ReservoirComputing.jl: An Efficient and Modular Library for Reservoir Computing Models

Francesco Martinuzzi 1, 2, 3
Chris Rackauckas 3, 4
Anas Abdelrehim 3
Miguel D. Mahecha 1, 2, 5
Karin Mora 1, 5

1 Center for Scalable Data Analytics and Artificial Intelligence, Leipzig University, Germany
2 Remote Sensing Centre for Earth System Research, Leipzig University, Germany
3 Julia Computing
4 Massachusetts Institute of Technology
5 German Centre for Integrative Biodiversity Research (iDiv), Leipzig, Germany

Editor: Sebastian Schelter

Abstract

We introduce ReservoirComputing.jl, an open source Julia library for reservoir computing models. It is designed for temporal or sequential tasks such as time series prediction and modeling complex dynamical systems. As such it is suited to process a range of complex spatio-temporal data sets, from mathematical models to climate data. The key ideas of reservoir computing are the model architecture, i.e. the reservoir, which embeds the input into a higher dimensional space, and the learning paradigm, where only the readout layer is trained. As a result the computational resources can be kept low, and only linear optimization is required for the training. Although reservoir computing has proven itself as a successful machine learning algorithm, the software implementations have lagged behind, hindering wide recognition, reproducibility, and uptake by general scientists. ReservoirComputing.jl enhances this field by being intuitive, highly modular, and faster compared to alternative tools. A variety of modular components from the literature are implemented, e.g. two reservoir types - echo state networks and cellular automata models, and multiple training methods including Gaussian and support vector regression. A comprehensive documentation, which includes reproduced experiments from the literature is provided. The code and documentation are hosted on Github under an MIT license https://github.com/SciML/ReservoirComputing.jl.

Keywords: reservoir computing, echo state networks, forecasting, machine learning, Julia

1. Introduction

Time series modeling is a very common technique throughout many areas of machine learning. However, many standard recurrent models are known to be susceptible to problems such as the vanishing gradient (Pascanu et al., 2013) or the extreme sensitivity of chaotic systems to their parameterization (Wiggins and Golubitsky, 2003). To counter these issues reservoir computing (RC) techniques were introduced as recurrent models, which can be
trained without requiring gradient-based approaches (Lukoševičius and Jaeger, 2009). Independently proposed as echo state networks (ESNs) (Jaeger, 2001) and liquid state machines (LSMs) (Maass et al., 2002), these architectures are based on the expansion of the input data using a fixed random internal layer, known as the reservoir, and the subsequent mapping of the reservoir to match an output. This output mapping can normally be written in a simple closed form, such as an $L_2$ minimization computed via a single QR-factorization. This allows the RC approach to achieve faster computational times with less parameter tuning compared to deep learning models which rely on local optimization. Numerous alterations have been made since their inception, such as deep architectures (Gallicchio et al., 2018) and different training approaches (Chatzis and Demiris, 2011; Shi and Han, 2007). Applications have confirmed that these improved architectures and training techniques can be impactful in many fields of study: from surrogates for stiff systems (Anantharaman et al., 2020) to reconstruction of chaotic attractors (Pathak et al., 2018a) and prediction of climate dynamics (Nadiga, 2021). This expanding landscape necessitates the development of production-quality software to empower scientists with all of the latest techniques without having to implement from scratch every novel variation on their own. While a number of libraries for ESNs and RC are available (Trouvain et al., 2020; Steiner et al., 2021; Pontes-Filho et al., 2020), they lack the hackability necessary to alter the provided architectures and achieve the top performances from the literature. To provide a truly modular library for RC models we present ReservoirComputing.jl, an efficient and flexible library written using the Julia language (Bezanson et al., 2017). This software provides a user oriented package with intuitive high level application programming interfaces (APIs), while maintaining a great level of customization. This flexibility is obtained from both the design choices of ReservoirComputing.jl and by leveraging the multiple dispatch architecture of Julia. This gives users the freedom to mix any Julia code with the reservoir computers in a seamless way to promote the latest performance and research.

2. Theoretical Background

The setup of an RC model (Konkoli, 2018) is based on leveraging the complex dynamics of a nonlinear system, the reservoir $\mathcal{R}$. The role of this layer is to expand the input data $\mathbf{u}(t)$ at time $t$ into a higher dimension. This operation can be aided by an input layer $\chi$, which maps the data onto the reservoir. The input data $\mathbf{u}(t)$ drives the reservoir $\mathcal{R}$ dynamics to a certain configuration. The resulting state $\mathbf{x}(t)$ represents the expansion of the input data. Once the states have been obtained for all the input length $T$ the RC model can be trained. The states $\mathbf{x}(t)$ can be manipulated at this stage: through concatenation with the input data $\mathbf{u}(t)$ such as $\mathbf{z}(t) = [\mathbf{x}(t), \mathbf{u}(t)]$ (Jaeger, 2007), using padding with some constant value $c$ usually set $c = 1.0$ such as $\mathbf{z}(t) = [c, \mathbf{u}(t)]$ (Lukoševičius, 2012), or through nonlinear transformations (Chattopadhyay et al., 2020). The training is performed in one step in order to find the best fit between the states $\mathbf{x}(t)$ and the desired output $\mathbf{y}(t)$g. The most common way to perform the training is linear regression, creating the readout layer $\psi$. This layer is finally used in the prediction phase to obtain the desired output $\mathbf{v}(t) = \psi(\mathbf{x}(t))$. 
ReservoirComputing.jl

Figure 1: ReservoirComputing.jl architecture. SV stands for support vector.

3. Library Overview

ReservoirComputing.jl features the implementation of different RC models, training methods, and state variations. The architecture with the main components are detailed below and illustrated in Figure 1. Given the modular nature of the software, more algorithms can be easily implemented on top of the existing ones. Any of the many libraries of the Julia community or custom codes can be used as part of the functions within the reservoir computer. Importantly, as Julia is a just-in-time (JIT) compiled language, these extensions do not sacrifice performance. For brevity, no code snippets are shown in this paper, but the documentation provides a large number of examples.

Building models. ReservoirComputing.jl provides a simple interface for model building that closely follows the workflow presented in the corresponding literature. It includes a standard implementation of ESNs (Lukoševičius, 2012) as well as a hybrid variation (Pathak et al., 2018b), gated recurrent unit ESN (Wang et al., 2020) and double activation function ESN (Lun et al., 2015). Multiple input layers $\chi$ and reservoirs $\mathcal{R}$ are also provided, ranging from weighted input layers (Lu et al., 2017) to minimally complex input layers and reservoirs (Rodan and Tino, 2010; Rodan and Tiño, 2012), including a reservoir obtained through pseudo single value decomposition (Yang et al., 2018). Reservoir computing with cellular automata (ReCA) (Yilmaz, 2014; Nichele and Molund, 2017) is another family of models available in the library leveraging the package CellularAutomata.jl (Martinuzzi, 2022).

Training. The training algorithms are implemented in a low-level fashion, supporting all RC models of the library. Multiple training methods to obtain the output layer $\psi$ can be obtained from open source libraries such as MLJLinearModels.jl (Blaom and Vollmer, 2020), GaussianProcesses.jl (Fairbrother et al., 2021) and LIBSVM.jl (Kornblith and Pastell, 2021), a Julia porting of LIBSVM (Chang and Lin, 2011).

Prediction. ReservoirComputing.jl leverages diverse prediction techniques. The generative approach allows the model to run autonomously: the predicted output $v(t)$ is fed back into the model to obtain the prediction for the next time step $v(t + 1)$. The predictive approach instead uses the standard feature-label setup.

States modifiers. Other lower level implementations are included, such as the possibility to modify the state vectors. All the approaches detailed in §2 are included in the library.
4. Code Quality

The library is made available on Github, where it is continuously tested through Github actions. The package is part of the scientific machine learning (SciML) community (sci), a NumFOCUS sponsored project since 2020 (num). An extensive online documentation is also provided with the library, where models are documented at length with multiple examples for different applications.

A feature comparison of ReservoirComputing.jl with similar libraries is provided in Table 1 and it also showcases the models available. Figure 2 illustrates the superior computational performance of the library. The speed of the Julia code is calculated without considering the JIT compilation. This mimics standard usage, in which a system image is used and the compilation is ahead of time and negligible for the types of applications seen here. ReservoirComputing.jl shows 1.5 times higher computational speed in the worst case scenario and 14.3 times higher in the best case scenario, both compared with the most performant times for reservoir size for the CPU. For GPU calculations the library ranges from being 1.4 to 3.0 times faster. The performance test task is a next step prediction of the Mackey-Glass system (Glass and Mackey, 2010) with time delay $\tau = 17$. The dense reservoir matrix and the dense input matrix are generated with uniform distribution sampled from $[-1,1]$. The spectral radius of the reservoir matrix is scaled by 1.25. The ridge regression parameter is set to $10^{-8}$. Training and prediction lengths are both equal to 4999. The time reported is the sum of training and prediction. For central processing unit (CPU) computations the precision is set to float64, for the graphics processing unit (GPU) computations it is float32. EchoTorch (Schaetti, 2018) is not present in the comparison as the version available on Github was not running on the examples provided. The versions tested are as follows: PyRCN v0.0.16, ReservoirPy v0.3.2, ReservoirComputing.jl v0.8 release candidate and pytorch-esn (Nardo, 2018) retrieved from Github on March 2022. All the simulations were run on a Dell XPS 9510 fitted with an Intel Core i7-11800H CPU, a Nvidia GeForce RTX 3050 Ti GPU and 16 GB of RAM.

5. Conclusion and Outlook

ReservoirComputing.jl is a comprehensive, modular and fast library for RC models with a strong focus on ESNs. In the future we plan to expand the model library by including LSMs and extreme learning machines (ELMs), while maintaining the intuitiveness of the current implementation.

Acknowledgments

This work was partially supported by the German Federal Ministry of Education and Research (BMBF, 1IS18026B) by funding the competence center for Big Data and AI “ScaDS.AI” Dresden/Leipzig. K.M. is funded by the German Centre for Integrative Biodiversity Research (iDiv) Halle-Jena-Leipzig via a FLEXPOOL project, funded by the German Research Foundation (DFG-FZT 118, 202548816). M.D.M. and K.M. thank the European Space Agency for funding the ”DeepExtremes” project (AI4Science ITT).
Figure 2: Speed comparison of RC libraries. The point timings for each reservoir size are obtained by averaging over 100 runs with different random initializations. Error bars are not shown as the variability is negligible on the log-scale.

Table 1: Comparison of RC libraries. The code quality section focuses on high level, quality of life library features. The subsections: Documentation indicates whether the library includes general examples or how-to guides; API/Tutorials indicates whether API documentation and step by step tutorials are provided. In the ESN section we list common models included in the libraries. Subsection Multiple reservoirs indicates whether native multiple reservoir matrix constructions are provided. In the training section, SV is short hand for support vector regression. In the Miscellanea section Data sets indicates whether ready to use data sets are provided in the library; Optimization stands for native hyperparameter optimization.

References


