Echo State Queueing Networks: a combination of Reservoir Computing and Random Neural Networks

Gerardo Rubino, Sebastián Basterrech

To cite this version:

Gerardo Rubino, Sebastián Basterrech. Echo State Queueing Networks: a combination of Reservoir Computing and Random Neural Networks. Probability in the Engineering and Informational Sciences, Cambridge University Press (CUP), 2017, 31 (4), pp.457-476. 10.1017/S0269964817000110 . hal-01663499

HAL Id: hal-01663499
https://hal.inria.fr/hal-01663499

Submitted on 14 Dec 2017

Echo State Queueing Networks: a combination of Reservoir Computing and Random Neural Networks

Sebastián Basterrech[*] and Gerardo Rubino[†]

Short title: Echo State Queueing Networks

Affiliations:

- S. Basterrech
  Department of Computer Science,
  Faculty of Electrical Engineering
  Czech Technical University
  Karlovo náměstí 13, 121 35 Prague 2
  Praha, Czech Republic
  `sebastian.basterrech@agents.fel.cvut.cz`

- G. Rubino
  Inria Rennes – Bretagne Atlantique
  Campus de Beaulieu
  35042 Rennes Cedex
  France
  `rubino@inria.fr`

[*] Corresponding author: S. Basterrech, Department of Computer Science, FEE, CTU, Prague, Czech Republic. Email: `sebastian.basterrech@agents.fel.cvut.cz`

[†] Inria, Rennes, France. Email: `rubino@inria.fr`

Abstract

This paper deals with two ideas appeared during the last developing phase in Artificial Intelligence: Reservoir Computing and Random Neural Networks. Both have been very successful in many applications. We propose a new model belonging to the first class, taking the structure of the second for its dynamics. The new model is called Echo State Queuing Network. The paper positions the model in the global Machine Learning area, and provides examples of its use and performances. We show on largely used benchmarks that it is a very accurate tool, and we illustrate how it compares with standard Reservoir Computing models.

1 Introduction

Artificial Neural Networks (ANNs) are a class of computational models which have been proven to be very powerful as statistical learning tools to solve different complicated engineering tasks. Several types of ANNs have been designed, some of them originating in the field of Machine Learning while others coming from biophysics and neurosciences. The Random Neural Network (RNN) proposed by E. Gelenbe in 1989 [10], is a mathematical model inspired by biological neuronal behavior which merges features of Spiking Neural Networks and Queueing Systems. It can be seen (and it is) as a new type of queuing system, with its own applications, for instance in performance evaluation. The network is a connectionist model where spikes circulate among the interconnected neurons. Each node has a state in $\mathbb{N}$, the number of customers if we are working in queuing problems, the neuron's potential if we see the model as a neural system. In the latter case, the neuron is said to be active if the potential is strictly positive. The firing times of the spikes are Poisson processes (conditioning with respect to the event "the queue is not empty"or, at the connectionist side, "the neuron is active"). The potential of each neuron increases when a spike arrives or decreases after the neuron fires. In order to use RNNs in supervised learning problems, a gradient descent algorithm has been described in [17], and Quasi-Newton methods have been proposed in [3, 28]. Additionally, the function approximation properties of the model were studied in [13, 14]. The structure of the model leads to efficient numerical evaluation procedures in many cases of interest, to good performance in learning tasks and to easy hardware implementations. Consequently, since its introduction the model has been applied in a variety of scientific fields. In particular, one of the authors has developed a technology for automatically measuring the perceptual quality of video or voice content received through the Internet, based on RNNs (see [34, 41]). See also [40] for a queueing-based performance evaluation work applying this approach (which in turn is based on G-networks, closing a funny loop).

Concerning connectionist models with recurrences (circuits) in their topologies, they are recognized as powerful tools for a number of tasks in Machine Learning (both in classic ANNs and in RNNs). However, they have a main limitation which comes from the difficulty, in the general case, in implementing efficient training algorithms: convergence is not always guaranteed, many parameters are involved, sometimes long training times are required [7, 30]. For these reasons, learning using recurrent neural networks is principally feasible for relatively small networks.

Recently, a new paradigm called Reservoir Computing (RC) has been developed which overcomes the main drawbacks of learning algorithms applied to networks with cyclic topologies. The two most important RC models are Echo State Networks (ESNs) [26] and Liquid State Machines (LSMs) [32]. Both exploit the idea of using recurrent neural networks *without* adapting the weight connections involved in recurrences. The network outputs are generated using very simple learning methods such as classification or regression tools. The RC approach has been successfully applied to many problems achieving goods results, specially in temporal learning tasks [30, 32, 48].

As announced before, in this paper we introduce a new type of RC method whose non-learning part has a dynamics based on RNNs. This article is an extended version of our previous work in [4]. Observe that in the Machine Learning literature, the acronym RNN is often used for Recurrent Neural Network, that is, a neural network with circuits. In this work we use it for Random Neural Network, since it is a central object in the paper.

The paper is organized as follows: we begin by describing in Section 2 the world of G-networks and their main characteristics, from the point of view of the objectives of this paper. The different subsections review G-queues and G-networks, then RNNs and finally, the problem of temporal supervised learning. Section 3 presents the RC paradigm and discusses the main properties of these models. Section 4 describes the main contribution of this article, a new RC model based on G-nets, and it explores the main model's parameters. Finally, we present some experimental results illustrating the interest of the proposal, and we end with some conclusions as well as a discussion regarding future lines of research.

2 G-nets

Taking inspiration from the behavior of neural systems, and the mathematical models representing them, Erol Gelenbe proposed, in the late 70s and early 80s, a new queueing paradigm where customers belong to two types, called *positive* and *negative* [11]. Positive customers behave as ordinary customers in classic queueing systems: they arrive, wait, are served, and move instantaneously to another queue or to the system's "outside". Negative customers are ephemeral objects: they disappear at the same time they arrive at a queue, but if the queue is not empty, they also remove a customer (necessarily a positive one) from it (which one depends on some rule specific to the considered system). When this happens, we also say that the positive customer has been *killed*. This means that negative customers can't be observed, only their effects on positive ones can; at any point in time, only positive customers can be in the system. Let us briefly review, in this section, the main characteristics of these models when they are isolated and when they are interconnected in networks. Then, we resume how we can see such a network as a statistical learning tool, the topic of this paper. The last subsection is different: we very briefly describe another learning situation where the target is a time series, that is, where there is a temporal dimension in the problem.

2.1 G-queues

Consider a stable queue in equilibrium, fed by two independent flows of customers, one of positive ones and the other one of negative customers (they not need to be Poisson for the moment, let's say that they are, for instance, renewal processes). Denote the respective throughputs by T^+ and T^-. The mean service time at the queue is S. Denote by ϱ the system's load, that is, the probability that, in equilibrium, the queue (and thus, the server) is not empty. Then, the mean intensity of the flow of positive customers leaving the system after service is ϱ/S, and of the flow of positive customers removed by negative ones is ϱT^-. The mean conservation flow equation then writes $T^+ = \varrho/S + \varrho T^-$, leading to

$$\varrho = \frac{T^+}{\dfrac{1}{S} + T^-} = \frac{T^+ S}{1 + T^- S}. \tag{1}$$

Observe that this expression only needs ergodicity assumptions, no need for any Exponential distribution anywhere in the model. If the two arrival processes are Poisson with respective parameters λ^+ and λ^-, and if the server is an Exponential one having rate μ, then the load is

$$\varrho = \frac{\lambda^+}{\mu + \lambda^-}. \tag{2}$$

In this case, the number of customers in the system at time t is Markov, it is ergodic if and only if $\lambda^+ < \mu + \lambda^-$, and when this inequality holds, the stationary distribution of the process is geometric: for any integer $k \geq 0$, the probability that, in equilibrium, there are k customers in the queue is $(1 - \varrho)\varrho^k$.

2.2 G-networks

When these these types of queues are organized in networks with classic Bernoulli switching, positive customers can become negative when moving from a queue to another. For any two queues (or nodes) i and j, we denote by $p^+_{i,j}$ the probability that a (positive) customer finishing its service at i, goes to j as a positive one, and $p^-_{i,j}$ the corresponding probability that it goes to j as a negative customer. Any customer removed from a node by a negative one disappears from the system. Observe that the probability that a customer having being served at i leaves the network is $d_i = 1 - \sum_{j=1}^n \big(p^+_{i,j} + p^-_{i,j}\big)$, where n is the number of nodes in the network. Of course, switching probabilities must satisfy connectivity constraints (as for Jackson networks) to avoid pathological situations where a node fills up and never sends customers (positive customers) to another node or to outside, or where a node can never receive customers (positive ones). The third situation that it is also uninteresting is the one where the network is actually composed of disjoint parts.

Assume a basic Markovian situation: the external arrival flows are Poissonian, and the service times at the nodes are Exponentially distributed. Also assume the usual independence conditions (between arrival processes and services times). Node i receives ordinary customers from outside with rate λ^+_i and negative ones with rate λ^-_i, and its service rate is $\mu_i > 0$. Process $X = \{X(t),\ t \ge 0\}$ with values in $\mathbb{N}^n$, defined by $X(t) = \big(X_1(t), \dots, X_n(t)\big)$, where $X_i(t)$ is the number of customers at node i, is then Markov (homogeneous, irreducible). Assume stability and consider the equilibrium regime. Denote by $x \to \pi_x$ the stationary distribution of X, $x \in \mathbb{N}^n$. The load at i, denoted by ϱ_i, that is, the probability that, in equilibrium, node i is not empty, is given by $\varrho_i = \sum_{x:x_i>0} \pi_x$. Then, the mean throughputs of positive and negative customers *arriving* at node i, denoted respectively T^+_i and T^-_i, must satisfy

$$T^+_i = \lambda^+_i + \sum_{j=1}^n \varrho_j \mu_j p^+_{j,i} \qquad \text{and} \qquad T^-_i = \lambda^-_i + \sum_{j=1}^n \varrho_j \mu_j p^-_{j,i}. \tag{3}$$

Writing now the mean flow conservation equation at i, we obtain, as in previous subsection, that the load at i satisfies the equation

$$\varrho_i = \frac{T^+_i}{\mu_i + T^-_i}. \tag{4}$$

See that, *necessarily*, if X is stable, for all node i we must have $T^+_i < \mu_i + T^-_i$. Always in this equilibrium situation, if we now replace π_x in the Chapman-Kolmogorov equilibrium equations for X by the product form $\pi_x = \prod_{i=1}^{n}(1-\varrho_i)\varrho_i^{x_i}$, we can verify that this is the stationary distribution of the system. Reciprocally, if we consider the non-linear system $\{(3),(4)\}$ in the variables $(T_1^+,\dots,T_n^+,T_1^-,\dots,T_n^-)$, or equivalently, in the variables $(\varrho_1,\dots,\varrho_n)$, if this system has a solution where for all i, $\varrho_i < 1$, then, the function $x \to \prod_{i=1}^{n}(1-\varrho_i)\varrho_i^{x_i}$ is a distribution probability on $\mathbb{N}^n$, strictly positive, and satisfying the equilibrium equations for X. Resuming, X is stable if and only if system $\{(3),(4)\}$ has a solution where for all i, $\varrho_i < 1$, and in that case, it has the given product form.

The reader can wonder if something can be said about the nonlinear system $\{(3),(4)\}$ alone. Define the *surviving* probability s_k at node k as the probability, always in equilibrium, that a customer that arrived at k gets served (that is, is not killed before the end of its service). We have

$$s_k = \frac{\varrho_k \mu_k}{T_k^+} = \frac{\mu_k}{\mu_k + T_k^-}.$$

We can then write another nonlinear system equivalent to $\{(3),(4)\}$:

$$T_i^+ = \lambda_i^+ + \sum_{j=1}^{n} T_j^+ s_j p_{j,i}^+, \qquad T_i^- = \lambda_i^- + \sum_{j=1}^{n} T_j^+ s_j p_{j,i}^-, \qquad s_j = \frac{\mu_j}{\mu_j + T_j^-}. \tag{5}$$

Using this form, in [15] Brower's fixed point on an appropriate function allows proving that this system has a fixed point $(T_1^+,\dots,T_n^+,T_1^-,\dots,T_n^-)$, and that it belongs to a specific rectangle of $\mathbb{R}^n$.

2.3 Random Neural Networks [12]

Assume that we want to learn (in the sense of Machine Learning) a function $\vec{f}\colon [0,1]^I \to [0,1]^O$ (if the function was from $A \subset \mathbb{R}^I$ to $B \subset \mathbb{R}^O$, we can move to the first case by scaling variables appropriately), from a data set composed of K pairs $(\vec{a}(k),\vec{b}(k))$, $k = 1..K$, where $\vec{b}(k) = \vec{f}(\vec{a}(k))$. For the sake of clarity, we make explicit vectors and matrices here. We then consider a G-network having $n \ge \max\{I,O\}$ nodes, called in this *supervised learning* context *neurons*, where only I of them, called *input* neurons, receive positive customers from outside, and O of them, called *output* neurons, send customers to outside. It is an usual assumption in this context that no negative customer arrives from outside; negative customers can randomly appear inside the network, when (positive) customers end service at some node and move to another one. Denote by $\mathscr{I}$ the set of input neurons and by $\mathscr{O}$ the set of output ones.

For any two neurons i and j, denote by $w^+_{i,j}$ and $w^-_{i,j}$ the reals $w^+_{i,j} = \mu_i p^+_{i,j}$ and $w^-_{i,j} = \mu_i p^-_{i,j}$. These numbers are called *weights* in this context. Observe that for any neuron i, we have $\sum_{j=1}^{n}\big(w^+_{i,j} + w^-_{i,j}\big) = (1-d_i)\mu_i$.

This model can be used to learn $\vec{f}$ in the following way. Denote by $\vec{\nu}$ the function mapping the I rates $(\lambda^+_i,\ i \in \mathscr{I}) = \vec{\lambda}^+$ to the O loads $(\varrho_j,\ j \in \mathscr{O}) = \vec{\varrho}$; we write $\vec{\varrho} = \vec{\nu}(\vec{\lambda}^+)$. Now, look at the weights as parameters of $\vec{\nu}$, write them generically as w, and make them appear explicitly in $\vec{\nu}$ saying that $\vec{\varrho} = \vec{\nu}(w;\vec{\lambda}^+)$. Then, we can define the quadratic error function of w

$$E(w) = \frac{1}{2}\sum_{k=1}^{K}\sum_{i\in\mathscr{O}}\Big[\nu_i\big(w;\vec{a}(k)\big) - b_i(k)\Big]^2,$$

and look for weights that minimize E. If w^* is one such argmin, and if $E(w^*)$ is small enough, a subsequent *validation* phase where another set of data is used, must verify that $\vec{\nu}(w^*;\vec{c}(k))$ is close enough to $\vec{d}(k)$, for the elements of the second data set $(\vec{c}(k),\vec{d}(k))$, $k = 1..K'$, where $\vec{d}(k) = \vec{f}(\vec{c}(k))$. In that case, learning was successful. Somehow, w^* (together with the whole queueing network) captured the way $\vec{f}$ maps its input to its output. If not, something went wrong in the process (not enough data, bad data composition, some problem in the optimization procedure, etc., or perhaps very little content to learn, because $\vec{f}$ has a strong random component), and we must start again after some appropriate change in the procedure. As stated before, this process has proved to be very successful in many areas. In [22], the power of RNNs for approximating any continuous function has been analyzed. It was found that it is possible to define a multilayer RNN with a bounded number of layers such that it can approximate any continuous function in a compact space [20]. Many variations of the canonical

RNN have also been introduced during the last two decades. For example, an extension of the model that simultaneously process multiple data streams was presented as Multiple Signal Class Random Neural Networks (MCRNNs) [21]. In this work, the gradient-based learning algorithm has been extended to cover "multiple classes" that can represent several colors in an image, or multi-sensory perception. The case of soma-to-soma interactions between neurons (rather than just the usual synaptic excitatory-inhibitory interactions) has also been considered and treated with a specific learning algorithm [23]. In addition, the RNN is frequently used for real-time applications with Reinforcement Learning [18, 5], rather than just gradient descent type algorithms. The model also have been applied as a tool in the recently established paradigms Deep Learning and Big Data. For an example of the RNN's application to deep learning yielding very high recognition accuracy see [24]. An approach for using the model on Big Data contexts is presented in [5]. For more details about other applications of RNN, see [1, 47].

While the name G-networks is sometimes used for RNNs, the G-network model is significantly more general [16, 19] and covers state transitions which are more complex in terms of the jumps involved and that can include several nodes in one single transition.

2.4 Learning in a temporal context

Assume now a completely different context. Consider real-valued time series $\big(\sigma(t),\ t \ge 0\big)$, we can observe its first values $\sigma(0), \sigma(1), \ldots, \sigma(T)$ for some time $T \in \mathbb{N}$, and we want to predict next value $\sigma(T+1)$. For this purpose, assume we have a dynamical system $\mathscr{S}$ in discrete time, characterized by some parameters globally designed by w, mapping some input $x(t)$ at time t to an output value $y(t)$, and having a state $s(t)$. Its dynamics is given by the parametric function

$$s(t) = F\big(w; s(t-1), x(t)\big), \quad t \ge 1 \tag{6}$$

giving the new state at t as a function of the old one $s(t-1)$ and the new input $x(t)$. The output $y(t)$ at time t is given by a second parametric function

$$y(t) = G\big(w;\, x(t-1), s(t)\big), \quad t \geq 1. \tag{7}$$

So, input and state values start at time 0; outputs start at time $t = 1$.

The idea is to use this dynamical system to predict, from the piece of the target sequence $\big(\sigma(t),\ 0 \leq t \leq T\big)$, its future after T. To simplify, we consider simply the prediction of $\sigma(T+1)$. As in the standard supervised case previously described, we minimize the error function

$$E(w) = \frac{1}{2}\sum_{t=1}^{T-1}\Big[G\big(w; \sigma(t-1), s(t)\big) - \sigma(t+1)\Big]^2,$$

where for each $t \in [1..T-1]$, $s(t) = F\big(w;\, s(t-1), \sigma(t)\big)$. Let w^* be an argmin of E in the space where w lives. Then, the prediction of $\sigma(T+1)$ to use will be $y(T) = G\big(w^*; \sigma(T-1), s(T)\big)$, where $s(t) = F\big(w^*;\, s(t-1), \sigma(t)\big)$, $1 \leq t \leq T$.

The description above was made for an uni-dimensional time series. Of course, this approach can be used in more general contexts, where, for instance, the series is multi-dimensional (that is, for some value $I > 1$ the series has values in $\mathbb{R}^I$). Another variation of the problem is such that the goal is to predict future values of a sequence from past ones. Instead of predicting one time step ahead, we use a free-run scheme for predicting values in an arbitrary interval of time. Typically, we learn the sequence until time T, and then, we predict values in a time window $T+1, \dots, T+W$. The predicted value at time $T+k$ is computed using the predicted values at time $T+k-1, T+k-2, \dots, T+1$, for any $k \in [1..W]$. The move from the previously discussed case to these situations is straightforward. Another case where the situation is more general is when there are other external variables that participate in the model prediction. The dynamical system has two types of inputs: explanatory variables (the variables to be predicted by the model, some authors call them endogenous variables), and external variables (also referred to as exogenous variables) [38].

The reason of this short description of temporal learning problems, somehow disconnected from the previous subsections, is because our paper proposes this type of dynamical system, for instance for the same temporal task. For that purpose, the paper merges two ideas, one of them being the Random Neural Network one.

3 Reservoir Computing

As recalled before, a Recurrent Neural Network is a Neural Network where there are circuits in the connections between neurons (we also say "recurrences"). The model is very useful to represent non-linear relationships between input sequences and output sequences. In particular, it is a quite powerful tool for modeling patterns that evolve with time, such as time series or sequential data. Besides, it has good theoretical properties. For instance, it has been shown that there exists a finite recurrent network composed by sigmoid neurons that is able to simulate an universal Turing machine [45]. However, in spite of their computational power, recurrent networks are much harder to use than structures with no circuits, basically because of problems arising in the learning phase (slow convergence, and even convergence difficulties, for instance). There are several optimization algorithms that work very well for training neural networks without recurrences that can fail in the case of networks with circuits. A typical example is the famous back propagation algorithm [42]. This technique and its variations work well for some feedforward networks, but they have problems for training networks with recurrences [35].

During the 90s, important effotts have been made in the community for developing learning algorithms able of properly training neural networks with recurrences. The most influential proposals include the Jordan network [27], Elman network [8], and the Long-Short Term Memory (LSTM) network [25]. At the beginning of the 2000s, a new approach for designing and training Recurrent Neural Networks was developed. The idea was presented in the following models: Liquid State Machine (LSM) [31, 33], Echo State Network (ESN) [26] and

Backpropagation Decorrelation (BPDC) [46]. All these models have been developed in parallel. They share the same basic principles, which have been aggregated under the global name of Reservoir Computing (RC) [48] that we now describe.

A RC model is composed of two different structures. One is a recurrent network whose parameters are non-adaptable, called *reservoir*: the parameters are randomly initialized and remain unchanged during the training and the utilization of the model. The other structure called *readout*, is a parametric model, in general a very simple supervised learning tool known to be fast and robust. It is the readout that is in charge of learning. Each structure then has a different role: the reservoir gives to the model the ability of memorizing the input data, and the readout provides the learning ability. This functional dissociation into two structures, of which only one learns, is the main well-distinguished characteristic of this family of methods.

3.1 Formalization of the model

Let us focus on the most used RC model, called Echo state Network (ESN). Consider a learning dataset $\mathscr{L} = \{(\vec{a}(t), \vec{b}(t)) : \vec{a}(t) \in \mathscr{A}, \vec{b}(t) \in \mathscr{B}, t = 1, \ldots, T\}$, where $\mathscr{A}$ and $\mathscr{B}$ are typically sets of (column) real vectors. A RC method is composed of two independent computational models. First, a Recurrent Neural Network $\psi_1(\cdot)$ that transforms a layout of points in the input space $\mathscr{A}$ into points of a (much) larger space $\mathscr{S}$. Once the projections from $\mathscr{A}$ to $\mathscr{S}$ are performed, a parametric function $\psi_2 : \mathscr{S} \to \mathscr{B}$ is computed. The results of $\psi_2(\cdot)$ are the predictions of the model. The reservoir is thus a hidden recurrent network $\psi_1(\cdot)$ composed of H neurons. The most popular RC models have a randomly initialized reservoir. So, we can see $\psi_1(\cdot)$ as a random projection from $\mathscr{A}$ to $\mathscr{S}$. A matrix $\mathbf{w}^{\mathrm{r}}$ of dimensions $H \times H$ collects the weight connections between the reservoir neurons. The model has an input layer of units that transforms the input patterns and feeds the reservoir. The forward weight connections between input neurons and reservoir neurons are collected in a matrix $\mathbf{w}^{\mathrm{in}}$ with dimensions $I \times H$, $I \ll H$. The discrete time dynamics of the reservoir at each time t is represented by a column vector $\vec{s}(t)$ of dimension H.

This vector represents the reservoir state at each unit of time, and is computed as follows:

$$\vec{s}(t) = \psi_1\big(\mathbf{w}^{\text{in}}\vec{a}(t) + \mathbf{w}^{\text{r}}\vec{s}(t-1)\big), \qquad t \geq 0, \tag{8}$$

where $\psi_1(\cdot)$ in the original ESN was the hyperbolic tangent function.

Most often, function $\psi_2(\cdot)$ is a linear model:

$$\vec{y}(t) = \psi_2(\vec{s}(t)) = \mathbf{w}^{\text{out}}\vec{s}(t), \tag{9}$$

where matrix $\mathbf{w}^{\text{out}}$ of dimensions $O \times H$ contains the forwards weights between the reservoir neurons and the output neurons. For the sake of notational simplicity, we avoid the bias terms in the linear regression.

The output weights $\mathbf{w}^{\text{out}}$ are the only adjustable parameters of both models (ESNs and ESQNs), which are trained using the learning dataset $\mathscr{L}$. A popular training algorithm used in the literature is the ridge regression offline training [26]. The offline version consists of running the reservoir $\psi_1(\cdot)$ on the whole input patterns. Then, we compute the parameters of $\psi_2(\cdot)$ such that a distance between target $\vec{b}$ and model's output $\vec{y}$ is minimized. Let $\mathbf{S}$ and $\mathbf{B}$ be two matrices of dimensions $H \times T$ and $O \times T$, respectively. Matrix $\mathbf{S}$ collects in their rows the reservoir states. This means that in row t we have the reservoir state $\vec{s}(t)$ that was computed when the input was $\vec{a}(t)$. The row t of matrix $\mathbf{B}$ has the target data $\vec{b}(t)$. The weight matrix $\mathbf{w}^{\text{out}}$ is then computed by

$$\mathbf{w}^{\text{out}} = \mathbf{B}\mathbf{S}^T(\mathbf{S}\mathbf{S}^T + \gamma^2\mathbf{Id})^{-1}, \tag{10}$$

where $\mathbf{Id}$ is the identity matrix of rank H and γ is a regularization parameter.

3.2 Stability conditions

A main characteristic of a RC model is that the recurrent trajectories created are generated by a randomly built matrix (the reservoir weights). Moreover, the initial state of the dynamics is arbitrary. Then, the reservoir matrix should satisfy some conditions, for the model to make accurate predictions. For instance, in the long term the output trajectories associated with a given input one, should become asymptotically independent of the initial conditions. In other words, the model needs to have some type of fading memory with respect to those initial conditions. In addition, two different sequences should be projected as two different sequences in the reservoir space, and similar reservoir states must been associated with similar input sequences. These characteristics are established in a property named Echo State Property (ESP) [26, 30], that was introduced for the ESN model. We avoid here technical details (see also [49, 50]), but this type of result is not valid for any type of RC model: there are conditions to be satisfied by the connections, by the set $\mathscr{S}$ of possible state values (it must be compact), etc. The idea is to show here that the properties known for the canonical ESN and some of its variations are rather limited and partial, in spite of the fact that the dynamics is significantly simpler than in the model we are going to describe in next section, whose theoretical analysis is open.

The common procedure for creating an ESN consists in random initializing the reservoir matrix $\mathbf{w}^{\mathrm{r}}_{\mathrm{initial}}$, then to scale the matrix:

$$\mathbf{w}^{\mathrm{r}} = \alpha \mathbf{w}^{\mathrm{r}}_{\mathrm{initial}}, \tag{11}$$

where α is a parameter named scaling factor. The value of α has an impact on the ESP. For instance, denoting msv the maximal singular value of a matrix, if $\alpha < 1/\mathrm{msv}(\mathbf{w}^{\mathrm{r}})$ (and under some conditions as previously mentioned), then we have the ESP [26]. It has been empirically observed that this sufficient condition is very conservative. On the other direction, to have ESP it is necessary that $\mathrm{sr}(\mathbf{w}^{\mathrm{r}}) \leq 1$, where $\mathrm{sr}(\cdot)$ denotes the spectral radius. The gap between these two conditions is unexplored [49]. Note that scaling the reservoir is a computational expensive procedure. According to [9], to rescale the reservoir matrix by the spectral radius has a $O(H^4)$ computational cost. Furthermore, it is known that different reservoirs with same spectral radius can have different prediction accuracy [43]. The scaling factor of the reservoir matrix is then an important parameter, but is not the only relevant one in the model.

4 Echo State Queueing Network

This section presents the main contribution of the paper, a new RC model named Echo State Queueing Network (ESQN) that can be seen as a new application of G-networks to predict the behavior of temporal systems, typically, time series. The approach consists in building a model that combines the central ideas of both RC and RNNs, with in particular the use by the latter of rational functions at all steps, allowing many straightforward mathematical work on the models.

4.1 Formalization of the model

Our model is called Echo State Queueing Network, precisely because it combines the Echo State Network approach with the richness of G-networks. An ESQN is a recurrent network with a G-network-like dynamics in the reservoir, inspired by the relationships occurring in a RNN in steady-state. As usual in the RC area, in an ESQN the readout outputs are computed using a linear regression.

The architecture of an ESQN consists of an input layer, a hidden reservoir and a readout layer. The input layer is composed of I input units. The input units are defined as in the Random Neural Network model. The input neurons process the input patterns and send spikes toward the reservoir. Let us index the input neurons from 1 to I, and the reservoir neurons from $I+1$ to $I+H$. Given an input pattern $\vec{x}$, we first set the rates of the external positive spikes with that input $\lambda_u^+ = x_u$, for each input neuron u. In a similar way than in the traditional Random Neural Network, we set the external negative spikes as $\lambda_u^- = 0$, for each input neuron u.

The state of the reservoir neurons is defined as in ESNs, adding a discrete clock t and making the state at t depend on what happened at $t-1$. Remember that in Random Neural Networks, the neuron's loads are computed using Expressions (3), (4). The state of the ESQN reservoir is given by the following dynamics:

$$s_u(t) = \frac{\displaystyle\sum_{v=1}^{I} \frac{x_v(t)}{\mu_v} w^+_{v,u} + \sum_{v=I+1}^{I+H} s_v(t-1)\, w^+_{v,u}}{\displaystyle\mu_u + \sum_{v=1}^{I} \frac{x_v(t)}{\mu_v} w^-_{v,u} + \sum_{v=I+1}^{I+H} s_v(t-1)\, w^-_{v,u}}, \qquad u \in [I+1, I+H],\ t \geq 1, \tag{12}$$

In the previous expression we are denoting the weights following the common usage in the community of G-nets, e.g. the weight $w_{i,j}$ is associated with the edge from i to j. Remember that, in the ESQN model the parameters $w^+_{v,u}$, $w^-_{v,u}$ and the service rate μ_u of neurons in the reservoir are fixed during the training. For this reason, we are not using a temporal reference in their notation. The state of an input neuron in the ESQN is simply computed as $s_u = x_u/\mu_u$.

In the setting of Subsection 2.4 and Relations (6) and (7), the dynamics of ESQNs can be written $\vec{s}(t) = F\big(w^r, \vec{s}(t-1), \vec{x}(t)\big)$, for $t \geq 1$, where w^r represents here all the input to reservoir and reservoir to reservoir weights. The output of the model is $\vec{y}(t) = G\big(w^O; \vec{s}(t)\big)$, $t \geq 1$, where w^O represents the reservoir to readout weights.

The adjusted parameters of the ESQN model using the training set are only the weights between the reservoir projections and the output layer. We generate the output of the model using a linear regression as in the ESN case: the readout part is computed by a linear function $\psi_2\big(\mathbf{w}^{\text{out}}, \vec{x}, \vec{s}\big)$, which receives the reservoir state $\vec{s}(t)$ at t and produces the network output $\vec{y}(t)$. Thus, the network output $\vec{y}(t)$ is computed performing Expression (9), which can be solved using ridge linear regression [37]. Note that it is very simple to generalize this implementation: instead of computing the outputs with a linear regression we can use any other type of parametric function $\psi_2\big(\mathbf{w}^{\text{out}}, \cdot\big)$, for instance a feed-forward RNN, due to the independence between reservoir dynamics and readout layer.

4.2 Discussion about the model parameters

In this part we discuss the most important global parameters of the ESQN model, and we compare their relevance according to the literature of RC methods.

- **Reservoir weights.** The reservoir is composed of two randomly created weight matrices. As we will see, in general uniform distributions are used, keeping some control on the sparsity of the matrix. Since in ESQNs we chose to stay close to the dynamics of a Random Neural Network, weights are positive reals, a difference with standard neural systems where weights can have any sign. Recall that G-nets can be seen a spiking networks, where the weights represent mean throughputs of spikes. The null value for a weight is interpreted as the fact that there is actually no such connection. In the experimental part of the paper, we use an Uniform distribution in $[0,1]$ for the reservoir weights.

- **Reservoir size.** The number of neurons in the reservoir impacts accuracy as the number of hidden neurons does in a classical ANN. Increasing it after having started by a small value, at the beginning accuracy increases. This has been already observed in many other RC models; for example, see [6, 48, 2, 29, 30]. However, there is a tradeoff between reservoir size and generalization ability of the network. If the reservoir has too many neurons, then the training error can become very low, but the *over-fitting* phenomenon can happen.

- **Spectral radius of the reservoir weight matrix.** As we have already seen above, the spectral radius is an important parameter of the reservoir. Its value has an impact on the stability of the model. The impact of the spectral radius of the reservoir matrix has been well studied in the case of ESNs. It has been noted that the spectral radius influences in the memory capacity of the model [6]. When the spectral radius $\mathrm{sr}(\mathbf{w}^{\mathrm{r}})$ is close to 1, then the model performs well when solving learning tasks that require long term memory (time series exhibiting some form of long correlations, for instance). On the other hand, when sr($\mathbf{w}^{\mathrm{r}}$) is small, then the model is adequate for tasks that require some form of short memory [26, 6]. However, there are still many open questions about the behavior of the spectral radius in the case of more complex reservoirs. For example, when the reservoir is composed by specific types of spiking neurons (LIF neurons, see [48, 36]), the way the spectral radius is related to accuracy is unclear. According to our empirical results, the spectral radius is also an important parameter for ESQN's accuracy, although the specific good values depend of the benchmark problem considered (see next section). We also noted that the spectral radius sr($\mathbf{w}^{\mathrm{r}+}$) has more impact on the accuracy than the spectral radius sr($\mathbf{w}^{\mathrm{r}-}$).

- **Sparsity of the reservoir weight matrix.** In the case of the ESN model, it is recommended to use a sparse matrix for the reservoir weights (around 15% to 20% of possible non-zero connections on the matrix [26]). The reason is more related to the computational speed than to the accuracy of the model (if the reservoir is spare, computing its state is faster). It is not clear how the density of the reservoir impacts the accuracy of the model. In this paper we didn't analyze the sparsity of the reservoir.

- **Topology and feedback connections.** In spite of numerous works exploring the topological structure of the reservoir, a useful topology for significantly improving the performance of the model is still unknown. The classic approach consists of generating random reservoirs. There are some models with feedback connections, that is, with arcs going from the outputs of the model back to the reservoir. This topology provokes new forms of impact on the stability of the model. To the best of our knowledge, there are no studies about the stability of these extensions in the RC literature.

5 Empirical evaluations

5.1 Benchmark problems

In this section we analyze the accuracy of the proposed approach on several experiments. The selected benchmark problems have been widely used in the area of dynamical systems, RC and temporal learning. In all examples, we first normalize the data in the range [0,1]. The list of considered benchmarks follows.

1. **Noisy Multiple Superimposed Oscillator (MSO) time-series [44].** The noisy MSO is a time-series dataset generated by

   $$a(t) = \sin(0.2t) + \sin(0.311t) + z, \qquad t = 1, 2, \ldots$$

   where z is a Gaussian random variable with distribution $\mathcal{N}(0, 0.01)$. We simulated 10000 samples for training the model. We present the performance of the trained model on 1000 unseen simulated samples.

2. **Lorenz attractor.** The series is based on the Lorenz differential equations:

   $$\frac{\partial x}{\partial t} = \sigma(y - x), \qquad \frac{\partial y}{\partial t} = rx - y - xz, \qquad \frac{\partial z}{\partial t} = xy - bz,$$

   we used the parameters $\sigma = 10$, $r = 28$, $b = 8/3$ and a step size of 0.01. The training set had 13107 samples and the testing set contained 3277 samples.

3. **Rossler attractor.** This is a classic time series with a sequence generated by following dynamics:

   $$\frac{\partial x}{\partial t} = -z - y, \qquad \frac{\partial y}{\partial t} = x + ry, \qquad \frac{\partial z}{\partial t} = b + z(x - c),$$

   where the parameters values are $r = 0.15$, $b = 0.20$, $c = 10.0$.

4. **Henon map.** The sequence is generated by

$$\frac{\partial x}{\partial t} = r + by - x^2, \qquad \frac{\partial y}{\partial t} = x,$$

where $r = 1.4$, $b = 0.3$ and the initial values are $x = y = 1$. This dynamical system has been modeled with ESNs in at least the following works: [2, 39].

5.2 Results

Table 1 presents the behavior of the ESQN and how it compares to the classical and widely used ESN, on the 4 benchmark datasets. The table shows the validation error[1] on the different considered problems, with two different number of neurons in the reservoir (40 and 80 neurons). In all cases, the weight matrices verify that $\mathrm{sr}(\mathbf{w}^{\mathrm{r}+}) = \mathrm{sr}(\mathbf{w}^{\mathrm{r}-}) = 0.5$. The regularization parameter of the linear regression in the readout of the ESQN was 10^{-4}. For the ESN, we used 10^{-3}, because the ESN obtained better results with a higher regularization parameter than for our proposal. In other words, we selected good values of this parameter for each model. We can see that in almost all the cases the ESQN performs better than the ESN. However, note that the ESN model has less free-parameters. In the case of ESQN, we need two weight matrices for representing the reservoir. This means that the number of parameters in both models is different. Anyway, we can affirm that when the reservoir contains similar number of neurons the accuracy of the ESQN was better than or similar to the accuracy of the ESN. The reader can compare the ESQN errors for 80 reservoir neurons to the errors in the corresponding models when analyzed using an ESN with 40 neurons.

Figure 1 shows two plots concerning the Lorenz time series. There are 750 time steps of the normalized Lorenz time series (plot on the left side), and there are 750 time steps of the state of 4 neurons chosen in the reservoir (plot on the right side). Each curve on the right plot represents the evolution of the state of a reservoir neuron. Those neurons have been

[1] We used a normalized version of the error, which is standard in these types of studies. It is called Normalized Root Mean Squared Error (NRMSE). See for instance [26].

randomly selected. Both graphics have the same time windows, the same 750 time steps on the horizontal axis. We can see how the reservoir neurons capture or memorize (in their states) a sort of pattern of the original time series. We can also see that the "reaction" of the reservoir neurons to the input behavior occurs after a small delay, because of the reservoir topology. Note that the state of the neurons have small values (for this range of time, the values are lower than 0.02), meaning that in Expression (12), the denominator is much larger than the numerator. Figure 2 presents the same information but working with the Henon map time series. At the left side we have the original time series (for a better visualization we present only 50 time-steps). Finally, the same phenomenon shown in Figure 3 for the Rossler attractor is exhibited.

The next Figures 4, 5, 6 and 7 present the obtained validation error of the ESQN model as a function of the spectral radius of the two matrices in the reservoir, $\mathrm{sr}(\mathbf{w}^{\mathrm{r}+})$ and $\mathrm{sr}(\mathbf{w}^{\mathrm{r}-})$, for a specific reservoir size. The left side of the figures contains the results for a reservoir with 50 neurons, and in the right side we use a reservoir with 100 neurons. Figure 4 corresponds to the MSO time series, Figure 5 corresponds to the Lorenz time series, Figure 6 displays information about the Henon time series, and the last one refers to the Rossler dataset. A common pattern on these graphics is that the value of $\mathrm{sr}(\mathbf{w}^{\mathrm{r}-})$ provokes a lower impact on the accuracy of the model than the value of $\mathrm{sr}(\mathbf{w}^{\mathrm{r}+})$. In the case of the MSO time series, larger values of $\mathrm{sr}(\mathbf{w}^{\mathrm{r}+})$ increase the error. Besides, the error variation along the $\mathrm{sr}(\mathbf{w}^{\mathrm{r}-})$ is almost constant, with exceptions occurring when the pairs $\big(\mathrm{sr}(\mathbf{w}^{\mathrm{r}+}), \mathrm{sr}(\mathbf{w}^{\mathrm{r}-})\big)$ are such that the value of $\mathrm{sr}(\mathbf{w}^{\mathrm{r}+})$ is large and the value of $\mathrm{sr}(\mathbf{w}^{\mathrm{r}-})$ is small.

Figure 4 also shows the impact of the reservoir size. An ESQN with 100 neurons in the reservoir obtains lower errors than with 50 neurons. The spectral radius of the positive weights also seems more relevant in Figure 5. The relevance of $\mathrm{sr}(\mathbf{w}^{\mathrm{r}+})$ is less clear in the case of the MSO time series. In Figure 5 we can see that the pairs $\big(\rho(\mathbf{w}^{\mathrm{r}+}), \rho(\mathbf{w}^{\mathrm{r}-})\big)$ with values in $[0.7, 1] \times [0.1, 03]$ present the worst performance. The best situation is obtained when $\mathrm{sr}(\mathbf{w}^{\mathrm{r}+})$ belongs to $[0.4, 0.6]$. In the case of the Lorenz map, an ESQN with 50 reservoir neurons performs pretty close to another one with 100 reservoir neurons. Figure 6 presents characteristics similar to those of Figure 4. We also see that large values of $\mathrm{sr}(\mathbf{w}^{\mathrm{r}+})$ and small values of $\mathrm{sr}(\mathbf{w}^{\mathrm{r}-})$ augment the error. Figure 7 also shows how $\mathrm{sr}(\mathbf{w}^{\mathrm{r}+})$ has more impact on the accuracy than $\mathrm{sr}(\mathbf{w}^{\mathrm{r}-})$. The best performance has been obtained with larger values of $\mathrm{sr}(\mathbf{w}^{\mathrm{r}+})$ and $\mathrm{sr}(\mathbf{w}^{\mathrm{r}-})$.

The next group of figures, Figures 8, 9, 10 and 11, present the normalized error (the NRMSE) as a function of the reservoir size and the spectral radius. The figures in the left side show the observed relationship between the reservoir size and $\mathrm{sr}(\mathbf{w}^{\mathrm{r}-})$, and those in the right side present the relationship between the reservoir size and $\mathrm{sr}(\mathbf{w}^{\mathrm{r}+})$. The vertical axis of all those figures is the NRMSE. The figures show how important is the reservoir size for the accuracy. They also show that in most of the cases, the value of $\mathrm{sr}(\mathbf{w}^{\mathrm{r}+})$ is more relevant than the value of $\mathrm{sr}(\mathbf{w}^{\mathrm{r}-})$. We can also conclude that the optimal values of the parameters $\big(H, \mathrm{sr}(\mathbf{w}^{\mathrm{r}+}), \mathrm{sr}(\mathbf{w}^{\mathrm{r}-})\big)$ strongly depend of the benchmark problem.

6 Conclusions

This article introduces the Echo State Queueing Network (ESQN), a new Reservoir Computing (RC) learning technique whose dynamics is inspired by the Random Neural Network (RNN), that is, a particular G-network of queues interpreted as a Machine Learning tool. The proposed approach is an attempt for combining the good properties of RC models, in particular, of its main representative, the Echo State Network (ESN), and G-nets, two successful procedures of the Machine Learning community.

The paper describes the context, presents the new model and performs an empirical analysis of its properties. In particular, we analyze the model's dynamics, its initialization, and its accuracy for solving some well-known benchmark problems.

The experimental results are built around the evaluation of the new technique on four standard dynamical systems used as benchmark in the Machine Learning literature. The results show that the ESQN is an accurate tool, and its accuracy is competitive with the already

Table 1: NRMSE of the ESQN and ESN models. The reservoir matrices of both models have spectral radius equal to 0.5 (sr($\mathbf{w}^{\mathrm{r}+}$) = sr($\mathbf{w}^{\mathrm{r}-}$) = 0.5 and the same 0.5 value for the reservoir matrix of the ESN).

Model	Problem	40 reservoir neurons	80 reservoir neurons
ESQN	MSO	1.320755×10^{-3}	1.333004×10^{-3}
	Lorenz	1.089686×10^{-4}	1.888829×10^{-5}
	Rossler	1.302032×10^{-3}	9.350732×10^{-4}
	Henon	5.538284×10^{-5}	3.257085×10^{-5}
Model	Problem	40 reservoir neurons	80 reservoir neurons
ESN	MSO	2.397251×10^{-3}	1.727307×10^{-3}
	Lorenz	9.094324×10^{-3}	1.067347×10^{-3}
	Rossler	7.230140×10^{-4}	7.011503×10^{-4}
	Henon	3.174407×10^{-4}	6.855022×10^{-4}

well-stablished ESN model. Due to the fact that the reservoir structure is fixed during the training algorithms, the initialization of the reservoir matrix has an important impact in the model's accuracy. We found two main global parameters of the reservoir that should be well adjusted for obtaining an accurate tool. The first one is the reservoir size, behaving approximately as the number of hidden neurons of a classical feed-forward Artificial Neural Network. In all observed cases, there exists a threshold value for the reservoir size. Reservoirs with a larger number of neurons than this threshold perform equal or worst that smaller reservoirs. In addition, observe that very large reservoirs are computationally expensive. The optimal region for the reservoir size depends of course of the problem, but in all our experiments, the best reservoir had always less than 80 neurons. The second parameter is the spectral radius of the reservoir matrices (positive and negative). In the case of the ESN model, the spectral radius has a direct impact on the stability and on the memory capacity of the method. In the case of our approach, the spectral radius of the reservoir with positive weight matrix seems to be more relevant to accuracy than the spectral radius of the negative weight matrix. According to our results, when the spectral radius of the reservoir with positive weight is close to 1, the model's accuracy degrades. Another interesting characteristic of the model is how the reservoir neurons somehow capture the way the input evolves. We show in a group of graphics how the neuron states evolve following similar patterns than those observed in the input sequence.

The ESQN model opens up further research directions in the well-developed G-nets and RC areas. In a similar way that other RC techniques, several extensions to the model can be considered. A theoretical analysis of the stability of the reservoir dynamics remains to be done. Finally, it is with its use that a practical procedure is understood and its properties identified. Much more experimental work is necessary, tests on real-world problems together with other benchmarks, and more comparisons. In particular, there is a significant number of very recent papers proposing improvements to the canonical ESN technique. The exploration of the possibility of applying them to, or of developing similar ones for ESQNs is an obvious research direction to be explored in the near future.

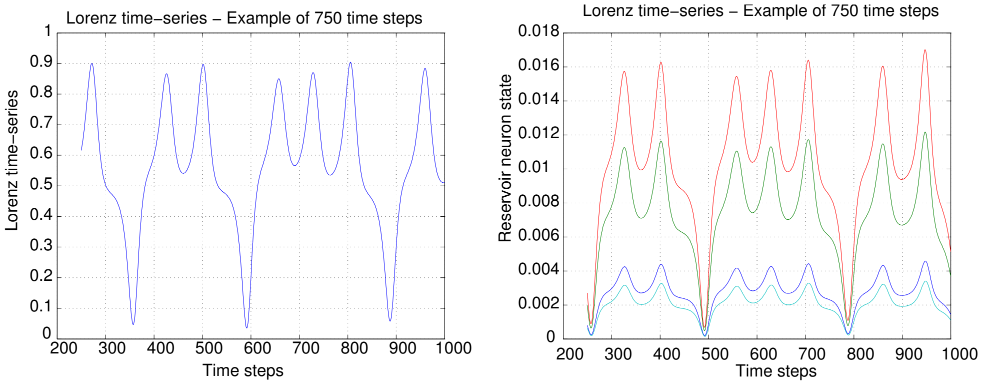

Figure 1: Lorenz map time series. The graphics on the left shows the original time series. The graphics on the right side shows the evolution in 750 time-steps of the reservoir states of 4 randomly selected neurons of the reservoir. The ESQN has been randomly created with a reservoir having 50 neurons, spectral radius of the positive weight matrix equal to 0.2 and spectral radius of the negative weight matrix equal to 0.5.

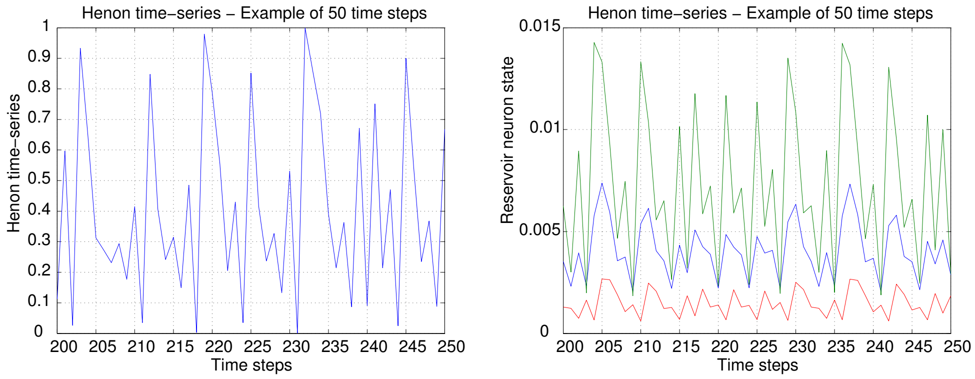

Figure 2: Henon map time series. The graphics on the left shows the original time series. The graphics on the right side shows the evolution in 50 time-steps of the reservoir states of 4 randomly selected neurons of the reservoir. The ESQN has been randomly created with a reservoir having 50 neurons, spectral radius of the positive weight matrix equal to 0.2 and spectral radius of the negative weight matrix equal to 0.5.

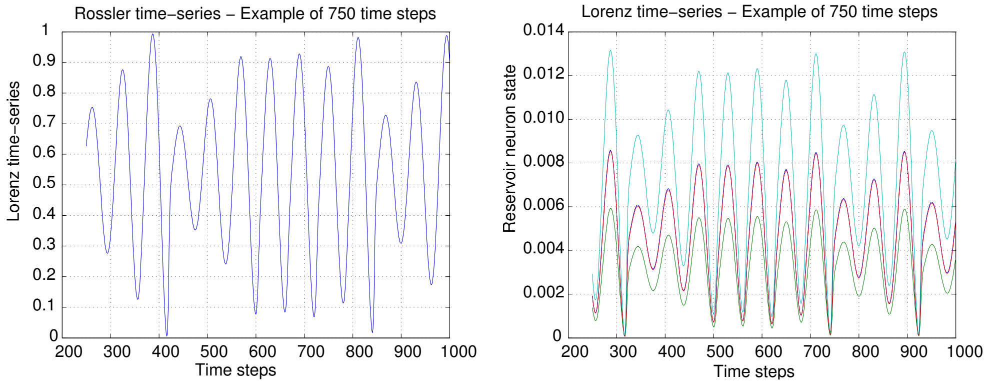

Figure 3: Rossler map time series. The graphics on the left shows the original time series. The graphics on the right side shows the evolution in 50 time-steps of the reservoir states of 4 randomly selected neurons of the reservoir. The ESQN has been randomly created with a reservoir having 50 neurons, spectral radius of the positive weight matrix equal to 0.2 and spectral radius of the negative weight matrix equal to 0.5.

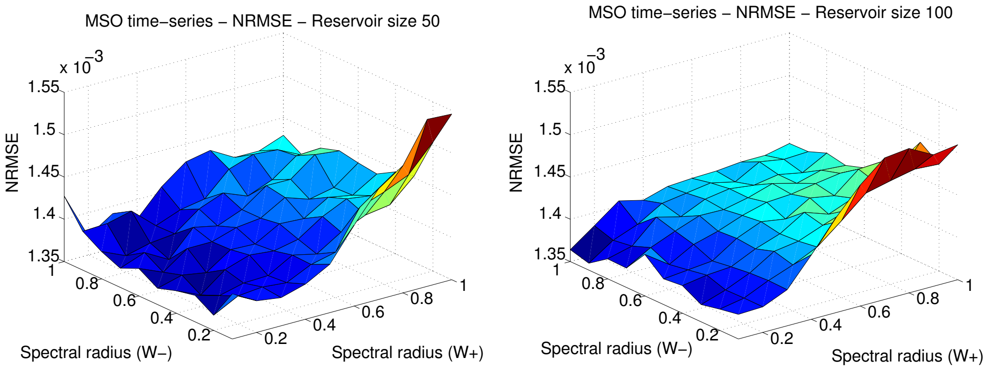

Figure 4: MSO time series. Evolution of the NRMSE computed with an ESQN having a reservoir with 50 neurons (plot on the left side) and 100 neurons (plot in the right side).

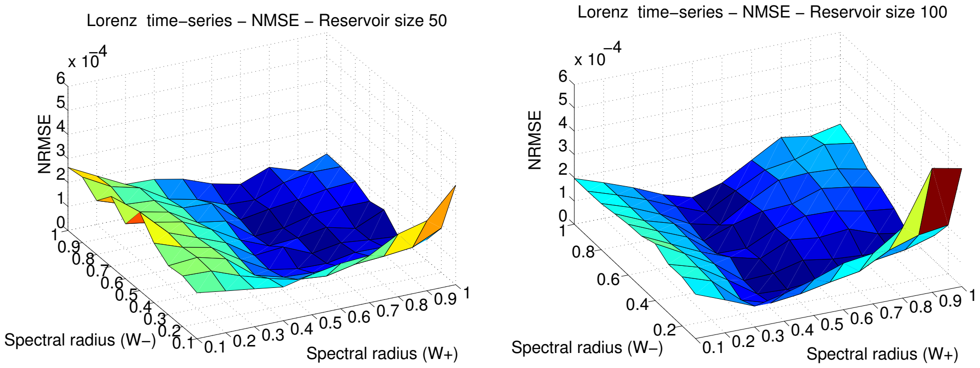

Figure 5: Lorenz time series. Evolution of the NRMSE computed with an ESQN having a reservoir with 50 neurons (plot on the left side) and 100 neurons (plot in the right side).

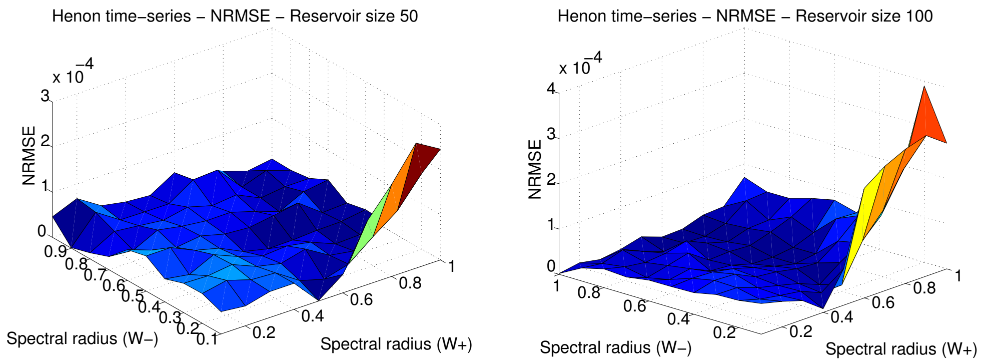

Figure 6: Henon map time series. Evolution of the NRMSE computed with an ESQN having a reservoir with 50 neurons (plot on the left side) and 100 neurons (plot in the right side).

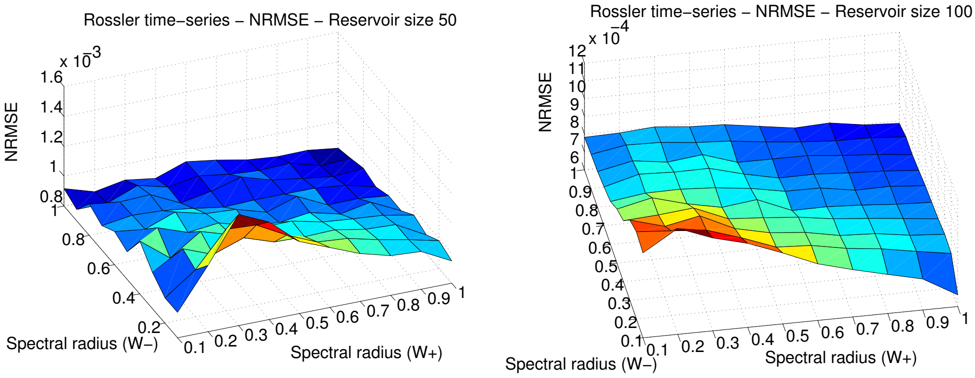

Figure 7: Rossler map time series. Evolution of the NRMSE computed with an ESQN having a reservoir with 50 neurons (plot on the left side) and 100 neurons (plot in the right side).

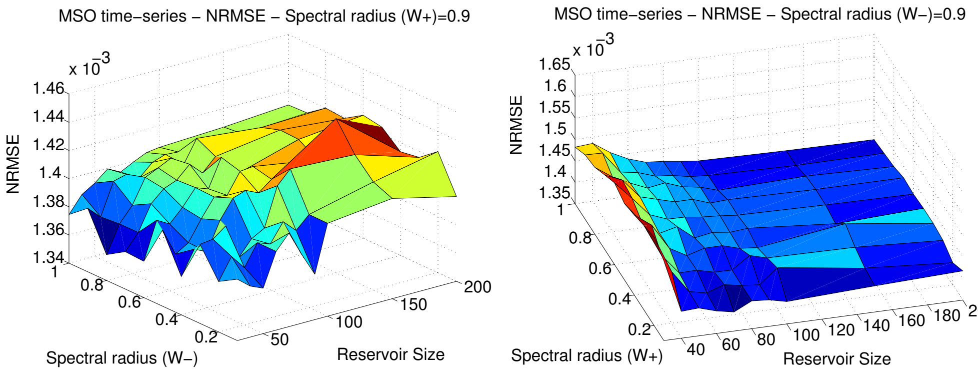

Figure 8: MSO time series. Evolution of the NRMSE computed with ESQNs having different reservoir sizes. Plot on the left side: normalized error as a function of the reservoir size and the spectral radius of $\mathbf{w}^{r-}$, when the spectral radius of $\mathbf{w}^{r+}$ is fixed to 0.9. Plot on the right side: evolution of NRMSE as a function of the reservoir size and the spectral radius of $\mathbf{w}^{r+}$, when the spectral radius of $\mathbf{w}^{r-}$ is equal to 0.9.

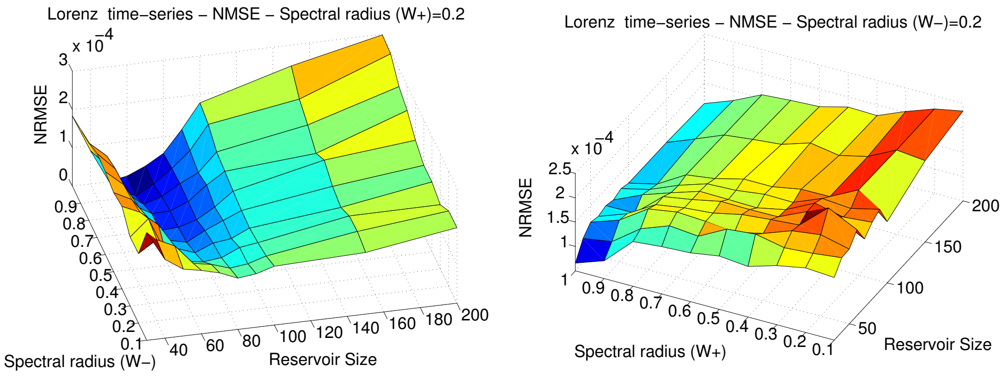

Figure 9: Lorenz time series. Evolution of the NRMSE computed with ESQNs having different reservoir sizes. Plot on the left side: normalized error as a function of the reservoir size and the spectral radius of $\mathbf{w}^{r-}$, when the spectral radius of $\mathbf{w}^{r+}$ is fixed to 0.2. Plot on the right side: evolution of NRMSE as a function of the reservoir size and the spectral radius of $\mathbf{w}^{r+}$, when the spectral radius of $\mathbf{w}^{r-}$ is equal to 0.2.

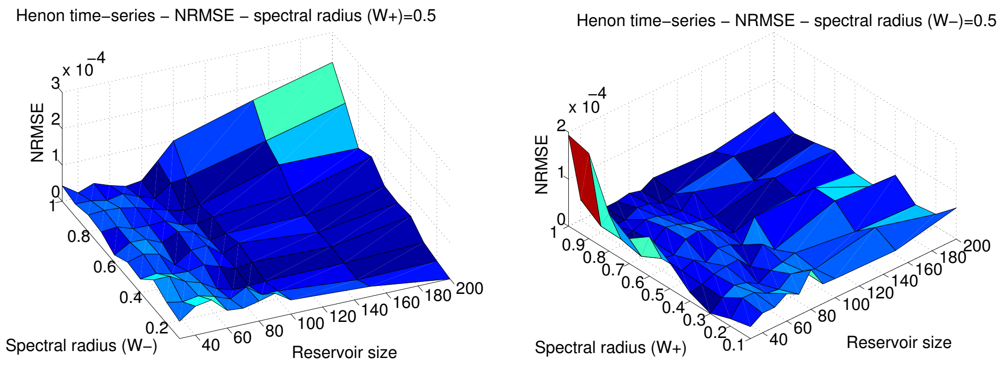

Figure 10: Henon time series. Evolution of the NRMSE computed with ESQNs having different reservoir sizes. Plot on the left side: normalized error as a function of the reservoir size and the spectral radius of $\mathbf{w}^{r-}$, when the spectral radius of $\mathbf{w}^{r+}$ is fixed to 0.9. Plot on the right side: evolution of NRMSE as a function of the reservoir size and the spectral radius of $\mathbf{w}^{r+}$, when the spectral radius of $\mathbf{w}^{r-}$ is equal to 0.9.

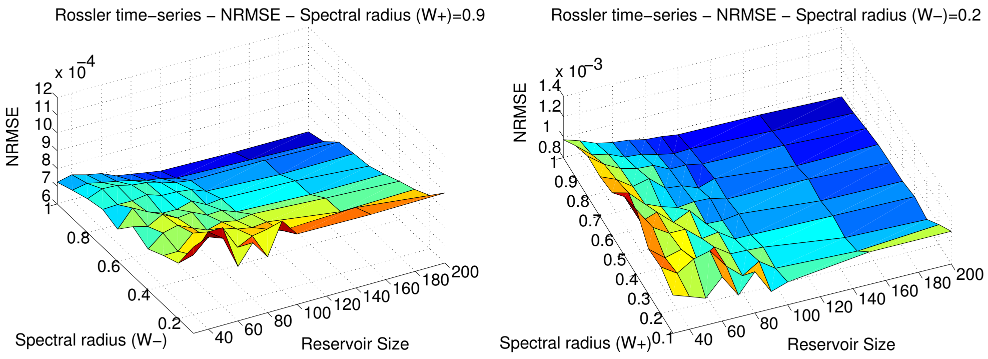

Figure 11: Rossler time series. Evolution of the NRMSE computed with ESQNs having different reservoir sizes. Plot on the left side: normalized error as a function of the reservoir size and the spectral radius of $\mathbf{w}^{r-}$, when the spectral radius of $\mathbf{w}^{r+}$ is fixed to 0.9. Plot on the right side: evolution of NRMSE as a function of the reservoir size and the spectral radius of $\mathbf{w}^{r+}$, when the spectral radius of $\mathbf{w}^{r-}$ is equal to 0.2.

References

[1] Hakan Bakircioğlu and Taşkin Koçak. Survey of Random Neural Network applications. *European Journal of Operational Research*, 126(2):319–330, 2000.

[2] Sebastián Basterrech, Colin Fyfe, and Gerardo Rubino. Self-organizing Maps and Scale-invariant Maps in Echo State Networks. In *11th International Conference on Intelligent Systems Design and Applications, ISDA 2011, Córdoba, Spain, November 22-24, 2011*, pages 94–99, November 2011.

[3] Sebastián Basterrech, Samir Mohamed, Gerardo Rubino, and Mostafa Soliman. Levenberg-Marquardt Training Algorithms for Random Neural Networks. *Computer Journal*, 54(1):125–135, January 2011.

[4] Sebastián Basterrech and Gerardo Rubino. Echo State Queueing Network: A new Reservoir Computing Learning Tool. In *10th IEEE Consumer Communications and Networking Conference, CCNC 2013, Las Vegas, NV, USA, January 11-14, 2013*, pages 118–123, 2013.

[5] Olivier Brun, Lan Wang, and Erol Gelenbe. Big data for autonomic intercontinental overlays. *IEEE Journal on Selected Areas in Communications*, 34(3):575–583, 2016.

[6] J. B. Butcher, D. Verstraeten, B. Schrauwen, C. R. Day, and P. W. Haycock. Reservoir Computing and Extreme Learning Machines for Non-linear Time-series Data Analysis. *Neural Networks*, 38:76–89, feb 2013.

[7] Kenji Doya. Bifurcations in the learning of Recurrent Neural Networks. In *IEEE International Symposium on Circuits and Systems*, pages 2777–2780, 1992.

[8] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[9] Aida A. Ferreira, Teresa B. Ludermir, and Ronaldo R. B. De Aquino. An Approach to Reservoir Computing Design and Training. *Expert Syst. Appl.*, 40(10):4172–4182, August 2013.

[10] E. Gelenbe. Random Neural Networks with Negative and Positive Signals and Product Form Solution. *Neural Computation*, 1(4):502–510, 1989.

[11] E. Gelenbe. Product-Form Queueing Networks with Negative and Positive Customers. *Journal of Applied Probability*, 28(3):656–663, September 1991.

[12] E. Gelenbe. Learning in the Recurrent Random Neural Network. *Neural Computation*, 5(1):154–511, 1993.

[13] E. Gelenbe. The Spiked Random Neural Network: Nonlinearity, Learning and Approximation. In *Proc. Fifth IEEE International Workshop on Cellular Neural Networks and Their Applications*, pages 14–19, London, England, april 1998.

[14] E. Gelenbe, Z. Mao, and Y. Li. Function Approximation by Random Neural Networks with a Bounded Number of Layers. *Journal of Differential Equations and Dynamical Systems*, 12:143–170, 2004.

[15] E. Gelenbe and M. Schassberger. Stability of product form G-Networks. *Probability in the Engineering and Informational Sciences*, 6:271–276, 1992.

[16] Erol Gelenbe. G-networks with signals and batch removal. *Probability in the Engineering and Informational Sciences*, 7:335–342, 1993.

[17] Erol Gelenbe. Learning in the recurrent random neural network. *Neural Computation*, 5:154–164, 1993.

[18] Erol Gelenbe. Steps toward self-aware networks. *Communications of the ACM*, 52(7):66–75, 2009.

[19] Erol Gelenbe and Jean-Michel Fourneau. G-networks with resets. *Performance Evaluation*, 49(1):179–191, 2002.

[20] Erol Gelenbe, Zhi hong Mao, and Yan da Li. Function approximation by random neural networks with a bounded number of layers. *Journal of Differential Equations and Dynamical Systems*, 12(1-2):143–170, 2004.

[21] Erol Gelenbe and Khaled Hussain. Learning in the multiple class random neural network. *IEEE Transactions on Neural Networks*, 13(6):1257–1267, 2002.

[22] Erol Gelenbe, Zhi-Hong Mao, and Yan-Da Li. Function approximation with spiked random networks. *IEEE Transactions on Neural Networks*, 10(1):3–9, 1999.

[23] Erol Gelenbe and Stelios Timotheou. Random neural networks with synchronized interactions. *Neural Computation*, 20(9):2308–2324, 2008.

[24] Erol Gelenbe and Yonghua Yin. Deep learning with random neural networks. In *SAI Intelligent Systems Conference 2016*. IEEEXpress, 2016.

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

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

[27] Michael Jordan. Serial order: A parallel distributed processing approach. Technical Report 8604, Institute for Cognitive Science, University of Califor- nia, San Diego, USA, 1986.

[28] A. Likas and A. Stafylopatis. Training the Random Neural Network using Quasi-Newton Methods. *Eur.J.Oper.Res*, 126:331–339, 2000.

[29] Mantas Lukoševičius. On self-organizing reservoirs and their hierarchies. Technical Report 25, Jacobs University, Bremen, 2010.

[30] Mantas Lukoševičius and Hebert Jaeger. Reservoir Computing Approaches to Recurrent Neural Network Training. *Computer Science Review*, 3:127–149, 2009.

[31] Wolfgang Maass. Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons. Technical Report TR–1999–037, Institute for Theorical Computer Science. Technische Universitaet Graz. Graz, Austria, 1999.

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

[33] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Computational models for generic cortical microcircuits. In *Neuroscience Databases. A Practical Guide*, pages 121–136, Boston, Usa, June 2003. Kluwer Academic Publishers.

[34] Samir Mohamed and Gerardo Rubino. A study of real-time packet video quality using random neural networks. *IEEE Trans. Circuits Syst. Video Techn.*, 12(12):1071–1083, 2002.

[35] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *Proceedings of the 30th International Conference on Machine Learning*, 28:37–48, 2013.

[36] H. Paugam-Moisy and S. M. Bohte. *Handbook of Natural Computing*, chapter Computing with Spiking Neuron Networks. Springer-Verlag, september 2009.

[37] William Press, Saul Teukolsky, William Vetterling, and Brian Flannery. *Numerical Recipes in C.* Cambridge University Press, Cambridge, UK, 2nd edition, 1992.

[38] Nikolay Robinzonov, Gerhard Tutz, and Torsten Hothorn. Boosting techniques for nonlinear time series models. *AStA Advances in Statistical Analysis*, 96(1):99–122, 2012.

[39] Ali Rodan and Peter Tiňo. Minimum Complexity Echo State Network. *IEEE Transactions on Neural Networks*, 22:131–144, 2011.

[40] Gerardo Rubino and Martín Varela. A new approach for the prediction of end-to-end performance of multimedia streams. In *1st International Conference on Quantitative Evaluation of Systems (QEST 2004), 27-30 September 2004, Enschede, The Netherlands*, pages 110–119, 2004.

[41] Â Gerardo Rubino. Quantifying the quality of audio and video transmissions over the internet: The psqa approach. In J. J. Barria, editor, *Design and Operations of Communication Networks: A Review of Wired and Wireless Modelling and Management Challenges*. Imperial College Press, 2005.

[42] D. E. Rumelhart, G. E. Hinton, and J. L. McClelland. A general framework for parallel distributed processing. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1 of *Computational Models of Cognition and Perception*, chapter 2, pages 45–76. MIT Press, Cambridge, MA, 1986.

[43] Benjamin Schrauwen, Marion Wardermann, David Verstraeten, Jochen J. Steil, and Dirk Stroobandt. Improving Reservoirs using Intrinsic Plasticity. *Neurocomputing*, 71:1159–1171, March 2007.

[44] C. Sheng, J. Zhao, W. Wang, and H. Leung. Prediction Intervals for a Noisy Nonlinear Time Series Based on a Boostrapping Reservoir Computing Network Ensemble. *IEEE Transactions on Neural Networks and Learning Systems*, 24(7):1036 – 1048, 2013.

[45] Hava T. Siegelmann and Eduardo D. Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77–80, 1991.

[46] Jochen J. Steil. Backpropagation-Decorrelation: online recurrent learning with O(N) complexity. *In Proceedings of IJCNN'04*, 1, 2004.

[47] Stelios Timotheou. The Random Neural Network: A Survey. *The Computer Journal*, 53(3):251–267, 2010.

[48] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks*, 20(3):287–289, 2007.

[49] Gilles Wainrib and Mathieu N. Galtier. A local Echo State Property through the largest Lyapunov exponent. *Neural Networks*, 76:39–45, 2016.

[50] Bai Zhang, David J. Miller, and Yue Wang. Nonlinear System Modeling with Random Matrices: Echo State Networks Revisited. *Neural Networks*, 76:39–45, 2016.