Supplementary Material: *ReservoirPy*: an Efficient and User-Friendly Library to Design Echo State Networks

Nathan Trouvain[1,2,3][0000−0003−2121−7826],
Luca Pedrelli[1,2,3][0000−0002−4752−7622],
Thanh Trung Dinh[1,2,3][0000−0003−0249−2080], and
Xavier Hinaut[1,2,3,*][0000−0002−1924−1184]

[1] INRIA Bordeaux Sud-Ouest, France.
[2] LaBRI, Bordeaux INP, CNRS, UMR 5800.
[3] Institut des Maladies Neurodégénératives,
Université de Bordeaux, CNRS, UMR 5293.
*Corresponding author: xavier.hinaut@inria.fr

1 Supplementary Material

1.1 *ReservoirPy* installation

ReservoirPy can be installed easily with `pip`[4]:

```
git clone https://github.com/neuronalX/reservoirpy
# Simple installation
pip install reservoirpy
# Developper installation
pip install -e <path/to/reservoipy/root/directory>
```

1.2 Echo State Network architecture

Echo State Networks (ESNs) are a class of Recurrent Neural Networks (RNNs) implemented according to Reservoir Computing (RC) paradigm. Figure 1 shows an example of ESN architecture. It is composed by a recurrent layer called *reservoir* and an output layer called *readout*. The reservoir is randomly initialized and left untrained, while, the readout weights are trained through *offline* learning or *online* learning. Please, see [2,4] for more information about ESN model.

[4] If you want to be able to modify the code of reservoirpy, debug it (hopefully it would not be necessary), or extend it, use the '-e' option. You will be able to inspect the code during execution with any debugger, and changes will be applied without any need to reinstall the package.

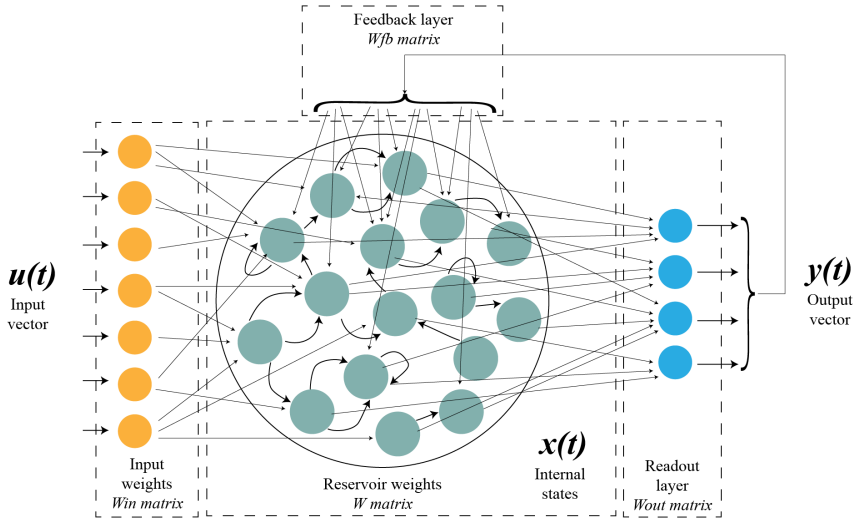

Fig. 1. Schematic representation of an ESN. For each time step t, an input vector $u(t)$ is fed to the model through the input matrix Win. The internal states vector $x(t)$ results from the inner dynamics of the reservoir and their reaction with the input data. A matrix $Wout$ then compute a readout from the state vector, to produce the $\hat{y}(t)$ output vector. Optionally, this vector can then be fed back to the reservoir, as a feedback vector for the next update of internal states with the vector $u(t+1)$.

1.3 Online learning: Details on FORCE learning

With FORCE learning[5], the output weights matrix (W_{out}) is updated for each time step, so as to keep prediction error as small as possible. The update of W_{out} is governed by equation (1), where $e\_(t)$ is the difference between the prediction output and ground truth at time t (i.e. prediction error), $r(t)$ is the state vector (of the reservoir) and $P(t)$ is computed via equation (2).

$$W_{out}(t) = W_{out}(t - \Delta t) - e\_(t)P(t)r(t) \tag{1}$$

$$P(t) = P(t - \Delta t) - \frac{P(t - \Delta t)r(t)r^T(t)P(t - \Delta t)}{1 + r^T(t)P(t - \Delta t)r(t)} \tag{2}$$

1.4 Display the results

After the end of the random search, all results can be retrieved from the report directory and displayed on a scatter plot, using the `plot_opt_results` function

[5] FORCE is a 2^{nd} order learning method similar to RLS (Recursive Least Squares), contrary to LMS (Least Mean Squares) which is 1^{nd} order.

(fig. 2). This function will load the results and extract the parameters and additional metric the user wants to display, specified with the `params` and `metric` arguments. Other parameters can be used to adjust the figure rendering, for instance by removing outliers or switching scales from logarithmic to linear.

```
fig = plot_opt_results("examples/report/hpt-mg"),
params=["sr", "leak", "ridge"],  metric="rmse")
```

In this example, we use the MSE as loss metric and the RMSE as additional metric for display. The default behaviour of the function is to use loss as metric. Every plot in the figure show the interaction between each couple of parameters, weighted by the normalized loss value (gradient of color) and the normalized additional metric (size of dots). The plots displayed on the diagonal of the figure display the relation between the loss function and the parameters, with the top five percent of trials, regarding to the additional metric, displayed in shades of green. The plot given as example display interesting results: the loss function have a convex profile, and variations in dots density in cross parameters scatter plots indicate acceptable ranges of parameters, for both spectral radius and leaking rate. The regularization coefficient does not seem to play an important role in model performance, and can therefore be fixed to a constant value for further explorations. Violin plots at the bottom of the figure can then help with choosing a range of acceptable parameters, by displaying the distribution of the top 5 percent of trials parameters regarding to the additional metric.

1.5 Example of random search visualization on the Canary dataset

The following plot was made using the same tool presented in 4.4. A random search is performed to find optimal ranges of parameters for a classification task over acoustic data representing canary songs. This data is fed to the ESN as two different vectors of features: a first vector represents the first order derivatives of an MFCC signal extracted from the acoustic data, and the second vector the second order derivatives of this MFCC signal. Each of these feature vectors have their own input scaling parameter, respectively `isd` and `isd2`. This visualization allows to quickly distinguish the best parameters range to use. Importantly, it gives insights on interactions between the two input scaling parameters used for the task and the leaking rate.

1.6 Ongoing and Future Work

In the future, there is several other features we want to include in *ReservoirPy*: more use-case examples (e.g. generative mode, hyperparameter search for more tasks, ...); more online learning methods (e.g. LMS); GPU computations (e.g. CuPy, JAX, ...); offline batch computation; batch direct approach for ridge

regression[6]; framework to build layers of reservoirs (e.g. deep reservoirs [1], hierarchical-task reservoirs [5]); Conceptors [3]; scikit-learn API compatibility.

References

1. Gallicchio, C., Micheli, A., Pedrelli, L.: Deep reservoir computing: a critical experimental analysis. Neurocomputing **268**, 87–99 (2017)
2. Jaeger, H.: The "echo state" approach to analysing and training recurrent neural networks. Tech. Rep. 148, GNRCIT GMD, Bonn, Germany (2001)
3. Jaeger, H.: Controlling recurrent neural networks by conceptors. arXiv preprint arXiv:1403.3369 (2014)
4. Jaeger, H., Lukoševičius, M., Popovici, D., Siewert, U.: Optimization and applications of echo state networks with leaky-integrator neurons. Neural Networks **20**(3), 335–352 (Apr 2007)
5. Pedrelli, L., Hinaut, X.: Hierarchical-task reservoir for anytime POS tagging from continuous speech. In: IJCNN (2020)

[6] An approach to incrementally compute the normal equations matrices in ridge regression. This allows the learning algorithm to compute the readout weights by saving memory in the case of large datasets.

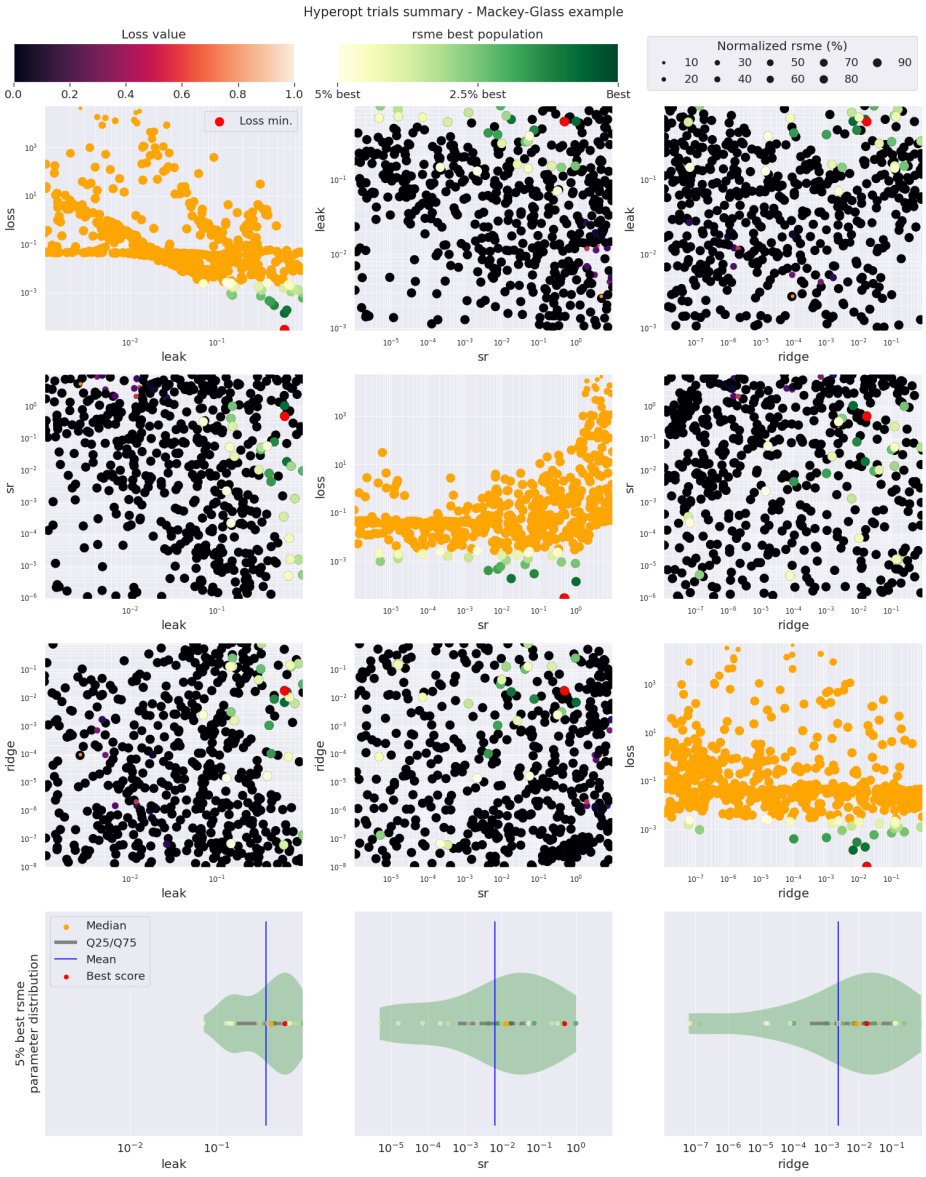

Fig. 2. (Same as the figure in main paper, here for comparison.) An example of figure obtained after 1000 trials over Mackey-Glass time series. The random search was performed on spectral radius (sr), leaking rate (leak) and regularization parameter (ridge). MSE and RMSE are displayed as evaluation metrics. Each trial point represent the averaged evaluation metrics over 10 sub-trials. Each sub-trial was performed on the same parameters combination within each trial, but with different ESN instances (e.g. different random weights initialization).

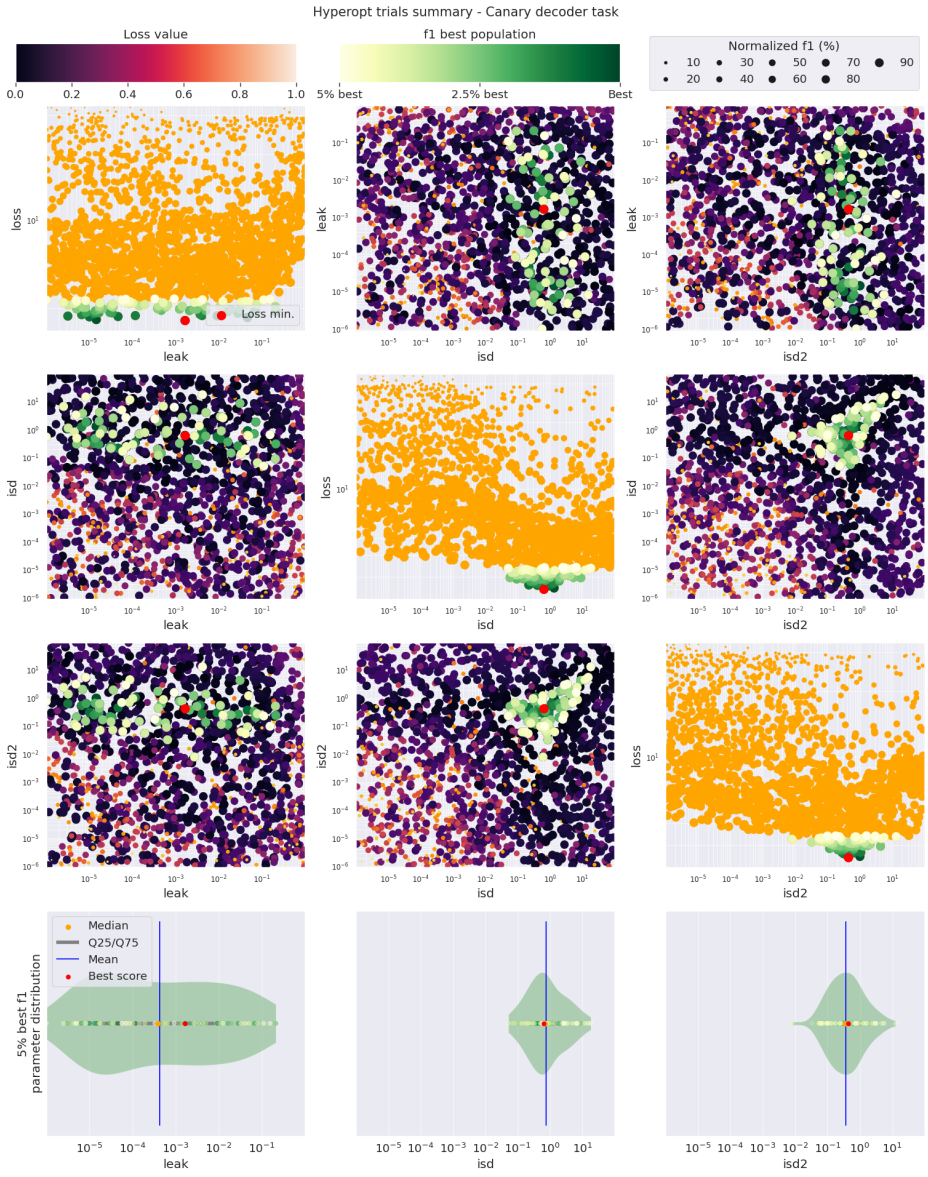

Fig. 3. An example of figure obtained after 1000 trials over the Canary dataset. The random search was performed on leaking rate (leak) and input scaling coefficients (isd and isd2), used to adjust two different sets of features. Cross-entropy and F1-score are displayed as evaluation metrics. For this experiment, the number of units was constant (N = 300), as the spectral radius (sr = 0.4) and the regularization coefficient (ridge $= 1.10^{-7}$).