JACOBS
UNIVERSITY

Mantas Lukoševičius, Herbert Jaeger

Overview of Reservoir Recipes

Technical Report No. 11
July 2007

School of Engineering and Science

Overview of Reservoir Recipes

A survey of new RNN training methods that follow the Reservoir paradigm

Mantas Lukoševičius[1], **Herbert Jaeger**

School of Engineering and Science
Jacobs University Bremen gGmbH
Campus Ring 12
28759 Bremen
Germany

Summary

Echo State Networks (ESNs) and Liquid State Machines (LSMs) introduced a simple new paradigm in artificial recurrent neural network (RNN) training, where an RNN (the *reservoir*) is generated randomly and only a readout is trained. The paradigm, becoming known as reservoir computing, made RNNs accessible for practical applications as never before and outperformed classical fully trained RNNs in many tasks. The latter, however, does not imply that random reservoirs are optimal, but rather that adequate training methods for them are yet to be developed. Thus much of the current research in reservoir computing is done on reservoir adaptation, redefining the paradigm as using different methods for training the reservoir and the readout. This report motivates the new definition of the paradigm and surveys the reservoir generation/adaptation techniques, offering a natural conceptual classification which transcends boundaries of the current "brand-names" of reservoir methods. The survey focuses more on methods relevant to practical applications of RNNs rather than modeling biological brains.

[1]Corresponding author, email: m.lukosevicius@jacobs-university.de

Contents

1 Introduction

Training recurrent neural networks (RNNs) has always been a difficult problem. This has impeded applications of RNNs for real life problems despite their lage theoretical potential. With Echo State Networks (ESNs) and Liquid State Machines (LSMs) a new, much less complex paradigm for training RNNs was introduced. The paradigm, becoming known as *reservoir computing*, was based on an observation, that as long as a randomly generated RNN (the reservoir) possess certain properties, it is not necessary to train it, and training only a recurrence-free linear readout is sufficient for many tasks. ESNs have been demonstrated to perform extremely well on several benchmark tasks, outperforming fully trained ESNs [Jae01]. The reservoir computing paradigm recently triggered many practical applications of RNNs and a whole new stream of research.

Having no recurrences and nonlinearities in the trained part of the model is what enables the fast and optimal training using linear regression. The same feature, however, reduces the expressivity of the model, compared to fully trained RNNs with the same number of neurons. Empirical simulations suggest that for highly nonlinear tasks, achieving a good internal dynamics of the reservoir which can be linearly mapped into desired output is hard, and increasing the number of neurons in the reservoir does not easily improve the situation. Thus much of the current research in reservoir computing is done on generating or training reservoirs that are not (completely) random. This report offers a conceptual classification and extensive survey of this stream of research. The paradigm differentiates itself from the classical error backpropagation approach for training RNNs in that the reservoir and the output are trained separately, using different techniques. The reservoir-readout separation is both conceptual and algorithmical.

More details on the ideas expressed so far are presented throughout the rest of this text, which is organized in the following way. We start by introducing our formalism and notation in Section 2, which will be used throughout the rest of this survey. More specifically, we define what we mean by "problem" or "task" in the context of machine learning in Section 2.1. Then we define a very general framework of notation for expansion (or kernel) methods for both non-temporal in Section 2.2 and temporal in Section 2.3 task. We will show the temporal case as a natural extension of the non-temporal one. Then we introduce our notation for recurrent neural networks in Section 2.4, and give some references to their classical training methods in Section 2.5.

In Section 3 we will briefly introduce the idea and conceptual motivation of Reservoir Computing and proceed by naming the most prominent methods that follow this line of thought.

In Section 4 we will introduce our classification of reservoir recipes together with some remarks on it. Then, following this classification we will review universal – Section 5, unsupervised – Section 6, and supervised – Section 7 reservoir generation/pretraining recipes. The recipes span through different reservoir meth-

ods, but most of them are related to Echo State Networks.

We will end with some closing remarks in Section 8.

2 Formalism

2.1 Formulation of the Problem

Let a *problem* or a *task* in machine learning be defined as a problem of learning a functional relation between a given input $u(n) \in \mathbb{R}^{N_u}$ and desired output $y_{\text{target}}(n) \in \mathbb{R}^{N_y}$, where $n = 1, \cdots, T$ and T is the number of data points in the training *dataset* $\{(u(n), y_{\text{target}}(n))\}$. A *non-temporal* task is where the data points are independent of each other and the goal is to learn the function $y(n) = y(u(n))$, such that $E(y, y_{\text{target}})$ is minimized, where E is a defined error function, e.g. normalized root-mean-square error (NRMSE)

$$E(y, y_{\text{target}}) = \sqrt{\frac{\left\langle \|y(n) - y_{\text{target}}(n)\|^2 \right\rangle}{\left\langle \|y_{\text{target}}(n) - \langle y_{\text{target}}(n) \rangle\|^2 \right\rangle}}, \tag{1}$$

where $\|\cdot\|$ stands for the Euclidean distance (or norm). A *temporal* task is where u and y_{target} are signals in a discrete time domain $n = 1, \cdots, T$ and the goal is to learn a function $y(n) = y(\cdots, u(n-1), u(n))$ such that $E(y, y_{\text{target}})$ is minimized.

Thus the main difference between the temporal and non-temporal task is that the function $y(\cdot)$ we are trying to learn has memory in the first case and is memoryless in the latter. In both cases the conjecture is that the function(-al dependence) we are trying to learn actually exists, i.e. in the case of the temporal task $y_{\text{target}}(n) = y_{\text{target}}(\cdots, u(n-1), u(n)) + \theta(n)$, where $\theta(n)$ is a noise term limiting the precision of the learning.

Normally one part of the T data points is used for training the model and the other part (unseen during the training) for testing it. We will not distinguish between these two phases/data parts throughout this report for brevity. When speaking about the output errors and "performance" or "precision" we have *testing* errors in mind (if not specified explicitly). Also n, denoting discrete time, will be used omitting its range.

2.2 Expansions in Non-temporal Tasks

There are many powerful mathematical methods for solving linear equations. Most of the nontrivial tasks, however, do not lend themselves to be expressed in a simple linear relation between the u and y_{target}. In more formal words, for many tasks $\{u, y_{\text{target}}\}$ a linear model $y(n) = Wu(n)$ (where $W \in \mathbb{R}^{N_y \times N_u}$) gives big errors $E(y, y_{\text{target}})$ for any W.

Many methods in statistical machine learning are based on the idea of expanding the input $u(n)$ into a high-dimensional feature vector $x(n) \in \mathbb{R}^{N_x}$ and then

combining those features in a linear fashion as a regression for real-valued outputs or a linear separator for classifiers.[2] They can be expressed in the form

$$y(n) = W_{\text{out}} x(u(n)), \tag{2}$$

where $W_{\text{out}} \in \mathbb{R}^{N_y \times N_x}$. Typically $N_x \gg N_u$. There is also typically a constant *bias* value added to (2), which is omitted here and in other equations for simplicity. The bias can be easily implemented, having one of the features in $x(n)$ constant (e.g. $= 1$) and a corresponding column in W_{out}). Some models are defined as

$$y(n) = f_{\text{out}}(W_{\text{out}} x[u(n)]), \tag{3}$$

where $f_{\text{out}}(\cdot)$ is some nonlinear function (e.g. sigmoid). We will consider this definition as equivalent to (2), since $f_{\text{out}}(\cdot)$ can simply be eliminated from y by redefining the target as $y'_{\text{target}} = {f_{\text{out}}}^{-1}(y_{\text{target}})$ (and the error function $E(y, y'_{\text{target}})$ if desired). Note, that (2) is a special case of (3), with the $f_{\text{out}}(\cdot)$ being the identity.

The *expansion* function $x(n) = x(u(n))$ in some methods is called *kernel function* and those methods – *kernel methods*. The *kernel methods* are often defined in literature as the methods in which the *kernel trick* (meaning that we don't need to calculate $x(n)$ explicitly) is applied. Thus we will use a broader notion of *expansion* function $x(u(n))$ and call any method corresponding to (3) (or (2) as a special case) an *expansion method*. These methods then include Support Vector Machines (SVMs) and Feed-Forward Neural Networks (FFNNs)[3] among many others.

2.3 Expansions in Temporal Tasks

Many temporal methods are also based on the same principle. The difference is that in temporal task the function to learn depends also on the history of the input, as discussed in Section 2.1. Thus, the expansion function has memory: $x(n) = x(\cdots, u(n-1), u(n))$, i.e. it is an expansion of the current input and its (potentially infinite) history. Since this function has an unbounded number of parameters, practical implementations often take an alternative, recursive, definition:

$$x(n) = x(x(n-1), u(n)). \tag{4}$$

The output is typically produced in the same way as for non-temporal methods (2) or (3).

[2]We will focus on real-valued outputs in this review. Linear separation can be seen as a thresholded linear readout, thus the results will also apply to some types of classification.

[3]The one-before-output layer of a FFNN can by considered as $x(u(n))$, and the output weights then correspond to W_{out} in (3).

2.4 Recurrent Neural Networks

The type of recurrent neural networks that we will consider most of the time in this report is a straightforward implementation of the latter definition. The nonlinear expansion with memory is done into the state vector

$$x(n) = f(W_{\text{in}}u(n) + Wx(n-1)), \quad n = 1, \cdots, T, \tag{5}$$

where $x(n) \in \mathbb{R}^{N_x}$ is a vector of internal neuron activations at a time step n, $f(\cdot)$ is the neuron activation function, usually the symmetric $\tanh(\cdot)$, or the positive logistic (or Fermi) sigmoid, applied element-wise, $W_{\text{in}} \in \mathbb{R}^{N_x \times N_u}$ is the input weight matrix and $W \in \mathbb{R}^{N_x \times N_x}$ is a weight matrix of internal network connections. The network is usually started with the initial state $x(0) = 0$. Bias values are again omitted in (5) in the same way as in (2). The readout $y(n)$ of the network is implemented as in (3).

Some models of RNNs extend (5) as

$$x(n) = f(W_{\text{in}}u(n) + Wx(n-1) + W_{\text{ofb}}y(n-1)), \quad n = 1, \cdots, T, \tag{6}$$

where $W_{\text{ofb}} \in \mathbb{R}^{N_x \times N_y}$ is an optional output feedback weight matrix. But we will most of the time consider model (5) which is less complicated.

2.5 Error Backpropagation in RNNs

The classical approach of training RNNs is by using error backpropagation (BP) methods and gradual iterative adaptation of the weights W_{out}, W, and W_{in}. The BP methods for RNNs are extensions of the PB methods in FFNNs. A systematic overview of the BP methods for RNNs is presented in [AP00]. However, applying BP for RNNs is significantly harder than for FFNNs [BSF94]. The [AP00] contribution also proposes a significant improvement to the previous BP methods for RNNs. The method is often referred to by other authors as the Atiya-Parlos recurrent learning (APRL).

3 Reservoir Methods

Reservoir computing is a recently emerging term describing a group of novel RNN methods that make the conceptual separation between the *reservoir* – an RNN as a nonlinear temporal expansion function, and the *readout*, which produces the desired output from the expansion.

A question might arise, why this conceptual separation between the expansion $x(n)$ and the final readout $y(n)$ is useful at all. It is true that e.g. in RNNs with error back-propagation training methods both $x(n)$ and W_{out} are trained in the same way, thus the separation does not make much sense. Reservoir methods as well as nontemporal kernel methods, however, are based on the understanding

that $x(n)$ and $y(n)$ serve different purposes. $x(n)$ expands the input $u(n)$ into a space where the data points can be linearly combined, while $y(n)$ does the latter. By "linearly combined" we mean, that for each dimension y_i of y a vector in the state space $\mathbb{R}^{N_x}$ is found where projection of each point $x(n)$ on the vector is equal to $y_i(n)$.

From a more philosophical point of view these two stages of the solution process could be called *analysis* and *synthesis*.

Another, more empirical, way of arriving to this separation in the context of RNNs is sketched in Section 3.4.

Since the expansion and the readout serve different purposes, training/generating them separately and even with different goal functions makes sense. For the linear *readouts* $y(n)$ as in (2), exact *regression* methods are well known that find optimal W_{out}, i.e. giving the minimal *mean-square* or equivalently (1) error.[4] But producing a good expansion function $x(n)$ usually involves more creativity. In many expansion methods the expansion function is chosen or generated "by hand" (most often through trial-and-error) and is fixed.

The different "brand names" of the *reservoir methods* mainly differ in types of units (neurons) that are used for the reservoir. Some have also distinctive reservoir generation or training algorithms attached. In the following subsections we will mention the most prominent reservoir methods.

3.1 Echo State Networks

Echo State Networks (ESNs) [Jae01][Jae02b] is one of the two pioneering reservoir methods that uses the most popular (and perhaps one of the most simple-minded) neuron models in machine learning, namely a neuron which adds its weighted inputs an applies a sigmoid function (usually $f(\cdot) = \tanh(\cdot)$) to the sum. The random sparse reservoir of such units is governed by (5). Classical "recipes" of producing the ESN reservoir are outlined in Section 5.1. The leaky integrator neuron model is another option for the classical ESNs. This option is outlined in Section 5.5.

The readout from the reservoir is usually linear as in (3), but also including direct mappings from $u(n)$ to $y(n)$:

$$y(n) = f_{\text{out}}(W_{\text{out}}[u(n)|x(n)]), \tag{7}$$

where $W_{\text{out}} \in \mathbb{R}^{N_y \times (N_u + N_x)}$ is the output weight matrix, that is learnt by linear regression, $f_{\text{out}}(\cdot)$ is the output neuron activation function (usually the identity) applied component-wise, and $\cdot|\cdot$ stands for a vertical concatenation of vectors.

[4] In fact, readouts of reservoir methods can be trained the same way as in Support Vector Machines [SGWG05] [SH07], following the analogy between the temporal and non-temporal expansion methods.

3.2 Liquid State Machines

Liquid State Machines (LSMs) [MNM02] is the other pioneering reservoir method developed independently from ESNs. While ESNs are developed with practical/engineering applications in mind, LSMs are developed aiming to realistically model the processes in biological brains. Thus LSMs use more sophisticated and biologically realistic models of spiking neurons for the reservoir. The connectivity among the neurons is based on their spacial embedding and follows biologically plausible distributions of connections with respect to their length. Inputs to LSMs also usually consist biologically plausible spike trains. In their readouts LSMs originally used FFNNs, but most current versions have linear readouts similar to ESNs. Some additional mechanisms for averaging the spike trains are often employed.

3.3 Evolino

Evolino [SWGG07] transfers the idea of ESNs from an RNN of simple sigmoidal units to a Long Short-Term Memory (LSTM) type of RNNs [HS97] constructed from units capable of preserving memory for long periods of time. In Evolino the weights of the reservoir are trained using evolutionary methods, as it is also done in some extensions of ESNs discussed in Section 7.2.

3.4 Backpropagation-Decorrelation

The idea of separation between the dynamical reservoir and a readout function has also been arrived at from the point of view of optimizing the performance of the RNN training algorithms that use error backpropagation, mentioned in Section 2.5.

There was an analysis made of the weight dynamics of an RNN trained by APRL [SS05]. It revealed that the output weights W_{in} of the network being trained are changing quickly, while the hidden weights W change slowly and in a case of single output $N_u = 1$ the changes are column-wise coupled. Thus in effect APRL decouples the RNN into a fast adapting output and a slowly adapting reservoir. Inspired by these findings a new iterative/online[5] RNN training method, called Backpropagation-Decorrelation (BPDC) was introduced [Ste04]. BPDC uses the reservoir update equation defined in (6) with the same type of units as ESNs, thus in contrast to other "brand names" of reservoir methods discussed so far, BPDC stands for a different approach to learning rather than a different type of neurons used.

[5] As opposed to "single shot" linear regression training. Reservoir methods with linear readout can also have an online version of training [Jae03] [JH04].

3.5 Other Types of Neurons

Using different activation functions inside a single reservoir might improve the richness of the echo states, as it is illustrated with inserting some neurons with a wavelet-shaped activation functions into the reservoir of ESNs [WYW06].

A hardware implementation friendly version of reservoirs composed of stochastic bitstream neurons was proposed in [VSS05].

3.6 Other Overviews of Reservoir Methods

An experimental comparison of LSM, ESN, and BPDC reservoir methods with different neuron models, even beyond the standard ones used for the respective methods, and different parameter settings is presented in [VSDS07].

A rather extensive but also quite subjective overview of current literature on (different flavors of) ESNs can be found in the appendix of [KÖ6].

A recent overview of many aspects of reservoir computing is presented in [SVVC07]. The overview is much briefer and broader (also less focused, sparser), covering literature on other methods for RNN training, applications and hardware implementations of reservoir methods, also covering more extensively the biological side of the reservoir research, than the overview your are currently reading.

4 Our Classification of Reservoir Recipes

Reservoir methods demonstrated (e.g. [Jae01]) that randomly generated RNNs with only trained outputs can (substantially) outperform the ones trained using computationally expensive state-of-art backpropagation methods. These results do not imply, however that randomly generated reservoirs are optimal and can not be improved. In fact "random" by definition can not be "the optimal". They rather indicate the need for some novel methods of training/generating the reservoirs, that are very probably not a direct extension of the way the output is trained (as in error backpropagation). Thus much of the current research on reservoir methods is done developing them. In fact, even the above mentioned "random" reservoirs of the classical ESNs are generated using carefully selected parameters, as we will see in Section 5.1.

It is well worth mentioning at this point, that the general "no free lunch" principle in supervised machine learning [Wol01] (similar to the one in optimization) states that there can exist no bias of a model which would universally improve the accuracy of the model for all possible problems it is solving. In our context this can be translated into a claim that no single reservoir can be optimal for all types of problems.

In this report we will try to survey all basic ideas that help us produce good reservoirs. We will classify those ideas into tree major groups based on their universality:

- *General* guidelines/methods of producing good reservoirs irrespective to the task (both the input $u(n)$ and the desired output $y_{\text{target}}(n)$);
- *Unsupervised* pre-training of the reservoir with respect to the given input $u(n)$, but not the target $y_{\text{target}}(n)$;
- *Supervised* pre-training of the reservoir with respect to both the given input $u(n)$ and the desired output $y_{\text{target}}(n)$.

These three classes of methods are discussed in the following three sections of this overview. Note that many of the methods to some extend transcend the boundaries of these three classes, but are classified according to their main principle.

Even though this overview aims at covering developments under all reservoir "brands", it is slightly (?) biased towards machine learning applications and different flavors of ESNs.

5 Universal Reservoir Recipes

While we have said, that this group of methods aims to produce good reservoirs *irrespective to the task* at hand, this is not entirely true. Some parameters for generating good reservoirs *are* more or less dependent on the task (following the "no free lunch" principle), but the methods discussed in this section preset them manually, rather than doing an automated systematic search optimizing a predefined goal function.

5.1 Classical ESN approach

The most fundamental guidelines of producing good reservoirs were presented with the very introduction of the ESNs [Jae01][Jae02b]. Motivated by an intuitive goal of producing "rich" set of dynamics the recipe is to generate a (i) *big*, (ii) *sparsely* and (iii) *randomly* connected reservoir. This means that (i) N_x is sufficiently large: the order ranging from tens to thousands, (ii) the weight matrix W is sparse: several to 20 percent of possible connections, and (iii) weights of the connections are usually generated randomly from a uniform distribution symmetric around the zero value[6] [Jae02b]. Such reservoir will have many, due to (i), activations, that are not tightly coupled, due to (ii), and different, due to (iii).

The input weights W_{in} and the optional output feedback weights W_{ofb} are usually dense (can also be sparse like W) generated randomly from a uniform distribution. The exact scaling of both matrices and an optional shift of the input are the few other free parameters that one has to choose, when "baking" an ESN. The vague rules of thumb for them are following. The scaling (W_{in}) and shifting of the input depends on how much nonlinearity of the processing unit the task needs:

[6] In some early experiments [Jae01] with ESNs the connections are assigned weight values of the same magnitude only choosing the sign of the value (positive or negative) randomly.

if inputs are close to 0, the tanh neurons tend to operate with activations close to 0, where they are virtually linear, while inputs far from 0 tend to drive them more toward saturation where they exhibit more nonlinearity [Jae02b]. The shift of the input might help to resolve tanh ESN's symmetry with respect to the sign of the signals. By the symmetry we mean, that if $x(u(n)) = x(n)$ and $y(u(n)) = y(n)$, then $x(-u(n)) = -x(n)$ and $y(-u(n)) = -y(n)$ (taking the constant bias input to the neurons as part of $u(n)$). The scaling of W_{ofb} is in practice limited by a threshold at which the ESN starts to exhibit an unstable behavior, i.e. the output feedback loop starts to amplify (the errors of) the output and thus enters a diverging generative mode.

A crucial element for ESNs to work is that reservoir should have the *Echo State Property* [Jae01]. This condition in essence states that the effect of an input $u(n)$ on the reservoir activation $x(n+k)$ should die out gradually as time passes $k \to \infty$, and not to persist or even get amplified. For practical purposes the condition is satisfied if the generated W is scaled so that its spectral radius $\rho(W) < 1$ [Jae01]. The optimal value of $\rho(W)$ should be set depending on the amount of memory which the given task requires. A rule of thumb is that $\rho(W)$ should be close to 1 for tasks that require long memory and accordingly smaller for the tasks where a too long memory might in fact be harmful.

Recently a less restrictive sufficient condition than $\rho(W) < 1$ has been derived [BY06].

It has also been shown that an ESN with a reservoir generally not satisfying the echo state condition can still work [OP05] reasonably well. The main principle of this result is that even though the reservoir exhibits meaningless (to the task) self-induced activations in absence of the input (i.e. the activations do not gradually die-out as in proper ESNs), a strong enough input $u(n)$ can "push out" this meaningless activity and make the activations dependent on the (fading) history of $u(n)$.

5.2 Different Topologies of the Reservoir

There have been attempts made to find alternative topologies of the ESN reservoir to the sparse random network. The famous models of small-world [WS98], scale-free [BA99], and biologically inspired networks generated by spatial growth [KH04] where tested for this purpose in [Lie04]. The NRMS error (1) of $y(n)$ as well as the eigenvalue spread of the cross-correlation matrix of the activations $x(n)$ (see Section 6.1 for more details on this) where used as the performance measures of the topologies. The datasets used were a synthetic NARMA (Nonlinear Autoregressive Moving Average) time series and the Mackey-Glass chaotic attractor. The contribution also explored an approach of exhaustive brute-force search of topologies of tiny networks (motives) of 4 units, and then combining successful motives (in terms of the eigenvalue spread) into larger networks. The investigation, unfortunately, concludes, that "*... none of the investigated network*

topologies was able to perform significantly better than simple random networks, both in terms of eigenvalue spread as well as testing error" [Lie04]. This, however, does not serve as a proof that similar approaches are futile.

It has been demonstrated that the reservoir can even be an unstructured feed-forward network if the finite limited memory window that it offers is enough for the task at hand [vM05].

A degenerate case of a "reservoir" composed of linear units and a diagonalized W and unitary inputs W_{in} is considered in [FE05][7].

A one-dimensional lattice (ring) topology was used for a reservoir, together with an adaptation of the reservoir discussed in Section 6.2, in [VSVC07].

5.3 Modular Reservoirs

One of the shortcomings of the conventional ESN reservoirs is that even though they are sparse, the activations are still coupled enough so that the ESN is poor in dealing with different time scales simultaneously, e.g. predicting several superimposed sine waves of different frequencies. This problem was successfully tackled by dividing the reservoir into decoupled sub-reservoirs and introducing inhibitory connections among all the sub-reservoirs [XYH07]. For this approach to be effective, the inhibitory connections must predict the activations of the sub-reservoirs one time step ahead. For this two different methods are proposed in the contribution.

The Evolino approach introduced in Section 3.3 can also be classified as belonging to this group, as the LSTM RNN used for its reservoir consists of specific small memory-holding modules (which could alternatively be regarded as more complicated units of the network).

5.4 Time-Delayed vs. Instantaneous Connections

Another time-related limitation of the classical ESNs pointed out in [Luk07] is that no matter how many neurons are contained in the reservoir, looking from a perspective of a simple recurrent networks the reservoir (like any other fully recurrent network) is only a single layer. Consider a problem where the mapping from $u(n)$ to $y(n)$ is a very complex one, and the data in neighboring time steps is almost independent (i.e. little memory is required), e.g. the "meta-learning" task as in [PFT02].[8] Inside a single time step n signals from the input $u(n)$ propagate only through one untrained layer W_{in}, influence the activations $x(n)$ and reach the output $y(n)$ through the trained weights W_{out}. Thus ESNs are not capable of producing a very complex *instantaneous* mapping from $u(n)$ to $y(n)$, which

[7] The contribution erroneously states that weights in regular ESNs are drawn from normal distributions.

[8] ESNs have been shown to perform well in a (significantly) simpler version of the "meta-learning" in [OLS05].

could (only) be done by a multilayer FFNN.[9] The effect of propagating signals through several, say k, layers of units can in principle be achieved in k time steps, i.e. mapping $u(n)$ to $y(n+k-1)$ (we could shift y_{target} accordingly), as in this time the signals "reverberate" or "cross" units of the reservoir k times (looking at a time-expanded ESN as a multilayer FFNN). But during this time the signals get "mixed" with the ones coming from the different time steps and being in the different virtual "layers". If by the nature of the problem time steps are almost independent, this mixing kicks in as a very strong noise (the bigger k, the more) for each individual time step, which ruins the complex mappings of individual time steps. As a possible remedy an idea of Layered ESNs was presented [Luk07], where part (up to almost half) of the connections can be instantaneous, and the rest take one time step for the signals to propagate as in normal ESNs. Randomly generated Layered ESNs, however, do not offer a consistent improvement for large classes of tasks, and pre-training methods of such reservoirs have not yet been investigated.

5.5 ESNs with Leaky Integrator Units

In addition to the basic sigmoid units, leaky integrator neurons were suggested to be used in ESNs from the point of their introduction [Jae01]. This type of neurons does a leaky integration of its activation, i.e. partially remembers its previous activation. Instead of (5), a reservoir of such units is governed by

$$x(n) = (1 - a\Delta t)x(n-1) + \Delta t f(W_{\text{in}}u(n) + Wx(n-1)), \tag{8}$$

where Δt is a compound time gap between two consecutive time steps divided by the time constant of the system and a is the decay (or leakage) rate [LPJS06] – the two new free parameters that the leaky integrator ESNs (LIESNs) have compared to simple ESNs. Other flavors of leaky integrator neurons (or a more advanced discretization than Euler's) can also be used instead of (8). Note that the simple ESN (5) is a special case of LIESNs (8) with $a = 1$ and $\Delta t = 1$. As a corollary, a LIESN with a good choice of the parameters can always perform *at least* as good as a corresponding simple ESN. With the introduction of the two new parameters a and Δt, th conditions for the echo state property is redefined [Jae01]. A natural constraint on the two new parameters is $a\Delta t \in [0, 1]$ – a neuron should neither retain, nor leak more activation than it had.

The additional parameters of the LIESN control the "speed" of the reservoir dynamics. Small values of a and Δt result in reservoirs that react slowly to the input. By changing these parameters it is possible to shift the effective interval of frequencies in which the reservoir is working.

Along these lines, time warping invariant ESNs (TWIESNs) – an architecture that can deal (e.g. detect temporal patterns) with strongly time-warped signals

[9] In theory, a big single hidden layer is also capable of producing arbitrary complex mappings, but that is not realistic if only the outputs are trained.

was framed [LPJS06] [JLPS07]. It varies Δt on-the-fly, directly depending on the speed at which the input $u(n)$ is changing.

6 Unsupervised Reservoir Pretraining

In this section we will describe reservoir training/generation methods that try to optimize some measure on the activations $x(n)$ of the reservoir, for a given input $u(n)$, but regardless of the desired output $y_{\text{target}}(n)$. In Section 6.1 we survey measures of reservoir activations $x(n)$ that are used to estimate the quality of the reservoir, detached from the methods optimizing them. Then the local (Section 6.2) and global (Section 6.3) unsupervised reservoir training methods are surveyed. Note, that not all the measures discussed in Section 6.1 are reported in the literature to be used as a targets for reservoir optimization. On the other hand some of the adaptation methods do not use the discussed measures as their target.

6.1 "Goodness" Measures of the Reservoir Activations

A much-desired measure to minimize is the eigenvalue spread (max. eigenvalue/min. eigenvalue) of the cross-correlation matrix of the activations $x(n)$ (EVSCCA), as small EVSCCA is necessary for an online training of the ESN output by stochastic gradient descent [Jae03] [JH04]. In classical ESNs EVSCCA sometimes reaches 10^{12} or even higher [Jae05], which makes such training unfeasible.

Other desired features of the reservoir could be small pairwise correlation of reservoir activations $x(n)$, or entropy of $x(n)$ distribution (e.g. [Jae05]). The latter is quite a popular measure, as seen later in this review. A measure for short term capability of reservoirs was introduced in [Jae02a]. A criterion for maximizing information transmission of each individual neuron was introduced (and pursued, see Section 6.2) in [Tri05].

Methods for estimating the computational power and generalization capability of neural reservoirs where presented in [MLB05]. The proposed measure for computational power or "kernel quality" is performed in a following way. Take k different segments of input(s) $u^i(n)$, $i = 1, \cdots, k$, and collect the resulting reservoir activations for each of the input sequence $x^{u^i}(n_0)$ into a matrix $M \in \mathbb{R}^{k \times N_x}$, where n_0 is some fixed time after the appearance of $u^i(n)$ in the input. The rank r of the matrix M is the measure. If $r = k$, this means that all the presented inputs can be separated by a linear readout from the reservoir, and thus the reservoir is said to have a *linear separation property*. For estimating the generalization capability of the reservoir the same procedure can be performed with s ($s \gg k$) inputs $u^j(n)$, $j = 1, \cdots, s$ that represent the set of all possible inputs. If the resultant r, is substantially smaller than the size of the training set, then this means that the reservoir generalizes well. These two measures are more targeted to tasks of time

series classification, but can also be beneficial in estimating the power of regression [LM07a].

The so called *Edge of Chaos* is a region of parameters of a dynamical system at which it operates at the boundary between the chaotic and non-chaotic behavior. It is observed that at the edge of chaos many types of dynamical systems, including dynamic reservoirs, possess high computational power [BN04] [LM07b].[10] It is intuitively clear that the edge of chaos in reservoirs can only arise when the effect of inputs on the reservoir state does not die out quickly, thus such reservoirs can potentially have high memory capacity, which is also demonstrated in [LM07b]. However this does not universally imply that such reservoirs are optimal [LM07a]. The edge of chaos can be empirically detected (even for biological networks) by measuring Lyapunov exponents [LM07b], even though such measurements are not trivial (and often involve a degree of subjectivity) for high-dimensional noisy systems. There is also an empirical observation, that while changing different parameter settings of a reservoir, the best performance with a given task correlates with a Lyapunov exponent specific to that task [VSDS07]. The optimal exponent is related to the amount of memory needed for the task as discussed in Section 5.1. It was observed in ESNs with no input, that when $\rho(W)$ is slightly greater than 1, the internally generated signals are periodic oscillations whereas for larger values of $\rho(W)$, the signals are more irregular and even chaotic [OP05]. Even though such reservoirs can still be used, no real benefit of such regimes was found in the latter contribution.

A tendency that higher ranks of the connectivity matrix W_{mask} (where $w_{\text{mask}i,j} = 1$ if $w_{i,j} \neq 0$, and $= 0$ otherwise, for $i, j = 1, \cdots, N_x$) correlate with lower output errors was observed in [BT05].

6.2 Unsupervised Local Methods

Local learning methods such as Hebbian or Anti-Hebbian learning did not prove useful in improving the reservoirs so far [Jae05].

Recently a significant interest is being shown to reservoir adaptation inspired by various kinds of plasticity of biological neurons. A local learning method inspired by intrinsic plasticity of the real-world neurons was recently shown to improve the performance of BPDC [Ste07]. This approach can also be formally derived by maximizing information transmission of each individual neuron [Tri05]. Even though the latter result, as most of the plasticity models, applies to Fermi (i.e. having a positive sigmoid activation function) or spiking neurons, the method is shown to have positive effect on ESN reservoirs, but can also cause stability problems [Ste07]. An adaptation of this result to tanh type of neurons, that results in Gaussian (instead of the original exponential) distribution of outputs is presented in [VSVC07].

[10] The significance and universality of the idea that dynamical systems are most powerful at the edge of chaos is disputed, e.g. [MHC93].

There are also investigations done on how several types of plasticity interact in a network. E.g. the effects of simultaneous intrinsic and synaptic plasticity on an individual Fermi neuron are explored in [Tri04]. A recent contribution [LPT07] investigates how the intrinsic and spike timing dependent interact in a reservoir of spiking neurons.

6.3 Unsupervised Global Methods

A biologically inspired unsupervised approach with a reservoir trying to predict itself is proposed in [MB04]. An additional output $z(n) \in \mathbb{R}^{N_x}$, $z(n) = W_z x(n)$ from the reservoir is trained on the target $z_{\text{target}}(n) = x'(n+1)$, where $x'(n)$ are the activations of the reservoir before applying the neuron transfer function $\tanh(\cdot)$, i.e. $x(n) = \tanh(x'(n))$. Then, in the second run the original activations $x'(n)$, that result from $u(n)$, W_{in}, and W are mixed with the predicted ones $z(n-1)$, that result from W_z with a certain ratio $(1-\alpha) : \alpha$. The coefficient α determines how much the reservoir is relying on the external input $u(n)$ and how much on the internal self prediction $z(n)$. With $\alpha = 0$ we have the classical ESN and with $\alpha = 1$ we have an "autistic" reservoir that does not react to the input. Intermediate values of α close to 1 where shown to enable reservoirs to generate slow changing highly nonlinear signals that are hard to get otherwise.

An algebraic unsupervised way of designing ESN reservoirs was proposed in [OXP07]. The idea is to linearize the ESN update equation (5) locally, around its current state $x(n)$ at every time step n to get a linear approximation of (5) as $x(n+1) = Ax(n) + Bu(n)$, where A and B are time (n) dependent matrices. The approach aims at making the complex eigenvalues of A be distributed uniformly within the unit circle on the $\mathbb{Z}$ plane. The reservoir matrix W is obtained analytically from the set of the predefined eigenvalues and a given input $u(n)$. The motivation for this is, that if the target $y_{\text{target}}(n)$ is unknown, it is best to have something like an orthogonal basis in $x(n)$, from which any $y_{\text{target}}(n)$ could be constructed well. The spectral radius of the reservoir is suggested to be set by hand (according to the correlation time of $u(n)$, which is an indication of a memory span needed for the task), or by adapting the bias value of the reservoir units to minimize the output error.[11] Reservoirs generated this way are shown to yield higher average entropy of $x(n)$ distribution, short term capacity (both measures mentioned in Section 6.1), and smaller output error with a couple of synthetic problems. Unfortunately, reservoirs of only very small size ($N_x = 20, 30$) have been considered, and the most popular version of classical ESNs with a sparse random (both sign and size of the weights) reservoir is not among the ones compared.

[11] The latter actually makes this a *supervised* method, but since it is initially motivated by having $y_{\text{target}}(n)$ unknown, we list it among the unsupervised ones.

7 Supervised Reservoir Pre-training

In this section we will discuss methods for training reservoirs to perform a specific given task, i.e. not only the concrete input $u(n)$, but also the desired output $y_{\text{target}}(n)$ is taken into account. Since a linear readout from a reservoir is easy to train, the suitability of a reservoir for a particular task (e.g. in terms of NRMSE (1) is also relatively inexpensive to check.

7.1 Optimization of Reservoir Global Parameters

In Section 5.1 we discussed guidelines for the manual choice of global parameters for reservoirs of ESNs. This approach requires some experience, intuition and luck in selecting good parameters, which makes using ESNs in practical applications a bit more difficult. A systematic gradient descent method of optimizing the global parameters of LIESNs (a generalization of ESNs, having more parameters, discussed in Section 5.5) to fit them to a given task is presented in [JLPS07]. The investigation shows that the error surfaces in the combined global parameter and W_{out} spaces are often nontrivial (i.e. have high curvature and multiple local minima), thus gradient descent methods are not always practical.

7.2 Genetically Modified Reservoirs

As one can see form the previous sections of this review, pretraining of the reservoirs is generally not an easy, and largely yet unsolved, problem. On the other hand training an output and checking the performance of a resulting ESN is relatively inexpensive. This brings in evolutionary methods for the reservoir pretraining as a natural candidate.

Recall that the classical method generates a reservoir randomly, thus the performance of the resulting ESN slightly (and in some cases not so slightly) varies from one instance to another. Then indeed, an "evolutionary" method as naive as "generate k reservoirs, pick the best" will outperform the classical method ("generate a reservoir") with probability $(k-1)/k$, even though the improvement might be not significant.

Several evolutionary approaches on optimizing reservoirs of ESNs where presented in [IvdZBP04]. In the first attempt the authors did an evolutionary search on only the three main global parameters used for generating W: the reservoir size N_x, the spectral radius $\rho(W)$, and the density of W. This search showed that for the task at hand (which was modeling the motion of an underwater robot) tiny reservoirs of only 5 units did best, which enabled the authors to do a more involving genetic search on the resulting weight matrices directly. An evolutionary algorithm [Hol92] with a truncation selection, 1% mutation, 50 individuals, and one-point crossover was used on the individuals consisting of all the weight matrices $(W_{\text{in}}, W, W_{\text{ofb}})$. In addition to this direct search, a variant with a reduced search space was also tried, where the weights, but not the topology of W was

explored, i.e. elements of W that where 0 initially, allways stayed 0, and only non-zero values where changed. In all the cases W was normalized back to $\rho(W) = 1$ whenever $\rho(W) > 1$ occured during the search, to maintain the echo state property. The results showed, that the methods used outperfomed other state-of-art methods, and ESNs with more paremeters adapted genetically performed better, than the ones with less, but the performance difference between the direct and topology-restricted search was small.

Another approach of genetically optimizing the reservoir W is presented in [BT05]. To reduce the search space, W was (similarly to the previously described approach) decomposed into the *topology* W_{mask} and the sizes of the *weights* W_{rand}. W is obtained by $W = W_{\text{mask}} \cdot W_{\text{rand}}$ (element-wise multiplication), and rescaled such that $\rho(W) = 0.9$. W_{rand} was a dense random matrix with all the elements uniformly sampled from $[-1, 1]$ and was fixed throughout all optimization. A local genetic search (contrary to the previously described approach) was only performed on the topology matrix W_{mask}, each element of which is either 0 or 1. The search starts by randomly generating the initial topology with a predefined sparsity. At each iteration N_x (out of ${N_x}^2$ possible) "mutations" of the original topology are generated, by flipping a single randomly chosen element. The mutation yielding the lowest output error is retained for the next iteration. If no mutation gives an improvement the search is terminated. This approach was demonstrated to yield on average 50% smaller (and much more stable) error in predicting the behaviour of a mass-spring-damper system, than the classical one. The experiments were done using only small ($N_x = 20$) reservoirs.

Yet another approach uses a multilayer FFNN as a readout from the reservoir and applies genetic search to find optimal weights of the FFNN [XLP05]. Such an ESN is applied for a hard task as direct adaptive control replacement of a classical indirect controller.

Evolino [SWGG07], introduced in Section 3.3, is another example of adapting a reservoir (in this case a LSTM network) using genetic search.

7.3 Combining with Backpropagation

It has been shown that the performance of a trained ESN can further be improved by further training it using gradient descent methods [Erh04].

The BPDC reservoir method introduced in Section 3.4 can also be viewed as lying in between the reservoir and error backpropagation methods.

7.4 Trained Auxiliary Feedbacks

A recent development in the theory of dynamic systems has shown, that trained feedbacks can endow fixed neural circuits with universal computational capabilities [MJS06]. This theory has direct implications for reservoir methods since reservoirs are fixed neural circuits. Different ideas on how the power of ESNs

could be improved along the lines of this theory are explored in [Luk07]. It is done by defining intermediate targets, and on these targets training additional outputs of ESNs, that are fed-back to the reservoir. The intermediate targets are constructed from $y_{\text{target}}(n)$ and/or $u(n)$. The intuition is that the feedbacks could shift the internal dynamics in the directions that would make them better linearly combinable into $y_{\text{target}}(n)$. The investigation showed that for some types of tasks there are natural candidates for such intermediate targets, that indeed improve the performance significantly. On the other hand there are no universally good methods for producing intermediate targets devised, such that the targets would be both easy to learn and improve accuracy of the final output $y(n)$. In addition, training multiple outputs with feedback connections W_{ofb} makes the whole procedure more complicated, as cyclical dependencies between the trained outputs as well as stability issues arise.

8 Closing Remarks

We have surveyed the current methods of RNN training based on the paradigm of separation between the RNN (the reservoir) and the readout which is a hot and lively research topic. This overview is not covering the different methods of readouts from the reservoirs, that are quite a few. Also some important reservoir recipes might have been missed since the field is very young and dynamic, the author apologies for that in advance. On the other hand we were not trying to put *every* contribution relating to reservoirs on this review, but only the ones highlighting the main research directions.

Following the analogy between the ESNs and non-temporal kernel methods, ESNs would be *shallow architectures of type-1* according to the classification recently proposed in [BC07]. The reviewed reservoir adaptation techniques would make ESNs *shallow architectures of type-3*, that are more expressive. However, authors of [BC07] argue that any type of *shallow* (i.e. non-hierarchical) architectures are incapable of learning complex intelligent tasks.[12] This suggests that for really complex tasks adaptation of a single reservoir might not be enough and a hierarchical architecture of ESNs [Jae07] might be needed. In any case, adaptation/generation of the reservoirs is still relevant in the hierarchical framework. On the other hand if we take adaptation of a single, perhaps structured, reservoir seriously enough, it just might result in containing dynamics of different levels of abstractions/timescales.

Acknowledgments

The author would like to thank his supervisor Prof. Herbert Jaeger for the patient discussions and sharing his expertise in the field.

[12] This also relates to the limitations of ESNs discussed in Section 5.4.

References

[AP00] Amir F. Atiya and Alexander G. Parlos. New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks*, 11(3):697–709, May 2000.

[BA99] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286:509, 1999.

[BC07] Yoshua Bengio and Yann Le Cun. Scaling learning algorithms towards ai. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large-Scale Kernel Machines*. MIT Press, Cambridge, MA, 2007. To appear.

[BN04] Nils Bertschinger and Thomas Natschläger. Real-time computation at the edge of chaos in recurrent neural networks. *Neural Computation*, 16(7):1413–1436, 2004.

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, March 1994.

[BT05] Keith Bush and Batsukh Tsendjav. Improving the richness of echo state features using next ascent local search. In *Proceedings of the Artificial Neural Networks In Engineering Conference*, St. Louis, MO, 2005.

[BY06] Michael Buehner and Peter Young. A tighter bound for the echo state property. *IEEE Transactions on Neural Networks*, 17(3):820–824, 2006.

[Erh04] Dumitru Erhan. Exploration of combining ESN learning with gradient-descent RNN learning techniques. Bachelor's thesis, International University Bremen, 2004. `http://www.eecs.jacobs-university.de/archive/bsc-2004/erhan.pdf`.

[FE05] Georg Fette and Julian Eggert. Short term memory and pattern matching with simple echo state networks. In *Artificial Neural Networks: Biological Inspirations ICANN 2005*, volume 3696/2005, pages 13–18. Springer Berlin / Heidelberg, 2005.

[Hol92] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[IvdZBP04] Kazuo Ishii, Tijn van der Zant, Vlatko Bečanović, and Paul Plöger. Identification of motion with echo state network. In *OCEANS '04 MTS/IEEE - TECHNO-OCEAN '04*, volume 3, pages 1205–1210, 2004.

[Jae01] 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.

[Jae02a] Herbert Jaeger. Short term memory in echo state networks. Technical Report GMD Report 152, German National Research Center for Information Technology, 2002.

[Jae02b] Herbert Jaeger. Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the "echo state network" approach. Technical Report GMD Report 159, German National Research Center for Information Technology, 2002.

[Jae03] Herbert Jaeger. Adaptive nonlinear system identification with echo state networks. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 593–600. MIT Press, Cambridge, MA, 2003.

[Jae05] Herbert Jaeger. Reservoir riddles: suggestions for echo state network research. In *Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on*, volume 3, pages 1460–1462, 2005.

[Jae07] Herbert Jaeger. Discovering multiscale dynamical features with hierarchical echo state networks. Technical Report No. 9, Jacobs University Bremen, 2007. to appear.

[JH04] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, pages 78–80, April 2004.

[JLPS07] Herbert Jaeger, Mantas Lukoševičius, Dan Popovici, and Udo Siewert. Echo state networks with leaky integrator neurons, and optimization of their global control parameters. *Neural Networks 20 (4)*, 2007.

[KÖ6] Ali U. Küçükemre. Echo state networks for adaptive filtering. Master's thesis, University of Applied Sciences Bohn-Rhein-Sieg, Germany, April 2006.

[KH04] Marcus Kaiser and Claus C. Hilgetag. Spatial growth of real-world networks. *Physical Review E*, 69:036103, 2004.

[Lie04] Benjamin Liebald. Exploration of effects of different network topologies on the ESN signal crosscorrelation matrix spectrum. Bachelor's thesis, International University Bremen, 2004. `http://www.eecs.jacobs-university.de/archive/bsc-2004/liebald.pdf`.

[LM07a] Robert Legenstein and Wolfgang Maass. Edge of chaos and prediction of computational power for neural microcircuit models. *Neural Networks*, 2007. In press.

[LM07b] Robert Legenstein and Wolfgang Maass. What makes a dynamical system computationally powerful? In S. Haykin, J. C. Principe, T. Sejnowski, and J. McWhirter, editors, *New Directions in Statistical Signal Processing: From Systems to Brain*, pages 127–154. MIT Press, 2007.

[LPJS06] Mantas Lukoševičius, Dan Popovici, Herbert Jaeger, and Udo Siewert. Time warping invariant echo state networks. Technical Report No. 2, International University Bremen, 2006.

[LPT07] Andreea Lazar, Gordon Pipa, and Jochen Triesch. Time series prediction, fading memory and error correction in recurrent networks shaped by plasticity. *Neural Networks 20 (4)*, 2007.

[Luk07] Mantas Lukoševičius. Echo state networks with trained feedbacks. Technical Report No. 4, International University Bremen, 2007.

[MB04] Norbert M. Mayer and Matthew Browne. Echo state networks and self-prediction. In *Biologically Inspired Approaches to Advanced Information Technology, BioADIT 2004. Revised Selected Papers*, pages 40–48, 2004.

[MHC93] M. Mitchell, P. T. Hraber, and J. P. Crutchfield. Dynamic computation, and the "edge of chaos": A re-examination. In G. Cowan, D. Pines, and D. Melzner, editors, *Integrative Themes*, Reading, MA, 1993. Addison–Wesley.

[MJS06] Wolfgang Maass, Prashant Joshi, and Eduardo D. Sontag. Principles of real-time computing with feedback applied to cortical microcircuit models. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 18*, Cambridge, MA, 2006. MIT Press.

[MLB05] W. Maass, R. A. Legenstein, and N. Bertschinger. Methods for estimating the computational power and generalization capability of

neural microcircuits. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 17, pages 865–872, Cambridge, MA, 2005. MIT Press.

[MNM02] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Comput.*, 14(11):2531–2560, 2002.

[OLS05] Mohamed Oubbati, Paul Levi, and Michael Schanz. Meta-learning for adaptive identification of non-linear dynamical systems. In *Proceedings of the Joint 20th IEEE International Symposium on Intelligent Control*, pages 473–478, June 2005.

[OP05] Mustafa C. Ozturk and José C. Príncipe. Computing with transiently stable states. In *Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on*, volume 3, pages 1467–1472, 2005.

[OXP07] Mustafa C. Ozturk, Dongming Xu, and José C. Príncipe. Analysis and design of echo state networks. *Neural Computation*, 19(1):111–138, 2007.

[PFT02] Danil V. Prokhorov, Lee A. Feldkamp, and Ivan Yu. Tyukin. Adaptive behavior with fixed weights in RNN: an overview. In *Proceedings of International Joint Conference on Neural Networks (IJCNN2002)*, pages 2018–2023, 2002.

[SGWG05] Jürgen Schmidhuber, Matteo Gagliolo, Daan Wierstra, and Faustino Gomez. Evolino for recurrent support vector machines. Technical Report 19-05 version 2.0, IDSIA, December 2005. Published in ESANN '06 — 14 th European Symposium on Artificial Neural Networks, M. Verleysen ed.

[SH07] Zhinwei Shi and Min Han. Support vector echo-state machine for chaotic time-series prediction. *IEEE Transactions on Neural Networks*, 18(2):359–72, 2007.

[SS05] Ulf D. Schiller and Jochen J. Steil. Analyzing the weight dynamics of recurrent learning algorithms. *Neurocomputing*, 63C:5–23, 2005.

[Ste04] Jochen J. Steil. Backpropagation-decorrelation: Recurrent learning with O(N) complexity. In *Proc. IJCNNNeural Networks, 2004. IJCNN '04. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 843–848, July 2004.

[Ste07] Jochen J. Steil. Online reservoir adaptation by intrinsic plasticity for backpropagation-decorrelation and echo state learning. *Neural Networks 20 (4)*, 2007.

[SVVC07] Benjamin Schrauwen, David Verstraeten, and Jan Van Campenhout. An overview of reservoir computing: theory, applications and implementations. In *Proceedings of the 15th European Symposium on Artificial Neural Networks*, pages 471–482, 4 2007.

[SWGG07] Jürgen Schmidhuber, Daan Wierstra, Matteo Gagliolo, and Faustino J. Gomez. Training recurrent networks by evolino. *Neural Computation*, 19(3):757–779, 2007.

[Tri04] Jochen Triesch. Synergies between intrinsic and synaptic plasticity in individual model neurons. In *Advances in Neural Information Processing Systems 17*, 2004.

[Tri05] Jochen Triesch. A gradient rule for the plasticity of a neuron's intrinsic excitability. In *Proceedings of the 15th European Symposium on Artificial Neural Networks*, pages 65–70, 2005.

[vM05] Michal Čerňanský and Matej Makula. Feed-forward echo state networks. In *Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on*, volume 3, pages 1479–1482, 2005.

[VSDS07] David Verstraeten, Benjamin Schrauwen, Michiel D'Haene, and Dirk Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks 20 (4)*, 2007.

[VSS05] David Verstraeten, Benjamin Schrauwen, and Dirk Stroobandt. Reservoir computing with stochastic bitstream neurons. In *Proceedings of the 16th Annual ProRISC Workshop*, pages 454–459, Veldhoven, The Netherlands, 11 2005.

[VSVC07] David Verstraeten, Benjamin Schrauwen, and Jan Van Campenhout. Adapting reservoirs to get gaussian distributions. In *Proceedings of the 15th European Symposium on Artificial Neural Networks*, pages 495–500, 4 2007.

[Wol01] David H. Wolpert. The supervised learning no-free-lunch theorems. In *Proc. 6th Online World Conf. on Soft Computing in Industrial Applications*, 2001.

[WS98] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, April 1998.

[WYW06] Se Wang, Xiao-Jian Yang, and Cheng-Jian Wei. Harnessing non-linearity by sigmoid-wavelet hybrid echo state networks (swhesn). *Intelligent Control and Automation, 2006. WCICA 2006. The 6th World Congress on*, 1:3014– 3018, June 2006.

[XLP05] Dongming Xu, Jing Lan, and Jose C. Principe. Direct adaptive control: An echo state network and genetic algorithm approach. In *Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on*, volume 3, pages 1483–1486, 2005.

[XYH07] Yanbo Xue, Le Yang, and Simon Haykin. Decoupled echo state networks with lateral inhibition. *Neural Networks 20 (4)*, 2007.