HAL
open science

Which Hype for my New Task? Hints and Random Search for Reservoir Computing Hyperparameters

Xavier Hinaut, Nathan Trouvain

To cite this version:

Xavier Hinaut, Nathan Trouvain. Which Hype for my New Task? Hints and Random Search for Reservoir Computing Hyperparameters. ICANN 2021 - 30th International Conference on Artificial Neural Networks, Sep 2021, Bratislava, Slovakia. hal-03203318v2

HAL Id: hal-03203318

https://inria.hal.science/hal-03203318v2

Submitted on 15 Dec 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Which Hype for my New Task? Hints and Random Search for Echo State Networks Hyperparameters

Xavier Hinaut[1,2,3,*][0000−0002−1924−1184] and Nathan Trouvain[1,2,3][0000−0003−2121−7826]

[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

Abstract. In learning systems, hyperparameters are parameters that are not learned but need to be set a priori. In Reservoir Computing, there are several parameters that needs to be set a priori depending on the task. Newcomers to Reservoir Computing cannot have a good intuition on which hyperparameters to tune and how to tune them. For instance, beginners often explore the reservoir sparsity, but in practice this parameter is not of high influence on performance for ESNs. Most importantly, many authors keep doing suboptimal hyperparameter searches: using grid search as a tool to explore more than two hyperparameters, while restraining the spectral radius to be below unity. In this short paper, we give some suggestions, intuitions, and give a general method to find robust hyperparameters while understanding their influence on performance. We also provide a graphical interface (included in *ReservoirPy*) in order to make this hyperparameter search more intuitive. Finally, we discuss some potential refinements of the proposed method.

Keywords: Reservoir Computing · Echo State Networks · Hyperparameters · Random Search · Grid Search · Effective Spectral Radius

1 Introduction

1.1 Disclaimer

This short paper aims to give some keys to reservoir newcomers to know how to set hyperparameters. It does not aim to be comprehensive like [12] or to discuss studies on hyperparameter search, but rather to give a few useful practical hints to always keep in mind. The main originality lies in the general method proposed, supplemented with ReservoirPy graphics, applied on two common tasks.

1.2 You have a new task

Suppose you have already some practical experience with Reservoir Computing (RC) [13], in particular with Echo State Networks [8]. For instance, you read the

nice guide of Lukoševičius [12] and you want to apply RC on a new task. How do you choose your HyperParameters (HPs) for that new task (for which you have no assumption of which HP values would work)?

If you only want good results and do not want to know how the HPs influence the performance of your task, you can use an optimizer (e.g. a Bayesian [17] or evolutionary one [1]) to set your HPs once and that's it. The problem is that if you want to know if you have found robust HPs with your optimized search, you will run the optimizer a few times more and will probably end up with several sets of HP values that are all different. Unfortunately, making an average of values for each HP is probably not a good idea given the interdependencies between HPs (see subsection 2.6). In the end, you would probably want to know a little more on the robustness of the HPs you found. Moreover, knowing which HPs are important will help you to adapt them if needed. For example if at some point you obtain more diverse data (e.g. on a classification task) and you would like to increase the reservoir size to gain performance. You will need to perform again the optimization but this time with more data and an increased size of reservoir: this will cost you much more time to optimize.

Conversely, you want to find good HPs for your task, and this time you are willing to know more about the robustness of the HP values found and understand which HPs are the most important for your task. Thus, you will probably do a grid search because that's what other people usually do, and because it provides you nicely discretized maps showing gradients of performance depending on your HPs. The problem is, by doing this, you will probably miss some of the best performances you could expect. Moreover, it will cost you a lot of computational time, much more than what you would need with an optimizer: because by exploring v values for each of your p HPs, you will need to perform p^v evaluations[4].

Thus, what should you do to disentangle this seemingly impossible trade-off? This is what we will argue in this paper. First, we discuss theoretical aspects of reservoir computing and interpret their impact on practical purposes while explaining why grid search is not a good idea. Then, we provide a general practical method for hyperparameter exploration along with graphical tools. Afterwards, we show how this method applies to two tasks of time series prediction. Finally, we discuss some potential refinements of the proposed method.

2 From theory to practice

2.1 At the edge of chaos

Like other dynamical systems (e.g. cellular automata [9]), reservoirs have rich and "interesting" dynamics at the edge of chaos, or *edge of stability* [2]. For most tasks, you want to have reservoir dynamics with contractive property in order not to have diverging dynamics that could destroy the behavior of your outputs. In other words, contractive property means that the reservoir will forget its current

[4] We call evaluations the training of instances of reservoirs for sets of HP values.

state and inputs after some time. While in chaotic regime small differences in states or inputs are amplified dramatically. Legenstein et al [10] have proposed an interesting computational predictive measure to find these edge of stability dynamics in the HP space. The measure is based on the idea that, in order to generalize well, a reservoir needs a mixture of two properties: (1) the ability to represent similar inputs in a similar way, and (2) the ability to separate inputs. Taking into account the fact that dynamics with contractive properties enable property (1), and chaotic dynamics enable property (2), what we need for a good generalization ability is to have both (1) and (2) at the same time, namely, at the intersection of contractive and chaotic dynamics: at the edge of stability. As they show in their computational study, their predictive generalization measure often overlaps with regions of the HP space with edge of stability dynamics.

That being said, remember that there is no such optimal reservoir dynamics for all tasks. Indeed, the desired dynamics depend heavily on your task. Thus, such computational predictive measures are probably useful only for some class of tasks.

2.2 Why you shouldn't care about the *Echo State Property*

In his seminal paper, Jaeger defines the Echo State Property (ESP) [8]. To simplify shortly, ESP states that a spectral radius (SR) following $SR < 1$ is a theoretical condition to have a contracting system in the absence of inputs. One should remember that originally this theoretical condition is interpolated from linear systems. However, we usually do not use linear reservoirs, as most people use hyperbolic tangent (*tanh*) activation function. Thus, even if refinements of the ESP were made, this condition should not be a *constraining* rule but only a rough guide, because reservoir dynamics depend on the inputs fed; it was shown that for some tasks optimal SR could be superior to 1.

Knowing that ESP is mainly a theoretical guide to set the SR is not enough. If you use reservoir with leaky neurons (i.e. with a *leak rate* inferior to one), then you should also know what the *effective spectral radius* is: it is what you should care about instead of the SR itself. Jaeger et al [3] introduced the notion of *effective spectral radius* for reservoirs with leaky-integrator neurons. The idea is roughly the following: the leak rate has a kind of "cooling" effect on the dynamics, thus for a constant spectral radius if one decreases the leak rate, the dynamics will "slow down": the same happens for the *effective spectral radius*. Consequently, the more your decrease the leak rate, the more you can increase the spectral radius, while still having the ESP.

2.3 The myth of grid search

Often, if people do not tune HPs by hand, they use grid search, and we have done it for too long too. Grid search means that you explore each HP by changing its values in a predefined grid, such as for each possible value of one HP you will have tested all possible values of the other HPs.

Bergsta et al. [6] explained why grid search is suboptimal compared to random search. In Figure 1 we can see a schematic explanation of why random search performs better. As Bergsta et al. [6] pointed out: "Grid search allocate too many trials to the exploration of dimensions that do not matter and suffer from poor coverage in dimensions that are important." Thus, you should prefer random exploration over grid search. One advantage of grid search is to be able to guide intuition by visualizing a map of the performance depending on the HPs values. As we will show in section 3, similar maps could be obtained with random search. Moreover, random search is particularly adapted to reservoir because of the little computation cost of the training compared to other methods based on back-propagation. A good view of the hyperspace can be obtained at low cost.

In fact, grid search wastes evaluations to explore unimportant HPs. Whereas, random search do not waste evaluations because for all HP no value is explored twice (i.e. no need to evaluate the same value several time for one HP in order to make a grid to explore other HPs). With grid search, by exploring v values for each of p HPs, one needs to perform p^v evaluations. While for random search, the number of values explored v is equal to the total number of evaluations.

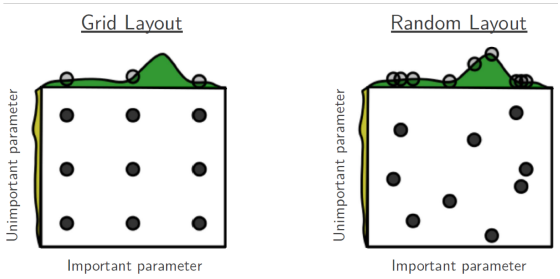


Fig. 1. Random search samples more efficiently than grid search. Image from [6].

2.4 Pieces of advice and hints on HPs choice

Like in other computing paradigms, all HPs do not have the same influence on the performance of your task. The most important HPs, like Lukoševičius suggests [12] are the *spectral radius*, *input scaling*, *leaking rate (or leak rate)*, *number of units in the reservoir* and *feedback scaling* (in case you have feedback from the readout units to the reservoir). If you do not want to explore too many HPs you should care only about them. For instance, newcomers often consider the sparsity of the reservoir to be important, but it is not, in particular if you generate the weights with a Gaussian distribution, many weights will be close to zero in any case. A rule of thumb that we use for sparsity is to keep 20% of non-zero connections in input and reservoir weights matrices, whatever the size of your reservoir is. For most HPs, in particular the ones recommended by

Lukoševičius, one should not use a uniform exploration, but rather a logarithmic exploration. This means that you will be likely to explore low values as much as high values of your searching ranges (with the same sampling at a given order of magnitude). For your first HPs search on a new task, we recommend to use to explore HPs with three levels of magnitude between the smallest value and the highest value (e.g. 1 and 1000).

2.5 A general method with random search

We consider a simple ESN with leaky integrator units, without feedback weights and with readout weights regularization. The main steps of the proposed method are (including two preliminary steps):

- Pre1: Take the minimal reservoir size N you would like to use (you can change N after the HP search)
- Pre2: Fix one of the following HP: IS (Input Scaling), SR (Spectral radius) or LR (Leak Rate) (to avoid conflicting interdependencies)
- A: 1st random search (large range): explore all HP within a wide range of values.
- B1: 2nd random search (narrow range): remove unimportant HPs and narrow the ranges of HP values to the $x\%$ best values (e.g. 10% best values is a good choice when doing more than 100 evaluations.).
- B2: Choose median or best HP values (but the ridge) depending on *loss vs. HP* cloud shape
- C: (optional) Check robustness of previously fixed HPs (in $2nd$ step) and adapt if needed
- D1: (optional) Explore the trade-off between loss and computation time when varying the reservoir size N
- D2: Find best regularization parameter (ridge) for chosen N
- E: Evaluate test set on chosen HPs for 30 reservoir instances

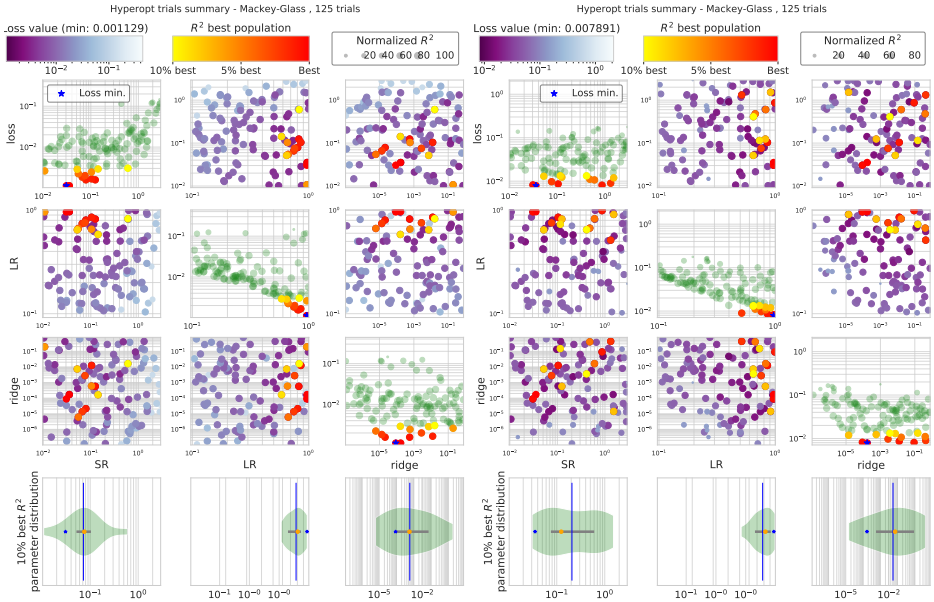
2.6 *ReservoirPy* HP interdependency plot

The proposed method uses HP interdependency plots in order to have a visual evaluation of the loss obtained as a function of all HP explored along with the interactions between HPs. Implementation details are provided here[5].

3 An illustrated example

We will make the exploration for two well known tasks of chaotic time series prediction: Mackey-Glass [14] and Lorenz [11] time series prediction. Both time series where generated using *ReservoirPy* library, and details about their characteristics can be found in the documentation.

[5] https://hal.inria.fr/hal-03203318/document



(a) Lorenz task. *(b)* Mackey-Glass task.

Fig. 2. (a) Small search dependence plot for Lorenz task. Two different patterns of interaction between error (or loss) and parameter value can be observed : the U-shaped clouds (top-left scatter plot and bottom right scatter plot) and the skewed-linear clouds (middle scatter plot). (b) Small search dependence plot for Mackey-Glass task. Here, if the dependence to the leak rate seems to be the same as in the Lorenz task, we can see that the SR has little influence on the error value, unlike in the Lorenz task, and is therefore task dependent.

For the other parameters that need to be set we do as follows. We set the *washout* (i.e. number of timesteps ignored at the beginning) to the size of the reservoir N. After N timesteps, it is indeed expected that all neural nodes in the reservoir graph have received signals from inputs and surrounding units. The random seeds used to initialize the 5 instances of the model used during cross validation are kept fixed for all evaluations. We finally fixed the reservoir size N to 100, to shorten the computation time during the searches.

We chose 5 parameters to explore: *spectral radius* (SR), *leak rate* (LR), *connectivity probability of W* (W proba), *connectivity probability of* W_{in} (W_{in} proba), and *ridge* regularization parameter. We included known "unimportant" HPs like the connectivity probability of matrices in order to illustrate how to deal with these kinds of parameter throughout the method we propose. We also chose to fix at least one of the more important HPs, to reduce the complexity of the search: IS will be kept constant and equal to 1 during this first step.

If we would have liked to do a grid search on 5 HPs and limit our search to about 100 evaluations, we could have taken 3 values per HP for a total of

$5^3 = 125$ evaluations. For grid search exploring only 3 values is very small, that is why exploring more than 2 HPs with grid search is too computationally expensive. Whereas for random search, 125 evaluations is enough to have a first idea of the good HP ranges. Taking 25 evaluations per HP explored is a good rule of thumb.

3.1 1st random search

For both tasks, we used a "task agnostic" range of HPs to be explored (with the type of distribution chosen *a priori*, specified in parentheses): SR = [10^{-2}, 10^{1}] (log-uniform); LR = [10^{-3}, 1] (log-uniform); W proba = [10^{-2}, 1] (uniform); W_{in} proba = [10^{-2}, 1] (uniform); ridge = [10^{-3}, 10^{-9}] (log-uniform). IS is kept fixed and equal to 1. A minimum error of 1.2×10^{-3} is obtained at this stage of the evaluation for the Lorenz task, and 7.8×10^{-3} for the Mackey-Glass task[6].

3.2 2nd random search

We can then use this information to redefine the range of possible values of parameters. One has to include the x% best results inside this range. In other words, the new range must include previous x% best values, with an error margin set depending on the trend observed in the influence of the parameter over performance, e.g. if it seems that expanding a bit the range towards positive side of a parameter axis would allow to explore possibly interesting values, we recommend expanding it. With 125 evaluations, $x = 10$ seems to be a good compromise. Also, one can fix parameters that seem to have no particular influence on the model performance (e.g. sparsity of W and W_{in}[7]). We can therefore fix these parameters to a specific value for the rest of the exploration. In both tasks, we chose to fix W proba and W_{in} to the median value of the top $x = 10$% trials.

Lorenz task For Lorenz task, we redefined the range of HPs as follows: SR = [10^{-2}, 3] (log-uniform); LR = [10^{-1}, 1] (log-uniform); W proba = 0.6 (fixed); W_{in} proba = 0.5 (fixed); ridge = [10^{-7}, 1] (log-uniform).

Mackey-Glass task For Mackey-Glass task, we redefined the range of HPs as follows: SR = [10^{-2}, 3] (log-uniform); LR = [10^{-1}, 1] (log-uniform); W proba = 0.6 (fixed); W_{in} proba = 0.5 (fixed); ridge = [10^{-6}, 10] (log-uniform). Figure 2 summarizes the results of this second random search for both tasks.

3.3 Best values obtained

During this second stage of search, the minimum error reached for Lorenz task lowered to 1.1×10^{-3}, while the minimum error reached for Mackey-Glass remained close to the value obtained during the large search.

[6] A figure summarizing the results of this first random search on the Lorenz time series prediction task is available here: https://hal.inria.fr/hal-03203318/document

[7] W proba and W_{in} proba have little influence on the error distribution (diagonal plots) and no linear dependence with the other parameters can be seen on figure available here: https://hal.inria.fr/hal-03203318/document

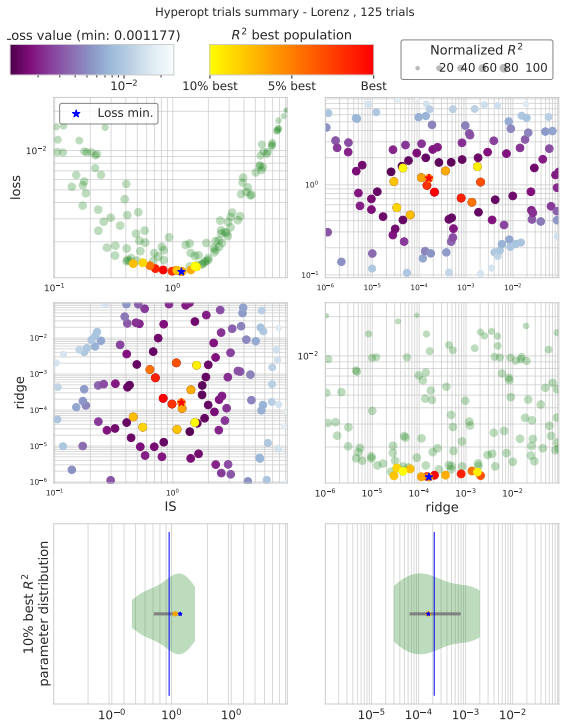


Fig. 3. Restricted search on the IS parameter, for the Lorenz task. With a value of 1, we can see that the IS yields the best results and reaches the minimum of the U-shaped error cloud that can be seen in the top-left plot.

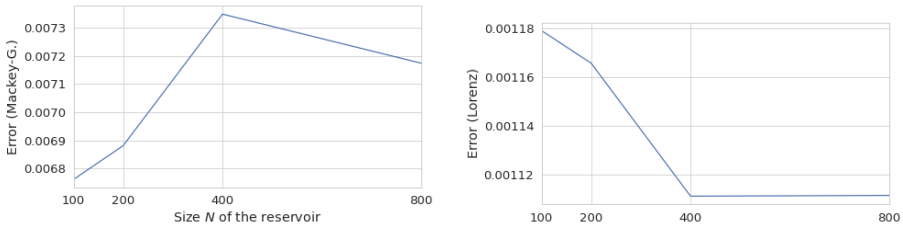


Fig. 4. Error obtained with the best ridge for sizes of reservoir in $\{100, 200, 400, 800\}$, on the Mackey-Glass task (left) and Lorenz task (right). Note that variations in error are really small for both tasks.

Looking at the graphs, we select values given the shape of the estimated distribution of the error metrics. Two patterns generally arise: the U-shape clouds and the skewed clouds. For the parameters provoking U-shaped clouds of error, we set the value of the parameter to the median of the $x = 10\%$ best values,

e.g. $SR = 7.10^{-2}$ for the Lorenz prediction task, as observed in figure 2. For the HPs clearly skewed with a linear trend (still on a logarithmic scale), we take the best parameter found, e.g. $LR = 0.98$, in the case of the Lorenz prediction task. We also double checked this values on the coupled parameters plots around the diagonal of plots, to make sure that no picked value would poorly interact with another parameter. We can also see in figure 2 that the spectral radius seems to play a not so important role in the Mackey-Glass task. The error distribution regarding to this parameter is quite flat, with unclear dependence compared to the Lorenz task. This shows that the SR can be a task dependent parameter, and be robust against variations for a wide range of values.

Lorenz task For Lorenz task, using the method described above, we finally chose the following parameters: SR = 7.10^{-2} (fixed); LR = 0.96 (fixed); W proba = 0.6 (fixed); W_{in} proba = 0.5 (fixed); ridge = $[10^{-6}, 10^{-1}]$ (log-uniform).

Mackey-Glass task For Mackey-Glass task, we finally chose the following parameters: SR = 0.12 (log-uniform); LR = 0.97 (log-uniform); W proba = 0.6 (fixed); W_{in} proba = 0.5 (fixed); ridge = $[10^{-6}, 10]$ (log-uniform).

The ridge was not fixed until the end of the process. We only kept narrowing its explored range. Regularization parameters should not be fixed until all other parameters are found: they are meant to optimize the model regarding to its complexity, in our case regarding to the number of neuronal units of the reservoir, the parameter N. We assess the impact of this parameter on the performance in section 3.5.

3.4 Checking the input scaling

Before evaluating the model using the parameters found, we recommend assessing the interdependence of the chosen parameters and the parameters kept constant during evaluation. In our case, the input scaling (IS) was fixed to 1, but as it controls the amplitude of the signals fed to the reservoir, it can have great influence on the model performance. We performed a simple check consisting in defining a range of one or two order of magnitude around the fixed value of the IS, and running a random search on this range. The ridge will also be kept variable, as it has not been fixed yet. Results of this evaluation for the Lorenz task can be found in figure 3 and on figure 2 for the Mackey-Glass task. Although it appears that a maximum of performance can be reached with a fixed value of 1 in the case of the Lorenz task, with this value placed just below the minimum of the error, we can see that the distribution of error is skewed towards the left of this value in the case of the Mackey-Glass task. We therefore redefine the IS value for Mackey-Glass to 0.3.

3.5 Searching for the best ridge

Finally, we propose to explore the dependence of the ridge on the reservoir size N. We explored the ridge in a range of $[10^{-5}, 10^{-1}]$, and used four different reservoir sizes N = $\{100, 200, 400, 800\}$. By selecting the most adapted ridge for each size, i.e. the ridge corresponding to the minimum error produced with each

reservoir size, we can see the relation existing between the error, the ridge and the reservoir size in figure 4. Using this information, one can then select the pair {ridge, N} that suits the most their needs and resources in terms of performance and computational power. Assuming that we have enough computational power to use a 800 neurons reservoir at least, we select the final values for ridge and N:

Lorenz task Because a size of 400 seems to already give good performance as shown in figure 4, we will keep this value instead of the maximum of 800. We finally chose: N = 400; ridge = 2×10^{-4}.

Mackey-Glass task In the case of Mackey-Glass, we obtained best results with a size of 100, as shown in figure 4. Even if counter-intuitive, this result is probably due to the simplicity of the task performed, or to a low dependency between the reservoir size and the chosen parameters. We finally chose: N = 100; ridge = 5×10^{-4}.

To conclude this experimentation, we evaluated the selected HPs on a test data set, with 30 different random initializations. The test data set is defined as the 2000 time steps following the 6000 time steps used during cross validation. This 6000 time steps are used as training steps during the evaluation. The results obtained are presented in table 1:

Table 1. Results averaged over 30 random instances on the test set with the chosen HPs.

Task	**NRMSE**	$\boldsymbol{R^2}$
Lorenz	4.5×10^{-4} ($\pm 1 \times 10^{-4}$)	0.99999 ($\pm 2 \times 10^{-6}$)
Mackey-G.	9.78×10^{-3} ($\pm 4.56 \times 10^{-3}$)	0.99805 ($\pm 2 \times 10^{-3}$)

Our final results outperformed the previously reached error values in both tasks. However, the difference in performance between previously reached values and final values remains relatively low, and could be attributed to chance. A better evaluation could be done using more folds during cross validation, and more random initializations to avoid any bias from both the data and the initialization. Nevertheless, our method helped finding ranges of HPs able to produce satisfying models in terms of performance, and without relying on too many assumptions and hypothesis, as the ones generally made about SR value for instance. Moreover, we evaluated the model regarding the influence of almost all possible parameters, ensuring robustness of the results.

4 Discussion

We first discussed theoretical aspects of reservoir computing and which impact they should have on practical purposes. Hopefully, we convinced the reader not to perform grid search for more than two hyperparameters (HPs), given the

high number of evaluations that such suboptimal method would require[8]. Then, we provided a general practical method for HP exploration. Finally, we showed how this method applies to the Lorenz and Mackey-Glass chaotic time series prediction tasks, and provided several figures illustrating the influence of HPs on performance. To our knowledge, neither grid search nor other optimizer provide such an amount of information and intuitions for reservoir computing HP exploration. In summary, we proposed a method that is better than a compromise between the graphical intuitions that grid search would provide, and the optimum performance that an optimizer provides. Our method probably uses more evaluations than most efficient optimizers (even if evolutionary methods are generally demanding in terms of number of evaluations). Given the precision of the performance obtained throughout the paper, the number of evaluations used could be reduced.

Our study illustrates that the importance of HPs depends on the task. Indeed, we showed in figure 2 that the spectral radius is more robust to changes (i.e. it offers a wider range of eligible values) for the Mackey-Glass than for the Lorenz task. Moreover, the interdependencies between HPs are visible, for instance on figure 2 for *SR vs. ridge* subplot we can see a red-orange diagonal: if the SR is changed, the ridge has to be adapted, and vice versa[9]. In future work, we would like to integrate these interdependencies within the HP search. For example by defining a HP Y as a linear function of a HP X. Searching the best values for Y would thus be redefined as finding the best values of a and b in the following equation: $Y = a.X + b$.[10] One can argue that this adds more HPs, but one can assume that finding such interdependencies in such a way is less demanding in number of evaluations needed.

The tasks we have chosen are famous tasks for reservoir benchmarking, however they are rather simple to solve, given that a 100-unit reservoir already solves the task well. In future work, we will apply the proposed method to more complex and more computationally demanding tasks which would not enable hundreds of evaluations to be performed. Indeed, in one of our recent study [7] we used parts of the method proposed here. We applied it to a more difficult task (i.e. language processing) and among other things, in supplementary material of [7] we showed the effect of the regularization on a *loss vs. reservoir size* plot. Given the point-clouds drawn, we could see the best loss given the reservoir size both with optimal regularization and without any regularization at the same time.

The method we proposed in this study is probably applicable to other machine learning paradigms than RC. Thus reducing the time and computational resources needed for HP search: this is a step towards more environment-friendly methods. In one of our recent studies [16], we explored a new way of viewing the influence of HPs. With UMAP visualizations (Uniform Manifold Approximation

[8] With grid search, by exploring v values for each of p HPs, one needs to perform p^v evaluations. While for random search, v is equal to the total number of evaluations.

[9] For interdependency for *IS vs. ridge* see https://hal.inria.fr/hal-03203318/document

[10] Of course as we plot many variables with log scales, the equation would often look like $log(Y) = a.log(X) + b$.

and Projection [15]) we explored the influence of HPs on the internal dynamics of the reservoir. For instance, increasing the spectral radius or the leak rate is fragmenting the UMAP representations of the internal states.

Finally, we would like to enjoin authors to think about simple rules while writing their next paper on Reservoir Computing: (1) please provide all the HPs you have used with the details of how you found them, in order to enable others to have a chance to reproduce your work; (2) demonstrate that the HPs you have found or chosen are robust again perturbations or slight modifications (like in the supplementary material of [7]), instead of just picking the results from a black-box optimization search.

References

1. Ferreira et al., A.: An approach to reservoir computing design and training. Expert systems with applications **40**(10), 4172–4182 (2013)
2. Schrauwen et al., B.: An overview of reservoir computing: theory, applications and implementations. In: Proc. of ESANN. pp. 471–482 (2007)
3. Jaeger et al., H.: Optimization and applications of echo state networks with leaky-integrator neurons. Neural Networks **20**(3), 335–352 (2007)
4. Bergstra et al., J.: Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In: Proc. Scipy. pp. 13–20. Citeseer (2013)
5. Trouvain et al., N.: Reservoirpy: an efficient and user-friendly library to design echo state networks. In: ICANN. pp. 494–505. Springer (2020)
6. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research **13**(Feb), 281–305 (2012)
7. Hinaut, X.: Which input abstraction is better for a robot syntax acquisition model? phonemes, words or grammatical constructions? In: ICDL-EpiRob (2018)
8. Jaeger, H.: The "echo state" approach to analysing and training recurrent neural networks. Bonn, Germany: GNRCIT GMD Tech. Rep. **148**, 34 (2001)
9. Langton, C.G.: Computation at the edge of chaos: Phase transitions and emergent computation. Physica D: Nonlinear Phenomena **42**(1-3), 12–37 (1990)
10. Legenstein, R., Maass, W.: Edge of chaos and prediction of computational performance for neural circuit models. Neural Networks **20**(3), 323–334 (2007)
11. Lorenz, E.N.: Deterministic Nonperiodic Flow. J. Atmo. Sci. **20**(2), 130–141 (1963)
12. Lukoševičius, M.: A practical guide to applying echo state networks. In: Neural Networks: Tricks of the Trade, pp. 659–686. Springer (2012)
13. Lukoševičius, M., Jaeger, H.: Reservoir computing approaches to recurrent neural network training. Computer Science Review **3**(3), 127–149 (2009)
14. Mackey, M.C., Glass, L.: Oscillation and chaos in physiological control systems. Science **197**(4300), 287–289 (Jul 1977)
15. McInnes, L., Healy, J., Saul, N., Großberger, L.: Umap: Uniform manifold approximation and projection. Journal of Open Source Software **3**(29), 861 (2018)
16. Variengien, A., Hinaut, X.: A journey in ESN and LSTM visualisations on a language task. arXiv preprint arXiv:2012.01748 (2020)
17. Yperman, J., Becker, T.: Bayesian optimization of hyper-parameters in reservoir computing. arXiv preprint arXiv:1611.05193 (2016)

5 Supplementary Material

5.1 Implementation details

We used the *ReservoirPy* library [5], which has been interfaced with *hyperopt* [4], to make the following experiments and figures in order to dive into the exploration of HPs. We illustrated our proposed method with two different tasks of chaotic time series prediction. These tasks consist in predicting the value of the series at time step $t+1$ given its value at time step t. To asses the performance of our models on these tasks, we performed cross validation. Each fold is composed of a train and a validation data set. The folds are defined as continuous slices of the original series, with an overlap in time, e.g. the validation set of the first fold would be the training set of the second, and the two sets would be two adjacent sequences of data in time in the series. For the last fold of the time series, the train set is defined as the last available slice of data, while the train set of the first fold is used as validation set.

For all tasks, we used a 3-fold cross validation measure on a time series composed of 6000 time steps, i.e. each fold is composed of 4000 time steps, with a train set and a validation set of 2000 time steps each. We used two metrics to perform this measure: the Normalized Root Mean Squared Error, defined in equation 1, and the R^2 correlation coefficient, defined in equation 2:

$$\mathrm{NRMSE}(y, \hat{y}) = \frac{\sqrt{\frac{1}{N}\sum_{t=0}^{N-1}(y_t - \hat{y}_t)^2}}{\max y - \min y} \tag{1}$$

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{t=0}^{N-1}(y_t - \hat{y}_t)^2}{\sum_{t=0}^{N-1}(y_t - \bar{y})^2} \tag{2}$$

where y is a time series defined over N time steps, $\hat{y}$ is the estimated time series predicted by the model, and $\bar{y}$ is the average value of the time series y over time. NRMSE was used as an error measure, which we expect to reach a value near 0, while R^2 was used as a score, which we expect to reach 1, its maximum possible value. All measures were made by averaging this two metrics across all folds, with 5 different initializations of the models for each fold.

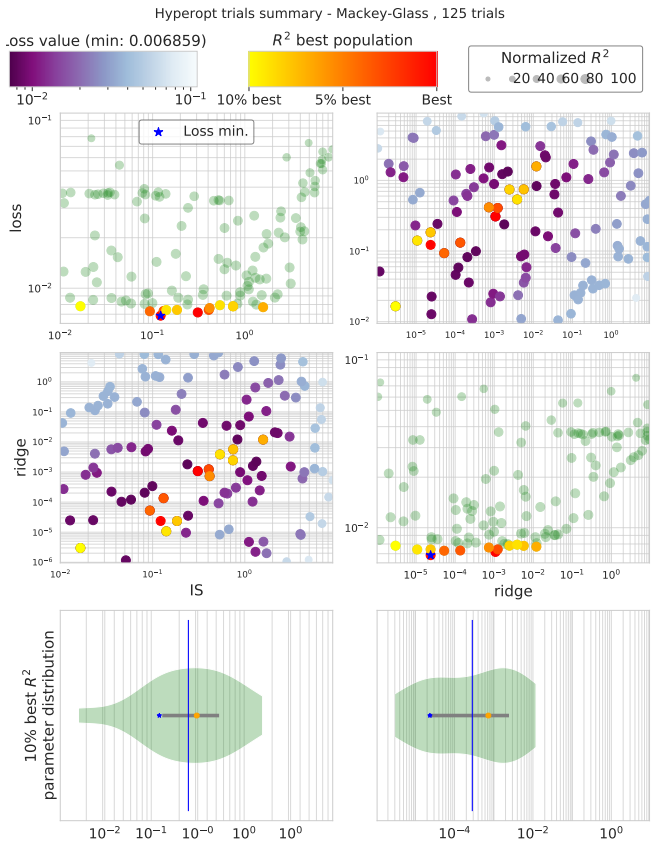


Fig. 5. Restricted search on the IS parameter, for the Mackey-Glass task. The fixed value of 1 defined at the beginning of the search for IS might not be the optimal value given all the other parameters chosen. In the case of the Mackey-Glass task, we can clearly see in the top-left and bottom-left plots that better results are achieved with lower values, with the top 10% trials distribution being placed around a median of 0.3.

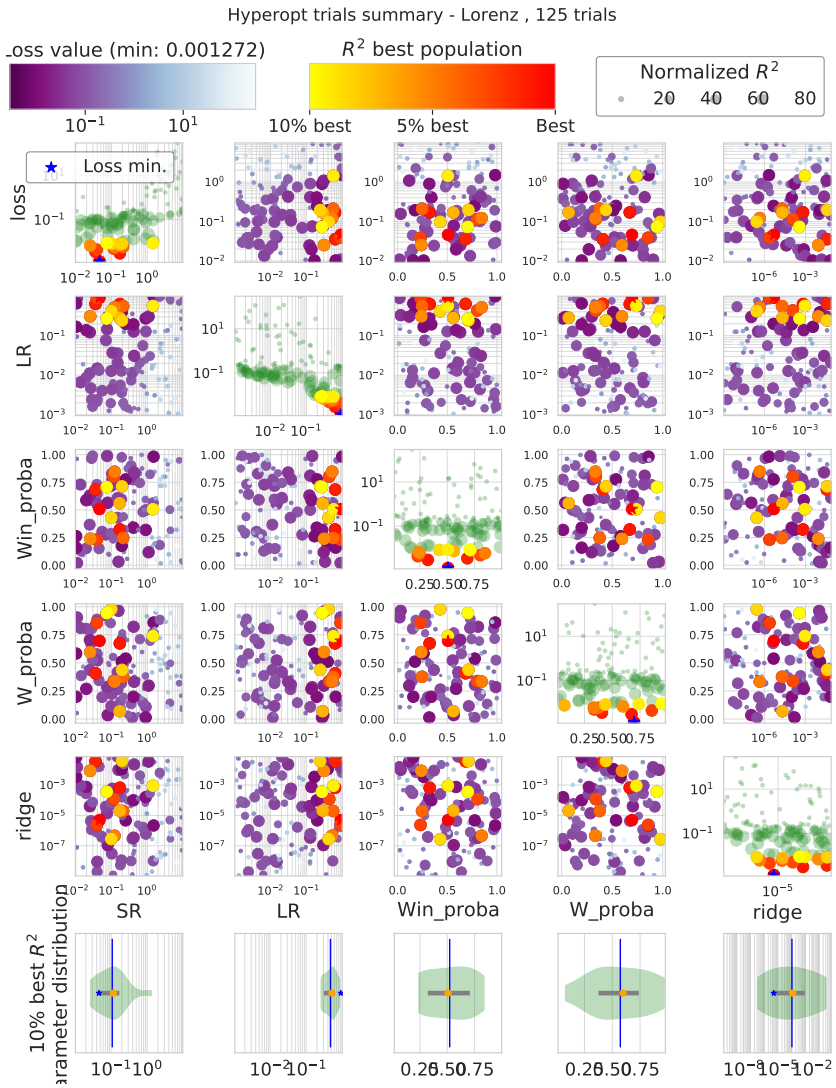


Fig. 6. Large search dependence plot for Lorenz task, with 125 trials. Diagonal of the plot matrix displays interactions between all parameters explored and the error value, also referred to as loss value. Top 10% trials in terms of score (here, R^2) are represented using colors from yellow (top 10) to red (top 1) in all plots. Other plots on the matrix display the interactions of all possible couple of parameters. In these plots, the value of the error is represented using shades of purple and blue, on a logarithmic scale, while score is represented using different circle sizes. Circle size is normalized regarding to the score values. Because R^2 can take values between $-\infty$ and 1, the smallest dots represents negative values. Finally, the bottom row of plots display the parameter distribution of the top 10% of trials, in terms of score, and for each parameter.