reservoirpy: A Simple and Flexible Reservoir Computing Tool in Python

Nathan Trouvain, Xavier Hinaut

reservoirpy: A Simple and Flexible Reservoir Computing Tool in Python

Nathan Trouvain nathan.trouvain@inria.fr
Inria Bordeaux Sud-Ouest, IMN, LaBRI

Xavier Hinaut xavier.hinaut@inria.fr
Inria Bordeaux Sud-Ouest, IMN, LaBRI

Editor: ?

Abstract

This paper presents `reservoirpy`, a Python library for Reservoir Computing (RC) models design and training, with a particular focus on Echo State Networks (ESNs). The library contains basic building blocks for a large variety of recurrent neural networks defined within the field of RC, along with both offline and online learning rules. Advanced features of the library enable compositions of RC building blocks to create complex "deep" models, delayed connections between these blocks to convey feedback signals, and empower users to create their own recurrent operators or neuronal connections topology. This tool is solely based on Python standard scientific packages such as `numpy` and `scipy`. It improves RC time efficiency with parallelism using `joblib` package, making it accessible to a large academic or industrial audience even with a low computational budget. Source code, tutorials and examples from the RC literature can be found at `https://github.com/reservoirpy/reservoirpy` while documentation can be found at `https://reservoirpy.readthedocs.io/en/latest/?badge=latest`

Keywords: Reservoir Computing, Echo State Networks, Recurrent Neural Networks, Machine Learning, Python, Online Learning, Offline Learning

1. Introduction

Reservoir Computing (RC) is a Machine Learning (ML) paradigm to train Recurrent Neural Networks (RNNs) while keeping the recurrent layer untrained. Training RNNs often reveals itself difficult and costly due to the challenge of keeping track of long range time dependencies using backpropagation through time (BPTT) algorithm. Solutions such as Long Short Term Memory (LSTM) cells (Hochreiter and Schmidhuber, 1997) use a gating mechanism to filter past information. They are an attempt to tackle vanishing or exploding gradient problem arising with long timeseries when training RNNs with BPTT. Nevertheless, this method comes at a high computational cost.

RC networks use a random recurrent layer of neurons, it is called a *reservoir* because it is literally a *reservoir of computations* based on non-linear combinations of inputs. As firstly formulated by Jaeger (2001) and Maass et al. (2002) within the ML community and Dominey (1995), Buonomano and Merzenich (1995) in the Neuroscience field, a single layer of neurons—the *readout*—is in charge with decoding the dynamics of the reservoir internal states. Only the connections between the reservoir and the readout are trained, using any offline learning rules, such as linear regression, or online learning rules, such as

Recursive Least Squares (Sussillo and Abbott, 2009) or reward-modulated Hebbian learning rules (Hoerzer et al., 2014). These methods display good performances on a great variety of tasks, and compete with state of the art RNN methods like LSTMs on several tasks Trouvain and Hinaut (2021).

While most RNN techniques can be easily replicated using popular Deep Learning frameworks, RC models are often implemented from scratch because there is no common library wells spread within the RC community. They also do not really benefit from the automatic differenciation features and training strategies of theses Deep Learning tools, as BPTT is not needed. On the other hand, libraries like `scikit-learn` (Buitinck et al., 2013) offer a very simple API to apply ML techniques, but lack of flexibility and are not meant to create complex neural networks operating on timeseries, with feedback loops and online learning rules.

In this paper, we present `reservoirpy`, a library written in Python 3, tailored for RC networks design. The library is in particular focused toward Echo State Networks (ESNs) (Jaeger, 2001), the most popular flavor of RC, which consists of a RNN of recurrently connected neurons that enables to project input data into a high-dimensional non-linear space, which is then decoded – *read-out* – by a single layer of neurons with trained connections. `reservoirpy` relies only on `numpy` and `scipy`, the Python standard scientific libraries, but its API is heavily inspired from `scikit-learn`, making it accessible to a large audience of researchers, students or any ML professionals. This library allows to design complex neural networks equipped with online or offline learning rules, add feedback connections, accelerate computations through parallelism, and quickly extend it with a mixed object-oriented/functional interface.

2. Library Overview

2.1 Architecture and API

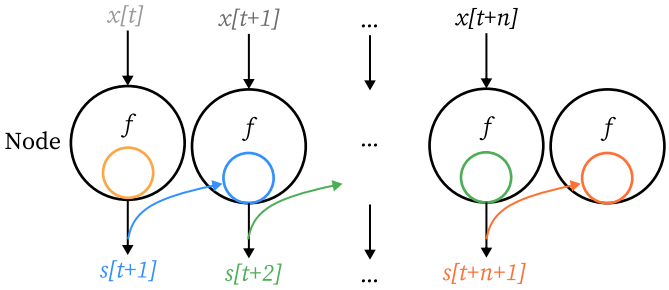

Figure 1: A node is a recurrent operator with an internal state $\mathbf{s}$, represented by a colored circle. This state $\mathbf{x}_t, \mathbf{s}_t \mapsto \mathbf{s}_{t+1}$, where $\mathbf{x}$ is an input timeseries. By default, a node outputs its internal state.

`reservoirpy` is centered around the `Node` base class. As schematised in Figure 1, a node is a recurrent operator which maps its internal state $\mathbf{s}_t$ and some input vector $\mathbf{x}_t$ at a timestep t with the next state $\mathbf{s}_{t+1}$. This mapping is defined by the function f, which can

be implemented by users as a simple Python function operating on a node and an input vector stored as a `numpy` array. A node can also carry parameters (e.g. a weight or bias matrix), stored as `numpy` arrays or `scipy` sparse matrices for efficiency. These parameters may be updated during the training phase of the node. Learning rules are also defined as Python functions passed to the node class constructor.

Finally, nodes can be connected together inside directed acyclic graphs to represent more complex operations. This mechanism allows to chain node functions. A graph of nodes is represented by the `Model` class, which is a subclass of the abstract `Node` class, and therefore share the same API as nodes. This allows to chain models together, or train them as a whole.

Connections between nodes may also be delayed. These type of connections are called *feedback connections*. Some node operators may therefore be of the form $f : \mathbf{s}_t, \mathbf{x}_t, \mathbf{y}_{t-1} \mapsto \mathbf{s}_{t+1}$, where $\mathbf{y}$ is an external signal, generally coming from other nodes in a model. $\mathbf{s}_t$ represents to internal state of the reservoir and $\mathbf{x}_t$ represent some input value. These type of delayed signal can also be used as target values in online learning, or as reward signals for reinforcement learning-like rules (Hoerzer et al., 2014).

A RC model can therefore be constructed with nodes within the library or by creation of new ones. This only requires to define the f operator, its parameters and a learning function, if needed. Nodes can be used as functions or as objects, and can also be created by subclassing the `Node` base class. Morevoer, one can also create a node by simply passing the functions and parameters as argument to this `Node` class constructor. This mechanism offers flexibility: one can switch between the high-level object-oriented API for end-to-end training, similar to what can be found in `scikit-learn`, or one can use nodes as functions and define their own training/inference policies from scratch.

2.2 Reservoir Computing Tools

`reservoirpy` already contains various implementations of RC tools, defined as `Node` subclasses, in the `reservoirpy.nodes` module. Some notable examples are:

- `Reservoir`, a recurrent pool of leaky neurons, to perform high dimensional embedding of timeseries;
- `Ridge`, a readout layer of neurons which connections are learnt through Tikhonov linear regression;
- `LMS`, a readout layer of neurons which connections are learnt using Least Mean Square, allowing online learning, as used in Hoerzer et al. (2014).
- `RLS`, a readout layer of neurons which connections are learnt using Recursive Least Square, allowing online learning, as used in Sussillo and Abbott (2009).

The library also provides nodes implementing the Intrinsinc Plasticity mechanism for reservoirs Steil (2007) Schrauwen et al. (2008) or recent reservoir reformulations like the Non-Linear Vector Autoregressive machine from Gauthier et al. (2021). New implementations are regularly proposed and added to the library *GitHub* repository, and we strongly encourage potential users to propose their own.

In addition to the main `reservoirpy.nodes` module, the library contains various chaotic timeseries generators for benchmarking (`reservoirpy.datasets`), hyperoptimization wrappers for `hyperopt` optimization tool (Bergstra et al., 2013) (`reservoirpy.hyper`), and some other utilities like weight matrix initialization functions (`reservoirpy.mat_gen`), activation functions (`reservoirpy.activationsfunc`) or simple metrics (`reservoirpy.observables`).

2.3 Comparison to Related Softwares

Table 1 summarizes the comparison between the presented library and some other open source software. We compared *PyRCN* (Steiner et al., 2021), *EchoTorch* (Schaetti, 2018), *ReservoirComputing.jl* (Martinuzzi et al., 2022), *Pytorch-esn* [1], *DeepESN* (Gallicchio et al., 2018), *RCNet* [2], *LSM* (Kaiser et al., 2017) and the historical package *Oger* (Verstraeten et al., 2012), no longer maintained.

`reservoirpy` is the only recent package able to handle delayed connections and providing a complete online learning API. Many other RC key features are spread across different implementations. EchoTorch allows users to manipulate conceptors, a mechanism enabling to control the dynamics of resevoirs proposed by Jaeger (2014). PyRCN defines a complete interface to train Extreme Learning Machine (ELM) (Huang et al., 2011), a network similar to ESN where recurrence has been removed. While LSM specializes in handling spiking neural networks. We make constant effort in gathering new features, and believe our flexible API will ease this process for outside contributors.

1. `https://github.com/stefanonardo/pytorch-esn`
2. `https://github.com/okozelsk/NET`

Library	**Language**	**Main dependency**	**Last activity**	**Package**	**Doc.**	**Tests**	**Off.**	**On.**	**Fb.**	**Model type**	**Deep**
PyRCN	Python 3	Scikit-learn	Jan. 2022	pip	✓	✓	✓	×	×	ESN	✓
EchoTorch	Python 3	PyTorch	Sep. 2021	pip	✓	✓	✓	×	×	ESN, Conceptors	✓
Res.Comp.jl [3]	Julia	Julia	Apr. 2022	Pkg	✓	✓	✓	×	×	ESN	×
Pytorch-esn	Python 3	Pytorch	Aug. 2021	×	×	×	✓	×	×	ESN	✓
DeepESN	Matlab	Matlab	Feb. 2019	Matlab	✓	?	✓	×	×	ESN	✓
RCNet	C#	C#	Aug. 2021	×	partial	×	✓	×	×	ESN, LSM	×
LSM	Python 3	Nest	Nov. 2020	×	×	×	✓	×	×	LSM	×
Oger	Python 2	mdp	2012 (obsolete)	×	×	✓	✓	✓	✓	LSM, ESN	✓
`reservoirpy` **(this package)**	Python 3	Numpy	Apr. 2022	pip	✓	✓	✓	✓	✓	ESN	✓

Table 1: Comparative table of some open source software for Reservoir Computing. This table might not be exhaustive. **Doc.** Complete documentation. **Off.** Offline learning strategies included. **On.** Online learning strategies included. **Fb.** Feedback and delayed connections. **Deep** The software allows the design of complex models where basic RC elements such as reservoirs and readouts can be stacked to form so-called "deep" networks.

3. ReservoirComputing.jl

3. Experiment with `reservoirpy`

In this section, we present a minimal example on how to use **`reservoirpy`** for chaotic timeseries–namely the Mackey-Glass timeseries–10 timesteps ahead forecasting with an ESN. Training results are displayed in Figure 2.

```
import reservoirpy as rpy
from reservoirpy.datasets import mackey_glass
from reservoirpy.observables import rmse
from reservoirpy.nodes import Reservoir, Ridge
rpy.set_seed(0)  # fix random state for reproducibility
X = mackey_glass(2500)
 # split dataset for training
X_train, Y_train = X[:2000], X[10:2010]
X_test, Y_test = X[2000:-10], X[2010:]
# Reservoir node (100 neurons, with custom leak rate and
# recurrent matrix spectral radius)
reservoir = Reservoir(100, lr=0.3, sr=0.9)
# Readout node (ridge linear regression)
readout = Ridge(ridge=1e-6)
# ESN creation (reservoir is connected to readout)
esn = reservoir >> readout
# Train and run
Y_pred = esn.fit(X_train, Y_train).run(X_test)
print("Root Mean Squared Error:", rmse(Y_test, Y_pred))
'>>> Root Mean Squared Error: 0.00866911443388836'
```

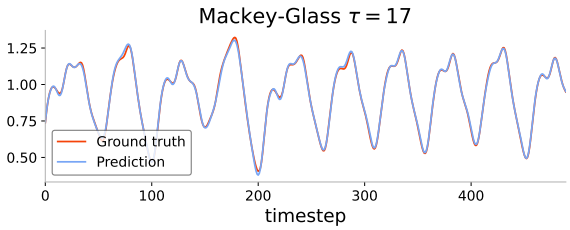

Figure 2: 10 timesteps ahead prediction of Mackey-Glass timeseries using an Echo State Network. The code above describe the training process of this ESN using **`reservoirpy`**.

4. Conclusion

`reservoirpy` is a complete toolbox to apply different Reservoir Computing techniques on any data where time carries information. The library offers both a high-level API similar to **`scikit-learn`** interface and a low-level API to create custom models using Python standard scientific stack. It contains several architectures of reservoirs and readouts, with online and offline learning rules, along with datasets, tutorials and documentation. The whole project

is delivered under the open source MIT licence, and is open to contribution from any RC enthusiast.

`reservoirpy` is still under development because we plan to integrate many features in the upcoming months/years. These features include the improvement of the computational capabilities of the tool by implementing an interface with other scientific backends such as *TensorFlow* or *PyTorch* and enabling computations over specific hardware like GPUs. Future features will also include a better interface with `scikit-learn`, embedding for instance `scikit-learn` tools within `reservoirpy` in a transparent way, spiking neural network support to implement Liquid State Machines (LSMs) (Maass et al., 2002), and the implementation of more tools and paper replication from the literature.

Acknowledgments

References

James Bergstra, Dan Yamins, and David D Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in Science Conference*, pages 13–20, 2013.

Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

DV Buonomano and M. M. Merzenich. Temporal information transformed into a spatial code by a neural network with realistic properties. *Science*, 267:1028 – 1030, 1995.

P. Dominey. Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning. *Biological cybernetics*, 73(3):265–274, 1995.

C. Gallicchio, A. Micheli, and L. Pedrelli. Design of deep echo state networks. *Neural Networks*, 108:33 – 47, 2018.

Daniel J. Gauthier, Erik Bollt, Aaron Griffith, and Wendson A. S. Barbosa. Next generation reservoir computing. 12(1):5564, 2021.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

Gregor M. Hoerzer, Robert Legenstein, and Wolfgang Maass. Emergence of Complex Computational Structures From Chaotic Neural Networks Through Reward-Modulated Hebbian Learning. 24(3):677–690, 2014.

Guang-Bin Huang, Dian Hui Wang, and Yuan Lan. Extreme learning machines: A survey. *Int. J. Mach. Learn. & Cyber.*, 2(2):107–122, June 2011.

H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. *Bonn, Germany: German National Research Center for Information Technology GMD Tech. Report*, 148:34, 2001.

Herbert Jaeger. Controlling recurrent neural networks by conceptors. *CoRR*, abs/1403.3369, 2014. URL `http://arxiv.org/abs/1403.3369`.

Jacques Kaiser, Rainer Stal, Anand Subramoney, Arne Roennau, and Rüdiger Dillmann. Scaling up liquid state machines to predict over address events from dynamic vision sensors. *Bioinspiration &amp Biomimetics*, 12(5):055001, sep 2017. doi: 10.1088/1748-3190/aa7663. URL `https://doi.org/10.1088/1748-3190/aa7663`.

W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.

Francesco Martinuzzi, Chris Rackauckas, Anas Abdelrehim, Miguel D. Mahecha, and Karin Mora. Reservoircomputing.jl: An efficient and modular library for reservoir computing models, 2022. URL `https://arxiv.org/abs/2204.05117`.

Nils Schaetti. Echotorch: Reservoir computing with pytorch. `https://github.com/nschaetti/EchoTorch`, 2018.

Benjamin Schrauwen, Marion Wardermann, David Verstraeten, Jochen J. Steil, and Dirk Stroobandt. Improving reservoirs using intrinsic plasticity. 71(7):1159–1171, 2008.

Jochen J Steil. Online reservoir adaptation by intrinsic plasticity for backpropagation–decorrelation and echo state learning. *Neural networks*, 20(3):353–364, 2007.

Peter Steiner, Azarakhsh Jalalvand, Simon Stone, and Peter Birkholz. Pyrcn: A toolbox for exploration and application of reservoir computing networks, 2021.

David Sussillo and L. F. Abbott. Generating Coherent Patterns of Activity from Chaotic Neural Networks. 63(4):544–557, 2009.

Nathan Trouvain and Xavier Hinaut. Canary song decoder: Transduction and implicit segmentation with esns and ltsms. In *International Conference on Artificial Neural Networks*, pages 71–82. Springer, 2021.

David Verstraeten, Benjamin Schrauwen, Sander Dieleman, Philemon Brakel, Pieter Buteneers, and Dejan Pecevski. Oger: modular learning architectures for large-scale sequential processing. *The Journal of Machine Learning Research*, 13(1):2995–2998, 2012.