RESERVOIR COMPUTING: A POWERFUL BLACK-BOX FRAMEWORK FOR NONLINEAR AUDIO PROCESSING

Georg Holzmann

Neural Information Processing Group
Technische Universität Berlin
D-10587 Berlin, Germany
http://grh.mur.at
grh@mur.at

ABSTRACT

This paper proposes reservoir computing as a general framework for nonlinear audio processing. Reservoir computing is a novel approach to recurrent neural network training with the advantage of a very simple and linear learning algorithm. It can in theory approximate arbitrary nonlinear dynamical systems with arbitrary precision, has an inherent temporal processing capability and is therefore well suited for many nonlinear audio processing problems. Always when nonlinear relationships are present in the data and time information is crucial, reservoir computing can be applied.

Examples from three application areas are presented: nonlinear system identification of a tube amplifier emulator algorithm, nonlinear audio prediction, as necessary in a wireless transmission of audio where dropouts may occur, and automatic melody transcription out of a polyphonic audio stream, as one example from the big field of music information retrieval. Reservoir computing was able to outperform state-of-the-art alternative models in all studied tasks.

1. INTRODUCTION

Most of the classical audio processing techniques, which resulted in numerous essential applications and are indispensable to life today, are founded on the assumptions of linearity, stationarity and second-order statistics with emphasis on Gaussianity. These assumptions are made for the sake of mathematical tractability [1]. Nevertheless, most or maybe all the physical signals that one has to deal with in real life are generated by dynamic processes, which are simultaneously nonlinear, nonstationary and non-Gaussian. With the introduction of digital methods in music production, many people missed the "warmth" and "dirt" of sounds generated by analog audio systems. Therefore it became more and more popular to bring back some nonlinearities and noise of the old equipment into the digital audio processing systems, because of their enjoyable characteristic distortions.

If one wishes to simulate, predict, classify or control nonlinear dynamical systems, one needs an executable system model. Sometimes it is hard to obtain an analytical description of the system and then one has to use black-box modeling techniques. Usually specific algorithms have been designed for specific nonlinear problems (for instance median and bilinear filters, special Volterra filter structures, time delay neural networks, ... see [2]) and it was a kind of art to find a good model. In theory recurrent neural networks (RNNs) can approximate arbitrary nonlinear dynamical system with arbitrary precision (universal approximation property [3]) and are also able to (re)produce temporal patterns. However, it is very hard to train RNNs and a number of specialized learning algorithms exist in literature, which are difficult to use and lead to suboptimal solutions.

A new and surprisingly easy to use network structure for recurrent neural networks was discovered independently by Jäger [4], who called these RNNs echo state networks (ESN), and by Maass [5], who developed a similar approach for spiking neural networks and called the structure liquid state machine (LSM). Both and a few other methods are subsumed under the more general term reservoir computing [6]. The common idea is that input signals are fed into a fixed nonlinear dynamical system, called dynamic reservoir or liquid, which is composed of randomly connected recurrent neurons. Only the output connections, the readout, are trained by simple linear regression. Figure 1 shows a schematic overview of the reservoir computing concept. The function of the reservoir can

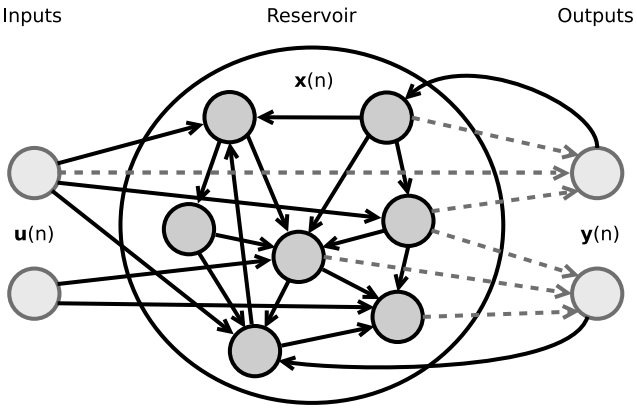

Figure 1: *Schematic overview of the reservoir computing approach with two inputs, two outputs and feedback from the outputs back to the reservoir. Solid black arrows indicate fixed, random connections and dotted arrows are trainable readout connections.*

be compared to that of the kernel in support vector machines [7], with the crucial difference that reservoirs are inherently temporal and therefore well suited for audio signals: inputs drive the nonlinear reservoir and produce a high-dimensional dynamical "echo response", which is used as a non-orthogonal basis to reconstruct the desired output signals 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 a local minima. It has been demonstrated on a number of benchmark tasks, that reservoir computing methods are often able to vastly outperform other state-of-the-art algorithms for nonlinear dynamical system modeling (see for example [8]).

In this paper, reservoir computing is introduced as an additional, powerful tool for nonlinear audio processing and Section 2 briefly presents the basic network equations and an overview over different architectures. Always when nonlinear effects are perceivable or important, linear modeling is insufficient and nonlinear techniques have to be applied. Reservoir computing 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. To show the possibilities of reservoir computing, examples from three different application areas are implemented in Section 3. The first example is a nonlinear system identification task of a tube amplifier emulator algorithm, the second an audio prediction task, where the reservoir computing network tries to forecast future samples out of a given history horizon. Afterwards a classification-based system for automatic melody transcription out of a polyphonic audio stream is presented, as one example of possible applications in music information retrieval. In all simulations, reservoir computing was able to outperform current state-of-the-art alternative models. Finally Section 4 gives a short conclusion and lists advantages and disadvantages of the proposed framework.

2. RESERVOIR COMPUTING

Reservoir computing is a term for a recently emerged supervised learning technique for recurrent neural networks. Its main representatives are echo state networks [4], liquid state machines [5] and a few other models like backpropagation decorrelation [9] and Evolino-based LSTM networks [10]. Most implementations of the liquid state machine use a spiking neuron model called leaky integrate and fire (LIF) neuron [11], whereas echo state networks are composed out of analog neurons, for instance linear, sigmoid or leaky integrator units. The common idea is to use a fixed reservoir with sparsely and randomly connected neurons, and only the output weights are adapted during training. Finding the right output weights is a linear regression task and if a mean squared error is used, the algorithm will find a global minimum of this error function. The output signals can also be fed back as additional inputs, which makes it possible to train these RNNs as oscillators or other generators. A schematic overview of the reservoir computing concept with output feedback is presented in Figure 1.

On a first view it might look, that random reservoirs must not work satisfactorily. However, [12] showed that also in traditional training methods for RNNs, where all weights are adapted, the dominant changes are only in the output weights. Furthermore classical RNN learning algorithms adapt the connections by some sort of gradient descent, which renders them slow and convergence cannot be guaranteed. Reservoir computing belongs to the family of computational methods with a clear biological footing and a related mechanism has been investigated in cognitive neuroscience [13] in the context of modeling sequence processing in mammalian brains, especially speech recognition in humans. Moreover it was proven in [5] and [14], 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 nonlinear signal processing tasks.

The next two subsections will introduce the basic network equations and an overview over different architectures. However, note that it is not possible in the scope of this paper to present a detailed and theoretical background, the interested reader should follow the references where necessary.

2.1. Notation and Training

Here the mathematical discussion is restricted to reservoir computing with basic analog neurons, such structures are also referred to as echo state networks (ESNs). An analog neuron receives inputs and computes a weighted sum of them to produce an output. Afterwards the output is passed through a so called activation function, which is usually a non-linear function with a sigmoid shape. In the examples of this paper, hyperbolic tangent (tanh) and linear activation functions, as commonly seen in multilayer perceptrons [15], are used.

Consider an echo state network with L inputs, M outputs and a reservoir with N neurons. At time $n = 1, 2, ..., n_{max}$ the input vector is $\mathbf{u}(n)$ and the output $\mathbf{y}(n)$. The activations (current outputs) of all neurons in the reservoir are collected each timestep in a N-dimensional vector $\mathbf{x}(n) = (x_1(n), ..., x_N(n))^\intercal$. Internal connection weights of the reservoir are stored in a $N \times N$ matrix $\mathbf{W}$, weights of input-to-reservoir connections in a $L \times N$ matrix $\mathbf{W}^{in}$ and possible output-to-reservoir (feedback) weights in a $M \times N$ matrix $\mathbf{W}^{fb}$. The only trainable weights are output weights, which include connections from the reservoir neurons and inputs to the outputs and are collected in a $(N + L) \times M$ matrix $\mathbf{W}^{out}$.
The activations of neurons in the reservoir are updated according to

$$\mathbf{x}(n+1) = \mathbf{f}(\rho\mathbf{W}\mathbf{x}(n)+\mathbf{W}^{in}\mathbf{u}(n+1)+\mathbf{W}^{fb}\mathbf{y}(n)+\mathbf{v}(n+1)) \quad (1)$$

where $\mathbf{v}(n + 1)$ is a small possible noise term and $\mathbf{f}$ is the nonlinear activation function of the reservoir neurons, for example tanh, a sigmoid or step function. If the largest absolute eigenvalue $\mid \lambda \mid_{max} (\mathbf{W}) = 1$ and the spectral radius ρ is $0 \leq \rho < 1$, the network states $\mathbf{x}(n)$ become asymptotically independent of initial conditions and depend only on a input history, which is called the "echo state property" [4].
After calculating the internal states, the outputs can be computed with

$$\mathbf{y}(n + 1) = \mathbf{g}(\mathbf{W}^{out}[\mathbf{x}(n + 1); \mathbf{u}(n + 1)]) \quad (2)$$

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

A training algorithm modifies only output weights $\mathbf{W}^{out}$, so that an optimal mapping from the internal states $\mathbf{x}(n)$ to a desired teacher signal is achieved. Finding the right mapping is a linear regression problem and if a mean squared error is used, the algorithm will find a global minimum of this error function. Commonly known online (e.g. LMS or RLS) and offline (e.g. pseudo-inverse, ridge regression) linear regression algorithms can be applied to this task.
In an offline training algorithm, based on the pseudo-inverse, the following steps calculate optimal output weights $\mathbf{W}^{out}$:

1. Randomly initialize the reservoir with a sparse connection matrix $\mathbf{W}$, so that its largest absolute eigenvalue $\mid \lambda \mid_{max} (\mathbf{W}) = 1$ and set the spectral radius ρ between $0 \leq \rho < 1$.

The network will be stable and has the "echo state property".[1]

2. Run the reservoir with a teaching input signal $\mathbf{u}_{teach}(n)$ to produce reservoir activations $\mathbf{x}(n)$ for each timestep according to Equation 1.
3. Dismiss data from an initial washout period n_{min}, where $n < n_{min}$, to get rid of initial transients. Collect the remaining network and input states $(\mathbf{x}(n), \mathbf{u}_{teach}(n))$ for each timestep row-wise into a $(n_{max} - n_{min}) \times (N + L)$ matrix $\mathbf{S}$, where n_{max} is the number of training examples.
4. Invert output activation function $\mathbf{g}$ and collect target signals $\mathbf{g}^{-1}(\mathbf{y}_{teach}(n))$ for each output neuron and timestep $n \geq n_{min}$ into a $(n_{max} - n_{min}) \times M$ matrix $\mathbf{T}$.
5. Compute pseudo-inverse $\mathbf{S}^{\dagger}$ and put
$$\mathbf{W}^{out} = (\mathbf{S}^{\dagger}\mathbf{T})^{\intercal} \tag{3}$$
where $(.)^{\dagger}$ denotes the pseudo-inverse and $(.)^{\intercal}$ the transposition of a matrix.
6. The network is now trained and Equations 1 and 2 can be used to calculate new outputs.

For more details on how to train echo state networks and the selection of model parameters (e.g. network size N and spectral radius ρ) the reader is referred to the ESN tutorial by Jäger [16].

2.2. Different Reservoir Computing Architectures

Reservoir computing can be seen more as a framework rather than one specific algorithm. Various models exist in literature, which are adapted to specific requirements. The primarily emerged two main flavors are echo state networks [4], composed of analog neurons as described in Section 2.1, and liquid state machines [5], which are made of biologically more plausible spiking neurons and have been developed in a computational neuroscience perspective.

From an engineering point of view, analog neurons are preferred most of the time, although liquid state machines were successfully applied to speech recognition problems [6]. Additional analog neuron types are for instance leaky integrator units [17], which also incorporate information from previous time steps, filter neurons [18], where regions tuned to different frequency bands can emerge in the reservoir, or long short-term memory (LSTM) neurons [10], which are one way to model long-term dependencies.

Furthermore there are many possibilities how to train the readout. These algorithms can be divided into online and offline methods. Offline algorithms are for instance the Wiener-Hopf solution [18], which is basically equivalent but faster than the pseudo-inverse based approach from Section 2.1, ridge regression [17], which incorporates an additional regularization factor, the delay & sum readout [18], which adds trainable delays to the readout connections and therefore vastly improves the memory capacity, or also linear support vector machines [10], which try to maximize the margin instead of a mean squared error and are often useful in classification tasks. Online learning rules are for instance the backpropagation-decorrelation (BPDC) algorithm for O(N) training [9] or the recursive least square (RLS) algorithm [8].

An other direction of research is reservoir adaptation, where one tries to pre-adapt the random reservoir to a specific task [19]. However, reservoir adaptation techniques were not used in the examples of this paper.

[1] Note that the "echo state property" ensures stability only for networks without output feedback. For a more detailed stability analysis see [4].

3. SIMULATIONS

The following subsections present examples from three main application areas of reservoir computing networks: audio system identification, audio prediction and music information retrieval or audio classification. All results are compared to alternative models. Network setups for all tasks are presented in Table 1. Other parameters, if necessary, will be described before presenting the simulation results. For a practical tutorial on how to set the individual parameters for a task and which neuron types to choose, one should consider the ESN tutorial by Jäger [16] and Section 7.1 of [18].

To be able to compare the results of different algorithms, the normalized root mean square error ($NRMSE$), a common error measure in machine learning, will be used in the first two experiments. The $NRMSE$ between a target signal $y_{target}(n)$ and a generated signal $y(n)$ for $n = 0, \ldots, \tilde{N} - 1$ is calculated with

$$NRMSE = \sqrt{\frac{1}{\tilde{N}\sigma^2_{target}} \sum_{n=0}^{\tilde{N}-1} (y_{target}(n) - y(n))^2}. \tag{4}$$

Here the squared difference of target and generated signal is normalized with the variance of the target signal σ^2_{target}, which is advantageous when using signals with different amplitudes. However, note the the $NRMSE$ might not be the best choice for comparing the perception of different sounds. Audio signals that are very close in terms of temporal waveforms may be very different from a perceptual point of view, therefore subjective listening tests should be used for evaluation in a more comprehensive study.

All simulation were implemented in Python[2] and a C++ library for reservoir computing networks, called aureservoir[3], was developed. The audio examples of the current section are downloadable at http://grh.mur.at/publications/reservoir-computing-for-audio.

3.1. System Identification of a Tube Amplifier Plugin

Nonlinear system identification tasks are necessary in many applications in audio signal processing. Always when distortions or saturations are perceivable, linear modeling is not sufficient and nonlinear methods have to be applied. Often Volterra systems [21] were used for applications like black-box modeling audio effects with nonlinear distortions [22], nonlinear echo cancellation [23], loudspeaker linearization using pre-distortion [24], nonlinear beamforming [25] or restoration of nonlinearly distorted audio [26]. In general also reservoir computing techniques could be applied to all those tasks, with the advantage that they are quite easy to use and have a much lower computational complexity than Volterra series with a large number of kernel terms. Furthermore it was theoretically proven in [27], that neural networks with time delays are able to approximate all filters that can be characterized by Volterra series.

This example presents a black-box nonlinear system identification of a tube amplifier emulator algorithm[4]. Volterra series were applied in literature for black-box modeling of tubes on digital computers [28], but usually only second or third order systems

[2] NumPy and Scipy (http://www.scipy.org) are packages for numerical and scientific computations under Python (http://www.python.org).

[3] aureservoir: http://aureservoir.sourceforge.net, efficient C++ library for analog reservoir computing neural networks with Python bindings

[4] The equipment and time for the identification of a real tube amplifier was unfortunately not available.

Table 1: *Network setups for the simulations of Section 3.*

Parameters	Nonlinear System Identification	Audio Prediction	Melody Extraction
Reservoir neurons N	100	100	1000
Reservoir connectivity	0.2	0.2	0.1
Spectral radius ρ	0.85	0.8	0.3
Reservoir neuron type	standard analog neurons (Eq. 1)	filter neurons [18]	leaky-integrator neurons [17]
Input weights $\mathbf{W}^{in}$ between	$[-4, 4]$	no inputs	$[-0.3, 0.3]$
Feedback weights $\mathbf{W}^{fb}$ between	no feedback	$[-0.5, 0.5]$	no feedback
Readout type	delay&sum readout [18]	delay&sum readout [18]	linear SVM readout [20]

with special structures for complexity reduction performed acceptable. Two different tubes, both implemented as open source digital audio plugins (LADSPA[5]), are identified with reservoir computing networks and compared to an identification with third order Volterra systems.

A challenging task was to choose the right training signals for the identification process, so that trained reservoir and Volterra systems performed well on arbitrary audio inputs. In linear signal processing impulses, swept sines, noise or maximum length sequences (see [29] 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 the training signal. 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[6]. A more detailed discussion of these training signals is given in [18], see also [30], [31] and [32] for nonlinear system identification examples.

In the first identification example a solo flute was used as test signal, discretized at a sampling rate of 44100 Hz. It was upsampled by a factor $L = 5$ to avoid aliasing, then processed by the "Valve Saturation" LADSPA plugin [33] and finally downsampled again by the same factor. The plugin had 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 $x(n)$

$$z(n) = \frac{x(n) - q}{1 - e^{-d(x(n)-q)}} + \frac{q}{1 - e^{dq}}, \quad x(n) \neq 0, \quad q \neq 0 \quad (5)$$

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 $y(n)$

$$y(n) = 0.999y(n-1) + z(n) - z(n-1). \quad (6)$$

The second tube example was simulated with a more complex test signal, including many instruments and a singer. This signal was processed with an oversampling factor $L = 10$ by the "TAP TubeWarmth" LADSPA plugin [34], which also had 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 passed through a static nonlinearity

$$\begin{aligned} z(n) &= a\left(\sqrt{|\, b_{1,1} + x(n)(b_{1,2} - x(n))\,|} + b_{1,3}\right), \\ &\qquad for \quad x(n) \geq 0 \\ z(n) &= -a\left(\sqrt{|\, b_{2,1} - x(n)(b_{2,2} + x(n))\,|} + b_{2,3}\right), \\ &\qquad for \quad x(n) < 0 \end{aligned} \quad (7)$$

with the parameters $a = 0.5305$, $b_{1,1} = 0.1896$, $b_{1,2} = 2.8708$, $b_{1,3} = -0.4354$, $b_{2,1} = 16.9468$, $b_{2,2} = 27.1424$, $b_{2,3} = -4.1166$. Afterwards a linear recursive filter is applied to get the output $y(n)$

$$y(n) = \frac{0.1 f_r}{0.1 f_r + 1}\left(y(n-1) + z(n) - z(n-1)\right) \quad (8)$$

where f_r is the sampling rate of the audio signal.

For both experiments a reservoir computing network as presented in Table 1 was used. It was first driven by 3000 samples of the training signal to washout initial transients and afterwards output weights were calculated from the next 5000 samples. Input, output target and the network output signal for the first identification example are shown in Figure 2.

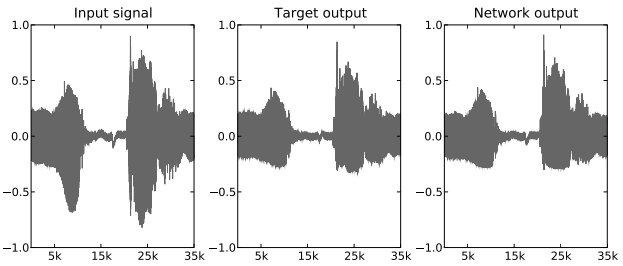

Figure 2: *Extract from the input, output target and network output signal of the first audio system identification example (flute signal).*

As a comparison, the same two examples were also identified using a third-order discrete time Volterra system [21]. The lowest generalization error was achieved with Volterra kernels h_1, h_2, h_3 of memory depth $N_1 = 100$, $N_2 = 40$, $N_3 = 5$ and the system was trained using a least squares approach [21] with

[5]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/`.

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

10000 samples of the training signal. The performance of both systems was measured with the normalized root mean square error as introduced in Equation 4. Table 2 shows the $NRMSE_{test}$ for both algorithms and identification examples, calculated from 100000 time steps of the test signals. One can see that especially in the second identification example, reservoir computing was able to outperform the Volterra system, while at the same time needing less computational power. However, it is also important to note that in both examples the nonlinear processing, a memoryless nonlinearity followed by a first-order filter, was quite simple. It was shown e.g. in [35] that reservoir computing techniques have advantages especially with higher-order nonlinear dynamical systems. Furthermore Volterra series are not preferred when strongly saturating nonlinearities are to be simulated due to the large number of required kernel terms.

Table 2: *Test error $NRMSE_{test}$ of the reservoir computing and Volterra system for both nonlinear audio system identification examples.*

Identification Example	Reservoir Comp.	Volterra
Valve Saturation (1)	0.0997	0.1277
TAP TubeWarmth (2)	0.1372	0.2353

The generalization performance of the reservoir approach was very constant, also when changing the system parameters from Table 1. In general it did not matter at all which exact values were chosen and it was easy to get good results. However, a delay&sum readout [18] was able to boost the performance, although there are no obvious long-term dependencies in the system.

3.2. Nonlinear Audio Prediction

The second example presents an audio prediction application with reservoir computing and compares the performance to commonly used alternative models. 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, or when impulsive noise appears. The scenario of this task [36] is a wireless signal transmission, where short dropouts with a length between 2 and 6 milliseconds can occur due to transmission problems. 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 Figure 4). Two different audio examples were taken from [36], 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. 27 dropout were generated at random locations within each audiofile, resulting in two percent of corrupted audio. A short extract with dropouts from the first audio example is shown in Figure 3.

The prediction performance of reservoir computing for both examples is compared to two commonly used alternative models: a pattern matching algorithm (PatMat) and a linear autoregressive (AR) model.
Pattern matching algorithms [37][38] use a signal template, usually just before the dropout, which is compared to areas in the past of the audio signal (the search window). Then the area within this search window with the highest similarity to the audio template is selected and copied into the dropout region. Many different versions of pattern matching algorithms exist and were evaluated in [36]. Here only the one with the best prediction result, which is based on waveform differences and defined in Equation 6 of [37], was considered.

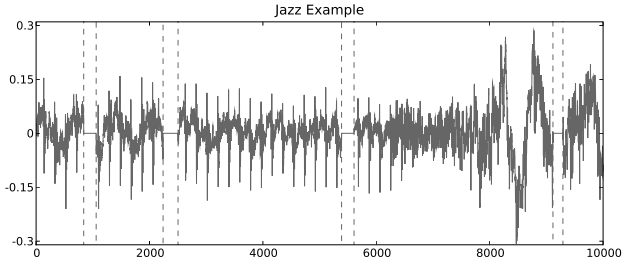

Figure 3: *Extract from the Jazz audio example of the nonlinear audio prediction task. Dropouts are marked with dotted lines.*

Additionally a linear autoregressive model [26], which is widely used for signal modeling, was implemented. The order of this linear predictor was set to 300, so that complex signals can be represented. Because no audio signal is truly stationary, it was necessary to train the model from a relatively short block of samples and a block length of 3000 was found to be optimal. Finally the coefficients of the AR model were estimated with a least squares approach based on those 3000 samples as described in Section 3.3.1 of [26].

The parameters of the reservoir computing network are given in Table 1. Here the reservoir neurons had additional built-in filters, to produce subregions in the network, which are specialized to process different frequency bands [18]. These band-pass filters were fixed when constructing the reservoir and spaced logarithmically between 60 Hz and 11000 Hz, to contain all frequencies important for the task.
In audio prediction the network is trained to be a generator of the audio signal, therefore the system needs output-feedback connections, indicated as $\mathbf{W}^{fb}$ in Equation 1. The network was first driven by 3000 samples to washout initial transients and afterwards output weights were calculated from 3000 time steps directly before the dropout.
Adding a small noise term $\mathbf{v}(n)$ in the state update equation 1 as regularization is often necessary for stability when networks are trained with output feedback. The amount of noise was chosen so that no more unstable simulations were produced and a uniform noise term $\mathbf{v}(n)$ between $[-0.0002, 0.0002]$ was found to be optimal.

The performance of all algorithms was benchmarked by calculating the normalized root mean square error between model output and original signals, only from data in the dropout regions. Results for both audio examples are presented in Table 3. Reservoir computing networks 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. Prediction examples of all models are shown in Figure 4.

In audio prediction it was very important to use filter neurons [18] in the reservoir. With standard neurons, as defined in Equation 1, the network was not able to produce that good results. The parameters of filter neurons were set to contain all necessary frequencies important for the task and logarithmically spaced bandpass filters from 60 Hz to 11000 Hz, with a bandwidth of 2 octaves,

Table 3: *Test error $NRMSE_{test}$ calculated from the dropout regions of both audio examples of the nonlinear audio prediction task.*

Example	Reservoir Comp.	AR Model	PatMat
Jazz (1)	0.601	0.671	0.734
Beethoven (2)	0.536	0.849	0.873

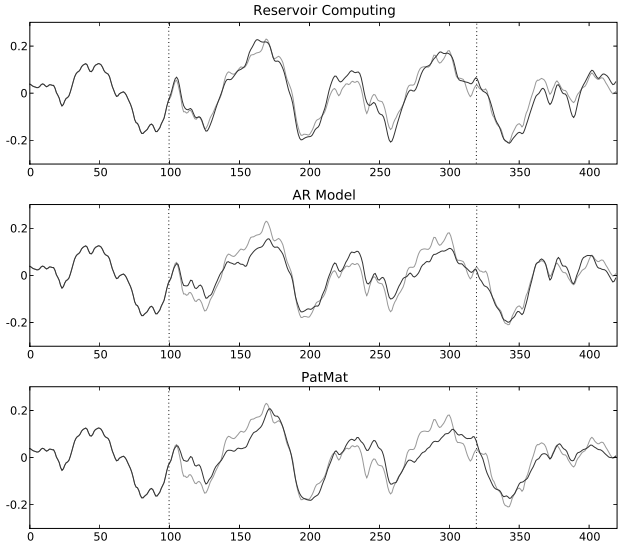

Figure 4: *Example dropout of the audio prediction task. The original signal is plotted in light gray, the predicted signal in dark gray. Beginning and ending of the dropout is marked with dotted lines.*

were found to be optimal.
With an additional delay&sum readout [18], it was possible to achieve a further performance boost. The exact values of all other parameters (spectral radius, feedback weights, connectivity) did not matter and it was easy to get good results.

3.3. Music Information Retrieval: Melody Extraction

Music information retrieval (MIR) is an emerging research field and many MIR applications use classifiers, like support vector machines (SVM) or hidden Markov models, to learn from existing music databases. Reservoir computing techniques are in principle similar to SVM classifiers [7], with the crucial difference that they also incorporate the temporal development of input signals. Therefore in tasks, where also time information is very important, reservoir computing should be able to boost the classification performance.

As one example from the pool of possible applications, a reservoir computing network was trained to extract the predominant melody (lead voice) of a polyphonic piece of music. Melody provides a very convenient and memorable description of popular music and is used as the basis for instance in query-by-humming systems. Two main techniques exist in predominant melody detection: a rule-based approach, where one assumes that individual notes are build of a set of harmonics of a particular fundamental, and a classification based approach [39], where the correct melody is estimated based on a training set with labeled examples. For a detailed comparison of different techniques see [40].

The setup of this task is similar as in [41]. Mono audio signals were first downsampled to 8kHz and then converted to frequency domain using a short-time Fourier transform (STFT) with a window length of 1024 (128ms) and an overlap of 944 samples (10ms grid). Only the first 256 bins, which correspond to frequencies below 2kHz, were considered. To remove some of the influence due to different instrument timbres, the magnitude of the STFT was normalized within each time frame to achieve zero mean and unit variance over a 51-frame local frequency window [41].

A supervised classifier can now use these 256 STFT features as input and calculates one output for each possible pitch class, quantized in semitones. In training the target output is set to 1 and all others to -1, and in testing the output with the highest value is considered as the pitch of the current frame. The biggest problem in a classification-based approach is to collect a suitable set of training data. Here a subset of the dataset from [41] was used and manually divided into a train and test set, so that every possible pitch class was included at least once in the training data. In especially, only the synthesized MIDI files, where the lead voice is a monophonic track and can be extracted easily, were utilized in this experiment[7].

First a SVM with a radial basis function kernel was implemented, using the popular LIBSVM library[8], to get a baseline. In order to classify the dominant melodic note with SVMs, it is assumed that each frame is independent of all others, because this classifier has no built-in memory and can only see the STFT data of the current frame. One SVM for each of the 19 target pitch classes and one additional SVM, which should indicate when background is detected and no melody note is present, were trained on the dataset. SVM parameters γ and C were varied in a grid search [20] within $C = 2^{-5}, 2^{-3}, \ldots, 2^{15}$ and $\gamma = 2^{-15}, 2^{-13}, \ldots, 2^{3}$ and the optimal values were found to be $C = 0.5$ and $\gamma = 0.125$.

The reservoir computing network had leaky-integrator neurons [17] in the reservoir and the detailed setup is shown in Table 1. As input, the 256 STFT features were used and again one output for each possible pitch class was trained.
The reservoir computing framework does not prescribe a particular optimality criterion for calculating the output weights. If instead of the mean squared error criterion from Equation 3, a maximum margin criterion is considered, using for instance a linear SVM, the classification performance can be improved significantly. Therefore each output neuron was implemented as a linear SVM [20].

To compare the performance of both algorithms, an error is counted for each frame where a classifier predicts a wrong note. The absolute number of frame errors and the percent error rate is presented in Table 4, furthermore an example classification output is shown in Figure 5. However, note that no additional features and post processing techniques [40] were applied and these values are the raw classification results.
Reservoir computing achieved a performance boost of about 36% compared to the SVM approach, therefore one can assume that time information between frames and memory is also important in this task. The assumption from [41], that each frame is independent of all other frames, is obviously a rough simplification.

[7]The multitrack audio recordings from [41] were removed from the dataset, because they were not hand-labeled and contained errors in the target data.

[8]LIBSVM: http://www.csie.ntu.edu.tw/˜cjlin/libsvm/, an open-source library for support vector machines.

Table 4: *Frame error rate of the melody extraction example, in absolute values and percentage.*

Misclassifications	Reservoir Comp.	SVMs
absolute errors	571	892
percent error rate	14.75%	23.05%

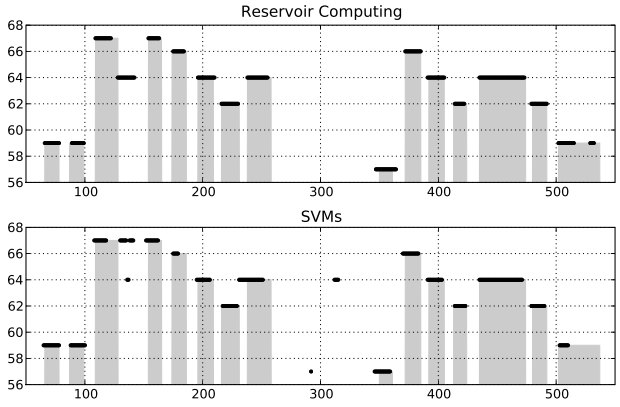

Figure 5: *Example classification results from both melody extraction algorithms. Target values are plotted in light gray boxes, model outputs in black dots.*

Leaky integrator neurons and linear SVM readouts were able to significantly improve the performance in this task.
In many classification applications, the maximum margin criterion of SVMs has advantages compared to the mean squared error criterion. The additional parameter C of a linear SVM was easily discovered by a line search and a value of $C = 0.001$ was found to be optimal [20].
Leaky integrator units [17] incorporate also information of the network states from previous time steps when calculating the current state. Such neuron types were useful in many classification tasks, because the dynamics in the reservoir are slowed down and the memory capacity is enhanced. The additional parameter *leaking rate* was set to 0.3 (same value as the spectral radius ρ), resulting in an effective spectral radius of 1 [17].
A further important parameter was the network size N of the reservoir. The bigger the network, the better was the classification performance. However, the size was finally fixed at a value of $N = 1000$, although it was possible to get better results when using even bigger networks.

4. CONCLUSIONS

This paper proposed reservoir computing as a general black-box framework for nonlinear audio processing. The presented simulations included three main application areas (nonlinear system identification, nonlinear time series prediction and classification) and it was demonstrated, that reservoir computing was able to outperform state-of-the-art alternative models in all studied tasks. Always when nonlinear relationships are present in the data and time information is crucial, these techniques can be applied. Therefore reservoir computing can be seen as a generalization of currently used linear methods in nonlinear domain, resulting in a tool which can be trained and adapted to many practical problems.
A number of pleasant properties of the model exist [42]:

Inherently temporal: The network itself is inherently temporal and therefore especially useful for audio processing. Because of recurrent reservoir connections, the influence of an input remains detectable in the reservoir and there is no need to manually encode time information in space (like for instance with tapped delay lines). The current reservoir state is a snapshot of the temporal development of inputs, therefore a readout function can instantly see the temporal information contained in all input signals.

Multiple readouts: If there is no feedback from outputs, it is possible to use multiple readouts on the same reservoir, as demonstrated in the melody extraction example from Section 3.3. This means that multiple tasks can be trained simultaneously on the same input signals, with the advantage that the costly nonlinear transformations only have to be calculated once. For instance, additional readouts for drum or baseline extraction could be easily added to the melody extraction example using the same reservoir.

Simple Training: There is no need to train the reservoir. Once it is constructed, it won't be modified by a learning algorithm. Only the readout functions need to be trained by simple and linear methods, which is usually much easier, more accurate and less costly than training a complete recurrent neural network or other nonlinear models.

A disadvantage of the proposed framework is, that it contains many extensions and different neuron types, which might be confusing for beginners. However, once one is familiar with the basic reservoir computing ideas, the parameter settings and neuron types are quite intuitive and even beginners will soon get good results.

Further possible audio applications of reservoir computing are:

- Long-term audio prediction tasks, as necessary in audio restoration.
- The delay&sum readout [18] should be well suited for nonlinear echo cancellation or beamforming applications, where a sparse system representation is necessary.
- Many classification tasks in music information retrieval could benefit from reservoir computing techniques.
- Reservoir computing networks can be used as audio synthesizers, when training specific control input signals to target audio samples.

Altogether reservoir computing offers an attractive framework for solving complicated nonlinear audio processing tasks and could be considered as an additional tool in many signal processing and classification applications. They are quite easy to use, robust against parameter changes and often have a much lower computational complexity compared to other nonlinear models.

5. ACKNOWLEDGMENTS

This work was written under support by the Integrated Graduate Program on Human-Centric Communication at the Technical University Berlin. The author would also like to thank Hubert Außerlechner for the dropout concealment audio examples of his master thesis [36], Graham Poliner for the polyphonic melody extraction dataset [41], Dmitriy Shutin for the Volterra system code of Section 3.1 and the anonymous reviewers for their valuable comments.

6. REFERENCES

[1] Simon Haykin, "Neural networks expand sp's horizons," *IEEE Signal Processing Magazine*, vol. 13, pp. 24–49, 1996.

[2] Aurelio Uncini, "Audio signal processing by neural networks," *Neurocomputing*, vol. 55, no. 3–4, pp. 593–625, 2003.

[3] H. T. Siegelmann and E. D. Sontag, "Turing computability with neural nets," *Applied Mathematics Letters*, vol. 4, pp. 77–80, 1991.

[4] Herbert Jaeger, "The "echo state" approach to analysing and training recurrent neural networks," Tech. Rep., 2001, German National Research Center for Information Technology Bremen, http://www.faculty.iu-bremen.de/hjaeger/pubs/EchoStatesTechRep.pdf.

[5] Wolfgang Maass, Thomas Natschlaeger, and Henry Markram, "Real-time computing without stable states: a new framework for neural computation based on perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[6] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt, "2007 special issue: An experimental unification of reservoir computing methods," *Neural Networks*, vol. 20, no. 3, pp. 391–403, 2007.

[7] Nello Cristianini and John Shawe-Taylor, *An Introduction to Support Vector Machines and other kernel-based learning methods*, Cambridge University Press, 2000.

[8] Herbert Jaeger and Harald Hass, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, pp. 78–80, 2004.

[9] Jochen J. Steil, "Backpropagation-decorrelation: Recurrent learning with o(n) complexity," in *Proc. IJCNN*, 2004, pp. 843–848.

[10] Juergen Schmidhuber, Daan Wierstra, Matteo Gagliolo, and Faustino Gomez, "Training recurrent networks by evolino," *Neural Computation*, vol. 19, no. 3, pp. 757–779, 2007.

[11] W. Maass and C. Bishop, *Pulsed Neural Networks*, MIT Press, 2001.

[12] Ulf D. Schiller and Jochen J. Steil, "Analyzing the weight dynamics of recurrent learning algorithms," *Neurocomputing*, 2005.

[13] Peter F. Dominey, "Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning," *Journal Biological Cybernetics*, vol. 73, no. 3, pp. 265–274, 1995.

[14] Wolfgang Maass, Prashant Joshi, and Eduardo Sontag, "Computational aspects of feedback in neural circuits," *PLoS Computational Biology*, vol. 3, no. 1, pp. 1–20, 2007.

[15] Simon Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice Hall, 1998.

[16] Herbert Jaeger, "A tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the "echo state network" approach," Tech. Rep., 2002, German National Research Center for Information Technology Bremen, http://www.faculty.iu-bremen.de/hjaeger/pubs/ESNTutorialRev.pdf.

[17] Herbert Jaeger, Mantas Lukoševičius, Dan Popovici, and Udo Siewert, "Optimization and applications of echo state networks with leaky-integrator neurons," *Neural Networks*, vol. 20, no. 3, pp. 335–352, 2007.

[18] Georg Holzmann, "Echo state networks with filter neurons and a delay&sum readout with applications in audio signal processing," M.S. thesis, Institute for Theoretical Computer Science, TU Graz, 2008, http://grh.mur.at/misc/MasterThesis.pdf.

[19] M. Lukosecicius and H. Jaeger, "Overview of reservoir recipes," Tech. Rep., 2007, Jacobs University Bremen, http://www.jacobs-university.de/imperia/md/content/groups/research/techreports/reservoiroverview_techreport11.pdf.

[20] C.W. Hsu, C. C. Chang, and C. J. Lin, "A practical guide to support vector classification," Tech. Rep., 2003, http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf.

[21] M. Schetzen, *The Volterra and Wiener Theories of Nonlinear Systems*, John Wiley, New York, 1980.

[22] Thomas Helie, "On the use of volterra series for real-time simulations of weakly nonlinear analog audio devices: application to the moog ladder filter," in *Proc. of the Int. Conf. on Digital Audio Effects (DAFx-06)*, 2006.

[23] A. Stenger and R. Rabenstein, "Adaptive volterra filters for nonlinear acoustic echo cancellation," Tech. Rep., 1999, http://www.ee.bilkent.edu.tr/~signal/Nsip99/papers/146.pdf.

[24] H. Schurer, C. Slump, and O. Herrmann, "Second order volterra inverses for compensation of loudspeaker nonlinearity," in *Proceedings of the IEEE ASSP Workshop on applications of signal processing to Audio & Acoustics*, 1995, pp. 74–78.

[25] W.G. Knecht, "Nonlinear noise filtering and beamforming using the perceptron and its volterra approximation," *IEEE Trans. Speech Audio Process*, pp. 55–62, 1994.

[26] Simon Godsill and Peter Rayner, *Digital Audio Restoration*, Springer, 1997, http://www-sigproc.eng.cam.ac.uk/%7Esjg/book/digital_audio_restoration.zip.

[27] W. Maass and E. D. Sontag, "Neural systems as nonlinear filters," *Neural Computation*, vol. 12, no. 8, pp. 1743–1772, 2000.

[28] Jyri Pakarinen and David Yeh, "A review of digital techniques for modeling vacuum-tube guitar amplifiers," *Computer Music Journal*, vol. 33, pp. 85–100, 2009.

[29] Swen Mueller and Paulo Massarani, "Transfer-function measurement with sweeps," *Audio Engineering Society*, vol. 49, pp. 443–471, 2001, http://www.anselmgoertz.de/Page10383/Monkey_Forest_dt/Manual_dt/Aes-swp.pdf.

[30] Stephan Moeller, Martin Gromowski, and Udo Zoelzer, "A measurement technique for highly nonlinear transfer functions," in *Proc. of the Int. Conf. on Digital Audio Effects (DAFx-02)*, 2002.

[31] Jonathan Abel and David Berners, "A technique for nonlinear system measurement," in *AES Convention 121*, 2006.

[32] Joern Schattschneider and Udo Zoelzer, "Discrete-time models for nonlinear audio systems," in *Proc. of the Int. Conf. on Digital Audio Effects (DAFx-99)*, 1999.

[33] Steve Harris, "Valve saturation ladspa plugin," Part of the plugin collection of Steve Harris, http://plugin.org.uk/ladspa-swh/docs/ladspa-swh.html, accessed Feb 22, 2009.

[34] Tom Szilagyi, "Tap tubewarmth ladspa plugin," Part of the TAP plugin collection of Tom Szilagyi, http://tap-plugins.sourceforge.net/ladspa/tubewarmth.html, accessed Feb 22, 2009.

[35] Herbert Jaeger, "Adaptive nonlinear system identification with echo state networks," *Advances in Neural Information Processing Systems 15, MIT Press*, pp. 593–600, 2003.

[36] Hubert Ausserlechner, "Untersuchungen von dropout-concealment-algorithmen," M.S. thesis, Institute for Electronic Music and Acoustics, Graz, Austria, 2006, http://www.iem.at/projekte/dsp/untersuchungen/index_html.

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

[38] Maciej Niedzwiecki and Krzysztof Cisowski, "Smart copying - a new approach to reconstruction of audio signals," *IEEE Transactions on Signal Processing*, vol. 49, pp. 2272–2282, 2001.

[39] Ellis, P. Daniel, Poliner, and E. Graham, "Classification-based melody transcription," *Machine Learning*, vol. 65, no. 2–3, pp. 439–456, 2006.

[40] G. Poliner, D.P.W. Ellis, A.F. Ehmann, E. Gomez, S. Streich, and Beesuan Ong, "Melody transcription from music audio: Approaches and evaluation," *IEEE Transactions on Audio Speech and Language Processing*, vol. 15, no. 4, pp. 1247–1256, 2007.

[41] G. Poliner and D. Ellis, "A classification approach to melody transcription," in *Proceedings of the 2005 International Conference on Music Information Retrieval*, 2005.

[42] Reservoir Lab Ghent, "Introduction to reservoir computing," http://snn.elis.ugent.be/rcbook, accessed March 30, 2009.