Echo State Networks for Signal Processing
Introduction
ESN Basics
ESNs vs. other RNNs
RNNs in Signal Processing

ESN Theory
Network Notation and Creation
Learning Algorithms
Additional Nonlinear Transformations
Short Term Memory Effects

Audio Signal Processing Examples
Nonlinear System Identification
Inverse Modeling
Audio Prediction

Conclusion
What are Echo State Networks?

Echo State Networks (ESNs) are special Recurrent Neural Networks (RNNs), with the following properties:

- a large, sparsely connected, RNN is used as a “reservoir” of dynamics (recurrent interconnected perceptrons)
- this Dynamic Reservoir can be excited by inputs and/or feedback of the outputs
- connection weights of the Reservoir are not changed by training
- only weights from the Reservoir to the output units are adapted, so training becomes a linear regression task
What are Echo State Networks?

Echo State Networks (ESNs) are special Recurrent Neural Networks (RNNs), with the following properties:

- a large, sparsely connected, RNN is used as a “reservoir” of dynamics (recurrent interconnected perceptrons)
- this Dynamic Reservoir can be excited by inputs and/or feedback of the outputs
- connection weights of the Reservoir are not changed by training
- only weights from the Reservoir to the output units are adapted, so training becomes a linear regression task
What are Echo State Networks?

Echo State Networks (ESNs) are special Recurrent Neural Networks (RNNs), with the following properties:

- a large, sparsely connected, RNN is used as a “reservoir” of dynamics (recurrent interconnected perceptrons)
- this Dynamic Reservoir can be excited by inputs and/or feedback of the outputs
- connection weights of the Reservoir are not changed by training
- only weights from the Reservoir to the output units are adapted, so training becomes a linear regression task
What are Echo State Networks?

Echo State Networks (ESNs) are special Recurrent Neural Networks (RNNs), with the following properties:

- a large, sparsely connected, RNN is used as a “reservoir” of dynamics (recurrent interconnected perceptrons)
- this Dynamic Reservoir can be excited by inputs and/or feedback of the outputs
- connection weights of the Reservoir are not changed by training
- only weights from the Reservoir to the output units are adapted, so training becomes a linear regression task
ESN example for a tuneable sinewave generator:

- black arrows are the fixed input and feedback connections
- red arrows are trainable output connections
- gray the recurrent interconnected Dynamic Reservoir
ESN example for a tuneable sinewave generator:

- black arrows are the fixed input and feedback connections
- red arrows are trainable output connections
- gray the recurrent interconnected Dynamic Reservoir
ESN example for a tuneable sinewave generator:

- black arrows are the fixed input and feedback connections
- red arrows are trainable output connections
- gray the recurrent interconnected Dynamic Reservoir
Problems with classical RNNs

Difficulties with existing algorithms have so far precluded supervised training techniques for RNNs from widespread use:

- unlike with Feedforward Neural Nets, several types of training algorithms are known for RNNs, no clear winner (e.g. Backpropagation Through Time, Real-Time Recurrent Learning, Extended Kalman Filtering Approaches, ...)
- usually these algorithms have a slow convergence and/or lead to suboptimal solutions
- ESN uses a large recurrent Reservoir (ca. 50-1000) whereas previous techniques typically use 5 to 30 neurons
Problems with classical RNNs

Difficulties with existing algorithms have so far precluded supervised training techniques for RNNs from widespread use:

- unlike with Feedforward Neural Nets, several types of training algorithms are known for RNNs, no clear winner (e.g. Backpropagation Through Time, Real-Time Recurrent Learning, Extended Kalman Filtering Approaches, ...)
- usually these algorithms have a slow convergence and/or lead to suboptimal solutions
- ESN uses a large recurrent Reservoir (ca. 50-1000) whereas previous techniques typically use 5 to 30 neurons
Problems with classical RNNs

Difficulties with existing algorithms have so far precluded supervised training techniques for RNNs from widespread use:

- Unlike with Feedforward Neural Nets, several types of training algorithms are known for RNNs, no clear winner (e.g. Backpropagation Through Time, Real-Time Recurrent Learning, Extended Kalman Filtering Approaches, ...)
- Usually these algorithms have a slow convergence and/or lead to suboptimal solutions
- ESN uses a large recurrent Reservoir (ca. 50-1000) whereas previous techniques typically use 5 to 30 neurons
ESN vs. classical RNNs

A: Schema of previous approaches to RNN learning

B: ESN approach
Why Recurrent Neural Networks?

- if one wishes to simulate, predict, filter, classify or control nonlinear dynamical systems, one needs an executable system model
- often it is infeasible to obtain analytical models, so one has to use blackbox modeling techniques
- for linear systems, efficient methods for blackbox modeling are available
- RNNs can be used to model nonlinear dynamical systems
Why Recurrent Neural Networks?

- if one wishes to simulate, predict, filter, classify or control nonlinear dynamical systems, one needs an executable system model.
- often it is infeasible to obtain analytical models, so one has to use blackbox modeling techniques.
- for linear systems, efficient methods for blackbox modeling are available.
- RNNs can be used to model nonlinear dynamical systems.
Why Recurrent Neural Networks?

- if one wishes to simulate, predict, filter, classify or control nonlinear dynamical systems, one needs an executable system model
- often it is infeasible to obtain analytical models, so one has to use blackbox modeling techniques
- for linear systems, efficient methods for blackbox modeling are available
- RNNs can be used to model nonlinear dynamical systems
Why Recurrent Neural Networks?

- if one wishes to simulate, predict, filter, classify or control nonlinear dynamical systems, one needs an executable system model
- often it is infeasible to obtain analytical models, so one has to use blackbox modeling techniques
- for linear systems, efficient methods for blackbox modeling are available
- RNNs can be used to model nonlinear dynamical systems
Why ESNs in Signal Processing?

- Classical methods of statistical signal processing are founded on three basic assumptions: linearity, stationarity and Gaussianity.
- These assumptions are made for the sake of mathematical tractability.
- Most real-life physical signals are generated by dynamic processes that are nonlinear, nonstationary and non-Gaussian.
Why ESNs in Signal Processing?

- classical methods of statistical signal processing are founded on three basic assumptions: linearity, stationarity and Gaussianity
- these assumptions are made for the sake of mathematical tractability
- most real-life physical signals are generated by dynamic processes that are nonlinear, nonstationary and non-Gaussian
Why ESNs in Signal Processing?

- Classical methods of statistical signal processing are founded on three basic assumptions: linearity, stationarity, and Gaussianity.
- These assumptions are made for the sake of mathematical tractability.
- Most real-life physical signals are generated by dynamic processes that are nonlinear, nonstationary, and non-Gaussian.
Why ESNs in Signal Processing?

- classical approaches for nonlinear signal processing consists of design specific algorithms for specific problems (e.g. median and bilinear filters, Volterra filters, polynomial filters, functional links, ...)

- ESNs and in general RNNs extend and generalize the simple adaptive linear filter in nonlinear domain and can be used to model any nonlinear dynamical system
Why ESNs in Signal Processing?

- classical approaches for nonlinear signal processing consists of design specific algorithms for specific problems (e.g. median and bilinear filters, Volterra filters, polynomial filters, functional links, ...)

- ESNs and in general RNNs extend and generalize the simple adaptive linear filter in nonlinear domain and can be used to model any nonlinear dynamical system
Possible Applications

Some possible applications of ESNs, especially in Audio Signal Processing, are:

- nonlinear adaptive filtering techniques
- audio signal prediction and restauration, quality enhancement
- system identification and simulation of nonlinear amplifiers, tubes, nonlinear transducer linearization, modelling of audio effects etc.
- learning based sound synthesis
- audio coding
- ...
Some possible applications of ESNs, especially in Audio Signal Processing, are:

- nonlinear adaptive filtering techniques
- audio signal prediction and restauration, quality enhancement
- system identification and simulation of nonlinear amplifiers, tubes, nonlinear transducer linearization, modelling of audio effects etc.
- learning based sound synthesis
- audio coding
- ...

Echo State Networks for Signal Processing
Some possible applications of ESNs, especially in Audio Signal Processing, are:

▶ nonlinear adaptive filtering techniques
▶ audio signal prediction and restoration, quality enhancement
▶ system identification and simulation of nonlinear amplifiers, tubes, nonlinear transducer linearization, modeling of audio effects etc.
▶ learning based sound synthesis
▶ audio coding
▶ ...
Some possible applications of ESNs, especially in Audio Signal Processing, are:

- nonlinear adaptive filtering techniques
- audio signal prediction and restoration, quality enhancement
- system identification and simulation of nonlinear amplifiers, tubes, nonlinear transducer linearization, modeling of audio effects etc.
- learning based sound synthesis
- audio coding
- ...
Possible Applications

Some possible applications of ESNs, especially in Audio Signal Processing, are:

- nonlinear adaptive filtering techniques
- audio signal prediction and restauration, quality enhancement
- system identification and simulation of nonlinear amplifiers, tubes, nonlinear transducer linearization, modelling of audio effects etc.
- learning based sound synthesis
- audio coding
- ...

Echo State Networks for Signal Processing
Possible Applications

Some possible applications of ESNs, especially in Audio Signal Processing, are:

- nonlinear adaptive filtering techniques
- audio signal prediction and restauration, quality enhancement
- system identification and simulation of nonlinear amplifiers, tubes, nonlinear transducer linearization, modelling of audio effects etc.
- learning based sound synthesis
- audio coding
- ...
SISO Systems

For simplifying notation we only address the task of modeling single-input, single-output systems without feedback connections:

These systems are quite common in signal processing.
Reservoir with $N$ internal network units

- at time $n \geq 1$ the input is $u(n)$ and the output $y(n)$
- activations of the internal units are a $N \times 1$ vector:
  \[ x(n) = (x_1(n), \ldots, x_N(n)) \]
- internal connection weights are collected in a $N \times N$ matrix $W$
- weights of input connections in a $N \times 1$ vector $w^{in}$
- output weights in a $(N + 1) \times 1$ vector $w^{out}$
  (input and network to output connections)
ESN Notation

- Reservoir with $N$ internal network units
- at time $n \geq 1$ the input is $u(n)$ and the output $y(n)$
- activations of the internal units are a $N \times 1$ vector:
  $x(n) = (x_1(n), ..., x_N(n))$
- internal connection weights are collected in a $N \times N$ matrix $W$
- weights of input connections in a $N \times 1$ vector $w^{in}$
- output weights in a $(N + 1) \times 1$ vector $w^{out}$
  (input and network to output connections)
ESN Notation

- Reservoir with $N$ internal network units
- at time $n \geq 1$ the input is $u(n)$ and the output $y(n)$
- activations of the internal units are a $N \times 1$ vector: $x(n) = (x_1(n), ..., x_N(n))$
- internal connection weights are collected in a $N \times N$ matrix $W$
- weights of input connections in a $N \times 1$ vector $w^{in}$
- output weights in a $(N + 1) \times 1$ vector $w^{out}$ (input and network to output connections)
ESN Notation

- Reservoir with $N$ internal network units
- At time $n \geq 1$ the input is $u(n)$ and the output $y(n)$
- Activations of the internal units are a $N \times 1$ vector:
  \[ x(n) = (x_1(n), \ldots, x_N(n)) \]
- Internal connection weights are collected in a $N \times N$ matrix $W$
- Weights of input connections in a $N \times 1$ vector $w^{in}$
- Output weights in a $(N + 1) \times 1$ vector $w^{out}$ (input and network to output connections)
ESN Notation

- Reservoir with $N$ internal network units
- at time $n \geq 1$ the input is $u(n)$ and the output $y(n)$
- activations of the internal units are a $N \times 1$ vector: $x(n) = (x_1(n), ..., x_N(n))$
- internal connection weights are collected in a $N \times N$ matrix $W$
- weights of input connections in a $N \times 1$ vector $w^{in}$
- output weights in a $(N + 1) \times 1$ vector $w^{out}$ (input and network to output connections)
Reservoir with $N$ internal network units
- at time $n \geq 1$ the input is $u(n)$ and the output $y(n)$
- activations of the internal units are a $N \times 1$ vector:
  \[ x(n) = (x_1(n), \ldots, x_N(n)) \]
- internal connection weights are collected in a $N \times N$ matrix $W$
- weights of input connections in a $N \times 1$ vector $w^{in}$
- output weights in a $(N + 1) \times 1$ vector $w^{out}$
  (input and network to output connections)
The activation of internal and output units are updated according to:

\[ x(n+1) = f(Wx(n) + w^{in}u(n+1) + v(n+1)) \]
\[ y(n+1) = f^{out}(w^{out}(u(n+1), x(n+1))) \]

\( v(n+1) \) ... small noise term
\( f \) ... nonlinearity in the Dynamic Reservoir (e.g. tanh)
\( f^{out} \) ... output unit activation function (also tanh or linear, ...)
Under certain conditions the network state becomes asymptotically independent of initial conditions and depends only on input history:

- let $W$ have a spectral radius $|\lambda_{max}| > 1$, with $\lambda_{max}$ as largest eigenvalue of $W$
- then the network does NOT have the echo state property for any input/output interval $UxD$ containing the zero input/output
Under certain conditions the network state becomes asymptotically independent of initial conditions and depends only on input history:

- let $W$ have a spectral radius $|\lambda_{\text{max}}| > 1$, with $\lambda_{\text{max}}$ as largest eigenvalue of $W$
- then the network does NOT have the echo state property for any input/output interval $UxD$ containing the zero input/output
In practice it was consistently found, that when the previous proposition is NOT satisfied, we do have an echo state network:

- usually $W$ is randomly generated from a uniform distribution over $[-1, 1]$.
- $W$ is then normalized to the spectral radius $\alpha < 1$ by scaling $W$ with $\alpha/|\lambda_{max}|$.
- the spectral radius $\alpha$ is a crucial parameter for the eventual success of an ESN: small $\alpha$ are for fast signals, big $\alpha$ for slow signals and a longer short term memory.
In practice it was consistently found, that when the previous proposition is NOT satisfied, we do have an echo state network:

- usually $W$ is randomly generated from a uniform distribution over $[-1, 1]$
- $W$ is then normalized to the spectral radius $\alpha < 1$ by scaling $W$ with $\alpha/|\lambda_{max}|$
- the spectral radius $\alpha$ is a crucial parameter for the eventual success of an ESN: small $\alpha$ are for fast signals, big $\alpha$ for slow signals and a longer short term memory
In practice it was consistently found, that when the previous proposition is NOT satisfied, we do have an echo state network:

- usually $W$ is randomly generated from a uniform distribution over $[-1, 1]$
- $W$ is then normalized to the spectral radius $\alpha < 1$ by scaling $W$ with $\alpha/|\lambda_{max}|$
- the spectral radius $\alpha$ is a crucial parameter for the eventual success of an ESN: small $\alpha$ are for fast signals, big $\alpha$ for slow signals and a longer short term memory
In training we compute the output weights, such that the training error is minimized in mean square sense:

\[ e_{\text{train}}(n) = (f^{\text{out}})^{-1} y_{\text{teach}}(n) - w^{\text{out}}(u_{\text{teach}}(n), x(n)) \]

where the effect of the output nonlinearity is undone by \((f^{\text{out}})^{-1}\). Therefore the computation of \(w^{\text{out}}\) is a linear regression task which can be calculated offline or online.
Offline Learning Algorithm

- init $W$ with spectral radius $\alpha < 1$ and run the ESN with the teaching input signal
- dismiss data from initial transients and collect remaining input and network states $(u_{teach}(n), x_{teach}(n))$ row-wise into a matrix $M$
- collect the training signals $(f_{out})^{-1}y_{teach}(n)$ into a vector $r$
- compute the pseudo-inverse $M^{-1}$ and put $w_{out} = (M^{-1}r)^T$
- the ESN is now trained
Offline Learning Algorithm

- init $W$ with spectral radius $\alpha < 1$ and run the ESN with the teaching input signal
- dismiss data from initial transients and collect remaining input and network states $(u_{\text{teach}}(n), x_{\text{teach}}(n))$ row-wise into a matrix $M$
- collect the training signals $(f^{\text{out}})^{-1}y_{\text{teach}}(n)$ into a vector $r$
- compute the pseudo-inverse $M^{-1}$ and put $w^{\text{out}} = (M^{-1}r)^T$
- the ESN is now trained
Offline Learning Algorithm

- init $W$ with spectral radius $\alpha < 1$ and run the ESN with the teaching input signal
- dismiss data from initial transients and collect remaining input and network states $(u_{teach}(n), x_{teach}(n))$ row-wise into a matrix $M$
- collect the training signals $(f^{out})^{-1}y_{teach}(n)$ into a vector $r$
- compute the pseudo-inverse $M^{-1}$ and put $w^{out} = (M^{-1}r)^T$
- the ESN is now trained
Offline Learning Algorithm

- init $W$ with spectral radius $\alpha < 1$ and run the ESN with the teaching input signal
- dismiss data from initial transients and collect remaining input and network states $(u_{teach}(n), x_{teach}(n))$ row-wise into a matrix $M$
- collect the training signals $(f^{out})^{-1}y_{teach}(n)$ into a vector $r$
- compute the pseudo-inverse $M^{-1}$ and put $w^{out} = (M^{-1}r)^T$
- the ESN is now trained
Offline Learning Algorithm

- **init** $W$ with spectral radius $\alpha < 1$ and run the ESN with the teaching input signal
- dismiss data from initial transients and collect remaining input and network states $(u_{teach}(n), x_{teach}(n))$ row-wise into a matrix $M$
- collect the training signals $(f_{out})^{-1}y_{teach}(n)$ into a vector $r$
- compute the pseudo-inverse $M^{-1}$ and put $w^{out} = (M^{-1}r)^T$
- the ESN is now trained
Additional Nonlinearities

- the modeling power of an ESN grows with network size
- for hard nonlinear tasks a cheaper way to increase the power is to use additional nonlinear transformations of the network state $x(n)$
- so the new internal state vector is $x_{\text{square}}(n) = (x_1(n), ..., x_N(n), x_1^2(n), ..., x_N^2(n))$ and the new input vector is $u_{\text{square}}(n) = (u(n), u^2(n))$
- this needs an output weight vector $w_{\text{out}}^{\text{square}}$ which is two times longer as $w^{\text{out}}$
the modeling power of an ESN grows with network size

for hard nonlinear tasks a cheaper way to increase the power is to use additional nonlinear transformations of the network state $x(n)$

so the new internal state vector is $x_{\text{square}}(n) = (x_1(n), ..., x_N(n), x_1^2(n), ..., x_N^2(n))$ and the new input vector is $u_{\text{square}}(n) = (u(n), u^2(n))$

this needs an output weight vector $w_{\text{out square}}$ which is two times longer as $w_{\text{out}}$
the modeling power of an ESN grows with network size

for hard nonlinear tasks a cheaper way to increase the power is to use additional nonlinear transformations of the network state $x(n)$

so the new internal state vector is $x_{\text{square}}(n) = (x_1(n), \ldots, x_N(n), x_1^2(n), \ldots, x_N^2(n))$ and the new input vector is $u_{\text{square}}(n) = (u(n), u^2(n))$

this needs an output weight vector $w_{\text{out}}^{\text{square}}$ which is two times longer as $w_{\text{out}}$
Additional Nonlinearities

- the modeling power of an ESN grows with network size
- for hard nonlinear tasks a cheaper way to increase the power is to use additional nonlinear transformations of the network state $x(n)$
- so the new internal state vector is $x_{\text{square}}(n) = (x_1(n), ..., x_N(n), x_1^2(n), ..., x_N^2(n))$ and the new input vector is $u_{\text{square}}(n) = (u(n), u^2(n))$
- this needs an output weight vector $w_{\text{out}}^{\text{square}}$ which is two times longer as $w^{\text{out}}$
Learning with Additional Nonlinearities

The learning procedure remains linear and now goes like this (similar for the online version):

- drive the ESN with training input, collect states $x_{square}(n)$ row-wise into $M$ and the teacher signal $(f^{out})^{-1}y_{teach}(n)$ into a vector $r$
- compute pseudo-inverse $M^{-1}$ and put $w^{out}_{square} = (M^{-1}r)^T$
- ESN is now trained, the output can be calculated with:

$$y(n + 1) = f^{out}(w^{out}_{square}(u_{square}(n + 1), x_{square}(n + 1)))$$
Learning with Additional Nonlinearities

The learning procedure remains linear and now goes like this (similar for the online version):

- drive the ESN with training input, collect states $x_{\text{square}}(n)$ row-wise into $M$ and the teacher signal $(f^{\text{out}})^{-1}y_{\text{teach}}(n)$ into a vector $r$
- compute pseudo-inverse $M^{-1}$ and put $w_{\text{square}}^{\text{out}} = (M^{-1}r)^T$
- ESN is now trained, the output can be calculated with:

\[ y(n + 1) = f^{\text{out}}(w_{\text{square}}^{\text{out}}(u_{\text{square}}(n + 1), x_{\text{square}}(n + 1))) \]
Learning with Additional Nonlinearities

The learning procedure remains linear and now goes like this (similar for the online version):

- drive the ESN with training input, collect states $x_{\text{square}}(n)$ row-wise into $M$ and the teacher signal $(f^{\text{out}})^{-1}y_{\text{teach}}(n)$ into a vector $r$
- compute pseudo-inverse $M^{-1}$ and put $w_{\text{square}}^{\text{out}} = (M^{-1}r)^T$
- ESN is now trained, the output can be calculated with:

$$y(n + 1) = f^{\text{out}} (w_{\text{square}}^{\text{out}}(u_{\text{square}}(n + 1), x_{\text{square}}(n + 1)))$$
Short Term Memory in ESNs

- short term memory means, how many of the previous input/output arguments \((u(n - k), y(n - k - 1))\) are actually relevant for the current output

- for a simple delay task, we can consider the squared correlation coefficient \(r^2(u(n - k), y_k(n))\) between the correct delayed signal \(u(n - k)\) and the network output \(y_k(n)\) trained on the delay \(k\)

- now a values of 1 means perfect correlation and 0 complete loss of correlation
Short Term Memory in ESNs

- Short term memory means, how many of the previous input/output arguments \((u(n - k), y(n - k - 1))\) are actually relevant for the current output.

- For a simple delay task, we can consider the squared correlation coefficient \(r^2(u(n - k), y_k(n))\) between the correct delayed signal \(u(n - k)\) and the network output \(y_k(n)\) trained on the delay \(k\).

- Now a values of 1 means perfect correlation and 0 complete loss of correlation.
Short Term Memory in ESNs

- Short term memory means, how many of the previous input/output arguments \((u(n - k), y(n - k - 1))\) are actually relevant for the current output.

- For a simple delay task, we can consider the squared correlation coefficient \(r^2(u(n - k), y_k(n))\) between the correct delayed signal \(u(n - k)\) and the network output \(y_k(n)\) trained on the delay \(k\).

- Now a values of 1 means perfect correlation and 0 complete loss of correlation.
we can define the memory capacity $MC$ of a network by

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

it was proven that for an ESN whose Dynamic Reservoir has $N$ nodes, the maximal possible memory capacity is bounded by $N$:

$$MC \leq N$$
we can define the memory capacity $MC$ of a network by

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

it was proofen that for a ESN whose Dynamic Reservoir has $N$ nodes, the maximal possible memory capacity is bounded by $N$:

$$MC \leq N$$
Forgetting Curves

When we plot the correlation coefficient against the delay $k$, we obtain the forgetting curves (here a 400-unit ESN):

- **A**: randomly created linear DR
- **B**: randomly created sigmoid DR
- **C**: like A, but with noisy state update of the DR
- **D**: linear, almost unitary weight matrix
- **E**: as D, but with noisy state update
- **F**: as D, but with spectral radius $\alpha = 0.999$
Practical suggestions

When one needs 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 linear update for the DR (might conflict with nonlinear modelling tasks)
- use DRs with almost unitary weight matrices
- use a spectral radius $\alpha$ close to 1 (would not work if one wants to have fast oscillating dynamics)
Practical suggestions

When one needs 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 linear update for the DR (might conflict with nonlinear modelling tasks)
- use DRs with almost unitary weight matrices
- use a spectral radius $\alpha$ close to 1 (would not work if one wants to have fast oscillating dynamics)
Practical suggestions

When one needs 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 linear update for the DR (might conflict with nonlinear modelling tasks)
- use DRs with almost unitary weight matrices
- use a spectral radius $\alpha$ close to 1 (would not work if one wants to have fast oscillating dynamics)
Practical suggestions

When one needs 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 linear update for the DR (might conflict with nonlinear modelling tasks)
▶ use DRs with almost unitary weight matrices
▶ use a spectral radius $\alpha$ close to 1 (would not work if one wants to have fast oscillating dynamics)
Nonlinear System Identification

- like in linear adaptive signal processing we try to identify an input-output system, but nonlinear in this case
- in the first example we simulate the behaviour a valve (tube) distortion
- in the second example we try to learn a hard nonlinear system (hard clipping)
Nonlinear System Identification

- like in linear adaptive signal processing we try to identify an input-output system, but nonlinear in this case
- in the first example we simulate the behaviour a valve (tube) distortion
- in the second example we try to learn a hard nonlinear system (hard clipping)
Nonlinear System Identification

- like in linear adaptive signal processing we try to identify an input-output system, but nonlinear in this case
- in the first example we simulate the behaviour a valve (tube) distortion
- in the second example we try to learn a hard nonlinear system (hard clipping)
the task of audio prediction is to compute future samples out of previous input samples

this task is e.g. necessary in audio restoration, whenever a sequence of consecutive samples is missing or when impulsive noise appears

usually a preproccession stage localizes the noise or missing sample position and then a reconstruction stage is needed
Audio Prediction

- the task of audio prediction is to compute future samples out of previous input samples
- this task is e.g. necessary in audio restoration, whenever a sequence of consecutive samples is missing or when impulsive noise appears
- usually a preprocession stage localizes the noise or missing sample position and then a reconstruction stage is needed
Audio Prediction

- the task of audio prediction is to compute future samples out of previous input samples
- this task is e.g. necessary in audio restoration, whenever a sequence of consecutive samples is missing or when impulsive noise appears
- usually a preproccession stage localizes the noise or missing sample position and then a reconstruction stage is needed
common methods for missing data reconstruction are based on the assumption that signals can be modeled as a P order autoregressive process

- the model parameters are taps of a linear predictor filter
- missing samples are predicted in forward and backward mode and crossfaded afterwards
- the main drawback of an autoregressive model based approach is the big model order of two or three times the length of missing data
common methods for missing data reconstruction are based on the assumption that signals can be modeled as a P order autoregressive process

- the model parameters are taps of a linear predictor filter
- missing samples are predicted in forward and backward mode and crossfaded afterwards
- the main drawback of an autoregressive model based approach is the big model order of two or three times the length of missing data
common methods for missing data reconstruction are based on the assumption that signals can be modeled as a P order autoregressive process

the model parameters are taps of a linear predictor filter

missing samples are predicted in forward and backward mode and crossfaded afterwards

the main drawback of an autoregressive model based approach is the big model order of two or three times the length of missing data
Common Prediction Methods

- common methods for missing data reconstruction are based on the assumption that signals can be modeled as a P order autoregressive process
- the model parameters are taps of a linear predictor filter
- missing samples are predicted in forward and backward mode and crossfaded afterwards
- the main drawback of an autoregressive model based approach is the big model order of two or three times the length of missing data
Forward-Backward Prediction Schema

Diagram showing the forward and backward prediction processes with cross-fade gain and reconstructed signal.
ESN Setup for Signal Prediction

For the signal prediction task, we want an ESN without input and one output unit.

- This output unit features connections that project back the teacher signal into the Dynamic Reservoir.
- The weights of these backprojections are also not changed during training.
- For training, the teacher signal is pumped into the DR and thereby excites activation dynamics within the DR (the “echos” of the signal).
ESN Setup for Signal Prediction

- For the signal prediction task we want an ESN without input and one output unit.
- This output unit features connections that project back the teacher signal into the Dynamic Reservoir.
- The weights of these backprojections are also not changed during training.
- For training the teacher signal is pumped into the DR and thereby excites activation dynamics within the DR (the “echos” of the signal).
ESN Setup for Signal Prediction

- For the signal prediction task, we want an ESN without input and one output unit.
- This output unit features connections that project back the teacher signal into the Dynamic Reservoir.
- The weights of these backprojections are also not changed during training.
- For training, the teacher signal is pumped into the DR and thereby excites activation dynamics within the DR (the “echos” of the signal).
ESN Setup for Signal Prediction

- For the signal prediction task, we want an ESN without input and one output unit.
- This output unit features connections that project back the teacher signal into the Dynamic Reservoir.
- The weights of these backprojections are also not changed during training.
- For training, the teacher signal is pumped into the DR and thereby excites activation dynamics within the DR (the “echos” of the signal).
Teacher Forcing

the ESN setup:

- to predict a signal we drive the ESN with the teacher audio signal (teacher forcing)
- then we let the ESN predict the future signal from the echos of the teacher signal
Teacher Forcing

the ESN setup:

▶ to predict a signal we drive the ESN with the teacher audio signal (teacher forcing)
▶ then we let the ESN predict the future signal from the echos of the teacher signal
I tried ESNs with 100 and 400 neurons in the DR (connectivity 20% and 5%)

to get a better short-term memory performance the DR was initiated with an “almost” unitary weight matrix $W$
	his was done by replacing the singular value diagonal matrix from octaves/matlab's SVD function by the identity matrix, then the resulting matrix was scaled with the factor $C = 0.9$
a linear activation function was used in the DR (again to get better short-term memory), the output neurons had a tanh activation function

feedback weights $w_{fb}$ were sampled from a uniform distribution over (-0.5, 0.5)
I tried ESNs with 100 and 400 neurons in the DR (connectivity 20% and 5%)

to get a better short-term memory performance the DR was initiated with an “almost“ unitary weight matrix \( W \)

this was done by replacing the singular value diagonal matrix from octaves/matlabs SVD function by the identity matrix, then the resulting matrix was scaled with the factor \( C = 0.9 \)

a linear activation function was used in the DR (again to get better short-term memory), the output neurons had a tanh activation function

feedback weights \( w_{fb} \) were sampled from a uniform distribution over \((-0.5, 0.5)\)
Signal Prediction, ESN Setup

- I tried ESNs with 100 and 400 neurons in the DR (connectivity 20% and 5%)
- to get a better short-term memory performance the DR was initiated with an “almost“ unitary weight matrix $W$
- this was done by replacing the singular value diagonal matrix from octaves/matlab's SVD function by the identity matrix, then the resulting matrix was scaled with the factor $C = 0.9$
- a linear activation function was used in the DR (again to get better short-term memory), the output neurons had a tanh activation function
- feedback weights $w_{fb}$ were sampled from a uniform distribution over (-0.5, 0.5)
I tried ESNs with 100 and 400 neurons in the DR (connectivity 20% and 5%)

to get a better short-term memory performance the DR was initiated with an “almost” unitary weight matrix $W$

this was done by replacing the singular value diagonal matrix from octaves/matlab's SVD function by the identity matrix, then the resulting matrix was scaled with the factor $C = 0.9$

a linear activation function was used in the DR (again to get better short-term memory), the output neurons had a tanh activation function

feedback weights $w_{fb}$ were sampled from a uniform distribution over (-0.5, 0.5)
I tried ESNs with 100 and 400 neurons in the DR (connectivity 20% and 5%)

- to get a better short-term memory performance the DR was initiated with an “almost“ unitary weight matrix $W$
- this was done by replacing the singular value diagonal matrix from octaves/matlabs SVD function by the identity matrix, then the resulting matrix was scaled with the factor $C = 0.9$
- a linear activation function was used in the DR (again to get better short-term memory), the output neurons had a tanh activation function
- feedback weights $w_{fb}$ were sampled from a uniform distribution over (-0.5, 0.5)
I tried two different test signals:

- first test signal was a steady tone of two violins with some noise
- second test signal was a band (drums, bass, keyboard, flute), which played music with a beat
- the ESN was first trained for forward prediction, then for backward prediction and finally these two generated signals were crossfaded
I tried two different testsignals:

- first testsignal was a steady tone of two violins with some noise
- second testsignal was a band (drums, bass, keyboard, flute), which played music with a beat
- the ESN was first trained for forward prediction, then for backward prediction and finally these two generated signals were crossfaded
I tried two different testsignals:

- first testsignal was a steady tone of two violins with some noise
- second testsignal was a band (drums, bass, keyboard, flute), which played music with a beat
- the ESN was first trained for forward prediction, then for backward prediction and finally these two generated signals were crossfaded
String Prediction 200 samples, DR Size 100

here we try to predict 200 samples, which is the theoretical limit of short term memory ($2^N$, forward and backward prediction)

$$\text{Forward } MSE_{test} = 0.0297, \text{ Backward } MSE_{test} = 0.0342, \text{ Combined } MSE_{test} = 0.0332$$
An other interesting thing is that the MSE is still the same, although the sound quality is different:

Forward $MSE_{\text{test}} = 0.0363$, Backward $MSE_{\text{test}} = 0.0287$, Combined $MSE_{\text{test}} = 0.0262$

for stationary sounds, like these strings, that behaviour is quite usefull
String Prediction 10000 samples

backward and forward prediction with DR size 100 (left) and 400 (right):

![Graphs showing prediction results](image)

DR100 Combined $MSE_{test} = 0.0365$
DR400 Combined $MSE_{test} = 0.0266$
Beat Sample Prediction

predicting a sample including beat music (at the short term memory limit); left with DR size 100 predicting 200 samples, right DR size 400 predicting 800 samples:

DR100 Combined $MSE_{test} = 0.0113$, DR400 Combined $MSE_{test} = 0.0225$
Beat Sample Prediction, 4000 samples, DR Size 400

backward and forward prediction with linear (left) and tansig (right):

linear Combined $MSE_{test} = 0.0403$

tansig Combined $MSE_{test} = 0.0544$
Beat Sample Prediction, 10000 samples

backward and forward prediction with DR size 100 (left) and 400 (right):

\[
\text{linear Combined } MSE_{test} = 0.0403 \\
\text{tansig Combined } MSE_{test} = 0.0544
\]
ESN in Audio Processing

In the presented examples following observations were made:

▶ the nonlinear system identification and audio prediction task works quite well
▶ however, a real comparison to other algorithms needs to be done
▶ in the nonlinear system identification task it was very important to use quadratic state updates
▶ in all these examples the algorithm was very robust (exception: inverse modeling, there the success heavily depends on training size)
In the presented examples following observations were made:

- the nonlinear system identification and audio prediction task works quite well
- however, a real comparison to other algorithms needs to be done
- in the nonlinear system identification task it was very important to use quadratic state updates
- in all these examples the algorithm was very robust (exception: inverse modeling, there the success heavily depends on training size)
ESN in Audio Processing

In the presented examples following observations were made:

▶ the nonlinear system identification and audio prediction task works quite well
▶ however, a real comparison to other algorithms needs to be done
▶ in the nonlinear system identification task it was very important to use quadratic state updates
▶ in all these examples the algorithm was very robust (exception: inverse modeling, there the success heavily depends on training size)
In the presented examples following observations were made:

- the nonlinear system identification and audio prediction task works quite well
- however, a real comparison to other algorithms needs to be done
- in the nonlinear system identification task it was very important to use quadratic state updates
- in all these examples the algorithm was very robust (exception: inverse modeling, there the success heavily depends on training size)
Problems

- the MSE is not always a useful measure of the sound quality, our perception works different
- it is easy to generate additional high partials of a sound, but hard to undo this process, maybe other activation functions are needed or a processing in frequency domain
- e.g. it was not possible to learn the inverse system of the tube simulation or hard clipping - the MSE and the plots were quite good, but the sound was totally distorted
- for long predictions the ESN output changes into a state reproducing only the periodic parts of an input sound - this might be useful or not
Problems

- the MSE is not always a useful measure of the sound quality, our perception works different
- it is easy to generate additional high partials of a sound, but hard to undo this process, maybe other activation functions are needed or a processing in frequency domain
- e.g. it was not possible to learn the inverse system of the tube simulation or hard clipping - the MSE and the plots were quite good, but the sound was totally distorted
- for long predictions the ESN output changes into a state reproducing only the periodic parts of an input sound - this might be useful or not
Problems

- the MSE is not always a useful measure of the sound quality, our perception works different
- it is easy to generate additional high partials of a sound, but hard to undo this process, maybe other activation functions are needed or a processing in frequency domain
- e.g. it was not possible to learn the inverse system of the tube simulation or hard clipping - the MSE and the plots were quite good, but the sound was totally distorted
- for long predictions the ESN output changes into a state reproducing only the periodic parts of an input sound - this might be useful or not
Problems

- the MSE is not always a useful measure of the sound quality, our perception works different
- it is easy to generate additional high partials of a sound, but hard to undo this process, maybe other activation functions are needed or a processing in frequency domain
- e.g. it was not possible to learn the inverse system of the tube simulation or hard clipping - the MSE and the plots were quite good, but the sound was totally distorted
- for long predictions the ESN output changes into a state reproducing only the periodic parts of an input sound - this might be useful or not
Simon Haykin; *Neural Networks Expand SP’s Horizons*; 1996, IEEE Signal Processing Magazine

Herbert Jaeger; *Adaptive nonlinear system identification with echo state networks*; 2003, Frauenhofer Institut for Autonomous Intelligent Systems

Herbert Jaeger; *A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the “echo state network” approach*; 2002, Frauenhofer Institut for Autonomous Intelligent Systems

Herbert Jaeger, Harald Hass; *Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication*; 2004, International University Bremen

Aurelio Uncini; *Audio signal processing by neural networks*; 2003, University of Rome
Questions