Available online at www.sciencedirect.com
ScienceDirect
Neural Networks 20 (2007) 391–403
Neural Networks
www.elsevier.com/locate/neunet

2007 Special Issue

An experimental unification of reservoir computing methods

D. Verstraeten*, B. Schrauwen, M. D'Haene, D. Stroobandt

Department of Electronics and Information Systems, Ghent University, Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium

Abstract

Three different uses of a recurrent neural network (RNN) as a reservoir that is not trained but instead read out by a simple external classification layer have been described in the literature: Liquid State Machines (LSMs), Echo State Networks (ESNs) and the Backpropagation Decorrelation (BPDC) learning rule. Individual descriptions of these techniques exist, but a overview is still lacking. Here, we present a series of experimental results that compares all three implementations, and draw conclusions about the relation between a broad range of reservoir parameters and network dynamics, memory, node complexity and performance on a variety of benchmark tests with different characteristics. Next, we introduce a new measure for the reservoir dynamics based on Lyapunov exponents. Unlike previous measures in the literature, this measure is dependent on the dynamics of the reservoir in response to the inputs, and in the cases we tried, it indicates an optimal value for the global scaling of the weight matrix, irrespective of the standard measures. We also describe the Reservoir Computing Toolbox that was used for these experiments, which implements all the types of Reservoir Computing and allows the easy simulation of a wide range of reservoir topologies for a number of benchmarks.

Keywords: Reservoir computing; Memory capability; Chaos; Lyapunov exponent

1. Introduction

Recurrent neural networks (RNNs) seem to offer an attractive method for solving complicated engineering tasks. They have the advantages of feedforward networks, which include robustness, learning by example and the ability to model highly nonlinear systems, and add to that an inherent temporal processing capability. Possible – and actual – applications are manifold and include the learning of context free and context sensitive languages (Gers & Schmidhuber, 2001; Rodriguez, 2001), control and modeling of complex dynamical systems (Suykens, Vandewalle, & De Moor, 1996) and speech recognition (Graves, Eck, Beringer, & Schmidhuber, 2004; Robinson, 1994).

RNNs have been shown to be Turing equivalent (Kilian & Siegelmann, 1996) for common activation functions and can approximate arbitrary finite state automata (Omlin & Giles, 1994). So, theoretically, RNNs are very powerful tools for solving complex temporal machine learning tasks. Nonetheless, several factors still hinder the large scale deployment of RNNs in practical applications. So far, not many learning rules exist (Haykin, 1999; Jaeger, 2002; Suykens & Vandewalle, 1998) and most suffer from slow convergence rates, thus limiting their applicability. Recently, however, three similar solutions for this problem have been proposed independently.

Maass et al. (Maass, Natschläger, & Markram, 2002), Jaeger (2001a) and Steil (2004) all describe the possibility of using an RNN without adapting the weights of the internal connections. In principle, the output can be generated using any type of classifier or regressor, ranging from a perceptron (Minsky & Papert, 1969) to a Support Vector Machine (Vapnik, 1995). In almost all applications, however, a simple linear discriminant or regression algorithm (Duda, Hart, & Stork, 2001) is used to compute the desired output. This type of readout function offers some quite convincing advantages – such as its ease of training and guaranteed optimality in a least squares sense – and yields very good results. From this viewpoint, the function of the reservoir is similar to that of a kernel in the case of kernel-based methods, by projecting the inputs into a high-dimensional space which enhances the separability. For traditional methods, the projection into the high-dimensional space does not need to be computed because of a convenient

* Corresponding author. Tel.: +32 9 264 34 04; fax: +32 9 264 35 94.
E-mail address: david.verstraeten@ugent.be (D. Verstraeten).
URL: http://www.elis.ugent.be/SNN (D. Verstraeten).

0893-6080/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2007.04.003

mathematical construction (the so-called *kernel trick*), while for reservoir methods the projection is explicit. A significant advantage of the reservoir approach, however, is the fact that the kernel is able to incorporate temporal information present in the inputs.

1.1. Using RNNs as reservoirs

The *Liquid State Machine* (LSM) by Maass et al. (2002) proposes a generic framework, where the reservoir can consist of a broad range of node types, and needs to obey a quite unrestrictive property (the *point-wise separation* property) (Maass et al., 2002) in order to be computationally useful. In practice, most descriptions of the LSM use reservoirs built from a relatively simple spiking neuron model called the Leaky Integrate and Fire (LIF) neuron (Maass & Bishop, 2001) with a dynamic synapse model (Gerstner & Kistler, 2002), but the use of threshold logic gates (TLGs) has also been described (Natschläger, Bertschinger, & Legenstein, 2004). Spiking neuron models are more complex than sigmoidal neurons but they have been shown to be computationally more powerful (Maass, 1997). The readout layer, on the other hand, is also very broadly specified: it needs to be able to approximate any continuous function over a compact set $X \subseteq \mathbb{R}^N$ of input values. When both conditions are fulfilled, the resulting LSM is guaranteed to be able to approximate any time-invariant function with fading memory[1] operating on timeseries. However, these sufficient conditions are too broad to offer a practical guideline for constructing reservoirs, and quite probably universal approximation of temporal functions is not needed to solve real world problems. So, while this result is very convincing from a theoretical point of view, the transition to practical applications is still unclear.

Jaeger independently proposed a very similar computational idea which he calls *Echo State Networks* (Jaeger, 2001a). The main difference with LSMs lies in the type of nodes that constitute the reservoir: where LSMs are usually built from spiking LIF neurons, these reservoirs are built from analog sigmoidal neurons. Here too, a theoretical property is defined for potentially interesting reservoirs called the *echo state* property (Jaeger, 2001a), which expresses – informally stated – the fact that the influence of inputs on reservoir states fades away gradually. Further, an upper and lower bound are defined for the echo state property that are very easy to compute and depend only on the weight matrix of the reservoirs. However, this property alone is not sufficient to guarantee optimal performance for a given problem, and the search for a good reservoir requires experience and can take some time. In Jaeger (2002), some guidelines are offered, but a structured method is still lacking.

Thirdly, Steil proposes an O(N) learning rule for RNNs called *Backpropagation Decorrelation* (BPDC) (Steil, 2004). The BPDC rule is an extension of a state-of-the-art learning algorithm for RNNs (Atiya–Parlos recurrent learning (Atiya & Parlos, 2000)). It appears that this APRL learning rule restricts the adaptation of the weights to the output layer, effectively splitting the RNN into a reservoir and a readout layer. Only the weights to the output layer and the recurrent connections inside the output layer are trained. Thus, the use of an RNN as a reservoir was attained here from an entirely different angle than the previous two approaches. Here too, sigmoidal neurons are used, but a significant difference between BPDC reservoirs and ESNs is the fact that feedback connections from the readout layer into the reservoir and into the readout layer itself are used, whereas in practice this is hardly ever the case for ESNs.

1.2. Applications of reservoir computing

Several successful applications of reservoir computing to both 'abstract' and real world engineering applications have been reported in the literature. Abstract applications include dynamic pattern classification (Jaeger, 2001b), autonomous sine generation (Jaeger, 2001a; Verstraeten, Schrauwen, & Stroobandt, 2005) or the computation of highly nonlinear functions on the instantaneous rates of spike trains (Maass, Natschlger, & Markram, 2004). In robotics, LSMs have been used to control a simulated robot arm (Joshi & Maass, 2004), to model an existing robot controller (Burgsteiner, 2005b), to perform object tracking and motion prediction (Burgsteiner, 2005a; Maass, Legenstein, & Markram, in press) or event detection (Jaeger, 2005). ESNs have been used in the context of reinforcement learning (Bush & Anderson, 2005). Also, applications in the field of Digital Signal Processing (DSP) have been quite successful, such as speech recognition (Maass, Natschläger, & Markram, 2003; Skowronski & Harris, 2006; Verstraeten, Schrauwen, Stroobandt, & Van Campenhout, 2005) or noise modeling (Jaeger & Haas, 2004). Finally, the use of reservoirs for chaotic timeseries generation and prediction has been reported in Jaeger (2001b, 2003), Steil (2005, 2006).

1.3. Comparing reservoir computing methods

In this contribution, we offer an experimental overview of the different implementations of reservoir computing described in the literature. In Section 2, we present experimental results for benchmarks with different characteristics. Section 2.1 shows the relation between the complexity of the reservoir nodes and the performance on three tasks. Section 2.2 investigates the effects of memory on the node level and the reservoir level. Thirdly, Section 2.3 links three existing bounds for the echo state property to the actual dynamics in the network, quantified using a Lyapunov-inspired method as a measure of the network dynamics. In Section 3, we present a novel toolbox we developed to do this work, that incorporates all reservoir implementations currently described and some novel ones. We outline its modular structure and describe which components are present. In Section 4, finally, we conclude and outline future work.

[1] This means that the current output depends on inputs from a finite time window in the past.