ReservoirPy: About as chaotic as they come

Kanjonavo Sabud
05.17.2025

[1] Reservoir Computers are a type of RNN Machine Learning Model which have been found to be especially useful for modeling chaotic/dynamic systems. They were first introduced by Jaeger and Haas (2004) as Echo State Networks (ESN), a popular paradigm of Reservoir Computing. Our package, ReservoirPy, is a Python library for defining, training, and using Reservoir Computing (RC) architectures in a few lines. Additionally, ReservoirPy also has many functions to obtain datasets for popular chaotic systems such as the Lorenz1963 and Kuramoto Sivashinsky. Overall, ReservoirPy aims to simplify the process of exploring chaotic systems and simulating them using the Reservoir Computer model of ESN.

[2] I have been working with this package for the past semester in my research project with Professor Ed Ott. As our work involves exploring dynamic systems, this package has been particularly helpful. Therefore, I wanted to explore this package more thoroughly, making it an ideal candidate for this project.

[3]

Age: The first public release (v0.2) appeared in January 2021, making the project over four years old.

Genealogy: ReservoirPy builds on the foundational ESN concept introduced by Jaeger (2001). Predecessor toolboxes include PyESN - a minimal pure-Python ESN implementation - and more structured toolboxes like PyRCN, which adopts a scikit-learn-compliant interface for RCNs.

Contemporaries: Other active libraries (easyesn, rescomp) also target RC tasks, but ReservoirPy stands out for its breadth of learning rules and composability.

Version I installed was: 0.3.13post1 using the command `"reservoirpy.__version__".`

[4] The original authors are Nathan Trouvain, Luca Pedrelli, Thanh Trung Dinh, Xavier Hinaut. However, in the contribution section only two email links for Xavier Hinaut and Paul Bernard are given which indicates that ReservoirPy is actively maintained by the Inria research group but only by them. The most recent commit on the source code git was only a couple weeks ago.

Contribution guidelines like bug reports, feature suggestions, and documentation improvements are detailed in the Developer Guide on Read the Docs. There is detailed information on how to submit a pull request, the proper documentation that must be provided.

[5][6]

```
pip install reservoirpy
```

Ease: Installation via PyPI is immediate, requiring only Python 3.7+ and no special compilation. Very easy installation procedure requiring only the standard pip command.

[7] The full source lives on GitHub at reservoirpy/reservoirpy, where you can browse the Python modules, CI pipelines, and examples folder

[8] Yes, I was able to find a package which uses the ReservoirPy package - Conn2Res. Conn2Res is a toolbox which requires reservoirpy for its use (included in requirements.txt). This package is part of the same domain of Reservoir Computing but extends the reservoirpy package by exploring how the model can be used to implement biological neural networks as artificial neural networks and explore the role of connections in the human brain.

[9] All interactions occur through Python scripts or Jupyter notebooks; no CLI or web UI is bundled, though tutorials provide Colab links. Users define Node objects (reservoirs, readouts), chain them, and call .fit()/.run() methods to train and predict.

[10] [12]

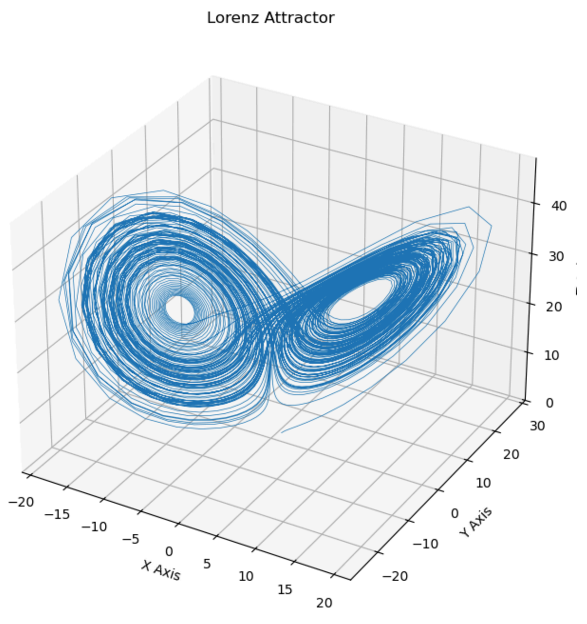
```
%pip install reservoirpy

from reservoirpy.datasets import lorenz

n_timesteps = 5000
data = lorenz(n_timesteps)

// created this method on my own and included in jupyter notebook
plot_lorenz_attractor(data)
```

OUTPUT:



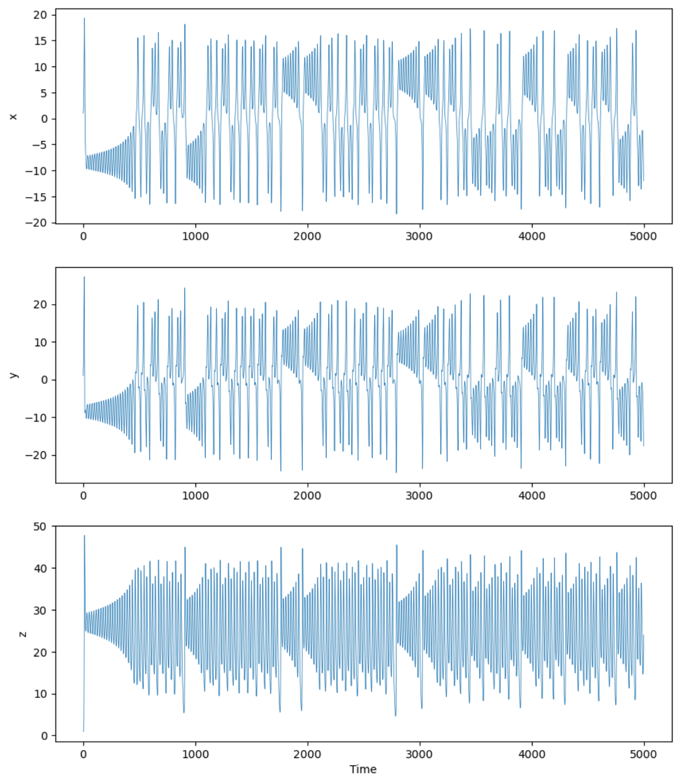


Figure 1 (left): shows the trajectory lorenz 1963 system in the 3d plane starting from point 1,1,1.

Figure 2 (right): shows the trajectory of individual x,y,z variables over time based on the lorenz1963 system.

(Lorenz 1963 is a very popular system in the domain of chaos. It is governed by 3 partial differential equations. We will not go deeply into the system but for our purposes a high level level explanation of the system would be that it is governs the trajectory of a particle in the 3D space over time based on the prior mentioned partial differential equations)

More CODE (SHOWS HOW TO TRAIN MODEL):

```
train_len = 4000
test_len  = 1000

train_data = data[:train_len]
test_data  = data[train_len : train_len + test_len + 1]

X_train = train_data[:-1]
X_test = train_data[1:] + [[0, 0, 0]]

reservoir = Reservoir(300, lr=0.5, sr=0.9)
ridge = Ridge(output_dim=3, ridge=1e-7)

esn_model = reservoir >> ridge

esn_model = esn_model.fit(X_train, X_test)
predicted_data = esn_model.run(test_data)
```

[11] ReservoirPy itself does not include plotting utilities; users rely on Matplotlib (and Seaborn for hyperopt reports) for visualization. This can be seen being done in the `plot_lorenz_attractor(data)` method.

[13][18]

Language: Pure Python (100% Python code). No C/C++/Fortran code needed.

By looking at its requirements.txt and setup.py files on Github we see that it mainly uses numpy (array operations), scipy (sparse matrices, and linear algebra), and tqdm (progress bar). It also makes heavy use of solve_ivp to obtain the datasets for different chaotic systems by solving their associated differential equations. Extras include hyperopt for hyper-parameter search and scikit-learn for interoperability.

[14] [15]

For obtaining datasets (shown in example), only the time range (n_timesteps) is necessary as input. The output to that function (lorenz) will be the dataset for the respective chaotic system (for lorenz it will be x,y,z coordinates at discrete time intervals).

For training the model the input will be Numpy arrays, either loaded via reservoirpy.datasets or some user's own customized solver function which will be used to train the model. The output will be the arrays of predictions; metrics returned as scalars. This is the standard for any ML model architecture and ReservoirPy follows it pretty strictly.

[16] They have an observable tab in their API guide which contains all the metrics (Mean Squared Error, Reduced Mean Squared Error, R square score, and even a Memory Capacity checker - an essential part of the Reservoir Computing Model). These metrics can be used to check the capability of the model in modeling a system. For checking the integrity of the model algorithm one would have to check the source code in the git repository. Similarly, for checking the integrity of the datasets, we would have to compare the ReservoirPy dataset with our own differential equation solver's results on the chosen system. There is still work being done in the benchmarking section as the contributing comments encourage developers to add to the comparison and benchmarking section.

[17] I am very confident in ReservoirPy's code because it was recommended by project director, Professor Ed Ott. Technical checks have also shown that the dataset for Lorenz that I received from ReservoirPy was approximately the same (numerical error always exists) as my solver_ip's solution to the Lorenz system. There is also no hidden code, and the package has been peer reviewed, not to mention that it is maintained by a reputable research group.

[19] They gave very good API documentation which included all of the classes and method headers, their argument descriptions and example code for basic usage. Overall, it took only a little experimentation to understand the workings of the chosen methods so I would say it was pretty easy to use.

[20] Yes they gave the following citation format:

> "Nathan Trouvain, Luca Pedrelli, Thanh Trung Dinh, Xavier Hinaut. ReservoirPy: an Efficient and User-Friendly Library to Design Echo State Networks. 2020. ⟨hal-02595026⟩ https://hal.inria.fr/hal-02595026"

[21] Jaeger and Haas (2004) paper introducing ESN (most popular Reservoir Computing paradigm) https://www.science.org/doi/10.1126/science.1091277

[22]

A paper which describes their toolbox conn2res and its usages (https://www.nature.com/articles/s41467-024-44900-4#author-information) cites reservoirpy. Another paper (https://www.sciencedirect.com/science/article/pii/S0957417424017238) uses the package to conduct its own exploration of optimizing reservoir computing methods. Google scholar shows a count of 80 total citations on the package paper (https://link.springer.com/chapter/10.1007/978-3-030-61616-8_40).

[23] No, I think the class did a pretty good job in teaching me the tools enough to make productive use of this package (descriptive graphs for the Lorenz1963 system). I did, however, learn how to make a 3d graph using the `projection="3d"` argument.

[24] I had prior experience as mentioned in the "why I chose this package" question. I also collaborated with Vaibhav Vasudevan and Isabella Pereira on this project but had my own example code and answers.