods as outlined in Y.Huang et al. [2006]. Additionally an online version should be developed, which could be done with an EM-based approach similar to Frenkel and Feder [1999], or as in Duro and Reyes [1999], where the backpropagation algorithm was extended for feedforward neural networks with weights and delays in their synaptic connections.
Another possibility would be to also add fixed delays in the synaptic connections of the reservoir or they could be pre-adapted to a specific task with a similar method as in Bone et al. [2000] or Bone et al. [2002].
Furthermore it was illustrated in Schmitt [1999] that the computational and learning capabilities of artificial neural networks are considerably enlarged when transmission delays become adjustable. It was also noted therein that tuning the delays is a task that is believed not to have efficient algorithms with provable performance guarantee.

Nevertheless, altogether one can say that ESNs with a delay&sum readout show a significant better performance for many tasks, regardless of the chosen learning algorithm, also for very simple systems where one might not think that long-term dependencies are relevant (see for instance section 6.1). Furthermore the computational complexity of the network is still the same, but some additional memory for delaylines is necessary.

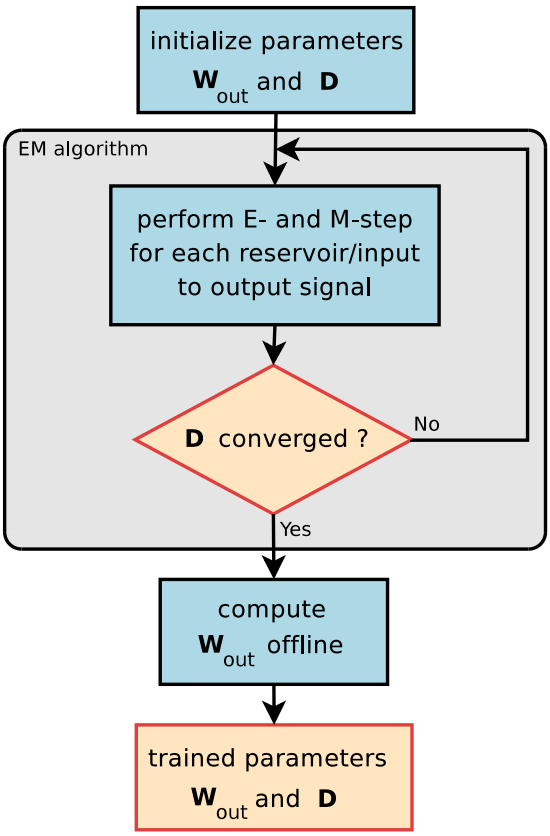

Figure 4.3: EM-based learning algorithm. First weights and delays are estimated with the EM method, afterwards delays are fixed and weights are recalculated offline. For a description of the individual steps see text.

Chapter 5

Simulations

This chapter presents three synthetic examples which will demonstrate where and how filter neurons and a delay&sum readout have advantages. The first example in section 5.1 is about multiple attractor learning and will show that filter neurons are very useful for this task. A single echo state network is trained to generate multiple sines and the results are compared to similar experiments from literature. Section 5.2 is about chaotic attractor learning where an ESN tries to predict the Mackey-Glass system, which is a standard benchmark task in research on time series prediction. Almost every technique for nonlinear system modeling is tested with the Mackey-Glass system and echo state networks significantly outperformed all other methods. Nonetheless a delay&sum readout is still able to boost the performance. Finally the third example in section 5.3 is a sparse nonlinear system identification task with long-term dependencies. A relatively small ESN will be able to identify nonlinear dynamical systems which have dependencies far in the past. No other similar method was found to handle such systems.

Four different error measures are used in that chapter, mainly to be able to compare the results to measures given in literature. The mean square error (MSE), as already introduced in chapter 2, between a target signal $y_{target}(n)$ and a generated signal $y(n)$ for $n = 0, \ldots, N-1$ is calculated as

$$MSE = \frac{1}{N} \sum_{n=0}^{N-1} \left(y_{target}(n) - y(n) \right)^2 . \tag{5.1}$$

Another possibility is to take the square root of the MSE, which results in the root mean square error ($RMSE$)

$$RMSE = \sqrt{\frac{1}{N} \sum_{n=0}^{N-1} \left(y_{target}(n) - y(n) \right)^2 }. \tag{5.2}$$

When using signals with different amplitudes it is more meaningfull to normalize the error with the variance of the target signal σ^2_{target}, which is called the normalized mean square error ($NMSE$)

$$NMSE = \frac{1}{N\sigma^2_{target}} \sum_{n=0}^{N-1} \left(y_{target}(n) - y(n) \right)^2 . \tag{5.3}$$

Finally, by taking the square root of the $NMSE$, one gets the normalized root mean square error ($NRMSE$)

$$NRMSE = \sqrt{\frac{1}{N\sigma^2_{target}} \sum_{n=0}^{N-1} \left(y_{target}(n) - y(n) \right)^2 }. \tag{5.4}$$

5.1 Multiple Superimposed Oscillations

In the multiple superimposed oscillations (MSO) task an echo state network is trained to be a generator of a superposition of sine signals. This sounds like a quite simple task, however, standard echo state networks are unable to learn functions composed of even two superimposed oscillators. The problem is that the dynamics of all reservoir neurons are coupled, while this task requires that both oscillations can be represented by the internal state vector. With filter neurons in the reservoir and with a delay&sum readout it will be possible to train ESNs as a generator for a superposition of sinewaves. The results will be compared to a similar recurrent neural network structure from Schmidhuber et al. [2007], called Evolino, and to the work of Xue et al. [2007], where they used ESNs with multiple decoupled reservoirs.

In Jaeger et al. [2007b] some general comments on the MSO task are given. It is easy for linear systems, for instance ESNs with only linear activation functions, to generate multiple sines. The training error will be close to machine precision as long as there are at least two reservoir neurons for one sine component. Also when using such a system as a generator it will be very precise after a teacher-forcing period. However, in a linear system the generated sine components are not stably phase-coupled and they are not asymptotically stable oscillations. In the presence of perturbations, both the amplitude and relative phase of the sine components will go off on a random walk (see figure 5.1 **(c)**).

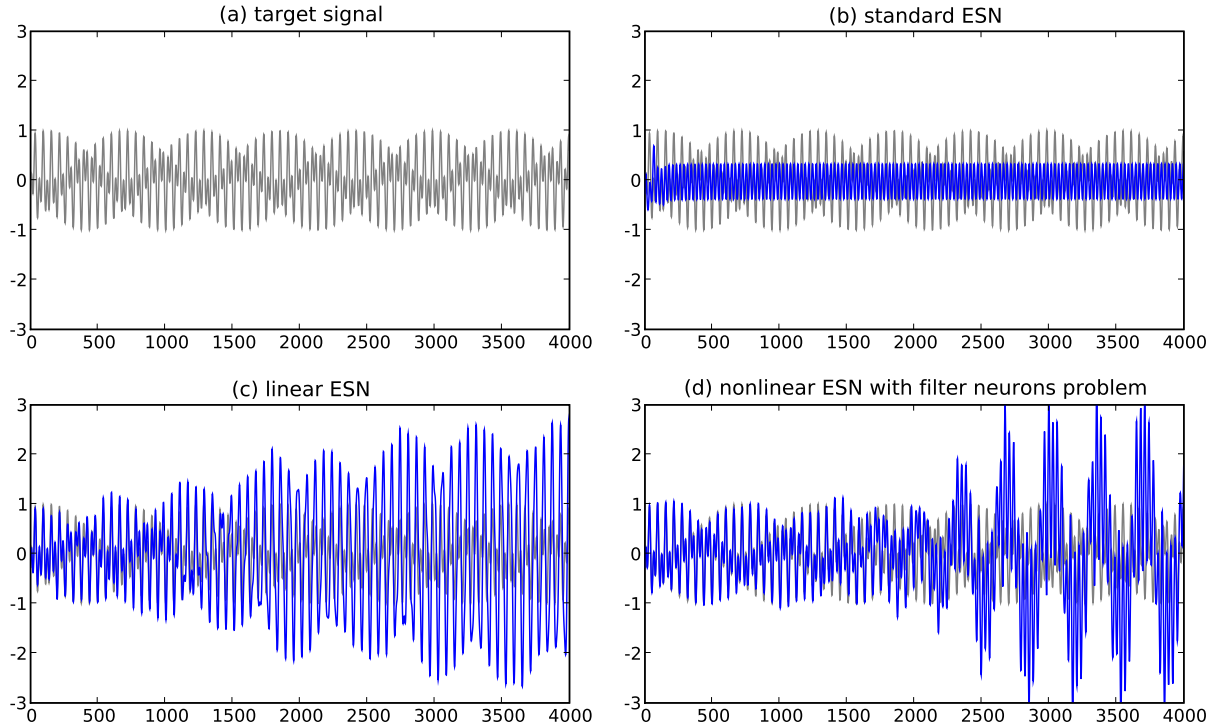

Figure 5.1: Some stability problems of the multiple superimposed oscillations task. Figure **(a)** shows the target signal, a superposition of two sines: $0.5sin(0.1x) + 0.5sin(0.211x)$. Figure **(b)** presents the output of a nonlinear standard ESN, which is not able to learn both sine components. In figure **(c)** a linear ESN is able to produce both sines at the beginning, but then it slides away and finally in **(d)** a nonlinear ESN with filter neurons is trained on the MSO task and switches into a different attractor after about 2000 timesteps.

For nonlinear systems the MSO task is quite difficult and often results in unwanted dynamic side effects (see figure 5.1 **(d)**). While the MSO signals can be seen as additive superpositions of sines in a linear systems view, nonlinear systems do not lend themselves to noninterfering additive couplings (Jaeger et al. [2007b]). Trained nonlinear networks show often very low test error for a few thousand time steps, but become unstable and switch to completely different attractors afterwards. Figure 5.1 shows some problems with nonlinear and linear standard ESNs and the output of an unstable nonlinear ESNs with filter neurons.

In the following the performance of ESNs with filter neurons will be compared to the work of Schmidhuber et al. [2007] and Xue et al. [2007] for a short-time prediction task. In Schmidhuber et al. [2007] an evolino-based long short term memory (LSTM) recurrent neural network is trained to learn functions composed of two to five sines. The following target signal

$$sin(0.2x) + sin(0.311x) + sin(0.42x) + sin(0.51x) + sin(0.74x)$$

was used, as shown in figure 5.2, with a washout time of 100 time steps, then the network weights were calculated from steps $101, \ldots, 400$ and the testing error was evaluated in time steps $701, \ldots, 1000$. The $NRMSE_{test}$ was calculated for two sines $(sin(0.2x) + sin(0.311x))$, three sines $(sin(0.2x) + sin(0.311x) + sin(0.42x))$, four sines $(sin(0.2x) + sin(0.311x) + sin(0.42x) + sin(0.51x))$, five sines $(sin(0.2x) + sin(0.311x) + sin(0.42x) + sin(0.51x) + sin(0.74x))$ and was compared to an LSTM network with subpopulations of size 40, 40, 40, 100, where the number of memory cells was 10, 15, 20, 20.

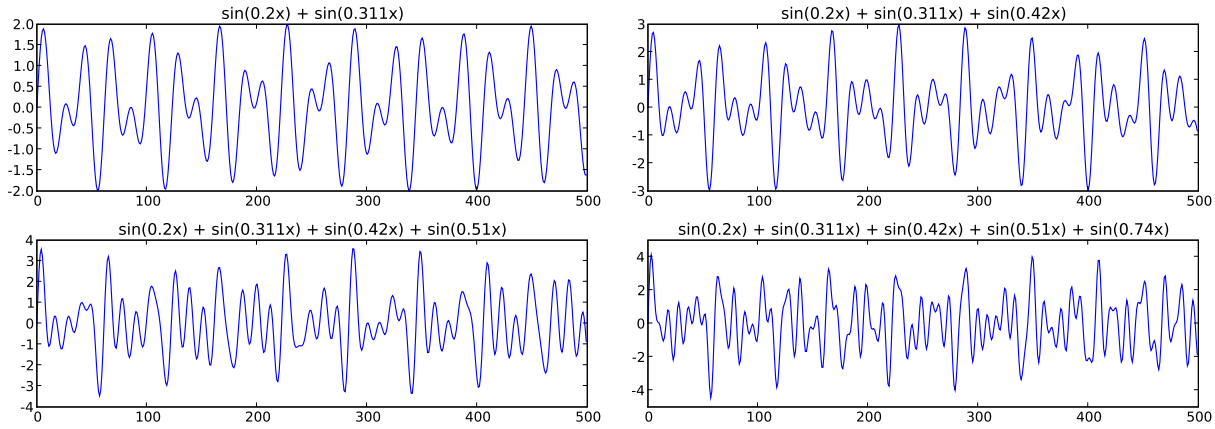

Figure 5.2: 500 timesteps of the target signal from Schmidhuber et al. [2007] to learn functions composed of two to five sines.

For all simulations the same echo state network of size 100 with logarithmically spaced bandpass filters over the frequency range of those five sines was trained, using the same washout, train and testsize as the LSTM network. Additionally the target signal was rescaled into a range of $[-1, 1]$ to be able to use the same ESN for all simulations. The detailed setup is given in table 5.1 and the $NRMSE_{test}$ for the ESN and LSTM network, averaged over 20 simulations, is shown in table 5.2. One can see that when using ESNs with filter neurons and a delay&sum readout it is also possible to train 5 sines as in the LSTM network example and the performance is significantly better.

In Xue et al. [2007] they trained the same two sines problem

$$sin(0.2x) + sin(0.311x)$$

reservoir neurons	100
reservoir connectivity	0.2
spectral radius	0.8
output-feedback weights	random between $[-0.1, 0.1]$
reservoir activation function	tanh
output activation function	linear
delay&sum readout	yes, maximum delay was restricted to 100 samples
reservoir neurons filter type	band-pass
filter spacing	logarithmically between frequencies $0.19/(2\pi)$ and $0.75/(2\pi)$
filter bandwidth	0.5 octaves
noise during training	no
noise during testing	no

Table 5.1: ESN setup for the same MSO task as in Schmidhuber et al. [2007].

Nr. of sines	$NRMSE_{test}$ LSTM	$NRMSE_{test}$ Filter-ESN
2	2.01×10^{-3}	3.65×10^{-9}
3	2.44×10^{-3}	3.57×10^{-7}
4	1.51×10^{-2}	1.40×10^{-5}
5	1.60×10^{-2}	7.24×10^{-5}

Table 5.2: Performance of the multiple superimposed oscillations task. Weights are trained from time steps $101, \ldots, 400$ and $NRMSE_{test}$ is calculated from steps $701, \ldots, 1000$, averaged over 20 experiments. Second column shows the results from Schmidhuber et al. [2007] and third column from echo state networks with filter neurons and a delay&sum readout.

with an "decoupled echo state network with lateral inhibition", which basically consists of four almost independent ESN reservoirs with 100 neurons each, where each one has a different spectral radius (0.6, 0.7, 0.8 and 0.9). Here the training set consists of 700 data points with a washout time of 100 steps and a test mean squared error, calculated from time steps 701 to 1000, of $MSE_{test} = 3 \times 10^{-3}$ was achieved. For the same washout, training and test data size and the ESN setup from table 5.1 the simulations resulted in an MSE_{test} of 3.49×10^{-12}, averaged over 20 experiments[1]. When also using a reservoir with 400 neurons the MSE_{test} is even 3.04×10^{-20} and therefore significantly lower as in Xue et al. [2007].

Finally some general notes on the parameter selection for the short-term prediction task are presented:

- *Filter parameters*:
  The frequency range was chosen for obvious reasons to just contain all sine frequencies of the task. Furthermore no significant difference in performance was noteable if a logarithmic or linear spacing for band-pass filters is used. The parameter which has a big influence on the performance is the filter bandwidth ϕ, figure 5.3 shows a comparison for different values of ϕ.

[1] The target signal was again scaled into a range of $[-1, 1]$ and afterwards rescaled to the original range for MSE_{test} calculation.

- *Noise*:
  Usually adding a small noise term $v_{train}(n)$ in the state update equation 3.9 is mandatory for stability when ESN's are trained as generators, as for instance analyzed in Jaeger [2001] and Jaeger et al. [2007b]. However, with the delay&sum readout it was not necessary for this simple task.

- *Delay&Sum readout*:
  The maximum delay was restricted to 100, because of the short washout time used in Schmidhuber et al. [2007]. Both learning algorithms, the simple method from section 4.3 and the EM-based algorithm from section 4.4, performed more or less similar and figure 5.4 shows a plot of the resulting delay distributions. However, it was important in both algorithms to use the cross correlation and not the GCC method for delay estimation. In general, when using the same setup without a delay&sum readout, the $NRMSE_{test}$ was larger approximately by a factor of 10.

- *Spectral radius and feedback weights*:
  The network was very robust to changes of the spectral radius and the exact value was not important. On the other hand the feedback weights had a big influence on the performance, small values resulted in a much lower $NRMSE_{test}$.

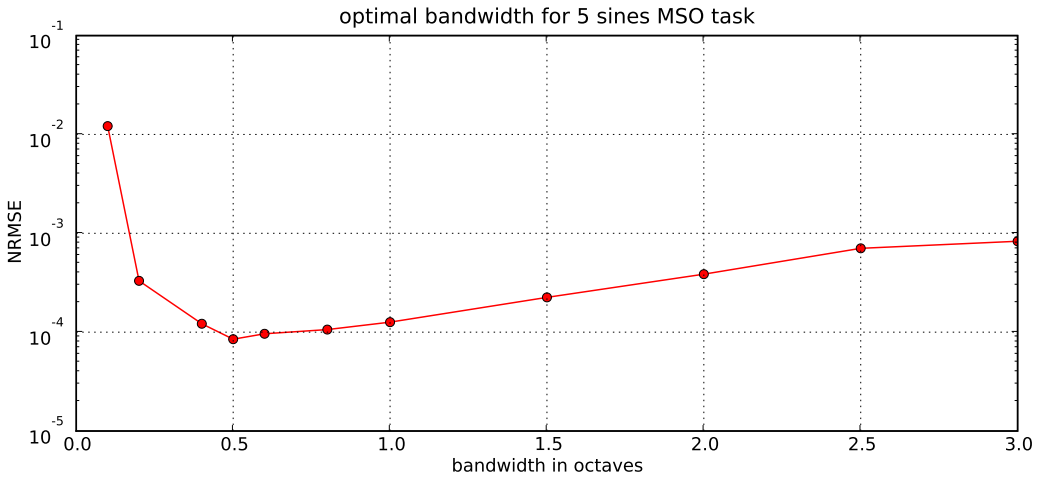

Figure 5.3: $NRMSE_{test}$ for the 5 sines problem and the ESN setup from table 5.1 with a variable bandwidth ϕ of the band-pass filters. The optimal bandwidth was found to be about 0.5 octaves between band-pass cutoff frequencies. Note that the y-axis is logarithmic.

In the previous examples the performance of a short-term prediction task was discussed, which can be measured easily by a mean squared error. When analyzing the long-term stability of such networks, there is no similar quantifiable criterion which can be computed with comparable ease. One strategy to achieve long-term stability is to use a quite high noise term $v_{train}(n), v_{test}(n)$ in the state update equations during training and testing. In training this helps to be immune against deviations and in testing the noise is used to test if the trained network is a dynamically stable periodic attractor. The number of successfully trained networks and the training and testing error depends of course on the amplitude of the noise terms.

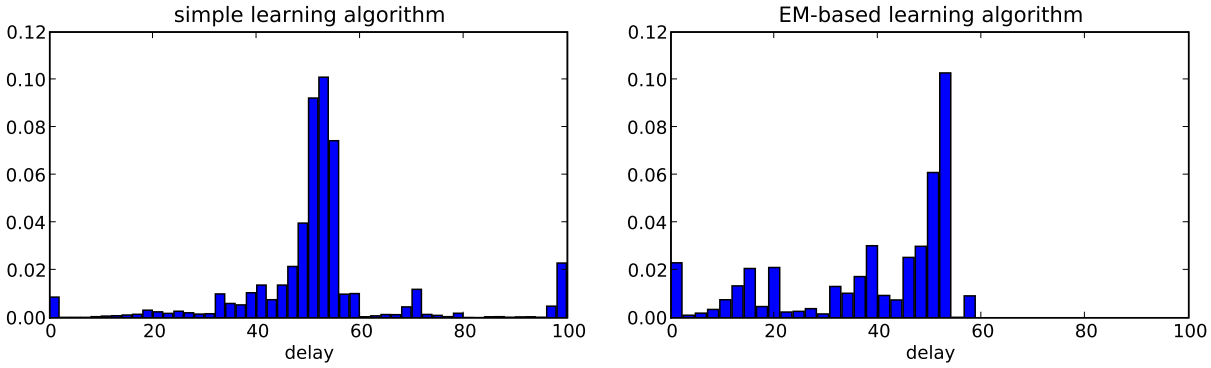

Figure 5.4: Distribution of the learned delays for the MSO task with 5 sines. Left image shows the result with the simple learning algorithm and the right image of the EM-based learning algorithm after 10 iterations.

To test the long-term stability of the ESN setup from table 5.1, the network was first run for 5000 time steps, then trained from steps 5001 to 10000 with an additional noise term $\upsilon_{train}(n)$ sampled from an uniform distribution between $[-0.0001, 0.0001]$ to be robust against small deviations and to keep the output weights low. Afterwards it was simulated for 100000 time steps with a high noise term $\upsilon_{test}(n)$ between $[-0.01, 0.01]$, which should test if the learned attractor is actually stable.

The noise parameter $\upsilon_{train}(n)$ is a control of the precision-stability tradeoff. When using $\upsilon_{train}(n)$ between $[-0.0001, 0.0001]$ all performed simulations were stable with an $NRMSE_{test}$ of 0.14 (averaged over 20 independent simulations)[2], even when simulating the network for 10^6 steps. As $\upsilon_{train}(n)$ gets lower also the $NRMSE_{test}$ will be lower, but sometimes the ESNs will become unstable. Besides noise, the reservoir size and a delay&sum readout also have an influence on the stability. Usually the more sines or multiple attractors one wants to model, the bigger the reservoir should be. When using more neurons in the reservoir or a delay&sum readout, it was not necessary to add that much noise during training.

[2]Note that the quite high $NRMSE_{test}$ is only because of the noise term $\upsilon_{test}(n)$ during simulation. When using ESNs without testing its stability one would set $\upsilon_{test}(n)$ to zero, which results in a $NRMSE_{test}$ of about 4×10^{-3} for a 100000 time steps simulation.

5.2 Mackey-Glass System

An often used benchmark for time series modeling is the Mackey-Glass delay differential equation. It is given for instance in Jaeger [2001] as

$$\dot{y}(t) = \alpha \frac{y(t-\tau)}{1+y(t-\tau)^{\beta}} - \gamma y(t) \tag{5.5}$$

where usually the parameters are set to $\alpha = 0.2$, $\beta = 10$ and $\gamma = 0.1$. The system has a chaotic attractor if $\tau > 16.8$ and in most of the benchmarks τ is set to $\tau = 17$ or $\tau = 30$, a plot of both settings is shown in figure 5.5. In discrete time the Mackey-Glass delay differential equation can be approximated through

$$y(n+1) = y(n) + \delta \left(0.2 \frac{y(t-\tau/\delta)}{1+y(t-\tau/\delta)^{10}} - 0.1 y(t) \right) \tag{5.6}$$

with a stepsize δ. Here the stepsize is set to $\delta = 1/10$ to compare the results to Jaeger [2001], Jaeger and Hass [2004] and the references in there[3]. To be able to use the timeseries with tanh-ESNs it was rescaled into a range of $[-1, 1]$ by transforming it with

$$y_{ESN}(n) = tanh(y(n) - 1)$$

as in Jaeger [2001].

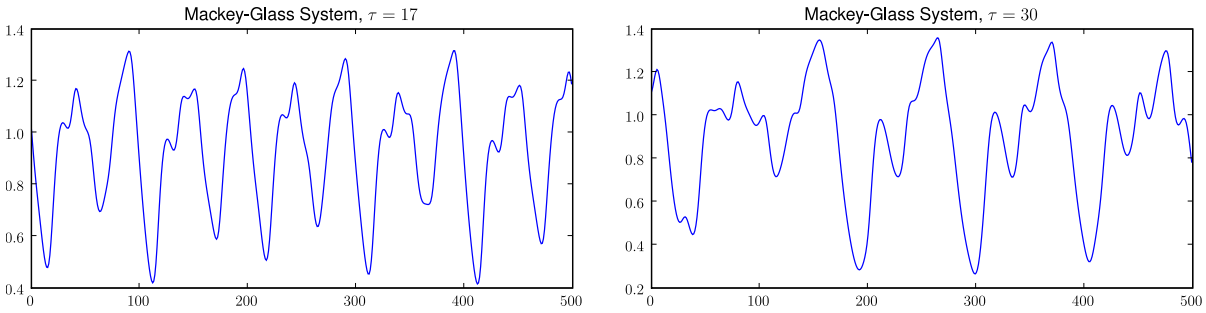

Figure 5.5: 500 steps of a Mackey-Glass sequence for $\tau = 17$ (left) and $\tau = 30$ (right). Both systems have a chaotic attractor and are standard benchmarks in research on time series prediction.

In this section mainly the $\tau = 30$ task will be studied, because it is much more difficult to learn and shows that a delay&sum readout boosts the performance of standard ESNs and also outperforms all other known methods. The ESN setup is the same as in Jaeger [2001][4] and summarized in table 5.3. Filter neurons were not useful in this task. They would divide the reservoir in autonomous sub-networks, which are able to learn multiple attractors. However, here only one single attractor should be trained and therefore the performance with filter neurons in the reservoir was worse.

As in Jaeger [2001] two training sequences with different length were generated. In the first example the network was trained from 2000 samples with an initial washout time of 1000 samples

[3] Actually the Mackey-Glass sequence was generated from the same code as in Jaeger and Hass [2004], which can be downloaded from http://www.faculty.iu-bremen.de/hjaeger/pubs/ESNforMackeyGlass.zip.

[4] The setup is not exactly the same, in Jaeger [2001] leaky-integrator neurons are used, here standard neurons.

reservoir neurons	400
reservoir connectivity	0.1
spectral radius	0.95
input weights	random between $[-0.14, 0.14]$
output-feedback weights	random between $[-0.56, 0.56]$
reservoir activation function	tanh
output activation function	linear
delay&sum readout	yes
reservoir neuron type	standard neurons, no filter
input signal	constant bias input signal of size 0.2
noise during training	yes, dependent on the task
noise during testing	no

Table 5.3: ESN setup for the Mackey-Glass system prediction task.

and in the second one a much bigger trainsize of 20000 time steps, with an additional washout of 1000 samples, was used. The training and test sequences were generated from equation 5.6 without using the first 1000 samples, to get rid of initial washouts.
Noise is also very important, as already mentioned in section 5.1 for the MSO task. A small noise term $\upsilon(n)$ during training is mandatory for ESNs used as a generator and there is again a precision-stability tradeoff. In general, the more complex the system and the shorter the training sequence, the more noise is needed.
Both delay learning algorithms from chapter 4 performed similar, using the cross correlation and not the GCC method for delay estimation. Figure 5.6 shows a plot of the delay distribution for the $\tau = 17$ and $\tau = 30$ task.

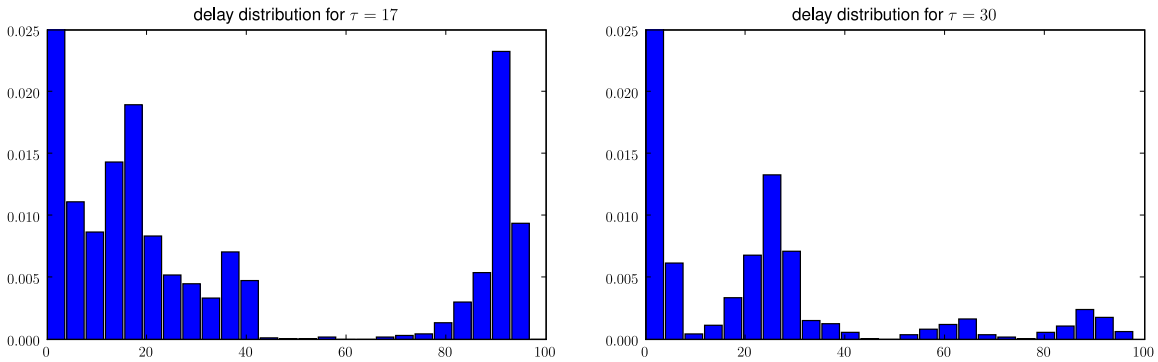

Figure 5.6: Distribution of the learned delays, left for $\tau = 17$ and right for the $\tau = 30$ task. Delays were calculated with the simple training algorithm, delay at time step zero (no delay) is omitted in the plot because of its high probability.

One qualitative way to compare the system output is an attractor plot. Figure 5.7 shows the original attractor plot for $\tau = 17$, $\tau = 30$ and the plot obtained from the trained network's output of an ESN trained with 2000 samples. All figures are generated from 6000 time steps and rendered in two dimensions by plotting $y(n)$ versus $y(n + 20)$. Additionally the output of the echo state network was retransformed to the original scale by

$$y(n) = arctanh(y_{ESN}(n)) + 1.$$

Such a plot can only show if the network is able to learn the essential structure of the attractor and if it stays stable, but it won't give a quantifiable or comparable measure of the performance.

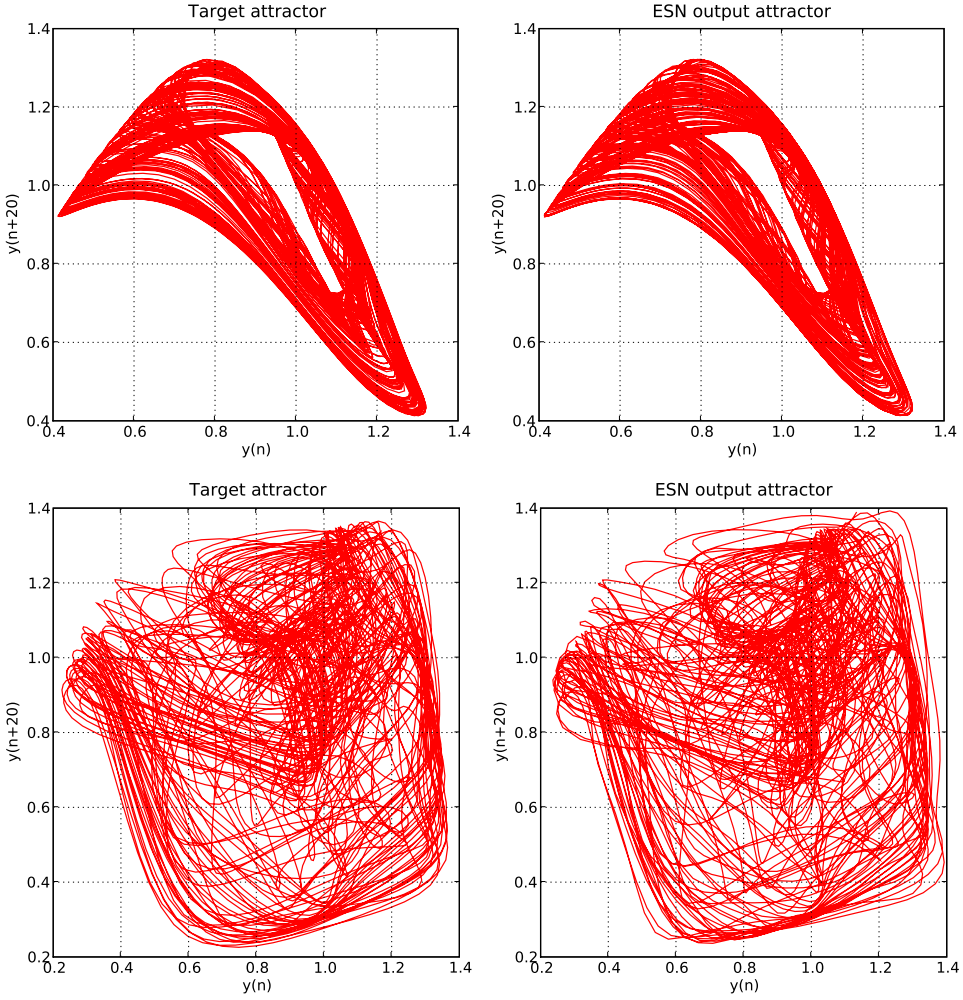

Figure 5.7: Attractor plots from 6000 timesteps of the Mackey-Glass sequence, $y(n)$ versus $y(n+20)$, ESN-setup as in table 5.3. First row shows plots for $\tau = 17$, where the left picture is the original sequence and the right picture generated from an echo state network trained with 2000 steps. Second row shows the same for the more complex case where $\tau = 30$. One can see that the echo state network is able to learn the essential structure of the original attractor.

Another possibility to compare the performace of different modeling techniques is to calculate an error measure of the system's time series evolution, as done in the MSO task. However, this is not really meaningful for chaotic systems because sometimes small deviations do not diverge quickly and sometimes the system is in a state where small deviations lead to big changes in the output. Therefore a mean squared error of a specific prediction horizon is usually used in literature to compare results of chaotic time series prediction, as also done in Jaeger [2001]. A common value for the Mackey-Glass system is a 84 or 120 steps prediction. To calculate for instance an $NRMSE_{84}$ the trained echo state network is teacher forced with a 1000 sample timeseries of different original Mackey-Glass signals and afterwards simulated for 84 time steps to get the predicted output $\hat{y}(n+84)$. This value can be compared, after retransformed to the

original scale, with the original Mackey-Glass signal $y(n+84)$. To average over different initial states the same echo state network was run 100 times with new teacher forcing signals and at each run i the predicted output $\hat{y}_i(n+84)$ and target signal $y_i(n+84)$ were collected and the $NRMSE_{84}$ was calculated with

$$NRMSE_{84} = \sqrt{\frac{1}{100\sigma^2}\sum_{i=1}^{100}(y_i(n+84) - \hat{y}_i(n+84))^2} \tag{5.7}$$

where σ^2 is the variance of the original attractor signal.

The results for a 84 and 120 step prediction with $\tau = 30$ are shown in table 5.4, where it can be seen that a delay&sum readout is able to improve the performance of ESNs. A uniform noise term $\upsilon(n)$ was added during training in the state update equation, for the 2000 step training sequence it was chosen between $[-10^{-5}, 10^{-5}]$ and for the 20000 step training between $[-10^{-8}, 10^{-8}]$. In Jaeger [2001] an $NRMSE_{84}$ of 0.11 for $\tau = 30$ was achieved, which is comparable to the results reported here for standard ESNs with an $NRMSE_{84}$ of 0.136. Figure 5.8 shows the further development of the error up to a prediction horizon of 400 time steps. The advantage of a delay&sum readout is smaller in the $\tau = 17$ task, a plot of the $NRMSE$ development is given in figure 5.9. As already mentioned, the $\tau = 30$ sequence is much more difficult to learn and it seems that a standard ESN already performs very well in the $\tau = 17$ task.

Mackey-Glass, $\tau = 30$				
method	$NRMSE_{84}$	$RMSE_{84}$	$NRMSE_{120}$	$RMSE_{120}$
standard ESN, 1K+2K	1.36×10^{-1}	3.86×10^{-2}	2.17×10^{-1}	6.14×10^{-2}
standard ESN, 1K+20K	7.18×10^{-2}	2.05×10^{-2}	1.20×10^{-1}	3.43×10^{-2}
D&S ESN, 1K+2K	3.11×10^{-2}	8.75×10^{-3}	4.86×10^{-2}	1.37×10^{-2}
D&S ESN, 1K+20K	2.00×10^{-2}	5.64×10^{-3}	3.17×10^{-2}	8.91×10^{-3}

Table 5.4: $NRMSE$ and $RMSE$ of the Mackey-Glass sequence with $\tau = 30$ and an ESN setup as in table 5.3. Results are for different prediction horizons (84-step and 120-step prediction) and for different training length (training from 2000 (2K) steps and 20000 (20K) steps, always with a washout time of 1000 (1K) steps). The error was calculated from 100 independent runs.

For a comparison of the Mackey-Glass time series prediction to other techniques one has to consider the work of Jaeger and Hass [2004], where it was shown that for the $\tau = 17$ task echo state networks vastly outperform all other known methods. A bigger ESN with 1000 reservoir neurons was trained in that paper and resulted in an $NRMSE_{84}$ of 6.309×10^{-5}, compared to a previously achievable accuracy of an $NRMSE_{84}$ of 1.995×10^{-2}. Results from various other techniques are summarized in Gers et al. [2001].

A comparison is more interesting for the $\tau = 30$ task. According to Jaeger and Hass [2004] the best previous result was in Feldkamp et al. [1998], where a multilayer perceptron with output feedback was trained with a refined version of backpropagation through time (extended Kalman filter multi-stream training). They calculated a root mean square error ($RMSE$) for a 120 step prediction and achieved an $RMSE_{120}$ of 0.04, which can be compared to the results of table 5.4 where an ESN with D&S readout achieved an $RMSE_{120}$ of 0.00891. When using bigger ESNs with a setup as in table 5.3, but with 700 neurons and a refined version[5] as in Jaeger and Hass

[5] In the refined version ten independently created ESNs, 700 neurons each, were trained one after the other,

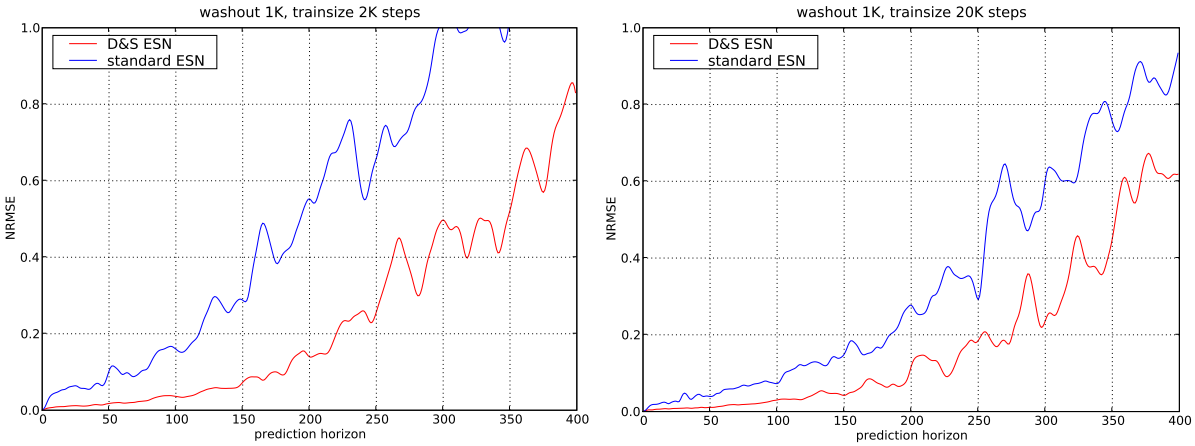

Figure 5.8: $NRMSE$ development of the Mackey-Glass sequence with $\tau = 30$ for prediction horizons up to 400 time steps, always calculated from 100 independent runs. An ESN setup as presented in table 5.3 was used with and without a delay&sum readout, the trainsize was 2000 (left plot) and 20000 (right plot) samples.

[2004], the error can be further reduced to an $RMSE_{120}$ of 0.0072 for 2000 and an $RMSE_{120}$ of 0.00265 for 20000 training steps, which improves the result in Feldkamp et al. [1998] by a factor of ≈ 15.

but in simulation the outputs of all 10 networks were averaged. The ten networks got also the output feedback from the averaged value.

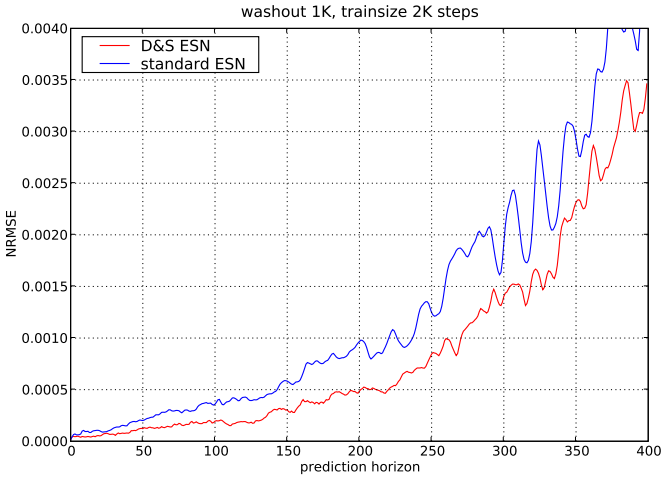

Figure 5.9: $NRMSE$ development of the Mackey-Glass sequence with $\tau = 17$ for prediction horizons up to 400 time steps, calculated from 100 independent runs. The ESN parameters are presented in table 5.3 and the network was trained from 2000 time steps. The difference in performance with or without a delay&sum readout is smaller as in the $\tau = 30$ task.

5.3 Sparse Nonlinear System Identification

One very useful feature when using D&S ESNs without feedback is, that the system is independent from a time-shift of the input signals. For instance the nonlinear systems $y(n) = x(n)x(n-1)$ and $y(n) = x(n-100)x(n-101)$ really need the same complexity in the reservoir, because the reservoir runs autonomous and a delay&sum readout simply learns the delay. Even better, with the D&S readout it is also possible to learn systems which depend on different time regions, as for example $y(n) = x(n)x(n-50)x(n-100)$, without the need of very big reservoirs for just modeling the time shifts. The examples of this section will show that an ESN with a delay&sum readout is well suited for a sparse nonlinear system identification with long-term dependencies.

Nonlinear systems can be classified in three types, as done for instance in Jaeger and Hass [2004]. Two major classes, which are studied in this section, are Nonlinear Moving Average (NMA) and Nonlinear Auto-Regressive Moving Average (NARMA) systems. In a NMA system the output $y(n)$ is a function of previous inputs $x(n)$

$$y(n) = f(x(n), x(n-1), x(n-2), \dots)$$

where f is a nonlinear system function. NARMA systems are more complex nonlinear dynamical systems where the current output $y(n)$ depends on the previous input and output history

$$y(n) = f(y(n-1), y(n-2), \dots, x(n), x(n-1), x(n-2), \dots).$$

A third class is the Nonlinear Auto-Regressive (NAR) system, for instance the Mackey-Glass system from section 5.2. In a NAR system the output $y(n)$ depends only on previous outputs

$$y(n) = f(y(n-1), y(n-2), \dots).$$

The first example is a modified version of the well known 10th-order NARMA system, which was introduced in Atiya and Parlos [2000] in a comparison of various recurrent neural network architectures and afterwards used in many papers about echo state networks, for instance in Jaeger [2003]. In Atiya and Parlos [2000] the original system is given as

$$y(n) = 0.3y(n-1) + 0.05y(n-1)\left[\sum_{i=1}^{10} y(n-i)\right] + 1.5x(n-10)x(n-1) + 0.1$$

and here a modified version is used, where every delay in the system equation is multiplied by a factor τ

$$y(n) = 0.3y(n-\tau) + 0.05y(n-\tau)\left[\sum_{i=1}^{10} y(n-i\tau)\right] + 1.5x(n-10\tau)x(n-\tau) + 0.1. \tag{5.8}$$

For example by setting $\tau = 10$ this results in a sparse NARMA system of order 100.

The same ESN setup as in Jaeger [2003] was employed and is given in table 5.5. This system was first driven with uniformly distributed white noise input between $[0, 0.5]$ for a washout time of 2000 time steps, then output weights and delays were calculated from the next 5000 steps. Aftwards the trained model was simulated with a new random input signal and after discarding initial values the test error was computed from 2000 time steps. It was also helpful to use an

reservoir neurons	100
reservoir connectivity	0.05
spectral radius	0.8
input weights	random between $[-0.1, 0.1]$
reservoir activation function	tanh
output activation function	linear
simulation algorithm	additional squared states as in section 2.4
delay&sum readout	yes
reservoir neuron type	standard neurons, no filter
additional input signal	constant bias input signal of size 1
noise during training	uniform noise between $[-0.0001, 0.0001]$
noise during testing	no

Table 5.5: ESN setup for the sparse nonlinear system identification task.

additional bias input, because equation 5.8 has a constant offset of 0.1. The performance of an ESN with and without a delay&sum readout, dependent on time lag τ, is shown in figure 5.10.

Additional squared state updates, as introduced in section 2.4, have usually boosted the performance in nonlinear system identification tasks. First delays are estimated between reservoir states $\mathbf{x}(n)$, input $u(n)$ and target output $y(n)$, aftwards the correctly delayed signals are squared and output weights are calculated in respect to the original and squared signals in the same way as in section 2.4.

This task demonstrates mainly the possibilities of a delay&sum readout, without delays in the readout the performance decreases very fast. Figure 5.10 shows also that for small τ the simple learning algorithm performs better than the EM-based learning algorithm. One problem with the EM-based method is, that some higher delays are detected, which are obviously not correct. However, on the other hand the performance stays approximately constant when increasing τ. Filter neurons in the reservoir were not useful here. It is not really known why, but when using filter neurons, the output usually drifted away from the target signal and the network became unstable.

The best achieved performance in Atiya and Parlos [2000] was an $NMSE_{train}$ of 0.241[6] for the original 10th-order system ($\tau = 1$), which corresponds to an $NRMSE_{train} \approx 0.49$. Note that they calculated the error directly form training and not from testing data as done here. One can see that the result of a delay&sum ESN for a 100th-order system ($\tau = 10$) is still better than the best performance for the 10th-order system in Atiya and Parlos [2000] and when using bigger reservoirs and more training data it is even possible to further reduce the test error. An additional unpublished result with a recurrent neural network trained as in Feldkamp et al. [1998] is mentioned in Jaeger [2003], which reached a slightly better precision[7] as the ESN setup from table 5.5 for the $\tau = 1$ task.

As a second example a less complex NMA system will be analyzed, which incorporates

[6] According to Jaeger [2003] the authors of Atiya and Parlos [2000] miscalculated the $NMSE_{train}$, because they used a formula for zero-mean signals. In Jaeger [2003] the value was recalculated to $NMSE_{train} \approx 0.241$, which also agrees with the plots given in Atiya and Parlos [2000].

[7] The exact error value is not given in Jaeger [2003].

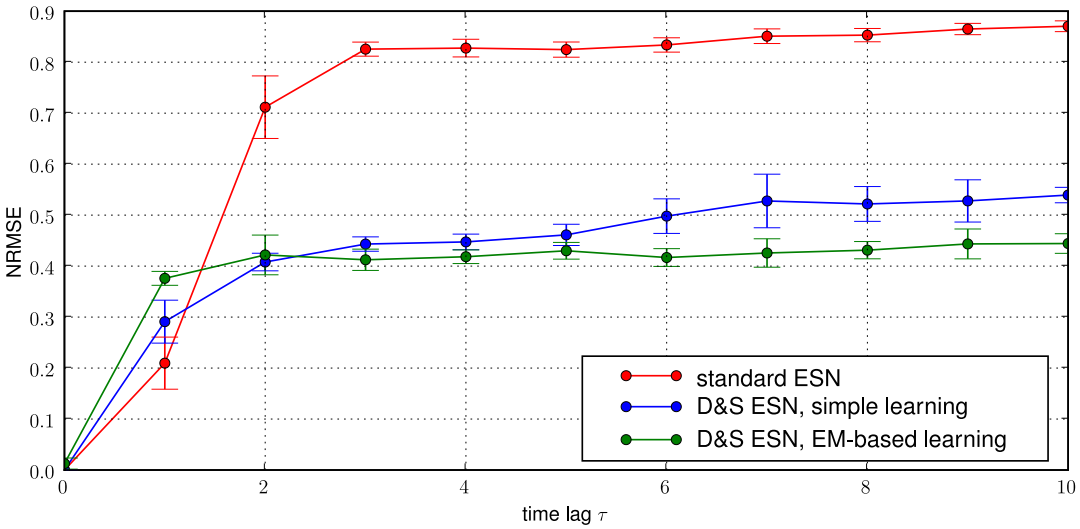

Figure 5.10: $NRMSE$ development for a nonlinear system identification task of equation 5.8 for different time lags τ. Each simulation was repeated ten times, mean and standard deviation for ESNs with a setup as in table 5.5 are plotted. The figure shows the different error development for a ESN with and without a delay&sum readout, using the simple and EM-based learning method.

information from different time regions. The system is given as

$$y(n) = \left(\sum_{i=0}^{3} x(n-i)\right)\left(\sum_{i=0}^{2} x(n-\tau-i)\right)\left(\sum_{i=0}^{2} x(n-2\tau-i)\right) \tag{5.9}$$

where parameter τ is the time lag between the three groups in time. Note that it is also possible to use a third-order volterra system for this identification task, however, it would need a very big memory or special structures if τ gets large.

Once more the setup from table 5.5 was used and the system was driven with uniformly distributed white noise input between $[0, 0.5]$. After a washout period of 2500 time steps the network was trained from the next $5000 + 2\tau$ steps[8]. The trained model was then simulated with a new random input signal and after discarding $100 + 2\tau$ initial values the test error was computed from 2000 time steps. Figure 5.11 shows the error development dependent on time lag τ for different delay learning algorithms.

The interesting result is that as time lag τ becomes bigger than 10 (which means a maximum delay of 22 steps in the system), the $NRMSE$ of D&S ESNs approximately remains at a value above 0.3, whereas the performance of standard ESNs further decreases. Figure 5.11 demonstrates also that the EM-based learning algorithm has a more constant performance and a much smaller standard deviation in different simulations. However, it detects also higher delays which are obviously not correct. An example plot of the delay distribution for $\tau = 1000$ can be seen in figure 5.12.

The simulations from this section demonstrate, that especially for system identification tasks the short term memory of ESNs, as defined in 2.5, can be vastly extended with a delay&sum

[8] Training size depends on τ to be able to compare higher values of τ.

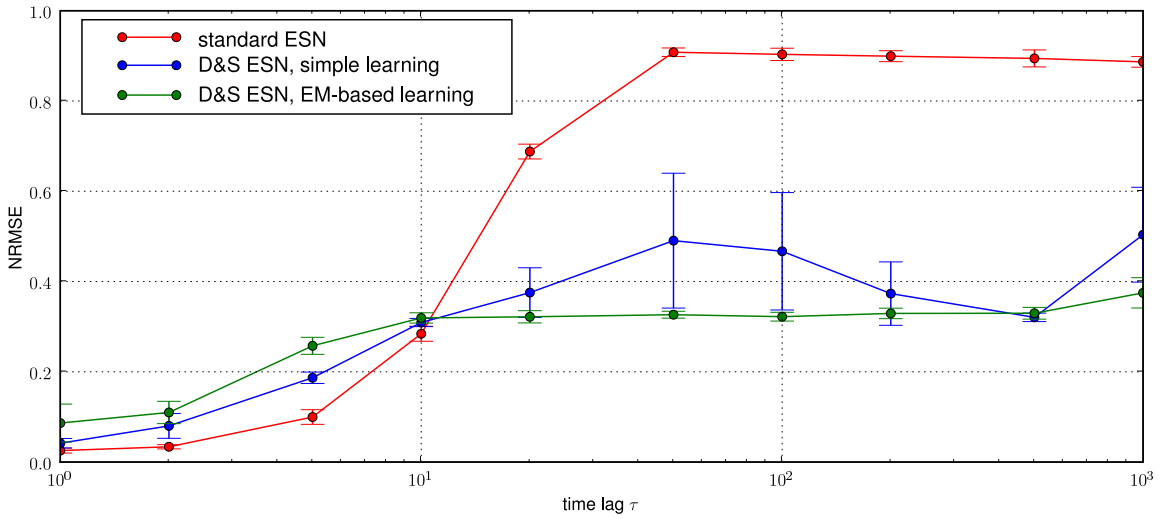

Figure 5.11: $NRMSE$ development for a nonlinear system identification task of equation 5.9 for different time lags τ. Each simulation was repeated ten times, mean and standard deviation for ESNs with a setup as in table 5.5 are plotted. The figure shows the different error development for a ESN with and without a delay&sum readout, using the simple and EM-based learning method. Note that the x-axis is logarithmic.

readout. They suggest also a different view on the short term memory. In Jaeger [2002a] short term memory is measured by training simple delays of the inputs, here the size of the dynamic reservoir only determines how complex a modeled system might be, more or less independent from delays of input or output feedback signals.

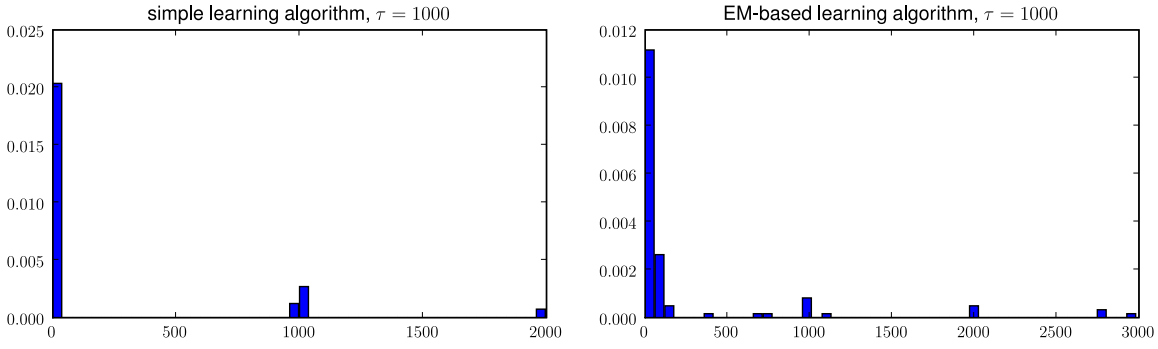

Figure 5.12: Distribution of the learned delays for time lag $\tau = 1000$, left calculated with the simple and right with the EM-based training algorithm. The maximum delay in the system is 2002, whereas the EM-based algorithm also finds values higher than that.

Chapter 6

Audio Examples

This chapter presents real-world applications of echo state networks in the area of audio signal processing and compares the performance to state-of-the-art alternative models. Nonlinear signal processing with neural networks has already been investigated in Haykin [1996], Kung [1997], Feldkamp and Puskorius [1998] or Uncini [2003]. Most of them used feedforward structures, but some also tried to model more complex systems with recurrent neural networks or other very special structures. Echo state networks could be seen as an unification of those investigations, which are very powerful, can cope with all the studied tasks, and are most of the time much easier to use.

Two signal processing examples are addressed in this section: nonlinear system identification and time series forecasting. A third big area of applications would be classification (music information retrieval, speech recognition, etc.), which is not studied in this work.
Section 6.1 presents an identification of a nonlinear tube amplifier and compares the results to an identification with volterra series. Finally in section 6.2 various methods for audio prediction are compared to an echo state network with filter neurons and a delay&sum readout. Especially for real-world problems the delay&sum readout and also filter neurons show advantages.

6.1 Nonlinear System Identification

Nonlinear system identification tasks are necessary in many applications in audio signal processing. Always when distorsions or saturations are perceiveable, linear modeling is not sufficient and nonlinear methods have to be applied. Usually volterra systems (Schetzen [1980]) were used in literature for applications like modeling audio effects with nonlinear distortions (Helie [2006]), nonlinear echo cancellation (Stenger and Rabenstein [1999]), loudspeaker linearization using pre-distorsion (Schurer et al. [1995]), nonlinear beamforming (Knecht [1994]) or restoration of nonlinearly distorted audio (Godsill and Rayner [1997]). In general also echo state networks could be utilized for all those tasks, with the advantage that they are quite easy to use and have a much lower computational complexity compared to higher order volterra systems without special structures for complexity reduction. Furthermore it was demonstrated in Maass and Sontag [2000], that neural networks with time delays are able to approximate all filters that can be characterized by volterra series. This section presents a nonlinear system identification of a tube amplifier as one example from the pool of possible applications.

With the introduction of digital methods in music reproduction the first major effort was to eliminate a number of technical artifacts produced by traditional analog audio systems so far (Schattschneider and Zoelzer [1999]). Nowadays it seems desirable to bring some nonlinearities

of the old equipment back into the all digital processing systems, because of their perceptible characteristic distorsions. Tube amplifiers are one example of nonlinear systems, which are still used today because of the "warmth" and "dirt" of its sound. Volterra series were utilized in literature to simulate tubes on digital computers (Helie [2006]), but usually only second or third-order systems with special structures for complexity reduction performed acceptable. Two different tubes, both implemented as open source digital audio plugins (LADSPA[1]) and frequently used in the linux audio community, are identified in this section and compared to an identification with volterra systems[2].

6.1.1 Measuring Nonlinear Systems

When measuring nonlinear systems on digital computers some special attention has to be paid to aliasing. Due to the bandlimited nature of discrete-time signal processing, the most obvious solution to the aliasing problem is oversampling (Schattschneider and Zoelzer [1999]), as illustrated in figure 6.1. In this case the input signal is first upsampled by a factor of L, using an interpolation filter H_1, then the nonlinear system of finite order $n < L$ is applied and spreads the spectrum of the oversampled input sequence by a factor n. After passing an anti-aliasing filter H_2, which reduces the bandwidth back to the Nyquist frequency, downsampling by a factor L finally leads to the output signal and only the harmonic components that fall inside the Nyquist range remain. Using this procedure the echo state network and the volterra system will just identify the bandlimited system without aliasing.

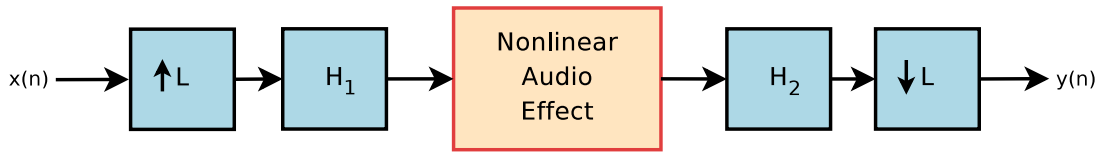

Figure 6.1: Oversampling to avoid aliasing problems produces by a nonlinear device. The input signal $x(n)$ is upsampled by a factor of L, using an interpolation filter H_1. Then the nonlinearity of finite order $n < L$ is applied and finally an anti-aliasing filter H_2, combined with downsampling by a factor of L, returns the output signal $y(n)$.

The next challenging task is to choose to right signals for the identification process, so that the trained ESN or volterra system performs well on arbitrary audio inputs. In linear signal processing usually impulses, swept sines, noise or maximum length sequences (see Mueller and Massarani [2001] for an overview) are used, because they include all frequencies of interest. However, in a nonlinear system identification all frequencies in all possible amplitudes should be present in this training signal. One common solution is to generate a swept sine, where the frequency slides over the area of interest and the amplitude is fixed. Afterwards the same procedure is repeated for many amplitude levels (see for instance Moeller et al. [2002], Abel and Berners [2006] and Schattschneider and Zoelzer [1999]). A second possibility is to use noise bursts, which include all frequencies of interest, with amplitude sweeps over the whole range (Moeller et al. [2002]).

[1] LADSPA: Linux Audio Developer's Simple Plugin API, a cross-platform standard that allows software audio processors and effects to be plugged into a wide range of audio synthesis and recording packages, see http://www.ladspa.org/.

[2] Of course it would be better to identify and measure a real tube amplifier and not its digital simulation. However, as always this is a question of available time.

After some experiments it turned out, that the by far best generalization error could be achieved with linear amplitude sweeps of Gaussian white noise, shaped with an additional low-pass filter[3]. Such a signal is presented in figure 6.2. The filter might improve the performance, because also real-world audio signals usually have a low-pass characteristic.

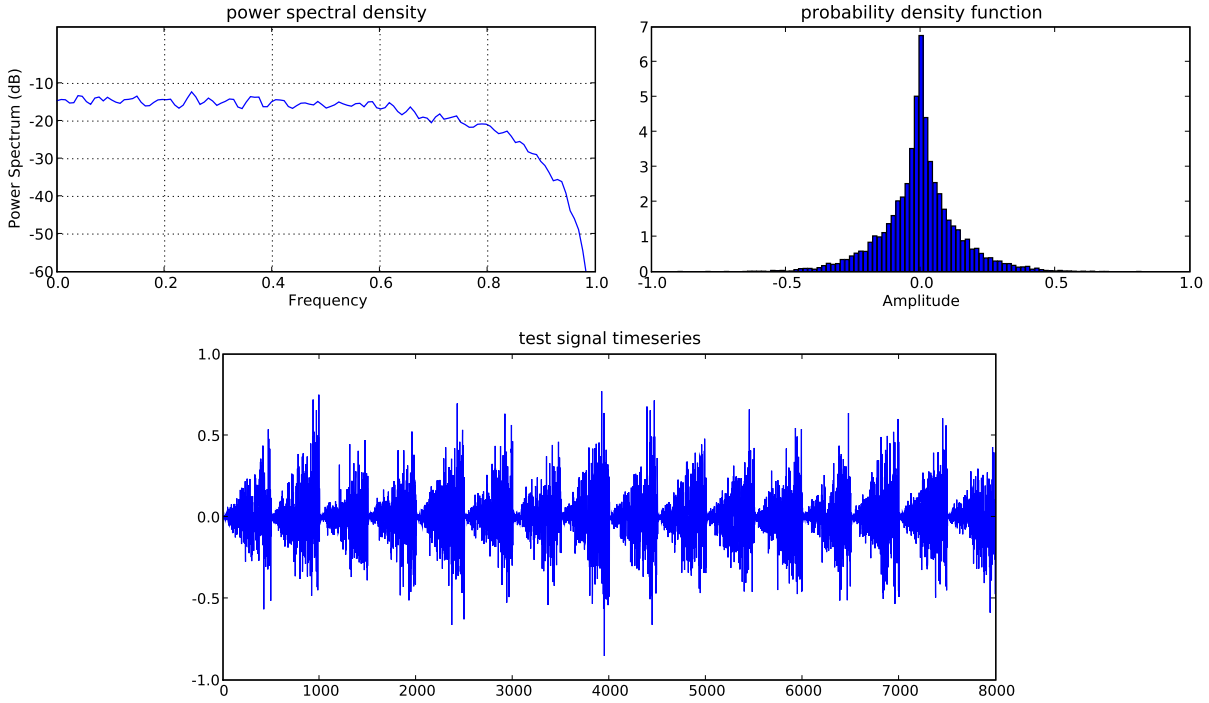

Figure 6.2: Input training signal for the nonlinear system identification task. It consists of linear sweeps of Gaussian white noise, processed with an additional low-pass filter. The first plot (from top left) shows the spectrum of the signal, then the probability density function of amplitudes is presented and the third picture illustrates the timeseries, with linear fades of 500 samples.

6.1.2 Identification Examples

In the first identification example the audio signal is a solo flute, discretized with a sampling rate of 44100 Hz. It was upsampled by a factor $L = 5$, then processed by the "Valve Saturation" LADSPA plugin (Harris [2002–]) and finally downsampled again by the same factor. The plugin has two adjustable parameters, which were set to *Distortion level = 0.8* and *Distortion character = 0.5*. After analyzing the implementation, the system can be divided into a static nonlinearity and a simple first order recursive linear filter. First the nonlinearity is applied to the input signal $x(n)$

$$z(n) = \frac{x(n) - q}{1 - e^{-d(x(n)-q)}} + \frac{q}{1 - e^{dq}}, \quad x(n) \neq 0, \quad q \neq 0 \tag{6.1}$$

[3]These signals worked much better than swept sines, for the ESN and the volterra system. It was also important to use Gaussian noise, uniformly distributed noise performed much worse.

where d and q can be calculated from the plugin parameters and were set to $q = -0.198$ and $d = 20.1$. Afterwards $z(n)$ is processed by a simple recursive filter producing the output $y(n)$

$$y(n) = 0.999y(n-1) + z(n) - z(n-1). \tag{6.2}$$

The second example is a more complex sound including many instruments and a singer, also with a sampling rate of 44100 Hz. This signal was processed with an oversampling factor $L = 10$ by the "TAP TubeWarmth" LADSPA plugin (Szilagyi [2004–]), which also has two adjustable parameters and they were set to *Drive* = *8* and *Tape–Tube Blend* = *8*.
In the plugin the input signal $x(n)$ is again first processed by a static nonlinearity

$$\begin{aligned} z(n) &= a\left(\sqrt{|\, b_{1,1} + x(n)(b_{1,2} - x(n))\,|} + b_{1,3}\right), \qquad for \quad x(n) \geq 0 \\ z(n) &= -a\left(\sqrt{|\, b_{2,1} - x(n)(b_{2,2} + x(n))\,|} + b_{2,3}\right), \qquad for \quad x(n) < 0 \end{aligned} \tag{6.3}$$

with the parameters

$$\begin{aligned} a &= 0.53050757809152127 \\ b_{1,1} &= 0.18958565330651478 \\ b_{1,2} &= 2.8708286933869704 \\ b_{1,3} &= -0.43541434669348522 \\ b_{2,1} &= 16.946763852588951 \\ b_{2,2} &= 27.142380373840449 \\ b_{2,3} &= -4.1166447323747715 \end{aligned}$$

and afterwards a linear recursive filter is applied to get the output $y(n)$

$$y(n) = \frac{0.1 f_r}{0.1 f_r + 1}\left(y(n-1) + z(n) - z(n-1)\right) \tag{6.4}$$

where f_r is the sampling rate of the audio signal.

6.1.3 System Setups and Results

For both experiments an ESN setup as presented in table 6.1 was used. The network was first driven by 3000 samples of the training signal from figure 6.2 and afterwards output weights were calculated from the next 5000 samples. Input, output target and the ESN output signal for the first identification example are shown in figure 6.3 and the corresponding spectrogram in figure 6.4. Results from the second identification example are illustrated in figure 6.5, where the ESN output signal is compared to the target signal, and in figure 6.6, which shows the probability density functions of audio input, target and ESN output signals. The generalization normalized root mean square error ($NRMSE_{test}$), calculated from 100000 time steps, for networks with and without a delay&sum readout and both identification examples are presented in table 6.2.

As a comparison the same two examples were also identified using a volterra system. A third-order discrete time volterra series can be written (Schetzen [1980]) as

$$\begin{aligned} y(n) = h_0 &+ \sum_{m_1=0}^{N_1-1} h_1(m_1)x(n-m_1) + \sum_{m_1=0}^{N_2-1}\sum_{m_2=0}^{N_2-1} h_2(m_1,m_2)x(n-m_1)x(n-m_2) \\ &+ \sum_{m_1=0}^{N_3-1}\sum_{m_2=0}^{N_3-1}\sum_{m_3=0}^{N_3-1} h_3(m_1,m_2,m_3)x(n-m_1)x(n-m_2)x(n-m_3) \end{aligned} \tag{6.5}$$

reservoir neurons	100
reservoir connectivity	0.2
spectral radius	0.85
input weights	random between $[-4, 4]$
reservoir activation function	tanh
output activation function	linear
simulation algorithm	additional squared states as in section 2.4
reservoir neuron type	standard neurons, no filter
noise during training	no
noise during testing	no

Table 6.1: ESN setup for the tube amplifier identification task.

where N_1, N_2, N_3 specify the memory depth of the volterra kernels h_1, h_2, h_3 and the lowest generalization error was achieved with the parameters

$$\begin{aligned} N_1 &= 100 \\ N_2 &= 40 \\ N_3 &= 5\ . \end{aligned}$$

The system was trained using a least squares approach (Schetzen [1980]) using 10000 samples of the training signal from figure 6.2, afterwards it was driven by the audio signal and the generalization error, calculated from 100000 samples, is presented in table 6.2. No special techniques for complexity reduction (see for instance Schetzen [1980]) were applied and therefore it was not possible to compute the audio output in reasonable time when using volterra systems with a much bigger memory[4].

Identification Example	$NRMSE_{test}$ ESN	$NRMSE_{test}$ D&S ESN	$NRMSE_{test}$ Volterra
Valve Saturation (1)	0.1522	0.0997	0.1277
TAP TubeWarmth (2)	0.1594	0.1372	0.2353

Table 6.2: Performance of the nonlinear system identification task of two tube amplifiers. First row shows test errors for the *Valve Saturation* LADSPA plugin (equations 6.1 and 6.2), second row for the *TAP TubeWarmth* plugin (equations 6.3 and 6.4), always calculated from 100000 samples of an audio signal. The system was identified with a standard echo state network, a delay&sum echo state network and a volterra system, using a training signal as shown in figure 6.2.

The performance of echo state networks was very constant, also when changing the system parameters from table 6.1. In general it did not matter at all which exact values were chosen and it was easy to get good results. Additional squared state updates from section 2.4 have usually been helpful in nonlinear system identification tasks, as already demonstrated in section 5.3. Moreover it is not clear why a delay&sum readout is able to boost the performance, because both systems are only first order recursive linear filters (see equations 6.2 and 6.4) and don't require long-term dependencies.
All in all one can say that when using relatively small echo state networks the generalization

[4]For these settings it took approximately one and a half day to simulate 100000 audio samples on recent hardware. In contrast to that, ESNs master this task in a few seconds.

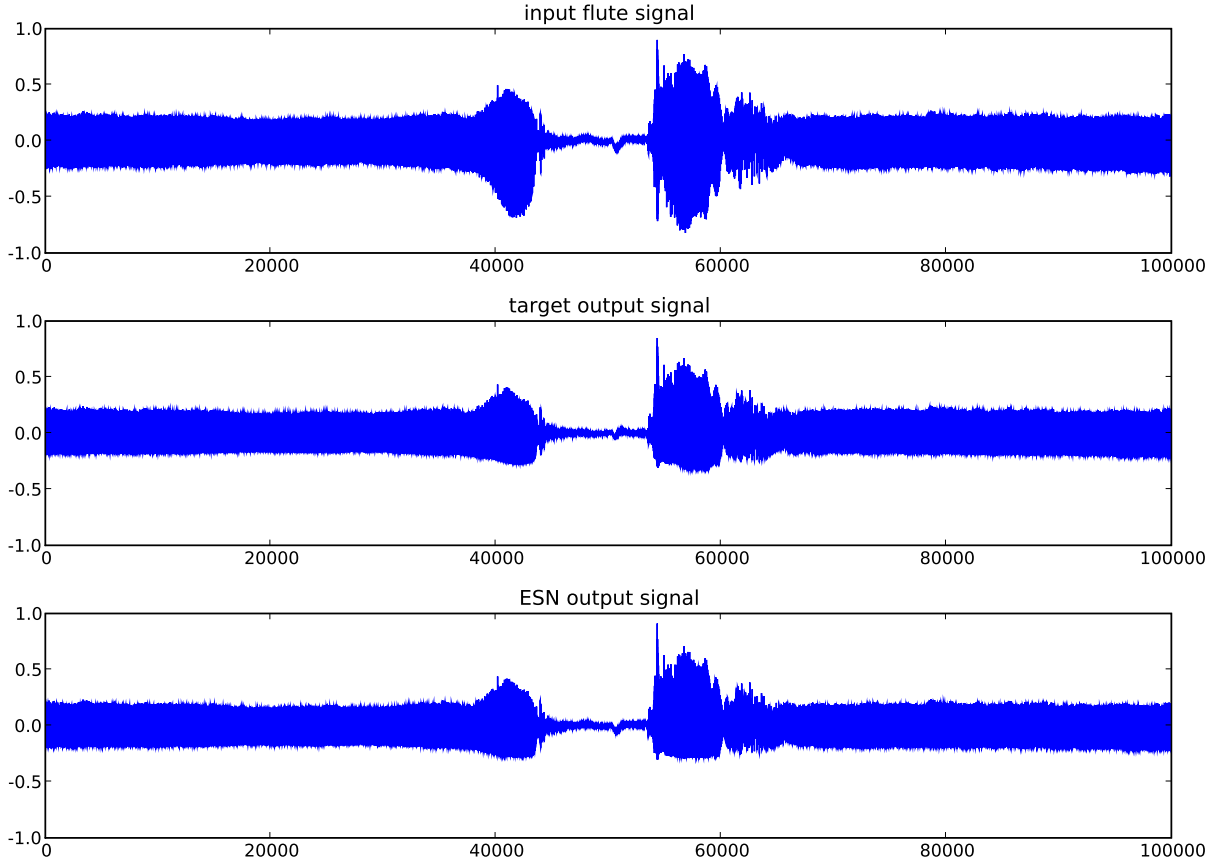

Figure 6.3: Audio input, target output and ESN output signal for the first identification example using the *Valve Saturation* LADSPA plugin (equations 6.3 and 6.4). The input signal is a flute sound with a small break at samples 45000 to 55000. One can see that the nonlinear system produces an asymmetric output signal.

error in nonlinear system identification is comparable or lower as with volterra systems and ESNs could be used for many applications where such tasks are required. Especially for higher order systems, as the NARMA system of section 5.3, volterra series will soon have practical limitations. Echo state networks are quite easy to use, robust against parameter changes and have a much lower computational complexity than higher order volterra systems without special structures for complexity reduction.

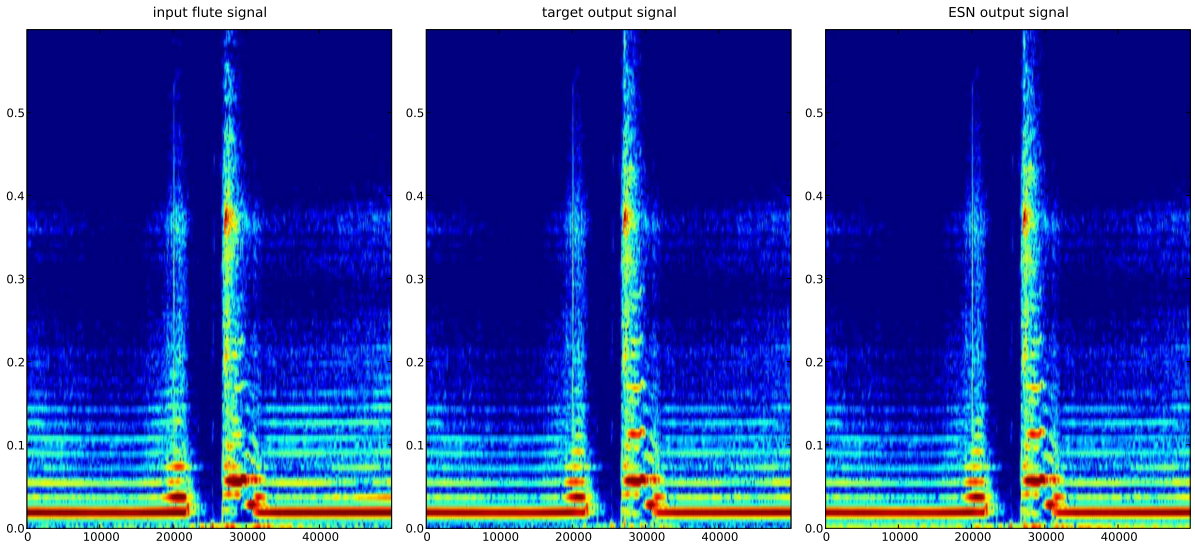

Figure 6.4: Spectrogram of input, target output and ESN output signal for the first identification example using the *Valve Saturation* LADSPA plugin, see also figure 6.3 for the corresponding time series. The nonlinear system emphasizes some partials and some are attenuated.

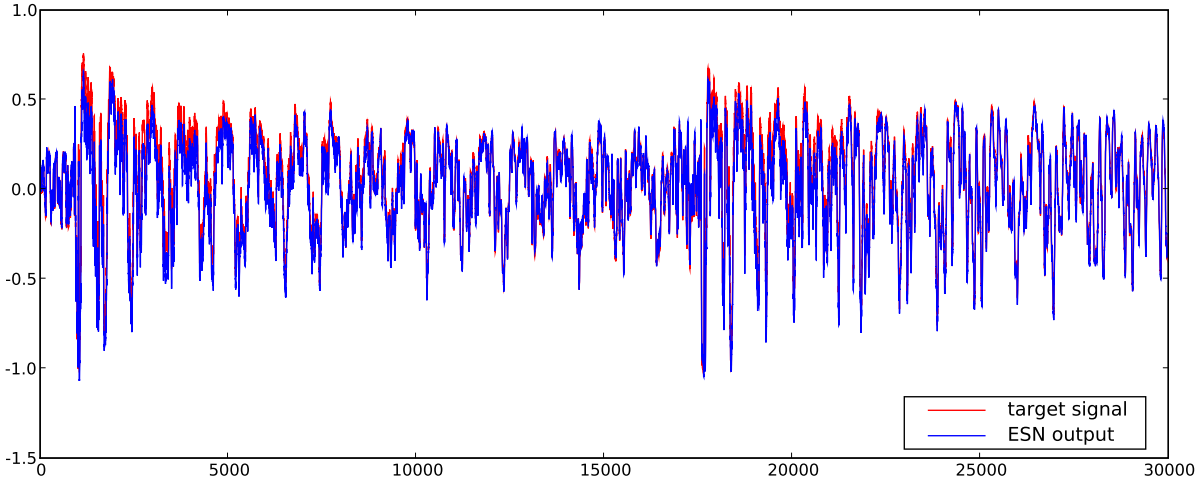

Figure 6.5: Target output and ESN output signal for the second identification example using the *TAP TubeWarmth* LADSPA plugin (equations 6.3 and 6.4). The input signal is a more complex sound including many instruments and a singer, plotted for 30000 time steps.

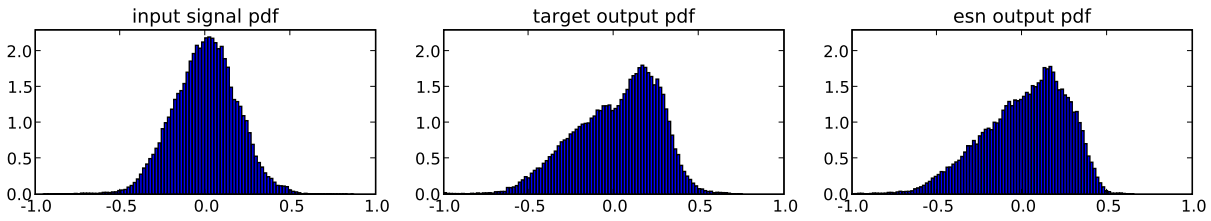

Figure 6.6: Probability density functions of the audio input, target output and ESN output signal for the second identification example using the *TAP TubeWarmth* LADSPA plugin, averaged over 100000 samples (see also figure 6.5 for a part of the time series). The audio input has approximately a Gaussian distribution, whereas after the nonlinearity the signals show a slant to the right, which corresponds to a non-zero skewness.

6.2 Nonlinear Audio Prediction

In audio prediction one tries to forecast future samples out of a given history horizon. Such methods are necessary for instance in audio restoration, whenever a sequence of consecutive samples is missing, when impulsive noise appears, or in the wireless transmission of digital signals where short dropouts can occur. Common methods try to model missing data with an autoregressive process (Godsill and Rayner [1997]), with the main drawback of a big model order when filling long dropouts, or use several forms of statistical prediction (Vaseghi [1996]). These linear models have advantages in implementation and interpretation, but they have serious limitations in that they cannot capture nonlinear relationships in the data which are common in many complex real-world problems (Uncini and Cocchi [2002]).
An alternative technique are pattern matching algorithms (see for instance Goodman et al. [1986], Wasem et al. [1988] or Niedzwiecki and Cisowski [2001]), where a signal template, usually just before the dropout, is compared to areas in the past of the audio signal (the search window). Now the area within this search window with the highest similarity to the audio template is selected and copied into the dropout region.

6.2.1 Algorithms and Audio Examples

In this section the prediction performance of echo state networks is compared to the work of Ausserlechner [2006], who evaluated different dropout concealment techniques based on pattern matching in subjective listening tests, and to a linear autoregressive model. Ausserlechner [2006] assumed short dropouts with a length of 2 to 6 milliseconds in a wireless signal transmission, therefore only past samples could be used to forecast the signal development. The exact number and length of dropouts is shown in table 6.3. After each dropout, the signal is predicted for further 200 samples and crossfaded with the original one to avoid discontinuities at the transition points (see for instance figure 6.8 or 6.9).
Two different audio examples were considered, both with a duration of five seconds and a sampling rate of 44100 Hz. The first one is a short recording of a jazz quartet and the second an orchestra composition by Beethoven, a short extract of both is shown in figure 6.7.

length (ms)	length (samples)	occurrences
2	88	9
3	132	6
4	176	5
5	220	4
6	264	3

Table 6.3: Dropout distribution in listening test three from Ausserlechner [2006]. The location was randomly chosen within an audio file with a duration of five seconds and a sampling rate of 44100 Hz, resulting in two percentage of corrupted audio.

Ausserlechner [2006] evaluated the performance of three pattern matching algorithms. The first one is based on waveform differences as defined in [Goodman et al., 1986, equation 6] and will be called *PatMat* in the rest of this section. A second algorithm compares the zero crossing rate (ZCR) of the audio template to areas in a search window and incorporates this information to select optimal patterns, for a detailed description see [Ausserlechner, 2006, section 2.3]. It will be called *ZCR* in this comparison. Finally a third technique uses the YIN-algorithm (de Cheveigné and Kawahara [2002]) for pitch detection, which consists of a combination of

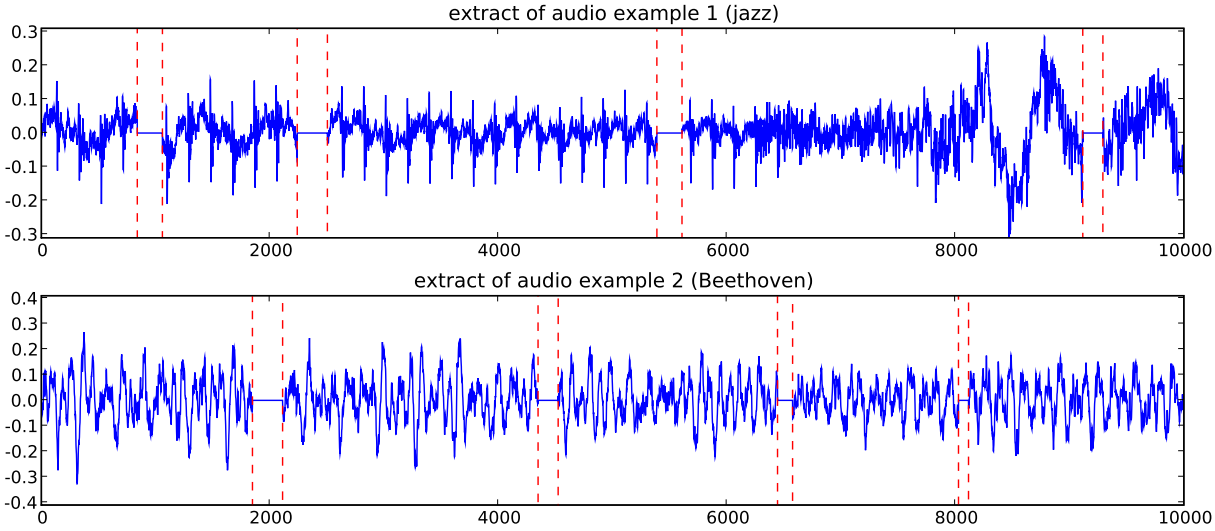

Figure 6.7: 10000 samples extract from two audio examples studied in Ausserlechner [2006]. The first one is a jazz quartet, the second an orchestra composition by Beethoven. Dropouts, generated from a distribution as presented in table 6.3, are marked with dotted red lines.

the autocorrelation and the average magnitude difference function (AMDF), and this additional knowledge is considered to detect optimal patterns (see [Ausserlechner, 2006, section 2.4]). Here this algorithm is denoted as *YIN*.

Additionally to pattern matching algorithms, which use some form of expert knowledge to detect optimal patterns, linear autoregressive models, as for instance described in Godsill and Rayner [1997], were considered. In a linear autoregressive interpolation the current signal value is represented as a weighted sum of P previous values

$$y(n) = \sum_{i=1}^{P} y(n-i)a_i + e(n) \tag{6.6}$$

where P is called the model order, a_i are the coefficients of a linear filter and $e(n)$ is an additional error term. The model order should be fixed to a value high enough, so that complex signals can be represented, here P was set to $P = 300$. Furthermore no audio signal is truly stationary, therefore it is necessary to train the model from a relatively short block of samples and a block length of 3000 samples was found to be optimal. Finally filter coefficients a_i are estimated with a least squares approach from those 3000 samples, for more details see [Godsill and Rayner, 1997, section 3.3.1].

For both audio examples a standard echo state network and an ESN with filter neurons and a delay&sum readout was trained, exact setups are presented in table 6.4 and table 6.5. The training size is in general the most critical parameter in this task, because audio signals are not stationary. Standard ESNs were first driven by 500 samples and afterwards output weights were calculated from the next 1000 steps. For D&S ESNs it was possible to use a washout time of 3000 samples and the readout was trained from the next 3000 time steps. Note that for standard ESNs the error was lower if the same amount of noise was added in training and testing, otherwise the output signal energy always decreased and the audible result was much worse.

reservoir neurons	100
reservoir connectivity	0.2
spectral radius	0.98
output-feedback weights	random between $[-1, 1]$
reservoir activation function	tanh
output activation function	linear
delay&sum readout	no
noise during training	uniformly distributed between $[-0.0001, 0.0001]$
noise during testing	uniformly distributed between $[-0.0001, 0.0001]$
washout samples	500
training samples	1000

Table 6.4: Standard ESN setup for the audio prediction task. Note that noise is added during training and testing.

reservoir neurons	100
reservoir connectivity	0.2
spectral radius	0.8
output-feedback weights	random between $[-0.5, 0.5]$
reservoir activation function	tanh
output activation function	linear
delay&sum readout	yes, maximum delay was restricted to 1000 samples
reservoir neurons filter type	band-pass
filter spacing	logarithmically between 60 Hz and 11000 Hz
filter bandwidth	2 octaves
noise during training	uniformly distributed between $[-0.0002, 0.0002]$
noise during testing	no
washout samples	3000
training samples	3000

Table 6.5: Delay&Sum ESN setup for the audio prediction task. The sampling rate of the audio signals was 44100Hz.

6.2.2 Evaluation

The performance of all algorithms was benchmarked by calculating the $NRMSE$ between model output and original signals only from data in the dropout regions and results for both audio examples are presented in table 6.6. One can see that standard ESNs are not able to produce good predictions, similar as in the multiple superimposed oscillations task from section 5.1, even when using much bigger reservoirs or more training data. ESNs with filter neurons and a delay&sum readout showed in general a slightly better performance than all other methods, especially in the second audio example. Predictions from a delay&sum ESN, a standard ESN, a linear autoregressive model and from the *PatMat* algorithm are shown in figure 6.8 and figure 6.9.

Unfortunately it was not possible in the scope of this thesis to make comparable subjective listening tests as in Ausserlechner [2006], where 21 persons evaluated the quality of the three pattern matching algorithms. However, the results there correspond more or less to the $NRMSE$ values from table 6.6. Test persons evaluated the *PatMat* algorithm in general better as *ZCR* and *YIN*, especially in the first audio example (jazz quartet) *PatMat* outperformed the others,

which can be also seen from table 6.6. In a quick personal listening test no major differences were perceived between results of the *PatMat*, the autoregressive model and the delay&sum ESN, whereas a standard ESN produced hearable annoying artifacts.

algorithm	$NRMSE$ example 1 (jazz)	$NRMSE$ example 2 (Beethoven)
YIN	1.0126	1.1663
ZCR	0.9275	0.9232
PatMat	0.7336	0.8729
AR-model	0.6708	0.8492
delay&sum ESN	0.6010	0.5358
standard ESN	1.0851	1.4055

Table 6.6: $NRMSE$ values calculated from the dropout regions of both audio examples. For a description of the algorithms see text. Standard ESNs were not able to produce good predictions, but ESNs with filter neurons and a delay&sum readout showed in general a slightly better performance than all other methods.

Finally some general notes on the parameter selection for the audio prediction task are presented:

- *Training and washout size*:
  This is the most important parameter in this task, for ESNs and also the autoregressive model. Audio signals are not stationary and therefore the training size should be relatively small, when using too many samples for training the output becomes very smooth and high frequencies are missing.
- *Filter parameters*:
  The range should contain all necessary frequencies important for the task. Logarithmically spaced band-pass filters from 60 Hz to 11000 Hz with a bandwidth of 2 octaves were found to be optimal.
- *Noise*:
  Adding a small noise term in the state update equation as regularization is necessary for stability when ESNs are trained as generators. The amount of noise was chosen so that no more unstable networks were produced in simulations.
- *Delay&Sum readout*:
  Delays were restricted to a maximum of 1000 samples, because the training size should be kept small. The EM-based learning algorithm from section 4.4 performed a little bit better as the simple method from section 4.3, both used the cross correlation for delay estimation.
- *Spectral radius, feedback weights and reservoir connectivity*:
  In general the exact values of those system parameters did not matter and it was easy to get good results.

This section evaluated the performance of echo state networks in a short-term prediction task up to 300 samples, as it is necessary when dropouts in a wireless signal transmission occur. It was shown that ESNs with a delay&sum readout and filter neurons are able to outperform other techniques, which sometimes even use additional knowledge like pitch information or zero crossing rates. Furthermore it would be especially interesting to implement a similar comparison for long-term prediction tasks, as they are necessary in audio restoration, where a delay&sum readout should be able to incorporate long-term dependencies and could give good results.

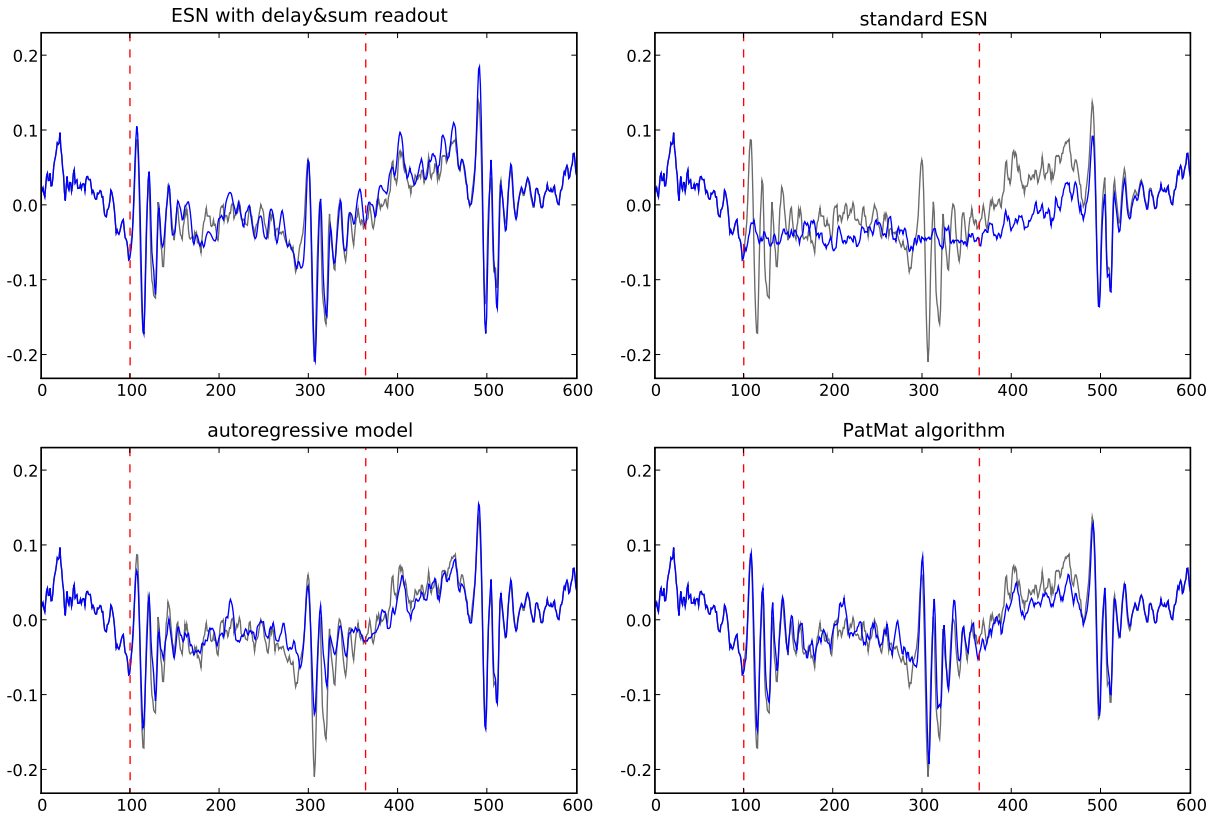

Figure 6.8: Dropout number 7 of the first audio example (jazz quartet). The beginning and ending is marked with dotted red lines, after the dropout the signal is predicted for further 200 samples and crossfaded with the original one to avoid discontinuities at the transition points. This figure presents results from a delay&sum ESN, a standard ESN, a linear autoregressive model and from the pattern matching algorithm *PatMat*. The original signal is plotted in gray, the predicted signal in blue.

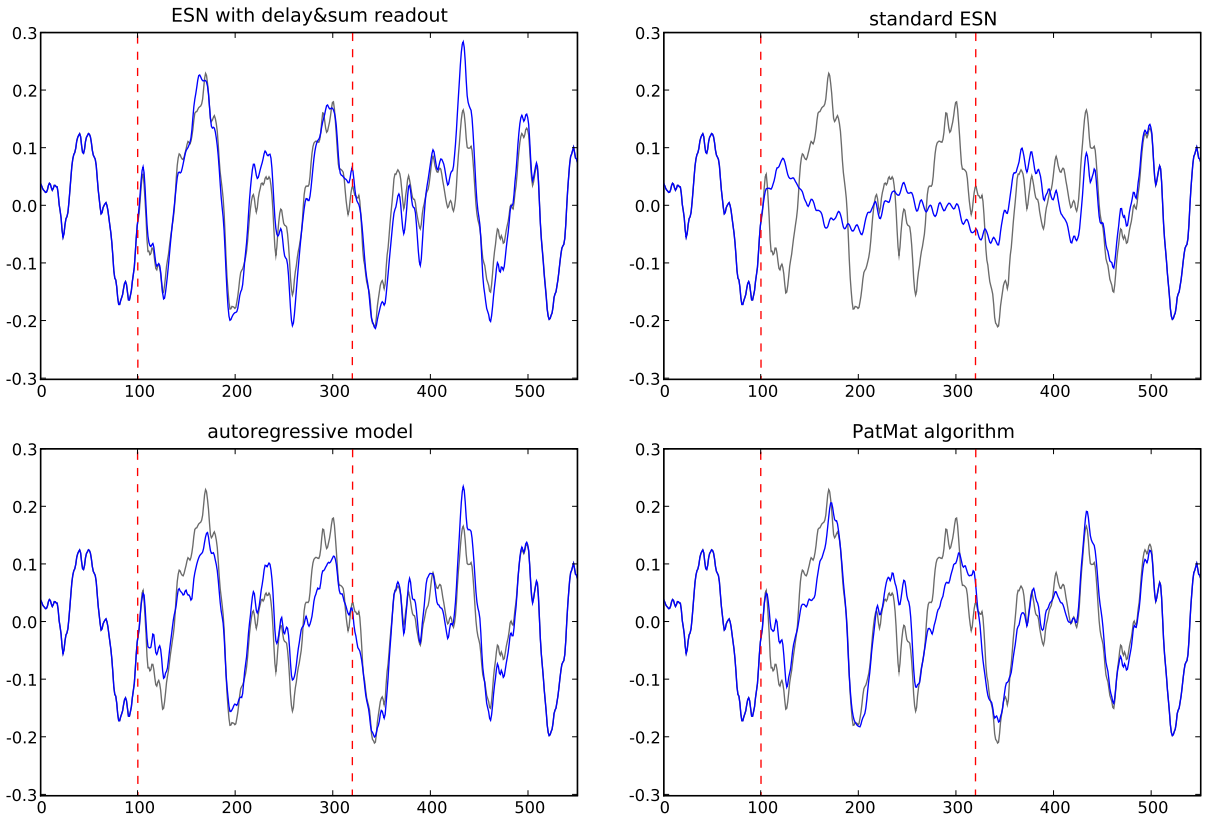

Figure 6.9: Dropout number 18 of the second audio example (Beethoven). The beginning and ending is marked with dotted red lines, after the dropout the signal is predicted for further 200 samples and crossfaded with the original one to avoid discontinuities at the transition points. The original signal is plotted in gray, the predicted signal in blue.

Chapter 7

Conclusion

This master thesis analyzed two enhancements of echo state networks: general IIR filter neurons in the reservoir and a delay&sum readout. Both methods were able to significantly boost the performance in some tasks, especially when using real-world, complex data. Simulations with synthetic signals and audio samples demonstrated the usefulness of the developed techniques.

With filter neurons in the reservoir, see chapter 3, it is possible to model multiple attractors with one single network. Real-world audio signals mostly consist of multiple sources, therefore filter neurons are mandatory in tasks like nonlinear audio prediction (section 6.2). When using filters, "specialized" neurons are tuned to different frequencies and more diverse echoes can evolve in the reservoir because not all units are coupled.

The delay&sum readout, see chapter 4, adds trainable delays in the synaptic connections of readout neurons and boosts the performance of ESNs in general. It vastly increases the short term memory capability of echo state networks and is an efficient approach to handle long-term dependencies. Simulations in chapter 5 showed that such a readout is especially useful for sparse nonlinear system identification with long-term dependencies or chaotic timeseries prediction tasks.

This work has demonstrated that echo state networks, or in general reservoir computing techniques, could be used as an additional tool in many signal processing or classification applications, where nonlinear relationships are present in the data. Classical approaches for nonlinear signal processing consists in general of designing specific algorithms for specific problems, whereas echo state networks with the proposed extensions can be trained and adapted to many applications. The following section will give further practical hints on how to use such systems and finally ideas for further research are presented in section 7.2.

7.1 Practical Hints

This section tries to give some practical hints, when and how one should use echo state networks with a delay&sum readout and filter neurons. First suggestions for system identification tasks are presented and afterwards some general tips for ESNs trained as generators are given. One should also consider the tutorial [Jaeger, 2002b, chapter 9], which contains many useful hints for standard echo state networks.

In nonlinear system identification tasks, where ESNs are utilized as nonlinear filters (see simulations from sections 5.3 and 6.1), the following suggestions can be given:

- The bigger the networks, the more complex systems can be modelled. However, also the training size should increase with network size.
- No output feedback connections are necessary, therefore stability is ensured if the echo state property from section 2.2 is fulfilled.
- Additional squared state updates, as defined in section 2.4, are most of the time helpful when identifying strong nonlinearities.
- Filter neurons are usually counterproductive in identification tasks. The networks become sometimes unstable.
- A delay&sum readout is able to boost the network performance in most of the simulations, even when no obvious long-term dependencies are present as for instance in the examples of section 6.1. Mostly the EM-based learning algorithm (section 4.4), using the cross correlation method for delay estimation, gives slightly better results as the simple training algorithm (section 4.3), where the GCC method for delay estimation should be used. With an EM-based algorithm the delays usually converge after two to five iterations.
- In general it is not necessary to add a noise term $v(n)$ in the state update equation 2.1 during training. Nevertheless, sometimes it is possible to improve the generalization performance with a small $v(n)$, because the output weights will become smaller.
- An additional bias input signal can be helpful when signals with non-zero means are considered.
- All other parameters (spectral radius, input weights, connectivity of the reservoir, ...) where not that important, often it did not matter at all which values were chosen and it was easy to get good results. More detailed hints for those parameters can be found in [Jaeger, 2002b, chapter 9].

Similar suggestions can be presented for tasks, where echo state networks are trained as generators (see simulations from sections 5.1, 5.2 and 6.2):

- Output feedbacks are mandatory, otherwise the network is not able to produce autonomous patterns.
- Echo state networks with output feedbacks can become unstable, therefore some kind of regularization is necessary to keep the readout weights small, for instance Tikhonov regularization (section 2.3.1) or adding a small noise term $v(n)$ in the state update equation 2.1 during training. The amount of noise should be chosen so that no more unstable networks are produced in simulations.
- When trying to learn multiple attractors/oscillators with a single network, filter neurons in the reservoir are mandatory. The frequency range, bandwidth and spacing of the used filters should be adapted to the target signal, so that all important frequencies can be represented. In general the filter bandwidth should not be too narrow, so that overlapping regions exist in the reservoir (see for instance figure 3.2).
- The bigger the reservoirs, the more independent attractors/oscillators can be modelled with one single network.

- A delay&sum readout was able to boost the network performance in all generator tasks studied in this thesis. Most of the time the simple learning algorithm (section 4.3) and the EM-based training (section 4.4) perform quite similar, both should use the cross correlation method for delay estimation. With an EM-based algorithm and ESNs with output feedback the delays won't converge as fast as without feedback, but good results are achieved when for instance performing ten EM iterations for delay calculation.
- An additional bias input is helpful when signals with non-zero means should be modelled.

7.2 Ideas for Future Work

Further research in the area of filter neurons and delay&sum readouts could go in the following directions:

- It should be possible to improve the quite simple delay learning algorithms from chapter 4. One way could be to use more advanced multipath time delay estimation methods, as for instance outlined in Y.Huang et al. [2006].
- An online version of the delay learning algorithm should be developed. This could be done with an EM-based approach similar to Frenkel and Feder [1999], or as in Duro and Reyes [1999], where the backpropagation algorithm was extended for feedforward neural networks with weights and delays in their synaptic connections.
- Adding delays in reservoir connections might be another possibility to improve the modeling power of ESNs. This feature is already implemented in the developed software library (see appendix A), but was not further studied. Such delays could be fixed or pre-adapted to a specific task with a similar method as in Bone et al. [2000] or Bone et al. [2002].
- All filter parameters could be automatically pre-adapted to a specific task, then no more manual tuning would be necessary. This can be implemented with a method inspired by intrinsic plasticity (IP), as presented in Verstraeten et al. [2007b], see also Lukosecicius and Jaeger [2007] for an overview of various reservoir adaptation techniques.
- Further filter structures should be considered and analyzed.

Moreover the following audio signal processing tasks, which were not studied in this work due to lack of time, would be interesting real-world examples:

- A similar comparison as in section 6.2 should be implemented for long-term audio prediction tasks, as they are necessary for instance in audio restoration. The delay&sum readout would be able to incorporate long-term dependencies and should give good results.
- With an adaptive delay learning algorithm the model would be well suited for nonlinear echo cancellation (Stenger and Rabenstein [1999]) or beamforming (Knecht [1994]) applications, where a sparse system representation is possible.
- Many classification tasks in music information retrieval could benefit from reservoir computing techniques. Some similar applications were already developed, for instance a composer and instrument classification using the liquid state machine (Pape et al. [2007]), some publications about speech recognition (for example Skowronski and Harris [2007] or Verstraeten et al. [2005]) or melody generating ESNs (Jaeger and Eck [2008]).

- Echo state networks could be used as audio and speech synthesizers. Preliminary experiments of an ESN-based audio synthesis were already made with sine signals as inputs, which were generated automatically by tracking the pitch and volume of a desired output sound. Afterwards the network was trained with the generated sine as input to the desired output samples. Echo state networks were able to add the other frequency and noise components present in the target spectrum, according to the volume and frequency of the input sine.

- More complex sound modeling techniques could be implemented with hierarchical layers of echo state networks, similar to Jaeger [2007b]. The different layers could run at multiple rates, where each level adds more details to the signal. Such a hierarchical model would also correspond to the structure of music itself, which is build out of information on different time scales (samples, sample blocks, tones, phrases, musical form, et cetera).
  The features of each layer could be extracted in an unsupervised way, as done in Jaeger [2007b] where the relevant features are discovered automatically, or in a supervised manner as shown in figure 7.1 (using for instance LPC, MFCC coefficients, pitch, spectral or temporal features).

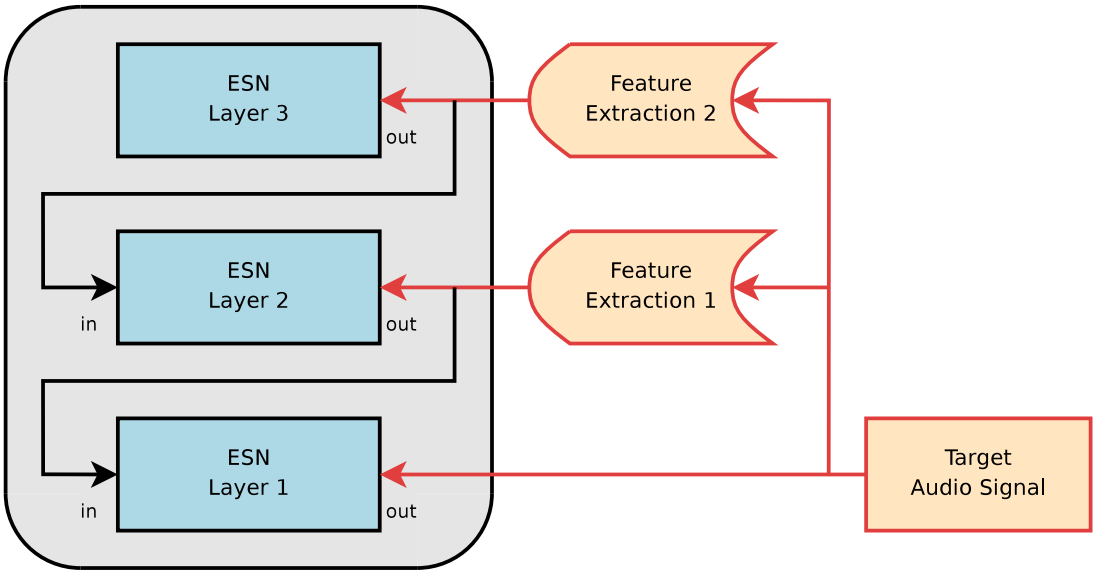

Figure 7.1: Hierarchical ESN model with three layers and supervised feature extraction, red lines and objects are used in training only (teacher forcing). In a top-down flow of information each level runs at a faster rate and adds more details to the signal. After training, control input values could be fed into an arbitrary layer, using all lower levels as synthesizer.

Appendix A

aureservoir - Analog Reservoir Computing C++ Library

Aureservoir is an open source (L-GPL) library for analog reservoir computing neural networks, source code and documentation is located at the sourceforge project page http://aureservoir.sourceforge.net/. Its main focus is on speed and a clear object oriented design, moreover it can be integrated in various scientific online and offline computation environments, in standalone programs, embedded systems et cetera. The library is free to use and no dependency on *Matlab* or other license fees are required.

A.1 Library Design

Aureservoir was designed with the use of object oriented and generic programming design patterns. Computation is possible with different floating point precisions (float, double, long double) and all methods for initialization, training, simulation, reservoir adaptation, activation functions and additional parameters are exchangeable at runtime. These algorithms are implemented as function objects, therefore it is easy to add new ones by just deriving from the appropriate base class. Furthermore a detailed doxygen[1] documentation was generated, which can be found at the sourceforge project page.

The algorithms are extensively tested. A comprehensive unit test suite was developed, which recalculates all algorithms in *python* and compares the results with the C++ implementation. This is done with different random starting states and network parameters, to test many possible combinations.

A.2 Performance

Special importance was given to performance, because the initial goal was to use the library for realtime audio applications. A chooseable floating point precision, usually single precision is sufficient for audio processing, speeds up the computation about a factor of two and uses less memory compared to a double precision implementation. Furthermore benchmarks with a number of sparse matrix computation libraries were executed in Holzmann [2007] to get the most efficient available solution.

[1] Doxygen is a documentation generator for C++ and many other languages. The project page is located at http://www.doxygen.org.

The winner of this benchmark was a library called *FLENS*[2], its design is presented in Lehn [2008]. *FLENS* is mainly a nice C++ interface to BLAS[3] (BLAS [2008]) and LAPACK[4] (Anderson et al. [1999]) with additional optimized sparse algorithms. It avoids virtual function calls by building a class hierarchy with static polymorphism (curiously recurring template pattern - Alexandrescu [2001]) and therefore achieves code reusability without sacrificing efficiency. Furthermore high-level notation and operator overloading can be used without temporal objects, allowed by a combination of closures (Stroustrup [1986]) and expression templates (Veldhuizen [1995]), therefore *FLENS* is basically as fast as direct Fortran BLAS calls.

A.3 Current Features

All listed features are a subject of change and represent the status of the time of this writing. So far bindings to *python* (*NumPy*/*SciPy*[5] - see Oliphant [2007] and Jones et al. [2001–]) and initial bindings to *Pure Data*[6] are provided.

Implemented simulation algorithms:

- standard simulation algorithm as in Jaeger [2001]
- simulation algorithm with leaky integrator neurons from Jaeger et al. [2007b]
- algorithm with bandpass style neurons as introduced in Wustlich and Siewert [2007]
- simulation algorithm with general IIR-filter neurons from chapter 3
- algorithm with IIR-filter before neurons nonlinearity
- ESNs with an delay&sum readout from chapter 4
- simulation algorithm with additional squared state updates as in Jaeger [2003]

Implemented training algorithms:

- offline training algorithm using the pseudo inverse as in equation 2.3
- training algorithm using the Wiener-Hopf solution from equation 2.4
- algorithm with Tikhonov regularization as in equation 2.5
- simple delay learning algorithm from section 4.3
- EM-based delay learning algorithm from section 4.4

Miscellaneous:

[2] *FLENS*: Flexible Library for Efficient Numerical Solutions, project page at http://flens.sourceforge.net/

[3] BLAS (Basic Linear Algebra Subprograms - BLAS [2008]) are highly optimized and tested matrix/vector algorithms and are used as the building block of all main scientific computation packages.

[4] LAPACK (Linear Algebra Package - Anderson et al. [1999]) uses BLAS to calculate standard linear algebra problems.

[5] *SciPy* (Jones et al. [2001–]) is an open source library for scientific computation in *python*. The *SciPy* library depends on *NumPy* (Oliphant [2007]), which provides convenient and fast N-dimensional array manipulation.

[6] *Pure Data* (aka *PD*) is a real-time graphical programming environment for audio, video, and graphical processing. The community portal is located at http://puredata.info.

- Gaussian-IP reservoir adaptation method for tanh neurons from Verstraeten et al. [2007b]
- ArrayESN class, where the output of multiple ESNs is averaged to boost the performance as described in Jaeger and Hass [2004]
- learning algorithm with relaxation stages as introduced in the appendix of Jaeger and Hass [2004]
- reservoir neurons with additional delays

All features are documented in detail at `http://aureservoir.sourceforge.net` and usage examples are included in the source package.

Bibliography

Jonathan Abel and David Berners [2006]. *A Technique for Nonlinear System Measurement.* AES Convention 121. (Cited on page 50.)

Andrei Alexandrescu [2001]. *Modern C++ Design.* Addison-Wesley. ISBN 0-201-70431-5. (Cited on page 68.)

E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen [1999]. *LAPACK Users' Guide.* ISBN 0-89871-447-8. http://www.netlib.org/lapack/lug/. (Cited on page 68.)

Keith Andrews [2006]. *Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science.* Graz University of Technology, Austria. http://ftp.iicm.edu/pub/keith/thesis/. (Cited on page xi.)

A. F. Atiya and A. G. Parlos [2000]. *New Results on Recurrent Network Training: Unifying the Algorithms and Accelerating Convergence.* IEEE-NN, 11(3), page 697. citeseer.ist.psu.edu/atiya00new.html. (Cited on pages 43 and 44.)

Hubert Ausserlechner [2006]. *Untersuchungen von Dropout-Concealment-Algorithmen.* Institute for Electronic Music and Acoustics, Graz, Austria. http://www.iem.at/projekte/dsp/untersuchungen/index_html. Diplomarbeit. (Cited on pages v, vii, xi, 57, 58 and 59.)

M. Azaria and D. Hertz [1986]. *Time delay estimation by generalized cross correlation methods.* Acoustics, Speech, and Signal Processing; IEEE Transactions on Signal Processing, 32, pages 380–285. (Cited on pages 24 and 26.)

BLAS [2008]. *Basic Linear Algebra Subprograms.* http://www.netlib.org/blas/. Online resource, accessed 22.5.2008. (Cited on page 68.)

R. Bone, M. Crucianu, and J.P. Asselin de Beauville [2000]. *Two constructive algorithms for improved time series processing with recurrent neural networks.* Neural Networks for Signal Processing, 1, pages 55–64. doi:10.1109/NNSP.2000.889362. (Cited on pages 29 and 65.)

R. Bone, M. Crucianu, and J.P. Asselin de Beauville [2002]. *Learning long-term dependencies by the selective addition of time-delayed connections to recurrent neural networks.* Neurocomputing, 48(1), pages 251–266. doi:10.1016/S0925-2312(01)00654-3. (Cited on pages 29 and 65.)

V. Bringuier, F. Chavane, L. Glaeser, and Y. Frégnac [1999]. *Horizontal propagation of visual activity in the synaptic integration field of area 17 neurons.* Science, 283, pages 695–699. doi:10.1126/science.283.5402.695. (Cited on pages 3 and 23.)

Robert Bristow-Johnson [2008]. *Cookbook formulae for audio EQ biquad filter coefficients.* http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt. Online resource, accessed 14.4.2008. (Cited on pages 20 and 21.)

Nello Cristianini and John Shawe-Taylor [2000]. *An Introduction to Support Vector Machines and other kernel-based learning methods.* Cambridge University Press. ISBN 0521780195. (Cited on page 2.)

Alain de Cheveigné and Hideki Kawahara [2002]. *YIN, a fundamental frequency estimator for speech and music.* Acoustical Society of America. doi:10.1121/1.1458024. (Cited on page 57.)

Peter F. Dominey [1995]. *Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning.* Journal Biological Cybernetics, 73(3), pages 265–274. doi:10.1007/BF00201428. (Cited on page 2.)

R.J. Duro and J.S. Reyes [1999]. *Discrete-time backpropagation for training synaptic delay-based artificial neural networks.* IEEE Transactions on Neural Networks, 10(4), pages 779–789. doi:10.1109/72.774220. (Cited on pages 29 and 65.)

B. Farhang-Boroujeny [1998]. *Adaptive Filters: Theory and Applications.* John Wiley & Sons, Inc., New York, NY, USA. ISBN 0471983373. (Cited on page 10.)

L. Feldkamp, D. Prokhorov, C. Eagen, and F. Yuan [1998]. *Enhanced multi-stream Kalman filter training for recurrent networks.* Nonlinear Modeling: Advanced Black-Box Techniques, pages 29–53. (Cited on pages 40, 41 and 44.)

L.A. Feldkamp and G.V. Puskorius [1998]. *A signal processing framework based on dynamic neural networks with application to problems in adaptation, filtering, and classification.* Proceedings of the IEEE, 86(11), pages 2259–2277. doi:10.1109/5.726790. (Cited on pages 3 and 49.)

J. Fessler and A. Hero [1994]. *Space-alternating generalized expectation-maximization algorithm.* IEEE Trans. Signal Processing, 42, pages 2664–2667. doi:10.1109/78.324732. (Cited on page 28.)

L. Frenkel and M. Feder [1999]. *Recursive expectation-maximization (EM) algorithms for time-varying parameters with applications to multiple target tracking.* IEEE Transactions on Signal Processing, 47, pages 306–320. ISSN 1053-587X. doi:10.1109/78.740104. (Cited on pages 29 and 65.)

Felix A. Gers, Douglas Eck, and Juergen Schmidhuber [2001]. *Applying LSTM to Time Series Predictable through Time-Window Approaches.* Lecture Notes in Computer Science, 2130, pages 669+. (Cited on page 40.)

Simon Godsill and Peter Rayner [1997]. *Digital Audio Restoration.* Springer. ISBN 978-3540762225. http://www-sigproc.eng.cam.ac.uk/storation.zip. (Cited on pages 4, 49, 57 and 58.)

D. Goodman, G. Lockhart, O. Wasern, and Wong Wai-Choong [1986]. *Waveform substitution techniques for recovering missing speech segments in packet voice communications.* Acoustics, Speech, and Signal Processing, IEEE Transactions, 34, pages 1440–1448. (Cited on pages 4 and 57.)

Steve Harris [2002–]. *Valve Saturation LADSPA Plugin*. http://plugin.org.uk/ladspa-swh/docs/ladspa-swh.html. Part of the plugin collection of Steve Harris, accessed 29.05.2008. (Cited on page 51.)

Simon Haykin [1996]. *Neural Networks Expand SP's Horizons*. 13, pages 24–49. doi:10.1109/79.487040. IEEE Signal Processing Magazine. (Cited on pages 1, 3 and 49.)

Thomas Helie [2006]. *On the use of volterra series for real-time simulations of weakly nonlinear analog audio devices: application to the moog ladder filter*. Proc. of the Int. Conf. on Digital Audio Effects (DAFx-06). http://www.dafx.ca/proceedings/papers/p_007.pdf. (Cited on pages 4, 49 and 50.)

Georg Holzmann [2007]. *Benchmark of C++ Libraries for Sparse Matrix Computation*. http://grh.mur.at/misc/SparseLibBenchmark.pdf. Accessed 22.5.2008. (Cited on page 67.)

John D. Hunter [2007]. *Matplotlib: A 2D Graphics Environment*. Computing in Science and Engg., 9(3), pages 90–95. ISSN 1521-9615. doi:10.1109/MCSE.2007.55. http://matplotlib.sourceforge.net/. (Cited on page xi.)

H. Jaeger, W. Maass, and J. Principe [2007a]. *Special issue on echo state networks and liquid state machines: Editorial*. Neural Netw., 20(3), pages 287–289. (Cited on page 2.)

Herbert Jaeger [2001]. *The "echo state" approach to analysing and training recurrent neural networks*. German National Research Center for Information Technology Bremen. http://www.faculty.iu-bremen.de/hjaeger/pubs/EchoStatesTechRep.pdf. Tech. Rep. No. 148. (Cited on pages iii, 1, 2, 5, 7, 8, 14, 35, 37, 39, 40 and 68.)

Herbert Jaeger [2002a]. *Short term memory in echo state networks*. German National Research Center for Information Technology Bremen. http://www.faculty.iu-bremen.de/hjaeger/pubs/STMEchoStatesTechRep.pdf. Tech. Rep. No. 152. (Cited on pages iii, 3, 11, 12, 13, 14 and 46.)

Herbert Jaeger [2002b]. *A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach*. German National Research Center for Information Technology Bremen. http://www.faculty.iu-bremen.de/hjaeger/pubs/ESNTutorialRev.pdf. Tech. Rep. No. 159. (Cited on pages 5, 7, 8, 11, 12, 63 and 64.)

Herbert Jaeger [2003]. *Adaptive nonlinear system identification with echo state networks*. Advances in Neural Information Processing Systems 15, MIT Press, pages 593–600. http://www.faculty.iu-bremen.de/hjaeger/pubs/esn_NIPS02.pdf. (Cited on pages iii, 7, 10, 11, 43, 44 and 68.)

Herbert Jaeger [2007a]. *Echo state network*. http://www.scholarpedia.org/article/Echo_state_network. Scholarpedia, accessed 8.4.2008. (Cited on pages iii, 5 and 9.)

Herbert Jaeger [2007b]. *Discovering multiscale dynamical features with hierarchical Echo State Networks*. International University Bremen. http://www.jacobs-university.de/imperia/md/content/groups/research/techreports/hierarchicalesn_techrep10.pdf. Tech. Rep. No. 10. (Cited on pages 10 and 66.)

Herbert Jaeger and Douglas Eck [2008]. *Can't Get You Out of My Head: A Connectionist Model of Cyclic Rehearsal*. Wachsmuth, G. Knoblich (eds.): Modeling Communication for Robots and Virtual Humans, pages 310–335. http://www.faculty.iu-bremen.de/hjaeger/pubs/2185_JaegerEck08.pdf. (Cited on page 65.)

Herbert Jaeger and Harald Hass [2004]. *Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication.* Science, 304, pages 78–80. doi:10.1126/science.1091277. http://www.faculty.iu-bremen.de/hjaeger/pubs/ESNScience04.pdf. (Cited on pages iii, 2, 3, 6, 10, 37, 40, 43 and 69.)

Herbert Jaeger, Mantas Lukoševičius, Dan Popovici, and Udo Siewert [2007b]. *2007 Special Issue: Optimization and applications of echo state networks with leaky-integrator neurons.* Neural Netw., 20(3), pages 335–352. ISSN 0893-6080. http://www.faculty.iu-bremen.de/hjaeger/pubs/leakyESN.pdf. (Cited on pages 2, 9, 14, 15, 17, 32, 33, 35 and 68.)

Eric Jones, Travis Oliphant, Pearu Peterson, et al. [2001–]. *SciPy: Open source scientific tools for Python.* http://www.scipy.org/. (Cited on pages xi and 68.)

C. Knapp and G. Carter [1976]. *The generalized correlation method for estimation of time delay.* Acoustics, Speech, and Signal Processing; IEEE Transactions on Signal Processing. http://homes.esat.kuleuven.be/~doclo/dspII/papers/knapp_carter_gcc.pdf. (Cited on pages 24, 25 and 26.)

W.G. Knecht [1994]. *Nonlinear noise filtering and beamforming using the perceptron and its volterra approximation.* IEEE Trans. Speech Audio Process, pages 55–62. (Cited on pages 49 and 65.)

S.Y. Kung [1997]. *Neural Networks for Intelligent Multimedia Processing.* Neural Networks for Signal Processing, pages 509–510. (Cited on pages 3 and 49.)

Michael Lehn [2008]. *Everything You Always Wanted to Know About FLENS, But Were Afraid to Ask.* http://flens.sf.net/flensdoc.pdf. Accessed 22.5.2008. (Cited on pages xi and 68.)

M. Lukosecicius and H. Jaeger [2007]. *Overview of Reservoir Recipes.* Jacobs University Technical Report. http://www.jacobs-university.de/imperia/md/content/groups/research/techreports/reservoiroverview_techreport11.pdf. Tech. Rep. No. 11. (Cited on page 65.)

W. Maass and C. Bishop [2001]. *Pulsed Neural Networks.* MIT Press. ISBN 0-262-13350-4. (Cited on page 2.)

W. Maass and E. D. Sontag [2000]. *Neural systems as nonlinear filters.* Neural Computation, 12(8), pages 1743–1772. http://www.igi.tugraz.at/maass/psfiles/107.pdf. (Cited on pages 4 and 49.)

Wolfgang Maass, Thomas Natschlaeger, and Henry Markram [2002]. *Real-time computing without stable states: a new framework for neural computation based on perturbations.* Neural Comput., 14(11), pages 2531–2560. ISSN 0899-7667. http://www.igi.tugraz.at/maass/psfiles/ms2469-figs.pdf. (Cited on pages 1, 2 and 14.)

Wolfgang Maass, Prashant Joshi, and Eduardo Sontag [2007]. *Computational Aspects of Feedback in Neural Circuits.* PLoS Computational Biology, 3(1), pages 1–20. doi:10.1371/journal.pcbi.0020165. http://www.igi.tugraz.at/maass/psfiles/168_PLOS_last_version.pdf. (Cited on pages 2 and 17.)

W. S. McCulloch and W. H. Pitts [1943]. *A logical calculus of the ideas imminent in neural nets.* Bulletin of Mathematical Biophysics. (Cited on page 1.)

Stephan Moeller, Martin Gromowski, and Udo Zoelzer [2002]. *A Measurement Technique for Highly Nonlinear Transfer Functions.* DAFx–02. http://www2.hsu-hh.de/ant/dafx2002/papers/DAFX02_Moeller_Gromowski_Zoelzer_measurement_nonlinear.pdf. (Cited on page 50.)

Swen Mueller and Paulo Massarani [2001]. *Transfer-Function Measurement with Sweeps*. Audio Engineering Society, 49, pages 443–471. http://www.anselmgoertz.de/Page10383/Monkey_Forest_dt/Manual_dt/Aes-swp.pdf. (Cited on page 50.)

Maciej Niedzwiecki and Krzysztof Cisowski [2001]. *Smart Copying - A New Approach to Reconstruction of Audio Signals*. IEEE Transactions on Signal Processing, 49, pages 2272–2282. (Cited on pages 4 and 57.)

Travis E. Oliphant [2007]. *Python for Scientific Computing*. Computing in Science and Engg., 9(3), pages 10–20. ISSN 1521-9615. doi:10.1109/MCSE.2007.58. http://www.computer.org/portal/cms_docs_cise/cise/2007/n3/10-20.pdf. (Cited on pages xi and 68.)

Leo Pape, Jornt de Gruijl, and Marco Wiering [2007]. *Democratic Liquid State Machines for Music Recognition*. Springer Berlin / Heidelberg. (Cited on page 65.)

Helene Paugam-Moisy, Regis Martinez, and Samy Bengio [2008]. *Delay learning and polychronization for reservoir computing*. Neurocomputing, 71(7–9), pages 1143–1158. http://liris.cnrs.fr/Documents/Liris-3399.pdf. (Cited on page 3.)

K.I. Pedersen, B.H. Fleury, and P.E. Mogensen [1997]. *High resolution of electromagnetic waves in time-varying radiochannels*. Personal, Indoor and Mobile Radio Communications, 1997. 'Waves of the Year 2000'. PIMRC '97, 2, pages 650–654. doi:10.1109/PIMRC.1997.631112. (Cited on page 28.)

Joern Schattschneider and Udo Zoelzer [1999]. *Discrete-Time Models For Nonlinear Audio Systems*. Proc. of the Int. Conf. on Digital Audio Effects (DAFx-99). http://www.hsu-hh.de/download-1.4.1.php?brick_id=ZCXQx8BdUQZKnUBW. (Cited on pages 3, 49 and 50.)

M. Schetzen [1980]. *The Volterra and Wiener Theories of Nonlinear Systems*. John Wiley, New York. ISBN 978-1575242835. (Cited on pages 49, 52 and 53.)

Ulf D. Schiller and Jochen J. Steil [2005]. *Analyzing the weight dynamics of recurrent learning algorithms*. Neurocomputing. doi:10.1016/j.neucom.2004.04.006. http://www.techfak.uni-bielefeld.de/ags/ni/publications/media/SchillerSteil2004-ATW.pdf. (Cited on page 2.)

Juergen Schmidhuber, Daan Wierstra, Matteo Gagliolo, and Faustino Gomez [2007]. *Training Recurrent Networks by Evolino*. Neural Comput., 19(3), pages 757–779. ISSN 0899-7667. (Cited on pages iv, vii, 32, 33, 34 and 35.)

Michael Schmitt [1999]. *On the implications of delay adaptability for learning in pulsed neural networks*. Proceedings of the Workshop on Biologically-Inspired Machine Learning, pages 28–37. http://www.neurocolt.com/tech_reps/2000/00069.ps.gz. (Cited on page 29.)

H. Schurer, C. Slump, and O. Herrmann [1995]. *Second order Volterra inverses for compensation of loudspeaker nonlinearity*. Proceedings of the IEEE ASSP Workshop on applications of signal processing to Audio & Acoustics, pages 74–78. (Cited on page 49.)

H. T. Siegelmann and E. D. Sontag [1991]. *Turing computability with neural nets*. Applied Mathematics Letters, 4, pages 77–80. (Cited on page 1.)

Mark D. Skowronski and John G. Harris [2007]. *2007 Special Issue: Automatic speech recognition using a predictive echo state network classifier*. Neural Netw. Springer Berlin / Heidelberg. (Cited on page 65.)

Julius O. Smith [2007a]. *Introduction to Digital Filters with Audio Applications*. W3K Publishing. ISBN 978-0-9745607-1-7. http://ccrma.stanford.edu/~jos/filters/. Online book, accessed 14.4.2008. (Cited on page 19.)

Julius O. Smith [2007b]. *Mathematics of the Discrete Fourier Transform (DFT)*. W3K Publishing. ISBN 978-0-9745607-4-8. http://ccrma.stanford.edu/~jos/mdft/. Online book, accessed 24.4.2008. (Cited on page 25.)

Steven W. Smith [1997]. *The scientist and engineer's guide to digital signal processing*. California Technical Publishing, San Diego, CA, USA. ISBN 0-9660176-3-3. http://www.dspguide.com/. Online book, accessed 15.4.2008. (Cited on page 17.)

Jochen J. Steil [2004]. *Backpropagation-Decorrelation: Recurrent Learning with O(N) Complexity*. pages 843–848. http://www.techfak.uni-bielefeld.de/~jsteil/publications.html. (Cited on page 2.)

A. Stenger and R. Rabenstein [1999]. *Adaptive Volterra Filters For Nonlinear Acoustic Echo Cancellation*. (Cited on pages 49 and 65.)

Bjarne Stroustrup [1986]. *The C++ Programming Language, Third Edition*. Addison-Wesley. ISBN 9780201120783. (Cited on page 68.)

H. Swadlow [1985]. *Physiological properties of individual cerebral axons studied in vivo for as long as one year*. J. Neurophysiol., 54, pages 1346–1362. (Cited on pages 3 and 23.)

Tom Szilagyi [2004–]. *TAP TubeWarmth LADSPA Plugin*. http://tap-plugins.sourceforge.net/ladspa/tubewarmth.html. Part of the TAP plugin collection of Tom Szilagyi, accessed 29.05.2008. (Cited on page 52.)

A. Uncini and G. Cocchi [2002]. *Subband neural networks for noisy signal forecasting and missingdata reconstruction*. Neural Networks, 1, pages 438–441. doi:10.1109/IJCNN.2002.1005512. (Cited on page 57.)

Aurelio Uncini [2003]. *Audio signal processing by neural networks*. Neurocomputing, 55(3–4), pages 593–625. doi:10.1016/S0925-2312(03)00395-3. (Cited on pages 1, 3 and 49.)

Saeed V. Vaseghi [1996]. *Advanced signal processing and digital noise reduction*. Queen's University of Belfast UK. ISBN 978-0471958758. (Cited on page 57.)

Todd L. Veldhuizen [1995]. *Expression templates*. C++ Report, 7(5), pages 26–31. (Cited on page 68.)

D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout [2005]. *Isolated word recognition with the Liquid State Machine: a case study*. http://citeseer.ist.psu.edu/744394.html. (Cited on page 65.)

D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt [2007a]. *2007 Special Issue: An experimental unification of reservoir computing methods*. Neural Netw., 20(3), pages 391–403. ISSN 0893-6080. doi:10.1016/j.neunet.2007.04.003. (Cited on page 1.)

David Verstraeten, Benjamin Schrauwen, and Jan Van Campenhout [2007b]. *Adapting reservoirs to get Gaussian distributions*. Proceedings of the 15th European Symposium on Artificial Neural Networks, pages 495–500. http://escher.elis.ugent.be/publ/Edocs/DOC/P107_046.pdf. (Cited on pages 65 and 69.)

Ondria Wasem, David Goodman, Charles Dvorak, and Howard Page [1988]. *The Effect of Waveform Substitution on the Quality of PCM Packet Communications*. IEEE Transactions on Acoustics, Speech, andSignal Processing, 36, pages 342–348. (Cited on pages 4 and 57.)

Welf Wustlich and Udo Siewert [2007]. *Echo-State Networks with Band-Pass Neurons: Towards Generic Time-Scale-Independent Reservoir Structures*. PLANET Intelligent Systems GmbH. http://snn.elis.ugent.be/sites/snn/files/EsnBandPass_20071017.pdf. Posted on the reservoir computing mailing list, accessed 23.3.2008. (Cited on pages 2, 17, 18, 19 and 68.)

Yanbo Xue, Le Yang, and Simon Haykin [2007]. *2007 Special Issue: Decoupled echo state networks with lateral inhibition*. Neural Netw., 20(3), pages 365–376. ISSN 0893-6080. doi:10.1016/j.neunet.2007.04.014. (Cited on pages 32, 33 and 34.)

Y.Huang, J.Benesty, and J.Chen [2006]. *Acoustic MIMO Signal Processing*. Springer-Verlag Berlin Heidelberg. ISBN 3-540-37630-5. (Cited on pages 3, 23, 26, 29 and 65.)