Mathematics and Physics Engineering
Bachelor's Final Project
Echo State Networks: implementation and applications

Alejandro Martínez Sánchez
directed by:
Daniel P. Palomar
UPC tutor:
Jordi Castro

May 2019

Abstract:
Reservoir computing has been one of the most prolific research fields during the first decade of 2000. One of the main representatives of this trend are Echo State Networks (ESN) which are a model for supervised learning. This model is older than the recent approaches of deep learning models but it still useful because of its simplicity and easy training.

This project makes a theoretical analysis of Echo State Networks some of the mos relevant aspects of ESN. It also presents some basic experiment with ESN and the development of a package using the R programming language. The project ends with the presentation of a possible application in finance of the Echo State Network.

Keywords: Supervised Learning, Echo State Networks, Neural Networks, Time-series forecasting.

Contents

1 Introduction

A wide variety of methods for non-linear operation are available thanks to advances in the latest trends such as the backpropagation algorithm presented in [1]. The aim of this project is exploring some technique for supervised learning with time series. The first option, and probably the most known, can be using Recurrent Neural Networks (RNN) which have had a significant improvement thanks to the application of the back propagation algorithm as presented by [2] and [3]. Nevertheless, the RNN present several disadvantages such as a huge computation time and the need of a very big amount of data for being trained. A viable alternative can be the Echo State Networks (ESN) that appeared a few years ago as can be seen in [4] and [5], with some of the first results exploring the memory capacity of the ESN [4] and [12].

The main idea of the ESN is similar to the kernel trick. This idea consists in make a modification of the input and go to a new space of bigger dimension where the task will be much more easy to solve, see Figure 1. This modification of the input will be called reservoir and the trend started with Echo State Networks and Liquid State Machines will be lately known as Reservoir Computing [6].

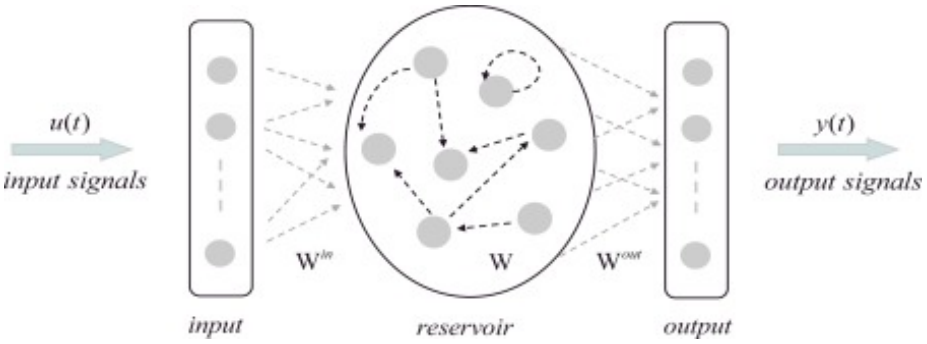


Figure 1: Basic diagram of a ESN.

Some great success of the ESN can be seen at [7], [8] and, specifically in the case of stock price prediction, [9]. During the last decade the performance of the RNN improved [10] despite this the ESN still useful being able to outperform RNN in some specific tasks [11]. Moreover, the ESN is a more intuitive and more interpretable model than the ESN [13].

2 Theory of Echo State Networks

2.1 Basic definitions and equations

First we start with the basic definitions. ESN are a model for supervised learning then an input and output signals are required. In our case we are going to call $\mathbf{u}(t) \in \mathbb{R}^K$ to the input signal and $\mathbf{y}(t) \in \mathbb{R}^L$ to the output and the internal units of the ESN will be $\mathbf{x}(t) \in \mathbb{R}^N$. Usually a unit with constant value 1 will be added to the input $\mathbf{u}(t)$ as bias term. Let $\mathbf{W}^{\mathsf{in}} \in \mathbb{R}^{N \times (K+1)}$ the weights connecting the input with the internal state and $\mathbf{W} \in \mathbb{R}^{N \times N}$ the connections of the internal state. The output weight matrix will be the only weights trained for the ESN, this weights connect the input and reservoir with the output of the net, so $\mathbf{W}^{\mathsf{out}}$

The equation for the the internal state is:

$$\mathbf{x}(t+1) = \mathbf{f}(\mathbf{W}^{\mathsf{in}} \begin{bmatrix} 1 \\ \mathbf{u}(t) \end{bmatrix} + \mathbf{W}\mathbf{x}(t+1)) \tag{1}$$

where $\mathbf{f}$ are the internal unit's output functions which typically will be $\mathsf{tanh}()$. The equation for the output of the ESN is:

$$\hat{\mathbf{y}}(t) = \mathbf{W}^{\mathsf{out}} \begin{bmatrix} 1 \\ \mathbf{u}(t) \\ \mathbf{x}(t) \end{bmatrix} \tag{2}$$

Let's consider all the possible input sequences lying on an input set $(\mathbf{u}(t))_{t \in I} \in U^I$, where I is just an index set. For all of the important results we require U to be compact. The notation for some input sequences will be $\bar{\mathbf{u}}^{\pm\infty}$, $\bar{\mathbf{u}}^{+\infty}$, $\bar{\mathbf{u}}^{-\infty}$ and $\bar{\mathbf{u}}^{h}$ to denote infinite sequences ($I = \mathbb{Z}$), right-infinite sequences ($I = k, k+1, \ldots$ for some $k \in \mathbb{Z}$) left-infinite sequences ($I = \ldots, k-1, k$ for some $k \in \mathbb{Z}$) and finite sequence of length h, ($I = k, k+1, \ldots, k+h-1$ for some $k \in \mathbb{Z}$).

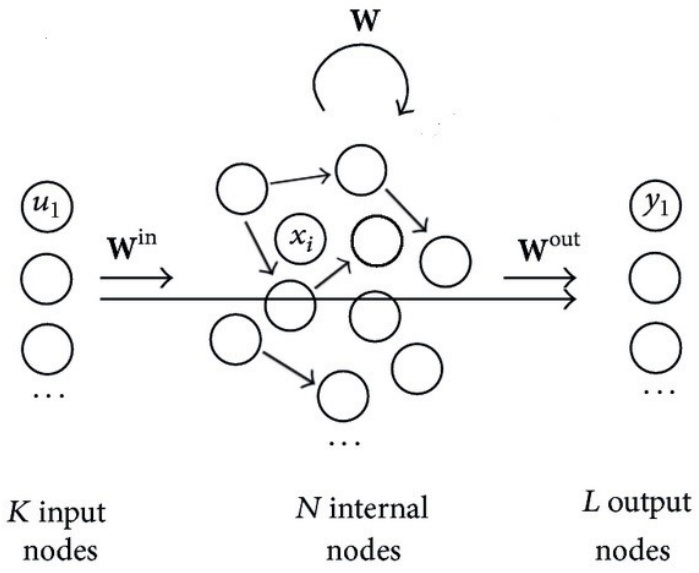


Figure 2: Diagram of an ESN for illustrating the equations mentioned above.

Let S the network state operator as a result of successively applying the equation (1), i.e. $\mathbf{x}(t+h) = S(\mathbf{x}(t), \bar{\mathbf{u}}^h)$. Let $A \subset \mathbb{R}^N$ the set of all admissible internal states. We require that A is closed under the operation of network update, that means $\mathbf{u} \in U, \mathbf{x} \in A$ implies $S(\mathbf{x}, \mathbf{u}) \in A$.

2.2 Echo states

Now we are going to introduce the concept of echo states but before doing that we need some previous definitions.

Definition 1: Let's assume that the sets U and A are compact. This situation will be called *standard compactness conditions.*

The standard compactness conditions will be the only conditions over the sets U and A for all the following results.

Definition 2: Assuming standard compactness conditions, the network has *echo states* if the network state $\mathbf{x}(t)$ is uniquely determined by any left-infinite input $\bar{\mathbf{u}}^{-\infty}$. This means that for every input sequence $\ldots, \mathbf{u}(t-1), \mathbf{u}(t) \in U^{-\mathbb{N}}$ and for all internal state sequences $\ldots, \mathbf{x}(t-1), \mathbf{x}(t) \in A^{-\mathbb{N}}$ and $\ldots, \mathbf{x}'(t-1), \mathbf{x}'(t) \in A^{-\mathbb{N}}$ where $\mathbf{x}(i) = S(\mathbf{x}(i-1), \mathbf{u}(i))$ and $\mathbf{x}'(i) = S(\mathbf{x}'(i-1), \mathbf{u}(i))$ it holds that $\mathbf{x}(t) = \mathbf{x}'(t)$.

And now there will be presented 3 equivalent characterizations of the echo states but before doing that two definitions are needed.

Definition 3: A state sequence $\bar{\mathbf{x}}^{-\infty} = \ldots, \mathbf{x}(t-1), \mathbf{x}(t) \in A^{-\mathbb{N}}$ is called *compatible* with an input sequence $\bar{\mathbf{u}}^{-\infty} = \ldots, \mathbf{u}(t-1), \mathbf{u}(t)$ if $\forall\ i < t$ the equality $\mathbf{x}(i+1) = S(\mathbf{x}(i), \mathbf{u}(i+1))$ is verified. In a similar way a infinite sequence $\bar{\mathbf{x}}^{\infty}$ is called *compatible* with an input sequence $\bar{\mathbf{u}}^{\infty}$ if $\forall\ i$ holds $\mathbf{x}(i+1) = S(\mathbf{x}(i), \mathbf{u}(i+1))$.

Definition 4: A network state $\mathbf{x} \in A$ is called *end-compatible* with an input sequence $\bar{\mathbf{u}}^{-\infty}$ if there exists a state sequence $\ldots, \mathbf{x}(t-1), \mathbf{x}(t)$ such that $\forall i\ \mathbf{x}(i+1) = S(\mathbf{x}(i), \mathbf{u}(i+1))$ and $\mathbf{x} = \mathbf{x}(t)$. In a similar way a network state $\mathbf{x} \in A$ is called *end-compatible* with a finite input sequence $\bar{\mathbf{u}}^{h}$ if there exists a state sequence $\ldots, \mathbf{x}(t-1), \mathbf{x}(t)$ such that $\forall i\ \mathbf{x}(i+1) = S(\mathbf{x}(i), \mathbf{u}(i+1))$ and $\mathbf{x} = \mathbf{x}(t)$.

Definition 5: Assuming standard compactness conditions:

1. The network is called *uniformly state contracting* if there exists a sequence $(\delta_h)_{h\geq 0}$ with $\lim_{h\to\infty} \delta_h = 0$ (i.e. null sequence) such that for all right-infinite input sequences $\bar{\mathbf{u}}^{+\infty}$ and for all states $\mathbf{x}, \mathbf{x}' \in A$, for all $h \geq 0$ for all sequence prefixes $\bar{\mathbf{u}}_h = \mathbf{u}(t), ..., \mathbf{u}(t+h)$ it holds that $d(S(\mathbf{x}, \bar{\mathbf{u}}_h), S(\mathbf{x}', \bar{\mathbf{u}}_h)) < \delta_h$, where d is the euclidean distance in $\mathbb{R}^N$.

2. The network is called *state forgetting* if for all left-infinite input sequences $\bar{\mathbf{u}}^{-\infty}$ there exists a null sequence $(\delta_h)_{h\geq 0}$ such that for all states $\mathbf{x}, \mathbf{x}' \in A$, for all $h \geq 0$ for all sequence suffixes $\bar{\mathbf{u}}_h = \mathbf{u}(t-h), ..., \mathbf{u}(t)$ it holds that $d(S(\mathbf{x}, \bar{\mathbf{u}}_h), S(\mathbf{x}', \bar{\mathbf{u}}_h)) < \delta_h$.

3. A network is called *input forgetting* if for all left-infinite input sequences $\bar{\mathbf{u}}^{-\infty}$ there exists a null sequence $(\delta_h)_{h\geq 0}$ such that for all $h \geq 0$, for all sequence suffixes $\bar{\mathbf{u}}_h = \mathbf{u}(t-h), ..., \mathbf{u}(t)$, for all left-infinite input sequences of the form $\bar{\mathbf{w}}^{-\infty}\bar{\mathbf{u}}_h$, $\bar{\mathbf{v}}^{-\infty}\bar{\mathbf{u}}_h$ for all states $\mathbf{x}$ end-compatible with $\bar{\mathbf{w}}^{-\infty}\bar{\mathbf{u}}_h$ and states $\mathbf{x}'$ end-compatible with $\bar{\mathbf{v}}^{-\infty}\bar{\mathbf{u}}_h$ it holds that $d(\mathbf{x}, \mathbf{x}') < \delta_h$

Proposition 1: Assuming standard compactness conditions, assume that S is continuous in state and input. Then being uniformly state contracting, state forgetting and input forgetting are all equivalent to the network having echo states.

Proof:
echo states $\Rightarrow$ *uniformly state contracting*
Let:

$$D = \{(\mathbf{x}, \mathbf{x}') \in A^2 : \exists \bar{\mathbf{u}}^{\infty} \in U^{\mathbb{Z}}, \exists\ \bar{\mathbf{x}}^{\infty}, \bar{\mathbf{x}}'^{\infty} \in A^{\mathbb{Z}}, \exists t \in \mathbb{Z} : \bar{\mathbf{x}}^{\infty}, \bar{\mathbf{x}'}^{\infty}$$

$$\text{compatible with } \bar{\mathbf{u}}^{\infty} \text{ and } \mathbf{x} = \bar{\mathbf{x}}(t) \text{ and } \mathbf{x}' = \bar{\mathbf{x}}'(t)\}$$

The idea of the D set is all the pairs compatibles with some input sequence. It is trivial that having echo states implies D only containing pairs of the form $(\mathbf{x}, \mathbf{x})$.
Let:

$$P^+ = \{(\mathbf{x}, \mathbf{x}', 1/h) \in A \times A \times [0,1] : h \in \mathbb{N}, \exists \bar{\mathbf{u}}^h \in U^h, \\ \mathbf{x} \text{ and } \mathbf{x}' \text{ are end-compatible with } \bar{\mathbf{u}}^h\}$$

And Let:

$$D^+ = \{(\mathbf{x}, \mathbf{x}') \in A^2 : (\mathbf{x}, \mathbf{x}', 0) \text{ is an accumulation point of } P^+\}$$

Let's prove that: $D \subseteq D^+$, taking $(\mathbf{x}, \mathbf{x}') \in D$ and taking $h \in \mathbb{N}$ then by definition of D exits $\bar{\mathbf{u}}^\infty$, $\bar{\mathbf{x}}^\infty$ and $\bar{\mathbf{x}}'^\infty$ such that $\bar{\mathbf{x}}^\infty$ and $\bar{\mathbf{x}}'^\infty$ are compatible with $\bar{\mathbf{u}}^\infty$, that means $\forall\, i\ \mathbf{x}(i) = S(\mathbf{x}(i-1), \mathbf{u}(i))$ and $\forall\, i\ \mathbf{x}'(i) = S(\mathbf{x}'(i-1), \mathbf{u}(i))$ it is clear for the sequences $x(t-h), \ldots, x(t) \in A^{h+1}$ and $x'(t-h), \ldots, x'(t) \in A^{h+1}$ holds $\mathbf{x}(i) = S(\mathbf{x}(i-1), \mathbf{u}(i))$ and $\mathbf{x}'(i) = S(\mathbf{x}'(i-1), \mathbf{u}(i))$ and, by definition this sequences are end compatible with $\bar{\mathbf{u}}'^h$. Then for all $h \in \mathbb{N}$ the point $(\mathbf{x}, \mathbf{x}', 1/h) \in P^+$ and then $\forall\, \epsilon > 0$ $h \in \mathbb{N}$ such that $d((\mathbf{x}, \mathbf{x}', 1/h), (\mathbf{x}, \mathbf{x}', 0)) < \epsilon$ and this is the definition of $(\mathbf{x}, \mathbf{x}', 0)$ being an accumulation point of P^+. So $D \subseteq D^+$ is proved.

Now let's prove $D^+ \subseteq D$: the firs observation is that if $(\mathbf{x}, \mathbf{x}') \in D^+$ and $u \in U$ then $(S(\mathbf{x}, \mathbf{u}), S(\mathbf{x}', \mathbf{u})) \in D^+$. This is clear by the definition of P^+, if exists a sequence $\bar{\mathbf{u}}^{\mathbf{h}}$ such that $\mathbf{x}$ and $\mathbf{x}'$ are end compatible with $\bar{\mathbf{u}}^{\mathbf{h}}$ then displacing the $\bar{\mathbf{u}}^{\mathbf{h}}$ in one unit in time is clear that $(S(\mathbf{x}, \mathbf{u}), S(\mathbf{x}', \mathbf{u})) \in P^+$ and for seeing that it lies in D^+ is enough to use the continuity property of S the fact of $(\mathbf{x}, \mathbf{x}', 0)$ is an accumulation point of P^+ extends to $(S(\mathbf{x}, \mathbf{u}), S(\mathbf{x}', \mathbf{u}), 0)$ and it is also an accumulation point so $(\mathbf{x}, \mathbf{x}') \in D^+$. Now let's prove that for every $(\mathbf{x}, \mathbf{x}') \in D^+$ exists $\mathbf{u} \in U$ and $(z, z') \in D^+$ such that $(S(\mathbf{z}, \mathbf{u}), S(\mathbf{z}', \mathbf{u})) = (\mathbf{x}, \mathbf{x}')$. For proving this let's take a sequence $(\mathbf{x}_n, \mathbf{x}'_n, 1/h_n)$ such that $\lim_{n\to\infty}(\mathbf{x}_n, \mathbf{x}'_n, 1/h_n) = (\mathbf{x}, \mathbf{x}', 0)$. Clearly $(\mathbf{x}_n, \mathbf{x}'_n, 1/h_n) \in P^+$ for being in P^+ for each of the $(\mathbf{x}_n, \mathbf{x}'_n)$ there exists $\mathbf{u}_i$ and $(\mathbf{z}_n, \mathbf{z}'_n) \in A \times A$ such that $(S(\mathbf{z}_n, \mathbf{u}_n), S(\mathbf{z}'_n, \mathbf{u}_n)) = (\mathbf{x}_n, \mathbf{x}'_n)$. Now using Bolzano-Weierstrass theorem the sequence $(\mathbf{z}_n, \mathbf{z}'_n, \mathbf{u}_n)$ has a convergent subsequence $(\mathbf{z}_{n_j}, \mathbf{z}'_{n_j}, \mathbf{u}_{n_j})$. As S is continuous we can make a pas to the limit and then obtain $(S(\mathbf{z}, \mathbf{u}), S(\mathbf{z}', \mathbf{u})) = (\mathbf{x}, \mathbf{x}')$, moreover the set $A \times A \times U$ is compact so any convergent sequence of $A \times A \times U$ will converge inside the space so by definition now $(\mathbf{z}, \mathbf{z}') \in D^+$ and using the property of being closed under network update, this finishes the proof of for every $(\mathbf{x}, \mathbf{x}') \in D^+$ exists $\mathbf{u} \in U$ and $(z, z') \in D^+$ such that $(S(\mathbf{z}, \mathbf{u}), S(\mathbf{z}', \mathbf{u})) = (\mathbf{x}, \mathbf{x}')$.
Using this result is easy check that for every $(\mathbf{x}, \mathbf{x}') \in^+$ exists an input sequence $\bar{\mathbf{u}}^\infty$ and state sequences $\bar{\mathbf{x}}^\infty$ and $\bar{\mathbf{x}}'^\infty$ compatibles with $\bar{\mathbf{u}}^\infty$ such that $\mathbf{x} = \mathbf{x}(t)$ and $\mathbf{x}' = \mathbf{x}'(t)$.

For proving the result assume that the network does not satisfy the property

of being uniformly state contracting, this implies that for every null sequence $(\delta_i)_{i\geq 0}$ exits an $h > 0$ and a finite input sequence of length h, $\bar{\mathbf{u}}_h$ and states $\mathbf{x}$, $\mathbf{x}' \in A$ such that:

$$d(S(\mathbf{x}, \bar{\mathbf{u}}_h), S(\mathbf{x}', \bar{\mathbf{u}}_h)) \geq \delta_h$$

Any compact set is bounded, so A is bounded, then defining the sequence:

$$\mu_i := \sup\{d(S(\mathbf{x}, \bar{\mathbf{u}}_i), S(\mathbf{x}, \bar{\mathbf{u}}_i)) : \mathbf{x}, \mathbf{x}' \in A, \bar{\mathbf{u}}_i \in U^i\}$$

is bounded, moreover μ_i can not be a null sequence because we are assuming the network not being uniformly state contracting. μ_i has a convergent subsequence and this subsequence has to converge to some $\epsilon > 0$. In any compact set the supremum is reached, then $\mathbf{x}, \mathbf{x}' \in A$. Now let's take the sequence $(\mathbf{x_{i_j}}, \mathbf{x'_{i_j}}) \in A^2$ such that:

$$(\mathbf{x_{i_j}}, \mathbf{x'_{i_j}}) \in \{S(\mathbf{x}, \bar{\mathbf{u}}_{i_j}), S(\mathbf{x}, \bar{\mathbf{u}}_{i_j}))|\bar{\mathbf{u}}_{\mathbf{i_j}} \in U^{i_j}, \mathbf{x}, \mathbf{x}' \in A, d(S(\mathbf{x}, \bar{\mathbf{u}}_{i_j}), S(\mathbf{x}, \bar{\mathbf{u}}_{i_j})) = \mu_{i_j}\}$$

As we said before A is compact, so A^2 is compact too, then exists a subsequence $(\mathbf{x}_{i_{j_k}}, \mathbf{x'}_{i_{j_k}})$ which converges to some $(\mathbf{y}, \mathbf{y}' \in A^2$. It is clear that $(\mathbf{x}_{i_j}, \mathbf{x'}_{i_j}, 1/i_j) \in P^+$ then $(\mathbf{y}, \mathbf{y}', 0)$ is an accumulation point of P^+ and by definition $(\mathbf{y}, \mathbf{y}') \in D^+$ and also:

$$0 < \epsilon = \lim_{k\to\infty} \mu_{i_{j_k}} = \lim_{k\to\infty} d(\mathbf{x_{i_j}}, \mathbf{x'_{i_j}}) = d(\mathbf{y}, \mathbf{y}')$$

That means that an element $(\mathbf{y}, \mathbf{y}') \in D^+$ with $\mathbf{y} \neq \mathbf{y}'$ which directly contradicts the echo state property.

uniformly state contracting ⇒ state forgetting
Let's assume not state forgetting i.e. exists an input sequence $\bar{\mathbf{u}}^{-\infty}$ and a strictly growing index sequence $(I_i)_{i\geq 0}$, the states $\mathbf{x}_i$, $\mathbf{x}'_i$ and an $\epsilon > 0$ such that:

$$\forall\ id(S(\mathbf{x}_i, \bar{\mathbf{u}}^{-\infty}[I_i]), S(\mathbf{x'}_i, \bar{\mathbf{u}}^{-\infty}[I_i])) > \epsilon$$

here $\bar{\mathbf{u}}^{-\infty}[I_i]$ is the sequence formed by the last I_i values of $\bar{\mathbf{u}}^{-\infty}$. Starting with the values $\bar{\mathbf{u}}^{-\infty}[I_i]$ and completing to having a right-infinite input sequence $\bar{\mathbf{v}}_\mathbf{i}$, now calling $\bar{\mathbf{v}}^{-\infty}[I_i]$ to the sequence formed by the firsts I_i values of $\bar{\mathbf{v}}_i$ it holds:

$$d(S(\mathbf{x}_i, \bar{\mathbf{v}}_i[I_i]), S(\mathbf{x'}_i, \bar{\mathbf{v}}_i[I_i])) > \epsilon$$

which directly contradicts the uniform state contraction property, this ends the proof of this result.

state forgetting ⇒ input forgetting
Let's take a left-infinite input sequence $\bar{\mathbf{u}}^{-\infty}$ and Let $(\delta_h)_{h\geq 0}$ it's associated null sequence of the state forgetting property. Let $\bar{\mathbf{u}}_h$ the finite sequence formed for the first h values of $\bar{\mathbf{u}}^{-\infty}$. Take two states $\mathbf{y}$ and $\mathbf{y}'$ from A. By the state forgetting property it holds that $d(S(\mathbf{y}, \bar{\mathbf{u}}_h), S(\mathbf{y}', \bar{\mathbf{u}}_h)) < \delta_h$. Let's consider now any left infinite sequences $\bar{\mathbf{w}}^{-\infty}$ and $\bar{\mathbf{v}}^{-\infty}$. Then the states $\mathbf{x}$ and $\mathbf{x}'$ end compatible

with $\bar{\mathbf{w}}^{-\infty}\bar{\mathbf{u}}_h$ and $\bar{\mathbf{v}}^{-\infty}\bar{\mathbf{u}}_h$ and it holds $d(\mathbf{x},\mathbf{x}') < \delta_h$ which, by definition, is the property of being input forgetting.

input forgetting $\Rightarrow$ *echo states*
Let's assume that the net does not verify the echo state property, then by definition exists a left-infinite input sequence $\bar{\mathbf{u}}^{-\infty}$ and two states $\mathbf{x}$, $\mathbf{x}'$ such that $d(\mathbf{x},\mathbf{x}') > 0$. Then the result is direct taking $\bar{\mathbf{w}}^{-\infty}\bar{\mathbf{u}}_h$ and $\bar{\mathbf{v}}^{-\infty}\bar{\mathbf{u}}_h$ equal to $\bar{\mathbf{u}}^{-\infty}$ directly contradicts the definition of input forgetting because the states $\mathbf{x}$, $\mathbf{x}'$ are end compatible with $\bar{\mathbf{w}}^{-\infty}\bar{\mathbf{u}}_h$ and $\bar{\mathbf{v}}^{-\infty}\bar{\mathbf{u}}_h$ and $d(\mathbf{x},\mathbf{x}') > 0$.

Proposition 2: Assume a network with tanh functions in the network update operator.

1. Let $\mathbf{W}$ the weight matrix satisfy $\sigma < 1$, where σ is its largest singular value, then $d(S(\mathbf{x},\mathbf{u}), S(\mathbf{x}',\mathbf{u})) < \sigma d(\mathbf{x},\mathbf{x}')$, for all inputs $\mathbf{u}$, for all states $\mathbf{x},\mathbf{x}' \in [-1,1]^N$
2. Let $\mathbf{W}$ the weight matrix with spectral radius $\rho(\mathbf{W})) > 1$. Then the network has no echo states for any input set U containing $\mathbf{0}$ and admissible set A.

Proof:

1. $$d(S(\mathbf{x},\mathbf{u}), S(\mathbf{x}',\mathbf{u})) = d(\mathbf{tanh}(\mathbf{W}^{\mathsf{in}}\mathbf{u} + \mathbf{W}\mathbf{x}), \mathbf{tanh}(\mathbf{W}^{\mathsf{in}}\mathbf{u} + \mathbf{W}\mathbf{x}')) \leq$$

   $$d(\mathbf{W}^{\mathsf{in}}\mathbf{u} + \mathbf{W}\mathbf{x}, \mathbf{W}^{\mathsf{in}}\mathbf{u} + \mathbf{W}\mathbf{x}') = d(\mathbf{W}\mathbf{x}, \mathbf{W}\mathbf{x}')$$

   For seeing this result we can use the mean value theorem which applied to the element wise **tanh** function holds $\tanh(b) - \tanh(a) = \tanh'(\xi)(b-a) = (1 - \tanh^2(\xi))(b-a) \leq b - a$ for each element.

   $$d(\mathbf{W}\mathbf{x}, \mathbf{W}\mathbf{x}') = ||\mathbf{W}(\mathbf{x} - \mathbf{x}') \leq \sigma d(\mathbf{x},\mathbf{x}')$$

   And now is easy to check that this shrinkage of the internal state by a factor $\sigma < 1$ implies echo state by using the definition of echo states.

2. Assume now $\bar{\mathbf{u}}^{-\infty} = \mathbf{0}$ it is clear that $\bar{\mathbf{x}}^{-\infty} = \mathbf{0}$ is a compatible state. Checking at the reference [18] section 3 if $\rho(W) > 1$ the null state is not asymptotically stable, then exits another internal state $\mathbf{x}' \neq \mathbf{0}$ compatible with the input sequence and this violates the echo state property.

2.3 Output feedback

The output feedback for an ESN consists in connecting the output signal to the internal state of the net using a weight matrix $\mathbf{W}^{\mathsf{fb}} \in \mathbb{R}^{N \times L}$. The form of the

main equation for the reservoir of the ESN in this case is:

$$\mathbf{x}(t+1) = \mathbf{f}(\mathbf{W}^{\text{in}} \begin{bmatrix} 1 \\ \mathbf{u}(t) \end{bmatrix} + \mathbf{W}\mathbf{x}(t+1) + \mathbf{W}^{\text{fb}}\hat{\mathbf{y}}(t)) \tag{3}$$

This makes the ESN a much more powerful tool because the dynamics of the system can be much more rich now, but this power has a price because some stability issues can appear when output feedback is used.

For solving the possible stability problems created because of the output feedback one possible strategy is the teacher forcing. It consists in breaking the feedback loop in the training phase doing:

$$\mathbf{x}_{\text{train}}(t+1) = \mathbf{f}(\mathbf{W}^{\text{in}} \begin{bmatrix} 1 \\ \mathbf{u}(t) \end{bmatrix} + \mathbf{W}\mathbf{x}(t+1) + \mathbf{W}^{\text{fb}}\mathbf{y}(t)) \tag{4}$$

Also a common strategy for making the net output robust is adding noise only during the training phase so the final equation for the training internal reservoir is:

$$\mathbf{x}_{\text{train}}(t+1) = \mathbf{f}(\mathbf{W}^{\text{in}} \begin{bmatrix} 1 \\ \mathbf{u}(t) \end{bmatrix} + \mathbf{W}\mathbf{x}(t+1) + \mathbf{W}^{\text{fb}}(\mathbf{y}(t) + \nu(t)) \tag{5}$$

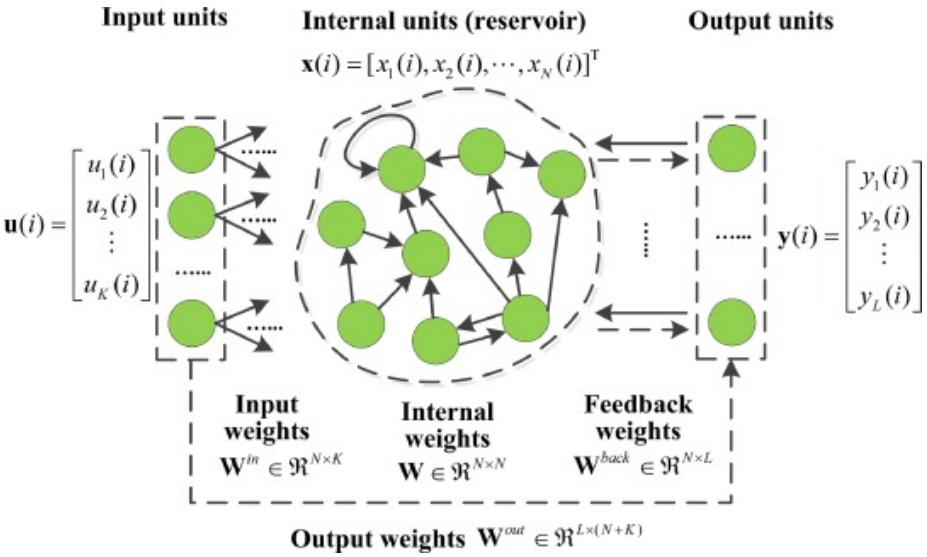


Figure 3: Diagram of an ESN with all the concepts mentioned until now.

For the testing it's not allowed using the real values $\mathbf{y}(t)$ and then testing the performance but it can be used a similar strategy used with the RNN for the testing which is compute the first values of the testing reservoir using the real values of the target function $\mathbf{y}(t)$ and then leave the ESN continue computing

the values of the reservoir in the test phase in a free run with the predicted values $\hat{\mathbf{y}}(t)$. All the values computed using the true values of the output should be discarded. An example of this technique will be provide in the section 3.2.

2.4 ESN with leaky integrator neurons

The performance of the ESN computed with the equation (1) or (3) perform well for discrete tasks, nevertheless this way of computation is not the most suitable for learning slowly changing systems. For this case the most adequate equation will be a continuous one, in our case we will use a discretized version of an continuous equation. The continuous equation is:

$$\dot{\mathbf{x}} = C(-\alpha\mathbf{x} + \mathbf{f}(\mathbf{W}^{\mathsf{in}}\begin{bmatrix}1\\\mathbf{u}\end{bmatrix} + \mathbf{W}\mathbf{x} + \mathbf{W}^{\mathsf{fb}}\hat{\mathbf{y}}) \tag{6}$$

Where C is a time constant and a is the leaking decay rate. Dicretizing with time step δ we obtain:

$$\mathbf{x}(t+1) = (1 - C\alpha\delta)\mathbf{x} + C\delta\left(\mathbf{f}(\mathbf{W}^{\mathsf{in}}\begin{bmatrix}1\\\mathbf{u}(t+1)\end{bmatrix} + \mathbf{W}\mathbf{x}(t) + \mathbf{W}^{\mathsf{fb}}\hat{\mathbf{y}}(t)\right) \tag{7}$$

The first observation in equation (7) is that the product $C\delta$ can be substituted by a single parameter $C\delta = \gamma$. A more interesting observation here is that γ is an irrelevant parameter, because for every ESN generated according to:

$$\mathbf{x}(t+1) = (1 - \alpha\gamma)\mathbf{x} + \gamma\left(\mathbf{f}(\mathbf{W}^{\mathsf{in}}\begin{bmatrix}1\\\mathbf{u}(t+1)\end{bmatrix} + \mathbf{W}\mathbf{x}(t) + \mathbf{W}^{\mathsf{fb}}\hat{\mathbf{y}}(t)\right) \tag{8}$$

exists other ESN with $\gamma = 1$ and exactly the same output. For proving this lets assume a ESN with weight some weight matrix $\mathbf{W}^{\mathsf{in}}$, $\mathbf{W}$ and $\mathbf{W}^{\mathsf{fb}}$ and some value of the parameters α and γ, now we introduce $a = \alpha\gamma$ and divide the equation (**??**) by γ

$$\frac{\mathbf{x}(t+1)}{\gamma} = (1-a)\frac{\mathbf{x}(t)}{\gamma} + \left(\mathbf{f}(\mathbf{W}^{\mathsf{in}}\begin{bmatrix}1\\\mathbf{u}(t+1)\end{bmatrix} + \mathbf{W}\mathbf{x}(t) + \mathbf{W}^{\mathsf{fb}}\hat{\mathbf{y}}(t)\right) \tag{9}$$

rewriting in a equivalent form:

$$\frac{\mathbf{x}(t+1)}{\gamma} = (1-a)\frac{\mathbf{x}(t)}{\gamma} + \left(\mathbf{f}(\mathbf{W}^{\mathsf{in}}\begin{bmatrix}1\\\mathbf{u}(t+1)\end{bmatrix} + \gamma\mathbf{W}\frac{\mathbf{x}(t)}{\gamma} + \mathbf{W}^{\mathsf{fb}}\hat{\mathbf{y}}(t)\right) \tag{10}$$

From (10) we can deduce that a new ESN with the same $\mathbf{W}^{\mathsf{in}}$ and $\mathbf{W}^{\mathsf{fb}}$ and a new internal weights $\mathbf{W}' = \gamma\mathbf{W}$ using (8) with $\gamma = 1$ will produce the internal state $\frac{\mathbf{x}(t)}{\gamma}$. Then the ouput produced by the new ESN will be:

$$\hat{\mathbf{y}}(t) = \mathbf{W}^{\mathsf{out}}\begin{bmatrix}1\\\mathbf{u}(t)\\\frac{\mathbf{x}(t)}{\gamma}\end{bmatrix} \tag{11}$$

So if the new ESN has a output weight matrix $\mathbf{W}'^{\mathsf{out}}$ with a scale of γ to the components corresponding to the internal state $\frac{\mathbf{x}(t)}{\gamma}$ the output of the original ESN an the new ESN will be exactly the same, and the new ESN has $\gamma = 1$. With this result we can conclude that the equation for the internal state with leaky integrator neurons is:

$$\mathbf{x}(t+1) = (1-\alpha)\mathbf{x} + \mathbf{f}(\mathbf{W}^{\mathsf{in}} \begin{bmatrix} 1 \\ \mathbf{u}(t+1) \end{bmatrix} + \mathbf{W}\mathbf{x}(t) + \mathbf{W}^{\mathsf{fb}}\hat{\mathbf{y}}(t)) \tag{12}$$

2.5 Generating the reservoir

As we have said before in a ESN the only weight trained are the weights in the last layer, $\mathbf{W}^{\mathsf{out}}$. The other matrices $\mathbf{W}^{\mathsf{in}}$, $\mathbf{W}$ and $\mathbf{W}^{\mathsf{fb}}$ are randomly generated. $\mathbf{W}$ is usually generated sparse, the performance of ESN with sparse reservoirs is usually better and the computations are much faster. The distribution used for generating $\mathbf{W}$ has to be with zero mean, the most common options are discrete distribution with two values, Gaussian or uniform. $\mathbf{W}^{\mathsf{in}}$ is also randomly generated but it has to be dense, otherwise useful information from the input can get lost. The distributions for generating $\mathbf{W}^{\mathsf{in}}$ can also be diverse. The weights $\mathbf{W}^{\mathsf{fb}}$ can be generated dense or sparse depending on the task again with a zero mean distribution.

One of the most important hyperparameters of the ESN is the spectral radius of the internal state weight matrix $\rho(\mathbf{W}))$. As we have seen before $\rho(\mathbf{W})) < 1$ does not guarantees having echo states but in most of the situations satisfying $\rho(\mathbf{W})) < 1$ the ESN will present echo states. The spectral radius of $\mathbf{W}$ is directly related with the memory capacity of the ESN and it has to bigger in tasks which require more memory span, for seeing an example of this behavior go to section 3. Other hyperparameter of the ESN is the scaling factor of the input matrix, bigger scaling of this weights usually works better for strongly non-linear tasks. The most common technique for selecting the hyperparameters is trying different values of the hyperparameters with small reservoirs which allows make the training in a very short amount of time, select the hyperparameters that give the best performance and use this values for the bigger reservoirs.

Usually in the dynamics of the neurons of the internal state appears an initial transient state which creates an unnatural behavior of the net. This transient is the result of setting to an arbitrary value the initial state $\mathbf{x}(0)$ (usually $\mathbf{x}(0) = 0$). For avoiding these modified dynamics the initial values of X are discarded, this implies also discarding the same amount of values from U and Y. For some tasks (specially classification or working with multiple sequences at the same time) this behavior can result useful and in this case no discard will be applied.

For For training the weights $\mathbf{W}^{\mathsf{out}}$ the most common strategy is minimize the

MSE between $\hat{\mathbf{y}}$ and $\mathbf{y}$. This is solving:

$$\mathbf{Y} = \mathbf{W}^{\mathsf{out}} \begin{bmatrix} 1 \\ \mathbf{U} \\ \mathbf{X} \end{bmatrix}$$

This problem has a closed form solution so solving it is very fast, moreover this problem is convex this this guarantees reaching the global optimum as opposed to other techniques such as the RNN which are solving a non-convex problem using gradient descend. If $T > 1 + L + N$ the system of equations will be overdetermined. A common s solution to this problem is the Tikhonov regularization, which in this case is:

$$\mathbf{W}^{\mathsf{out}} = \mathbf{Y} \times \begin{bmatrix} \mathbf{1} & \mathbf{U} & \mathbf{X} \end{bmatrix} \times \left(\begin{bmatrix} \mathbf{1} \\ \mathbf{U} \\ \mathbf{X} \end{bmatrix} \begin{bmatrix} \mathbf{1} & \mathbf{U} & \mathbf{X} \end{bmatrix} + \beta \mathbf{I} \right)^{-1} \tag{13}$$

this adds a new hyperparameter to the ESN which is β the regularization parameter for the training of $\mathbf{W}^{\mathsf{out}}$.

3 Basic experiments

This section corresponds to replicated experiments of different papers. The experiments range from simple toy examples to illustrate basic concepts to some more complex experiment using different approaches and real data. In some of the cases the experiments had some little modification to check the performance in a similar but not exactly the same situation. The ideas for experiments from 1 to 3 are obtained from [4] and the idea for the fourth one from [8].

3.1 Sinusoidal wave

The idea of this experiment is checking the capacity of an ESN of learning a non-linear function, in this case $f(x) = \frac{1}{2}x^7$. The input signal for this experiment will be a sinusoidal wave $u(t) = \sin(t/5)$ and output will be the non-linear modification of this wave $y(t) = \frac{1}{2}\sin^7(t/5) = \frac{1}{2}u^7(t)$. The size of the internal state for this experiment is fixed to $N = 100$. The training size is 300 we discard the first 100, the matrix $\mathbf{W}$ is generated only with values $-0.4, 0.4, 0$ with probability $0.025, 0.025, 0.95$ respectively (the value 0.4 is just a scaling factor to obtain $\rho(\mathbf{W}) < 1$). The matrix $\mathbf{W}^{\text{in}}$ is created with random values -1 and 1 with equal probability.

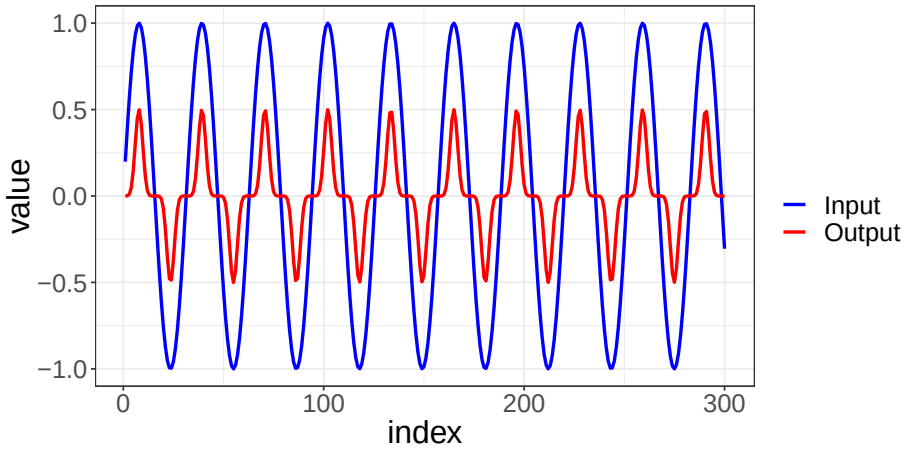
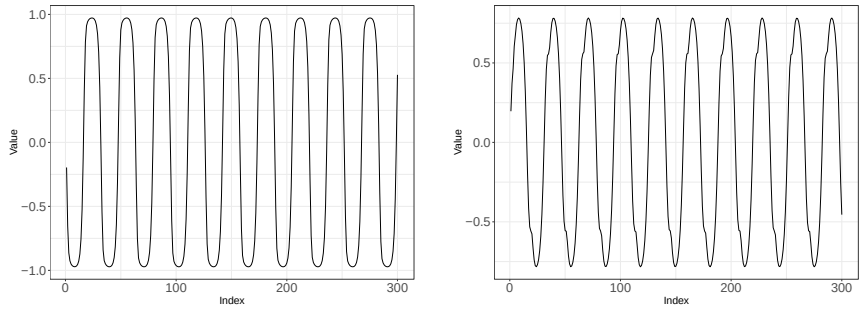


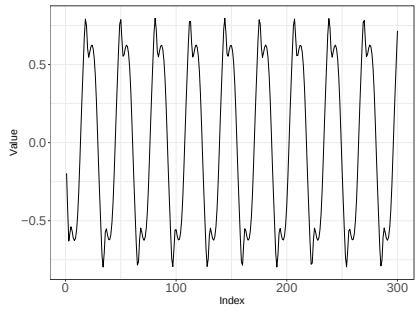
Figure 4: Input and output of the training data of the first experiment. In the y-axis of the plot we can see the value of each signal and in the x-axis the index of the samples.

The obtained results where $\text{MSE}_{\text{train}} \approx 10^{-14}$ and $\text{MSE}_{\text{train}} \approx 10^{-14}$, which is a similar performance than in the original experiment. As we have mentioned before working with ESN is important discard the first samples to remove the original transient mode of the neurons of the internal state. The transient state can be seen for some neurons in the following plot:

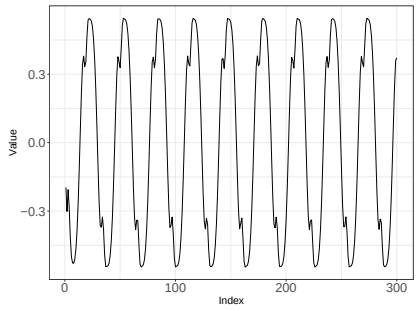


neuron1

neuron10



neuron30



neuron45

An interesting observation is that without discarding the results are $\text{MSE}_{\text{train}} \approx 10^{-13}$ and $\text{MSE}_{\text{train}} \approx 10^{-13}$ but discarding only in the training phase $\text{MSE}_{\text{train}} \approx 10^{-14}$ but $\text{MSE}_{\text{test}} \approx 0.018$. So in general the best results are obtained discarding some samples but the most important thing is treat in a similar way training and testing data.

3.2 Zero input and output feedback

The main idea of this experiment is illustrate that the output feedback can work in a similar way than the input. The input for this task will be the zero signal $\mathbf{u} = \mathbf{0}$ and the output will be the same than the last experiment $y(t) = \frac{1}{2}\sin^7(t/5)$, we discard the first 100 samples and train with the following 200. And we use the same $\mathbf{W}$ with values $-0.4, 0.4, 0$ with probability $0.025, 0.025, 0.95$ and $\mathbf{W^{fb}}$ with values $-1, 1$ with equal probability. The obtained results are $\text{MSE}_{\text{train}} \approx 10^{-9}$ but testing after with 300 values we have $\text{MSE}_{\text{test}} > 100$.

The error occurs because small disturbances in the prediction result accumulate because the equation designed for the intermediate state with output feedback uses the real value $\mathbf{y}(t)$:

$$\mathbf{x}_{\mathsf{test}}(t+1) = \tanh(\mathbf{W}\mathbf{x}(t) + \mathbf{W}^{\mathsf{fb}}\hat{\mathbf{y}}(t))$$

Small errors in the prediction $\mathbf{y}(t)$ accumulate until the error blows up.
The idea for solving this is make the system output robust. One way of doing this is adding feedback noise during the training phase:

$$\mathbf{x}_{\mathsf{train}}(t+1) = \tanh(\mathbf{W}\mathbf{x}(t) + \mathbf{W}^{\mathsf{fb}}\left(\mathbf{y}(t) + \nu(t)\right))$$

For this problem ν was noise between -10^{-4} and 10^{-4} sampled using a uniform distribution. It is important adding this noise only in the training because the test is always noisy by how is constructed. The noise $\nu(t)$ is random noise sampled between -10^{-4} and 10^{-4}.

In combination with the strategy of adding noise is important using teacher forcing steps to stabilize the network. For the testing phase the internal state was calculated using 1000 teacher forcing steps, using the equation:

$$\mathbf{x}_{\mathsf{test}}(t+1) = \tanh(\mathbf{W}\mathbf{x}(t) + \mathbf{W}^{\mathsf{fb}}\mathbf{y}(t))$$

After 1000 steps the teacher forcing is removed and the net goes in an free run for the remaining test samples, now for computing the internal state the equation is:

$$\mathbf{x}_{\mathsf{test}}(t+1) = \tanh(\mathbf{W}\mathbf{x}(t) + \mathbf{W}^{\mathsf{fb}}\hat{\mathbf{y}}(t))$$

For checking the performance of the net all the values of $\hat{\mathbf{y}}(t)$ corresponding to the teacher forcing phase were discarded. The performance with this new strategy is a training and testing MSE of approximately 10^{-7}. The performance for testing now is much better but the error of the training raises a bit.

3.3 Periodic sequence learning

For this task the signal is the melody of the song "The House of the Rising Sun", the musical notes of this melody are represented by numbers between -1 and 14 assigning -1 to the g note and going up in the scale order.

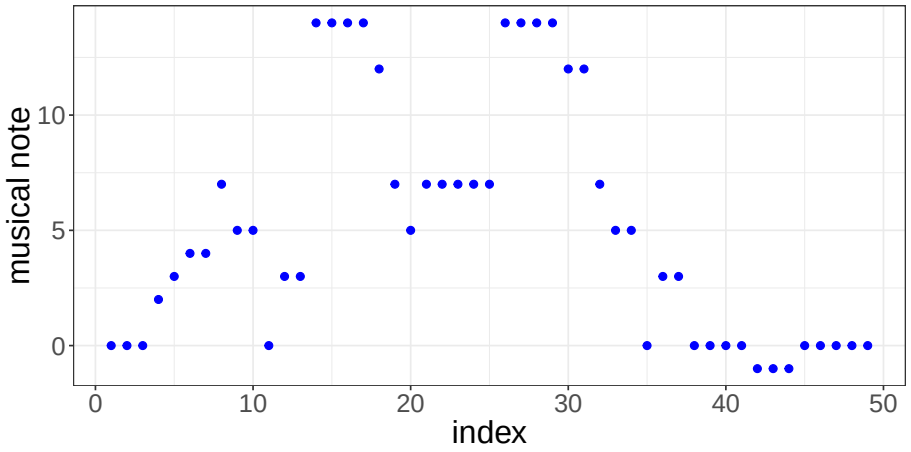


Figure 5: Melody for the task of this experiment before the squashing in [-1, 1].

The target signal for this task consist in concatenation of the values showed in Figure 5. And like in the previous experiment there is no input signal.

The complexity of this task is higher than in the two previous ones, for this reason the internal size of the network is fixed to $N = 400$. The values of the weight matrix $\mathbf{W}$ are randomly generated to 0, -1 and 1 with probabilities 0.9875, 0.0065 and 0.0065 respectively. And then rescaling this weight matrix to obtain $\rho(\mathbf{W}) = 0.9$. This network also has output feedback sampled using a uniform distribution in [-2, 2]. For this task is needed that the network has enough short term memory capacity because the target signal contains eight consecutive 0's and if the net has not a memory of at least 9 values the ESN will get lost and return always a wrong value.

As we have output feedback also using noise for making the output robust training and output feedback is needed. So the equation here for the training internal state will be:

$$\mathbf{x}_{\mathsf{train}}(t+1) = \tanh(\mathbf{W}\mathbf{x}(t) + \mathbf{W}^{\mathsf{fb}}\left(\mathbf{y}(t) + \nu(t)\right))$$

And for this experiments the net will run using 1500 teacher forcing steps. And free run until complete all the testing values. The performance in this experiment was a test MSE of 10^{-7}.

An interesting observation is that the choice of the parameters is robust. In the case of $\mathbf{W}$ the performance is similar for any $\mathbf{W}$ with $\rho(\mathbf{W})) > 0.7$ however for values $\rho(\mathbf{W})) < 0.7$ the memory span of the ESN is not enough and the network gets lost returning always the mean of all the values. The scaling parameters of the matrix $\mathbf{W}^{\mathsf{fb}}$ can be chosen from [-0.1, 0.1] to [-10, 10] the performance is similar in all the cases. The same happens to the noise that can be sampled from $[-0.01, 0.01]$ to $[-10^{-6}, 10^{-6}]$ also with a similar performance in testing, in this case bigger noise leads to a slightly worse performance but the difference is about one order of magnitude so is not relevant for this case.

3.4 Japanese vowel dataset

The "Japanese Vowels" is a dataset used for time series classification. This dataset consist in the speech of nine Japanese male speakers. Each utterance is represented by 12 LPC cepstrum coefficients, there is a total of 270 samples used for the training phase and 370 samples for the testing phase. For this problem different configurations of ESN were used:

The first one and the most simple one consists in using 9 output units $(y_m)_{m=1,\dots,9}$, observe that in this case the output of the ESN is not a time series. For computing the internal state $\mathbf{X}$ is used the equation (1) and the output is normalized in the interval [0, 1]. The advantage of this approach are that it is simple and intuitive and the results are good with a misclassification test error of 1%. The disadvantage of this method is that large ESN are required for solving the task, $N \approx 10^3$. For this setup was no clear difference between using leaky integrator neurons or not.

The idea of the second approach airses from the fact that not every speech has the same length. The idea es choosing a small integer D and take one sample of internal state and input each l_i/D where l_i is the length of each sequence. That means taking the values corresponding to the index $l_i/D, 2l_i/D, \dots, l_i$ using interpolation if l_i is not multiple of D. The advantage of this setup is that the training error are as lower as the first approach but with an ESN much smaller than the ESN used in the first case, here with a dimension of the internal state about 20. One of the most important disadvantages is that this setup trends to overfitting and the results in the testing phase are similar (and in some cases worse) than in the last setup.

The last approach presented here is the called *combined classifiers*. The architecture consist in create several ESN with an small reservoir, for this case with $N = 4$, with only 4 neurons the dynamics of the reservoir are quite different for different realizations (this usually does not happen when we are working with bigger reservoirs). The key idea is that with different dynamics the probability of several ESN getting wrong in the same way is very low. So using several ESN each of them with one vote for the predicted output we can exploit this fact. The performance for this experiment is te best one obtaining a 100% of testing

accuracy and only 1 misclassification in training, the performance can be seen in Figure 6 where is plot the number of misclasifications vs the number of ESN used in paralel for generating the output.

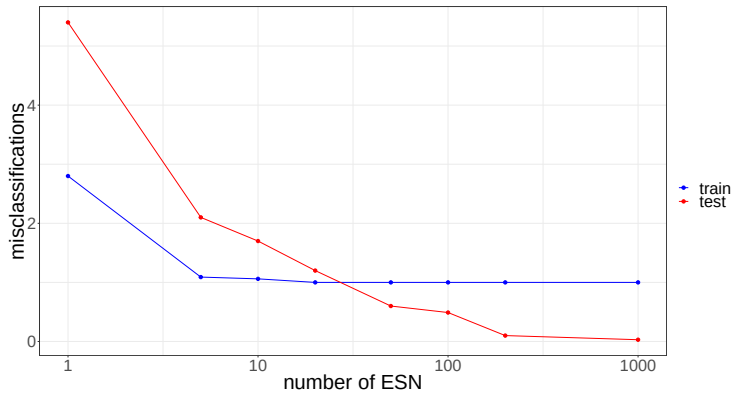


Figure 6: Diagram of the performance of ESN vs the number of ESN used in the classification task. These values are calculated averaging 10 different realizations

Taking a close look on the data the train sample misclassified is always the same, so this sample can be suspect of being corrupted.

4 R package

4.1 Motivation for creating an R package

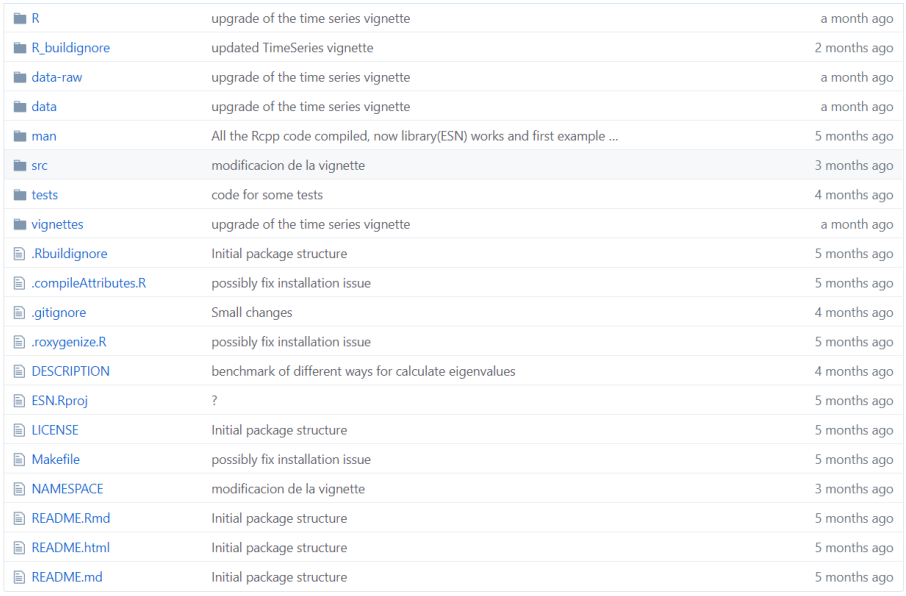
The use of ESN for the different tasks is tedious in many cases since it requires generating all the weight matrices with the desired conditions, calculating the intermediate state for the train and the test, training the last layer of weights and checking the performance of the net.

All these reasons make using ESN a long code and normally difficult to modify to test new configurations since an echo state network can be modified in many ways, generating the weight matrices with different distributions, changing rescaled values, different spectral radio, number of samples discarded, different values of noise among others.

For all of this reasons one of the main objectives of this project was the implementation of an official R package that will be soon uploaded to CRAN. The aim of the package is to allow, on the one hand, to use the ESN model in a simple way and, on the other hand, to allow a great freedom and number of options when creating an ESN. In the following sections the structure of the package and the methods and classes will be explained in more detail, however I would like to illustrate with an example the simplicity that allows to quickly test an ESN.

```
net <- ESN(u = u_train, y = y_train, num_neurons = 100,
           density = 0.05, spectral_radius = 0.8)
net <- train(net)
res <- predict(net, u_test, y_test)
```

Figure 7: Example of an easy use of ESN by using the R. As can be seen with three lines of code a simple model of an ESN can be trained.



R	upgrade of the time series vignette	a month ago
R_buildignore	updated TimeSeries vignette	2 months ago
data-raw	upgrade of the time series vignette	a month ago
data	upgrade of the time series vignette	a month ago
man	All the Rcpp code compiled, now library(ESN) works and first example ...	5 months ago
src	modificacion de la vignette	3 months ago
tests	code for some tests	4 months ago
vignettes	upgrade of the time series vignette	a month ago
.Rbuildignore	Initial package structure	5 months ago
.compileAttributes.R	possibly fix installation issue	5 months ago
.gitignore	Small changes	4 months ago
.roxygenize.R	possibly fix installation issue	5 months ago
DESCRIPTION	benchmark of different ways for calculate eigenvalues	4 months ago
ESN.Rproj	?	5 months ago
LICENSE	Initial package structure	5 months ago
Makefile	possibly fix installation issue	5 months ago
NAMESPACE	modificacion de la vignette	3 months ago
README.Rmd	Initial package structure	5 months ago
README.html	Initial package structure	5 months ago
README.md	Initial package structure	5 months ago

Figure 8: Screenshot of the GitHub repository containing all the code files of the R package. In this image we can see that the structure of the code is the same that in the official CRAN packages.

4.2 General Structure of the package

R allows a wide variety of options for implementing object oriented programming such as S3, S4 or R6. For the implementation of this package the option chosen was S3 objects.

S3 implements a style of object oriented programming called generic-function object oriented. This is different from most programming languages, like Java, C++, and C#, which implement message-passing object oriented programming. While computations are still carried out via methods, a special type of function called a generic function decides which method to call. S3 is a very casual system, it has no formal definition of classes, but for our purpose is the most simple way of coding the class structure of the ESN and the most convenient.

Other important aspect of the package is the usage of C code to speed up the calculations. As we said before training the last layer of weights of an ESN is fast because the problem which is solved has closed form solution. Nevertheless, computing the internal state $\mathbf{X}$ can be slow especially if the amount of samples is big. A solution for this problem was using the package "Rcpp" to include C code for making this calculations, thanks to the C code we reduce the time of computation in approximately a factor of 10.

4.3 Methods

In this subsection we will explain the most important methods of the R package giving a short description, an explanation of the arguments and the return of each method.

4.3.1 ESN()

This is the core package method of the package, it allows to create an ESN. The return of this method is an object from the class "ESN" to which the methods that we will see later can be applied to it. The usage of this method is:

```
ESN(u, y, num_neurons, density, spectral_radius,
    fb_density = NULL, alpha = 1, pre_proc = "none",
    scale_in = 1, scale_fb = 1,  scale_noise_in = 0,
    scale_noise_gen = 0, scale_noise_fb = 0,
    type_Win = "uniform", user_define_Win = NULL,
    type_W = "uniform", user_define_W = NULL,
    type_Wfb = "uniform", user_define_Wfb = NULL,
    bias = TRUE, output_feedback = FALSE,
    initial_state = "zero", user_define_initial_state = NULL)
```

This method has a wide variety of arguments, here we will give a short explanation about them. The arguments `u` and `y` are the input and output matrices respectively. `num_neurons` refers to the size of the internal reservoir of the ESN, throughout this project we have called to this argument N but the name `num_neurons` is more intuitive and this is good for an user who is not familiarized with the notation and `spectral_radius` is the value for the spectral radius of the internal state weight matrix $\mathbf{W}$. The previous arguments are the only ones which don not have a default value. `alpha` is the value of α the leaking rate parameter described at section 2.4, the default value is $\alpha = 1$ i.e. ESN with no leaky integrator neurons. The `pre_proc` argument is an option for making a preprocessed of the output data, the options for this parameter are `"none"` which will make no preprocessed, `"linear"` that will substract the mean and the divide by the standard deviation and `"tanh"` that will made the same preprocessed than in the last option and then apply a tanh function to each value. `scale_in` and `scale_fb` refer to the scaling factor of the input weight matrix $\mathbf{W}^{\text{in}}$ and the feedback weight matrix $\mathbf{W}^{\text{fb}}$. The argument `type_Win` is used for selecting the distribution used for generating the weights $\mathbf{W}^{\text{in}}$, the options are `"discrete"` (for a bi-valued discrete distribution with values -1 or 1), `"uniform"` (for a uniform distribution in [-1, 1]), `"gaussian"` (for a Gaussian distribution with $\sigma = 1$) and `"user_define"`, an important observation is that the values of the discrete, the boundaries of the uniform and the variance of the Gaussian are fixed to 1 because this parameter can be changed using the scaling factor for the input `scale_in`. The argument `user_define_Win` is only important if `type_Win = "user_define"`, in this case the argument `user_define_Win` has to be a matrix provided by the user, with the correct dimension, that will be

used as input weight matrix. This allows to the user a total control over the weights if desired. And the same discussion can be extended to the arguments `type_W`, `user_define_W`, `type_Wfb` and `user_define_Wfb` but here the discrete option is called `"discreteZero"` to emphasize the fact that the matrix will also contain null value elements. The `bias` is a boolean parameter that chooses if a bias term is added to the input $\mathbf{u}$ or not. The `output feedback` is also a boolean parameter selecting if the net will have output feedback or not. The `initial_state` refers to the value that will be fixed for $\mathbf{x}(0)$ with a default option `"zero"` for $x(0) = 0$ also the option `"gaussian"` for being generated according to a Gaussian distribution with $\mu = 0$ and $\sigma = 1$ and the option `"user_define` that allows to the user to set the desired value for the initial state with the argument `user_define_initial_state`.

4.3.2 train()

This method us used for training an object from the class "ESN". The strategy for training is the one described at section 2.5 which is minimizing the MSE between $\mathbf{y}$ and $\hat{\mathbf{y}}$ with a Tikhonov regularization. The return of this method is the same trained ESN that is given as input. An example of usage:

```
train(net, discard = NULL, discard_pct = 0.2,
      beta = 1e-12, tol = 1e-30)
```

The argument `net` is the most important one, it refers to the ESN that will be trained. The `net` given has to be from the class ESN otherwise the `train()` will raise an error. `discard` refers to the number of samples that will be discarded during the training phase and the `discard_pct` refers to the fraction of samples (over the total amount of training samples) that will be discarded, this argument has to be between 0 and 1. The argument `beta` is the regularization coefficient β of the Tikhonov regularization and the `tol` is the tolerance of the R function `solve()` for solving this minimization problem.

4.3.3 predict()

This method is useful for seeing the performance and making predictions with a trained ESN, before using this method the train method must have been used, otherwise the code will raise an error. The return of this method is a list with two elements, the first element of the list is test MSE and the second is the predicted values for the given test input. The usage of this method:

```
predict(net, u, y, discard = NULL, discard_pct = 0.1,
        teacher_forcing_steps = NULL, teacher_forcing_pct = 0.1,
        reset_state = TRUE)
```

As in the train method the argument `net` refers to the ESN that will be tested. The `net` has to be from the class ESN otherwise the `predict()` will raise an error. The arguments `u` and `y` are the matrices with the test data for input and ouput respectively, the format of this matrices is the same that in the `ESN()`

method. `discard` refers to the number of samples that will be discarded during the testing phase and the `discard_pct` refers to the fraction of samples (over the total amount of testing samples) that will be discarded, this argument has to be between 0 and 1. In a similar way `teacher_forcing_steps` refers to the number of samples that will be computed using teacher forcing and `teacher_forcing_pct` is the fraction of samples (over the total amount of testing samples) that will be computed with teacher forced, once again this argument has to be between 0 and 1. `reset_state` is a useful argument when the train and test data come from the same sequence, if `reset_state` is fixed to `FALSE` then the initial testing state $_{\text{test}}(0)$ will be the last value from the training internal state. If `reset_state = TRUE`, which is the default option, the value of $_{\text{test}}(0)$ will be fixed arbitrarily.

```
# Create a new object ESN
net <- ESN(u = synthetic_sinusoidal$u_train,
           y = synthetic_sinusoidal$y_train,
           num_neurons = 100,
           density = 0.05,
           fb_density = 0.05,
           scale_in = 1,
           spectral_radius = 0.9,
           scale_fb = 1,
           type_Win = "discrete",
           type_W = "discreteZero",
           type_Wfb = "discreteZero",
           output_feedback = TRUE)

# Then use the train method in this case discarding the first 100 samples
net <- train(net, discard_pct = 0.2)

# And then use the predict method. In this case we are using 30 steps of teacher forcing.
res <- predict(net,
               synthetic_sinusoidal$u_test,
               synthetic_sinusoidal$y_test,
               discard_pct = 0.3)
print(res$MSE)
#> [1] 3.158626e-14
```

Figure 9: Example from the package vignette of using the 3 methods mentioned above.

4.3.4 print()

This method gives a summary of the properties of a ESN.

```
print(net)
```

It has only one argument which is `net` it has to be an object from the class ESN and refers to the network that will be printed.

```
print(net)
#> Ecuation:
#> x(n) = (1 - alpha)*x(n - 1) + tanh(Win*u(n) + W*x(n - 1) + Wfb*y(n - 1))
#>
#> Net trained, train MSE: 1.360432e-14
#>
#> Number of neurons: 100
#> Spectral Radius: 0.9
#> alpha: 1
#> Output feedback: yes
#>
#> Type input weights: discrete
#> Type internal weights: discreteZero
#> Type output feedback weights: discrete
```

Figure 10: Example from the package vignette of the print() method.

5 Finance applications

5.1 VAR Models

The first part of this chapter consists in the definition of the VAR model which are a simple and useful model in the finance field. Let p_t be the price of an asset at (discrete) time index t, we define the returns as:

$$R_t := \frac{p_t - p_{t-1}}{p_{t-1}} = \frac{p_{t-1}}{p_t} - 1 \tag{14}$$

Working with assets is more common working with the log-returns, its definition is:

$$r_t = log(1 + R_t) = y_t - y_{t-1} \tag{15}$$

The VAR model is one of the simplest models which has memory of the past to predict the future. The notation of a VAR model is VAR(p) where p is the order of the model. A Var(p) is define in the following way:

$$\mathbf{r}_t = \boldsymbol{\phi}_0 + \sum_{i=1}^{p} \boldsymbol{\Phi}_i \mathbf{r}_{t-i} + \mathbf{w}_t \tag{16}$$

where Φ_0 and $\boldsymbol{\Phi}_i$ with $i = 1, 2, \ldots, p$ are the coefficients of the VAR model. As can be seen from the definition the value of a VAR(p) model r_t at time t directly depends on the values $r_{t-1}, r_{t-2}, \ldots, r_{t-p}$ but each of the r_{t-k}values depends also on the last p values so the value r_t is correlated with all the past values r_i with $i = 1, 2, \ldots, t-1$.

For the experiments in this chapter generating synthetic data using a VAR model will be needed, but a VAR model can be unstable so for checking the stability the following result can be useful.
A VAR(p) model given by (**??**) is stationary if

$$\det(\mathbf{I}_k - \sum_{i=1}^{p} \boldsymbol{\Phi}_i z^i) \neq 0 \quad \forall\ |z| \leq 1 \tag{17}$$

And (17) is verified if and only if the modulus of the eigenvalues of $\mathbf{F}$ lie at the unit circle, where $\mathbf{F}$ is:

$$\mathbf{F} = \begin{bmatrix} \boldsymbol{\Phi}_1 & \boldsymbol{\Phi}_2 & \ldots & \boldsymbol{\Phi}_{p-1} & \boldsymbol{\Phi}_p \\ \mathbf{I}_k & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{I}_k & \mathbf{0} \end{bmatrix} \tag{18}$$

5.2 Comparison of the ESN with VAR models

In this section we have generate synthetic data using different VAR models, specifically a VAR(1) a VAR(3) and a VAR(5). The parameters of this models where randomly generated but always checking the stability condition using (17) and (18). Also a Gaussian noise was added with $\sigma^2 = 10^{-4}$ The idea was predict the time series generated by the VAR model using different models. The naive model is simply a mean with a small rolling window. The clairvoyant is the true model for each case with the true parameters. Also a VAR(1), VAR(3) and VAR(5) where used for predicting each time series. An ESN was used for each time series too, this are the obtained results:

MSE ($\times 10^{-3}$)	VAR(1)	VAR(3)	VAR(5)
Naive	2.4	2.2	2.8
Clairvoyant	0.093	0.094	0.094
Estimated VAR(1)	0.093	0.12	0.16
Estimated VAR(3)	0.093	0.094	0.12
Estimated VAR(5)	0.093	0.094	0.095
ESN	0.10	0.11	0.13

Table 1: Table showing the results of the comparison of the VAR models with the ESN

We have tried different configurations of ESN for each model and the value that appears at Table 1 is always the one which give the best performance. For training each model 1000 samples were used and 200 different samples were used for testing. For the VAR of the small order the ESN chosen had a small reservoir $N = 30$. And for the bigger VAR models usually a reservoir of $N = 100$. As can be seen the performance of the ESN is not the best on this synthetic datasets but this is not a strange result because usually a VAR model with the same or higher order of the estimated one will perform better than the ESN because this model has capacity of predicting well the VAR model and has a much smaller amount of parameters. The interesting result here is that the performance of the ESN is slightly better than the one obtained predicting with a VAR model of smaller order than the original one.

6 Conclusions and future work

As conclusions we can said that the characterization of echo states showed in section 2.2 is hard to check when we are using the ESN in a practical way. The result of Proposition 2 can be easily checked but it is not a if and only if result, so in most of the cases the technique used is trial and error. ESN can perform well a wide variety of tasks from regression to classification in very different fields. Also the R package created in this project is such an useful tool that can speed up future research on this topic. The performance of the ESN in the finance application is not as good as we expected but more comparisons using different data should be done.

As future work related with the R package a significant improvement will be add the option of different metrics for the training phase, now the strategy is always minimize the MSE between $\mathbf{y}$ and $\hat{\mathbf{y}}$ but in some cases different metrics can be more suitable like binary crossentropy for classification tasks. Also the case of weights generated with a bi-valued binary distribution can be speed up due to the simplicity of this distribution.

An new interesting application will be using the ESN for predicting the volatility clustering of the time series formed by the price of an asset. Usually the models used for this task are the GARCH and its performance is good but they need a lot of data for training and also for prediction and maybe the ESN can obtain a similar performance with less test data. The last task for future work presented here is using ESN for classification of financial time series, throughout this project we have focused in using ESN for forecasting of time series but as we have seen in section 3 the performance of the ESN can be really good also in classification task so this path should be explored in the finance field.

References

[1] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning internal representations by error propagation.* In Neurocomputing: Foundations of research, pages 673-695. MIT Press, Cambridge, MA, USA, 1988.

[2] Paul J. Werbos. *Backpropagation through time: what it does and how to do it.* Proceedings of the IEEE, 78(10):1550-1560, 1990.

[3] Ronald J. Williams and David Zipser. *A learning algorithm for continually running fully recurrent neural networks.* Neural Computation, 1:270-280, 1989.

[4] Herbert Jaeger. *The "echo state" approach to analysing and training recurrent neural networks.* Technical Report GMD Report 148, German National Research Center for Information Technology, 2001.

[5] Herbert Jaeger. *Echo state network.* Scholarpedia, 2(9):2330, 2007.

[6] David Verstraeten, Benjamin Schrauwen, Michiel D'Haene, and Dirk Stroobandt. *An experimental unification of reservoir computing methods.* Neural Networks, 20(3):391-403, 2007.

[7] Herbert Jaeger and Harald Haas. *Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication.* Science, 304(5667):78-80, 2004.

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

[9] Xiaowei Lin, Zehong Yang, Yixu Song. *Short-term stock price prediction based on echo state networks* Expert Systems with Applications Volume 36, Issue 3, Part 2, April 2009, Pages 7313-7317.

[10] James Martens and Ilya Sutskever. *Learning recurrent neural networks with Hessian-free optimization.* In Proc. 28th Int. Conf. on Machine Learning, 2011.

[11] Herbert Jaeger. *Long short-term memory in echo state networks: Details of a simulation study.* Technical Report No. 27, Jacobs University Bremen, 2012.

[12] Herbert Jaeger. *Short term memory in echo state networks.* Technical Report GMD Report 152, German National Research Center for Information Technology, 2002.

[13] Alireza Goudari, Alireza Shabani, Darko Stefanovic. *Product Reservoir Computing: Time-Series Computation with Multiplicative Neurons* arXiv:1502.00718v2 26 Apr 2015.

[14] Mantas Lukoševičius *A Practical Guide to Applying Echo State Networks* Neural Networks: Tricks of the Trade, Reloaded.

[15] Qianli Ma, Lifeng Shen, Weibiao Chen, Jiabin Wang, Jia Weia, Zhiwen Yu *Functional echo state network for time series classification* Information Sciences Volume 373, 10 December 2016, Pages 1-20.

[16] Rikke Amilde Løvlid *A Novel Method for Training an Echo State Network with Feedback-Error Learning* Advances in Artificial Intelligence Volume 2013.

[17] Zhong, Shisheng, Xie, Xiaolong, Lin, Lin, Wang, Fang *Genetic algorithm optimized double-reservoir echo state network for multi-regime time series prediction* Neurocomputing Volume 238, 17 May 2017, Pages 191-204.

[18] H. K. Khalil. *Nonlinear Systems (second edition).* Prentice Hall, 1996.