EchoBay: Design and Optimization of Echo State Networks under Memory and Time Constraints

L. CERINA and M. D. SANTAMBROGIO, Dipartimento di Elettronica Informazione e Bioingegneria, Politecnico di Milano, Italy

G. FRANCO, C. GALLICCHIO, and A. MICHELI, Department of Computer Science, University of Pisa, Italy

The increase in computational power of embedded devices and the latency demands of novel applications brought a paradigm shift on how and where the computation is performed. Although AI inference is slowly moving from the cloud to end-devices with limited resources, time-centric recurrent networks like Long-Short Term Memory remain too complex to be transferred on embedded devices without extreme simplifications and limiting the performance of many notable applications. To solve this issue, the Reservoir Computing paradigm proposes sparse, untrained non-linear networks, the Reservoir, that can embed temporal relations without some of the hindrances of Recurrent Neural Networks training, and with a lower memory occupation. Echo State Networks (ESN) and Liquid State Machines are the most notable examples. In this scenario, we propose **EchoBay**, a comprehensive C++ library for ESN design and training. EchoBay is architecture-agnostic to guarantee maximum performance on different devices (whether embedded or not), and it offers the possibility to optimize and tailor an ESN on a particular case study, reducing at the minimum the effort required on the user side. This can be done thanks to the Bayesian Optimization (BO) process, which efficiently and automatically searches hyper-parameters that maximize a fitness function. Additionally, we designed different optimization techniques that take in consideration resource constraints of the device to minimize memory footprint and inference time. Our results in different scenarios show an average speed-up in training time of 119x compared to Grid and Random search of hyper-parameters, a decrease of 94% of trained models size and 95% in inference time, maintaining comparable performance for the given task. The EchoBay library is Open Source and publicly available at <https://github.com/necst/Echobay>.

CCS Concepts: • **Computing methodologies** → Supervised learning; • **Computer systems organization** → *Embedded software*; • **Theory of computation** → **Mathematical optimization**; • **Computing methodologies** → *Machine learning*;

Additional Key Words and Phrases: Echo State Networks, EchoBay

ACM Reference format:

L. Cerina, M. D. Santambrogio, G. Franco, C. Gallicchio, and A. Micheli. 2020. EchoBay: Design and Optimization of Echo State Networks under Memory and Time Constraints. *ACM Trans. Archit. Code Optim.* 17, 3, Article 22 (August 2020), 24 pages.

<https://doi.org/10.1145/3404993>

Authors' addresses: L. Cerina and M. D. Santambrogio, Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Via G. Ponzio 34/5, 20133, Milano, Italy; emails: {luca.cerina, marco.santambrogio}@polimi.it; G. Franco, C. Gallicchio, and A. Micheli, Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo, 3, 56127 Pisa, Italy; emails: g.franco4@studenti.unipi.it, {gallicch, micheli}@di.unipi.it.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.

1544-3566/2020/08-ART22

<https://doi.org/10.1145/3404993>

1 INTRODUCTION

The rise of Cloud services and the consolidation of computational resources *at-scale* is without any doubt one of the major technological outbreaks since the dawn of the Internet. However, the amount of data transferred is increasing (e.g., videos and sensors), whilst the latency requirements are falling down, with autonomous robotics and industry 4.0 that should react in real-time. These concerns, along with the congestion of wireless bandwidth and novel efficient processors, pushed for a redistribution of computing resources between the Cloud and endpoints to perform analyses near to the source of data (e.g., Fog Computing [33]).

However, this approach required to transfer algorithms and frameworks already optimized for large-scale systems or energy-hungry GPUs into devices with an exiguous amount of resources. For example, Deep Neural Networks (DNN) and their most widespread embodiment, the Convolutional Neural Networks (CNN), separate training and inference across different devices. While training is computationally intensive and must be performed at high precision levels with Cloud-scale machines, inference can be executed on low-end devices through the reduction of model complexity, weight compression or precision reduction among others [21].

On the other hand, every application that requires the learning of temporal relations, e.g., pattern recognition in time-series, is usually performed using Recurrent Neural Networks (RNN). RNN architectures require more complex neurons, such as Gated Recurrent Units (GRU) [10], which embeds knowledge through time-dependent states. This intrinsic complexity is quite difficult to be transferred onto small constrained devices without architectural changes [31], and the larger amount of trained weights slows down the process, making training unfeasible at the edge.

In this scenario, the Reservoir Computing (RC) paradigm [35, 46] offers valuable alternatives in the RNN field to these complex architectures, with the ESN model [25] representing the most known and used implementation. The RC/ESN approach proposes a peculiar solution: instead of learning a large set of weights in its dynamical (recurrent) component, it exploits a sparse matrix of random and *untrained* weights, which transforms the input stream into a (hidden) state embedding through a highly-dimensional non-linear mapping. This component of the RNN architecture is called *Reservoir*, as it is supposed to provide the overall system with a rich pool of dynamics representing the input streams. After *encoding* the inputs through the Reservoir, the training happens only on the fully connected linear readout layer, which transforms the Reservoir states into the output. The readout training is done with very efficient approaches. More details on ESN structure and equations will be provided in the next sections.

Under this assumption, ESN provides a notable advantage over classical RNN in memory and computational time, even for complex tasks such as polyphonic music predictions [19]. This is noteworthy on embedded devices, where it will be possible to deploy more complex models without increasing computational requirements or drain too much power. The same paradigm applies to large-scale systems, reducing costs in time and hardware capacity.

Unfortunately, as the quasi-randomness of the Reservoir minimizes the free hyper-parameters to be trained, it shifts the abstraction from the single neurons up to high-level parameters such as Reservoir size, sparsity, and topology. All these parameters are, *conceptually*, loosely coupled with input/output relations of the network, and they cannot be trained with usual DNN gradient techniques. The choice of many hyper-parameters typically resorts to the experience of the ESN designer or expensive search methods, limiting the Design Space Exploration to few selected parameters.

In this scenario, we propose **EchoBay**: an efficient and comprehensive C++ library for the design and training of ESN. The main contribution of this article is that the Design Space Exploration is performed using BO controlled by a zero-code configuration file. Compared to other optimization

methods, such as Grid Search, Random Search, and Genetic Algorithms, BO searches and selects the samples in the parameter space that are *expected* to maximize an objective function, vastly reducing the computational and time budget necessary to reach the best configuration.

Furthermore, we decided to design the ESN system from scratch with constrained devices in mind, such as microcontrollers and small board computers, and then scaled-up toward multi-core processors. This methodology enriched the EchoBay runtime options enabling multi-threaded computing, or parallel acquisition of Bayesian samples, and allowed us to reach a high efficiency across different devices.

Other than these architectural solutions, we designed multiple strategies to improve the quality of BO solutions: a pre-computed map of target devices' capabilities provide the upper boundaries of the parameter space, with respect to memory footprint and time constraints. Upon that, we built different rules that penalize the objective function and enforce hard or soft boundaries to the parameters space, reducing the overall size of the model or the inference time.

To summarize, the major contributions of this work are as follows:

- An efficient, portable and modular implementation of ESN core functions.
- A set of semi-automatic code optimizations that maximize ESN performance in constrained systems.
- A Bayesian-based architecture for the Design Space Exploration of ESN hyper-parameters.
- A multifaceted optimization strategy, valuable across multiple targets (eg. x86 CPUs or ARM microcontrollers), which improves the behavior of the Bayesian Optimizer.

Each optimization method was tested separately on different tasks and different ESN topologies, resulting in an average speed-up in training time of 119x compared to Grid and Random search of hyper-parameters, a decrease of 94% of trained models size and 95% in inference time, maintaining first-class performance. The rest of the article is organized as follows. We briefly introduce ESNs characteristics in Section 2, while Section 3 outlines the current state-of-the-art in ESN and RNN. Section 4 presents the EchoBay library and the optimization methodologies. Experimental results are presented in Section 5, and our conclusions are summarized in Section 6.

2 THEORETICAL BACKGROUND: RESERVOIR COMPUTING

Dynamical recurrent neural models are widely adopted Machine Learning tools for dealing with dynamically varying information in the form of time-series. Mathematically, they can be understood as state-space models where the computation is performed by two components. The first is a dynamical *recurrent* hidden layer that encodes the time-series information in a step-by-step fashion into the activation of its (typically non-linear) neurons. On top of that, an output *readout* layer (typically made up of linear neurons) is applied to compute the output of the model. This general kind of RNN architecture is flexible and powerful [29], yet training the weights in an end-to-end fashion typically involves a number of difficulties, which require ad-hoc solutions in the learning algorithm design [39]. An alternative view to the design and construction of RNNs was proposed around the 2000s independently by several research groups. Major instances are given by ESNs [24, 25] in the context of discrete-time evolving RNNs, Liquid State Machines (LSMs) [36] in the context of spiking neural networks, and Fractal Prediction Machines (FPMs) [44] in the field of iterated function systems. Despite the different terminology and formulation, the core idea of all these models is shared and has been later popularized under the name of RC [35, 46]. Essentially, RC drives the research attention to the dynamical nature of the recurrent hidden layer in the architecture, called *Reservoir*. This component can indeed be understood as a dynamical system that traverses the phase space describing trajectories, which directly depend on the external input signals that excite the system. The crucial aspect of RC is that the Reservoir can be left untrained

Fig. 1. Graphical depiction of the ESN architecture and of the involved weight matrices. The dimensions N_r and N_o can be different.

after proper initialization subject to stability constraints. The burden of training, then, only affects the readout component, with a striking computational advantage with respect to fully trained RNNs. Interestingly, RC models exploit the intrinsic ability of recurrent neural architectures to discriminate among different input histories in a suffix-based way even in the absence of training [15, 45]. Recent developments of the RC methodology include deep architectures [18], generalizations to deal with graph data structures [17], and photonic-accelerated hardware implementations [42].

In this article, we focus on the RC model that is the most widely used in applications, i.e., ESN, introduced in the following.

2.1 Echo State Networks

We can formulate the equations of an ESN with leaky feedback integration [26] as follows:¹

$$\begin{cases} \mathbf{x}(t) = (1 - \alpha) \mathbf{x}(t - 1) + \alpha \sigma(\mathbf{W}_{\text{in}} \mathbf{u}(t) + \mathbf{W}_{\text{r}} \mathbf{x}(t - 1)) \\ \mathbf{y}(t) = \mathbf{W}_{\text{out}} \mathbf{x}(t), \end{cases} \quad (1)$$

where $\mathbf{u}(t) \in \mathbb{R}^{N_u}$, $\mathbf{x}(t) \in \mathbb{R}^{N_r}$, and $\mathbf{y}(t) \in \mathbb{R}^{N_o}$ respectively denote the the input vector, the internal (Reservoir) state vector, and the output vector at time t . Here, $\mathbf{W}_{\text{in}}$ is an $N_r \times N_u$ weight matrix that expands the dimensionality of the input and modulates its influence in driving the Reservoir dynamics. Moreover, $\mathbf{W}_{\text{r}}$ is $N_r \times N_r$ highly sparse weight matrix acting as a recurrent matrix of the system, and whose sparsity is determined by the s density parameter, here expressed in the range $(0, 1]$. Additionally, the term $\alpha \in (0, 1]$ represents the *leaky factor* that controls the dynamic of the Reservoir with respect to the variations of $\mathbf{u}(t)$. Lastly, $\sigma(\cdot)$ is a non-linear sigmoid activation function, generally $\tanh(\cdot)$. The readout (second line in Equation (1)) is implemented in a feed-forward layer of linear neurons, where $\mathbf{W}_{\text{out}}$ is a $N_o \times N_r$ weight matrix. Figure 1 graphically illustrates the architectural elements of an ESN.

Given the sparsity of $\mathbf{W}_{\text{r}}$, the N_r dimension and density s define the number of active connections, or the elements with *weight* $\neq 0$ as $R_{\text{conn}} = \lceil s * N_r^2 \rceil$. These connections are mostly equivalent to artificial neurons, and we can see that the complexity of a state update increases roughly linearly with R_{conn} (see Figure 5(a)). Regarding the range of $\mathbf{u}(t)$, one possible downside of ESN is the necessity to rescale them in order to correctly excite the $\tanh$ activation function and avoid saturating effects. On the other hand, $\mathbf{W}_{\text{out}}$ constraints are less stringent and $\mathbf{y}(t)$ can keep its original values.

¹We dropped the bias terms for the ease of notation.

The fundamental aspect of ESNs is that the matrices $\mathbf{W}_{in}$ and $\mathbf{W}_r$ are left untrained after initialization, and only the elements in $\mathbf{W}_{out}$ are adapted supervisedly on a training set. The steps of Reservoir initialization, readout training, and hyper-parameters setting are briefly recalled in the following.

Reservoir Initialization. Due to the high non-linearity of the system, an ESN should respect some theoretical assumptions in order to properly function and avoid divergence of states (resulting from sensitivity to perturbations to the driving input or initialization conditions). In particular, a valid ESN has to satisfy a stability property, the so-called Echo State Property (ESP) [25, 50], according to which the dependence on the initial state is progressively lost and the Reservoir states depend only on the driving input signals.

A common procedure in literature, in case of leaky ESN, is to initialize the Reservoir weights in $\mathbf{W}_r$ randomly from a uniform distribution in $[-1, 1]$, and then re-scale them to control their magnitude. This operation normalizes the effective spectral radius of the Reservoir, defined as:

$$\rho = \max(|\text{eig}((1 - \alpha) \mathbf{I} + \alpha \mathbf{W}_r)|), \quad (2)$$

where $\text{eig}(\cdot)$ indicates the eigenvalues of its matrix argument, and $\mathbf{I}$ is the identity matrix. The spectral radius ρ identifies a key hyper-parameter of the ESN. Although the values considered for ρ are usually below unity, it has been shown that values of $\rho > 1$ can in some cases lead to an increase in performance of the network [14, 46]. As regards the weights in matrix $\mathbf{W}_{in}$, these are randomly initialized as well, with values chosen from a uniform distribution in $[-\omega^{in}, \omega^{in}]$, where ω^{in} is an input-scaling hyper-parameter of the system. Finally, the state of the Reservoir is initialized to zero, i.e., $\mathbf{x}(0) = \mathbf{0}$.

Training. The only set of weights of the ESN that needs to be trained is that pertaining to the readout matrix $\mathbf{W}_{out}$. This operation is typically performed efficiently in closed form, by using ridge-regression [35]. Considering the simple case of a training set composed by an input time-series $\mathbf{u}(1), \dots, \mathbf{u}(T)$, and by an output time-series $\mathbf{y}_{tg}(1), \dots, \mathbf{y}_{tg}(T)$, the Reservoir is driven by the input signals, and the corresponding internal states are collected column-wise into a matrix $\mathbf{X}$. Depending on the application, a certain number of states, called *washout*, is excluded to avoid unwanted transient dynamics in the Reservoir. The target outputs are also collected column-wise into a matrix $\mathbf{Y}_{tg}$. Training of the readout is then performed as follows:

$$\mathbf{W}_{out} = \mathbf{Y}_{tg} \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1}. \quad (3)$$

This training procedure requires another hyper-parameter, λ , which represents the Tikhonov's regularization factor.

Hyper-parameters setting. Considering the randomness associated with the ESN initialization, the appropriate tuning of network's hyper-parameters is fundamental for successful applications. The most common approach in literature [35] is to select the hyper-parameters whose effect is most evident on the final performance of the validation set, then perform a blind grid-search among predefined values for the set of hyper-parameters. At the end of the process, the best performing configuration is the chosen one. However, this procedure is both naive and time-consuming, given the exponential relationship between the number of hyper-parameters and the combinations to be tested. A slightly better approach is Random Search [5]. Given the same computational budget, Random Search can explore a larger space, and it is not limited by the dimension of bins in Grid Search.

However, high computational resources (e.g., compute clusters or time budget) are still demanded when the hyper-parameter space is large. A more elegant Bayesian-driven approach, described in detail in Section 4, could be more appropriate either for highly parallel clusters or more limited systems, such as desktops or IoT devices.

Applications of Echo State Networks. Given that ESN Reservoirs couple a large non-linear expansion of the input with internal state memory, we can summarize their applications in four different fields:

- Regression and prediction [24]: Here, the Reservoir embeds the temporal relationships of the inputs and $\mathbf{W}_{\text{out}}$ is trained to predict the trajectory of the signal n -steps forward in time. It emits a continuous stream as output.
- Classification [4]: The Reservoir learns the input features in the high-dimensional space and $\mathbf{W}_{\text{out}}$ is trained to discriminate different classes using internal states. The classification can be continuous, emitting a class at a certain rate of the input stream, or it can be used to classify sequences with arbitrary length.
- Autoencoder [9]: The Reservoir tries to develop the most faithful representation of the input itself. Some indexes evaluate the *richness* of the Reservoir, while the real output are the Reservoir states. This enriched representation of the input stream could be then fed as input to classifiers, neural networks, clustering systems, and the like.
- Generators: [24] These are similar to the regression task, but with a *teacher* signal with precise characteristics controlled by the input. This methodology is not widely used as simpler methods are available.

With respect to the data type, ESN are used mainly on sequential data, time-series, or any dataset that presents an evolution in time. However, some experiments in literature [30] explored also the analysis of 2D hyperspectral images with preliminary, but noteworthy results.

3 STATE-OF-THE-ART

If we compare the specific field of ESN against known DNN reductions, here, the researchers avoided compression or quantization in favor of more high-level optimizations. As an example, the number of active connections in a Reservoir could be minimized if random sparse initialization is substituted with more ordered topologies. Specifically, the authors in Ref. [40] explored minimum complexity Reservoirs, such as Delay Lines and Cyclic Reservoir (CR). These topologies not only require at most $2 * N_r + 1$ active connections, whereas a random Reservoir needs up to $0.5N_r^2$ of them, but they also need a maximum of 1 or 2 weights in total to be memorized, as all active connections share the same weight. The same work was extended [41] with Cyclic Reservoir with Jumps (CRJ), which adds connections between connections with a step > 1 . More recently, a research [28] started from CRJ and neuroscience knowledge to test Small World (SWT) topologies: here, a set of cyclic connections is rewired under a probabilistic parameter to mimic brain connectivity. These Reservoirs obtained interesting results in both learning performance and network stability.

These compact topologies were extensively used to deploy ESN on *exotic* devices where it would be still too difficult to implement a general Sparse Matrix by Dense Vector multiplication (SpMDV) such as $\mathbf{W}_r \mathbf{x}(t - 1)$. Notable examples include: photonic devices [32], memristor-based designs [48], or stochastic networks on Field-Programmable Gate Arrays (FPGA) [1].

Regarding the optimization of ESN, we identified two main tracks: *Gradient-based* optimizers [43], which require a profound analysis of the loss function and therefore difficult to adapt, and *evolutionary* optimizers, such as naive Genetic [49], Particle Swarm [47], and Pigeon-inspired [12] optimization. We opted for Bayesian Optimization as it works more efficiently and requires fewer samples than evolutionary algorithms if the number of hyper-parameters is small, such in our case. On the other hand, its level of parallelization is more limited since the next sample is dependent from the previous ones.

Fig. 2. Graphical representation of the EchoBay library.

For the sake of completeness, we include here some works that explored the deployment of RNN on low-power portable devices. Application-Specific Integrated Circuit (ASIC) solutions are not considered in the discussion. The authors in Ref. [6] offload Long-Short Term Memory (LSTM) computing on a mobile phone's Graphic Processing Unit (GPU) without quantization, but obtained limited improvements with respect to multithreaded CPU processing. The work in Ref. [8] proposes a similar model along with 8-bit weight quantization on a shallow LSTM (2 layers, 128 hidden units) to minimize memory footprint, rather than inference time. An interesting approach resembling ESN is available in Ref. [38], where a hard pruning process increases the sparsity of GRU weights, with notable results in model compression (up to 90%) and inference speed (2–7x). Unfortunately, tests were reproduced on a NVIDIA TitanX, with resources unfeasible for mobile devices.

4 ECHOBAY LIBRARY

This section provides a brief description of the EchoBay optimization library. After a brief introduction of the general workflow in Section 4.1, Sections 4.2 and 4.3 present the major novelties introduced by our library, focusing respectively on the algorithmic and architectural aspects.

4.1 Workflow and Features

Our proposal aims to embed the behavior of an ESN in an efficient and powerful system, complete of scalable runtime management and a set of fitness functions that make it suitable for different case studies, such as regression and classification learning tasks. The ESN computation core is enclosed in our optimizer, based on BO, which searches the optimal hyper-parameters for the task at hand and exports a portable representation of the final configuration. A brief representation of the overall system is visible in Figure 2.

Our proposed library is built in C++ using Eigen library² and Limbo Bayesian optimization engine [11]. Both libraries are compatible with ARM and x86 architectures with minimal dependencies. The ESN components are collected across multiple classes in order to guarantee a high level of modularity compared to a single monolithic system. Furthermore, the BO engine and the computation of ESN are separated, which allows the user to distribute the entire system over different devices (e.g., nodes that collect data and process it, a central remote architecture, a mixed system where optimization is performed remotely and novel ESN models are distributed).

The operations of the library are guided by a YAML³ configuration file, which can be edited manually or compiled semi-automatically following user's decision. The user has access to most of the ESN parameters, such as Reservoir size Nr , the effective spectral radius ρ , the input scaling ω^{in} , and the leaky factor α . The value of numeric parameters can be either fixed by the user or managed dynamically by the BO exploring the range between lower and upper bounds. Limbo library represents parameters with continuous values $p \in [0, 1]$ and the EchoBay frontend rescales them accordingly to the YAML file and collapses them to integer values for some parameters.

²eigen.tuxfamily.org.

³<https://yaml.org/>.

Another important aspect that can be chosen by the user is the preferred ESN topology, since EchoBay supports not only the common, Random topology, but also the more particular ones such as CR, CRJ, and SWT. In the last two cases, it is also possible to optimize parameters specific to the topology itself, such as the number of *jumps* for CRJ or *edges* for SWT.

Furthermore, the file contains options that are target-dependent, such as the number of threads available to system, memory limits, or number of network guesses (see below for details).

The EchoBay library supports OpenMP,⁴ where available on targets, to increase state update and BO training speed. As an example, increasing the number of threads available on a x86 machine led to an average speedup of 12x with respect to the single-thread execution. When the ESN and the BO engine are coupled together, the configuration file controls two levels of parallelism: the number of threads available to Eigen for state update and the parallelization of BO random samples.

4.2 Algorithmic Optimizations

In order to grant robustness to the final output of EchoBay, different algorithmic strategies have been adopted, aimed at solving a series of issues: as already stated, it is fundamental to choose the best set of hyper-parameters suited for the learning task at hand; moreover, given the intrinsic randomness associated to ESNs, the final configuration chosen should be also robust with respect to different initializations of the system itself. Finally, the shape of the fitness curve, and thus the set of hyper-parameters associated to its minimum, is strongly dependent on the function used to evaluate the performance of the model.

Bayesian Optimization. The Bayesian Optimization [13] procedure is driven by a simple, yet powerful paradigm. Instead of searching randomly in the hyper-parameter space or by performing exhaustive Grid Search (which computational time explodes as the number of parameters increases), instead, BO starts from a certain amount of *uncorrelated* random bootstrap samples, which can be computed in parallel, and then employs the obtained information against an objective function to suggest the next sample in the parameter space. The objective function is mapped to a Gaussian Process that provides an optimal posterior probability for the BO toward samples that maximize the objective function. The general formulation can be expressed as:

$$\begin{aligned} \max_{x \in A} f(x) \\ f(x_{1:k}) \sim \text{Normal}(\mu_0(x_{1:k}), \mathbf{s}_0(x_{1:k}, x_{1:k})), \end{aligned} \quad (4)$$

with $f(x)$ being the objective function and $f(x_{1:k})$ its prior distribution after k samples. The set A describes the space of explorable parameters with the hyper-rectangle $\{A \in \mathbb{R}^d\}$.

The BO process is optimal for the design of ESN [7] because the dimension of hyper-parameters d is relatively small (<15) and $f(x)$ is expensive. With expensive, we refer to an operation that is relatively efficient or can be done in parallel, namely the single state update of ESN. But that is lengthy to evaluate because the computation of a state $\mathbf{x}(t+1)$ requires all preceding states $\mathbf{x}(1:t)$ to be calculated sequentially.

In an unconstrained scenario, the BO will be focused only on the minimization of the error between the ESN output and the supervised *true* label or value. However, given a certain target device, our method will focus on four complementary objectives:

- The minimization of output error, given a certain loss function.
- A hard upper bound for Reservoir size that does not saturate the device memory.
- A soft boundary that pushes toward a better utilization of R_{conn} .
- A time constraint for each state update.

⁴www.openmp.org.

Moreover, the EchoBay framework adopts some procedures and optimizations, such as the possibility to use appropriate fitness functions or reduce the memory impact of large datasets, that make the use of ESN easier and more accessible to the final user. These optimizations, together with the rationale behind the four main objectives, are presented in the next paragraphs.

Performance stability. Given the randomness associated to the Reservoir $\mathbf{W}_r$ and the input weight matrix $\mathbf{W}_{in}$, it is possible to observe some variability between networks with the same hyper-parameters caused by topology and random weights. To counteract this issue, we introduced another layer of testing that replicates multiple guesses for each set of hyper-parameters under the following formulation:

$$\tilde{fitness}(\mathbf{p}) = E[fitness(\mathbf{p})]_{N_{guesses}} \quad (5)$$

under the hypothesis that the mean of fitness functions better reflect the behavior of the ESN with a set of parameters $\mathbf{p}$. The replication coefficient $N_{guesses}$ is provided by the user, but, since it linearly increases the training time, it is preferable to increase the guesses only on devices that are not constrained in time or with small networks. As a rule of thumb, this random error decreases with the number of guesses, and, depending on the users needs, an $N_{guesses}$ between 3 and 10 could be a good choice.

If $N_{guesses} > 1$, the network's final fitness on test set is picked from the worst-case evaluation instead of the mean to provide a more clear interpretation of its quality.

Loss function minimization. The first and most important optimization objective is, of course, the minimization of the prediction error of the ESN. Currently, the library supports regression and classification tasks, both with supervised learning. Normalized Root Mean Squared Error (NRMSE) and Median Symmetric Accuracy (MSA) are available for regression tasks, while Maximum Coefficient Accuracy and F1-mean are used in classification. We provide also the evaluation of Memory Capacity [24], an important measure of ESN recall. Thanks to Python bindings, the users have the possibility to write their own custom fitness function without changing the underlying code or knowing C++.

The two regression functions are defined as:

$$NRMSE = 100 * \frac{\sqrt{\frac{1}{N} \sum_{t=1}^N (y(t) - y_{target}(t))^2}}{std(y)} \quad (6)$$

$$MSA = 100 * \left(\exp \left(\text{median} \left(\left| \ln \left(\frac{y_{target}}{y} \right) \right| \right) \right) - 1 \right) \quad (7)$$

Since ESN are generally employed in the analysis of chaotic systems with fast and large fluctuations (e.g., Laser dataset in Section 5), MSA loss function could be more robust to the model's noise, as it is less sensible to isolated outliers compared to mean fluctuations of NRMSE.

Although the Bayesian Optimization works irrespective of the chosen loss function (assumed that it is a realization of a Gaussian Process), choosing the right loss function for the specific problem could lead to very different optimizations. A toy example in Figure 3: here, the MSA exhibits a clear relation with the Reservoir dimension, while the NRMSE does not show a precise dependency with N_r or the density of the Reservoir s . Other hyper-parameters were fixed to: $\rho = 0.958$, $\alpha = 0.5$, $\omega_{in} = 0.75$.

4.3 Architectural Optimizations

Moving the computation toward *edge devices* requires a set of constraints that should take in consideration the resources available on the system in terms of memory and computational capability.

Fig. 3. (color online) Comparison of loss function maps with varying N_r and s on 1-step Laser prediction task. Median Symmetric Accuracy exhibits a dependence on N_r ; NRMSE is more scattered on the range. (a) NRMSE loss function. (b) MSA loss function.

For this reason, we designed a set of tools that allow to identify the best working conditions for each device, thus allowing the user to better select the model characteristics. These constraints are entirely dependent on the Reservoir characteristics and not on the particular dataset used. However, giving the typical batch training of ESN, larger dataset implies the necessity to save large state matrix before being able to compute $\mathbf{W}_{out}$. To overcome this issue, we adopted a simple but efficient technique that allows to reduce considerably the size of the matrices that need to be stored.

Hard memory constraints. To test target-dependent optimizations, we provide a *probe* that measures two things: the size of the Reservoir that can be *safely* allocated on the machine (e.g., no *bad_alloc* exceptions), and the time necessary to update a single state and compute a prediction of the Reservoir with given R_{conn} . This approach is independent from the input properties; therefore, it can be calculated once for every target device.

Given the compressed representation of sparse matrices, the number of active connections suggests how much memory is occupied by the Reservoir. The *EchoBay* library provides an estimation of total size occupied by the Reservoir as:

$$\begin{aligned}
 R_{size} = & R_{conn} * (sizeof(float) + sizeof(idx)) \\
 & + (N_r * N_u) * sizeof(float) \\
 & + (N_r) * sizeof(float),
 \end{aligned} \tag{8}$$

where *float* represents the chosen floating-point precision. The first term takes into consideration the memory occupied by the active connections and the index of each element of $\mathbf{W}_r$ ($idx = Uint16$ in the current implementation); the last two terms account for the size of $\mathbf{W}_{in}$ and the state $\mathbf{x}$, respectively.

Although in *normal* working conditions, the memory occupied by Reservoir may be insignificant, when using *hard*-constrained devices, namely microcontrollers with less than 4MB of RAM, the Reservoir size, and thus the upper bounds of N_r and s , should be carefully tuned to control the memory occupation. This approach excludes networks that could generate unwanted exceptions or failures on the device. These limits are determined by R_{conn} , but they are asymmetrical with respect to N_r (which grows quadratically) and s , and generally target dependent. Therefore, a map

Fig. 4. (color online) Heap memory occupied by a random ESN Reservoir on an ESP32 microcontroller. The white region shows an empirical region that generates a `bad_alloc` exception on the device. *Single* floating-point weights increases the maximum size at expense of numerical precision. (a) Heap memory @ 64-bit floating points. (b) Heap memory @ 32-bit floating points.

shows to the user the heap space allocated by the ESN (see Figure 4) and its limits on a certain device. Upon that, the user can choose three complementary approaches:

- High N_r , low s (left upper corner)
- Low N_r , high s (right lower corner)
- Balanced on the curve

The system checks that $\max R_{conn}(N_r, s)$ remains under the hard limit, but the best coupling is still dependent on the learning task at hand and the user's expertise on ESN. For example, tasks that require remembering longer past sequences benefit more from large and highly sparse Reservoirs, since memory capacity increases almost linearly with N_r , as shown in Ref. [46]. Conversely, complex classification tasks could prefer a balanced approach (see also Ref. [34]), since the Reservoir exploits smaller time intervals, but it should be able to expand and separate the input as $\mathbb{R}_u^N \mapsto \mathbb{R}^{N_r}$ with $N_r \gg N_u$.

Soft memory constraints. As the memory available on the device increases, the constraint on R_{conn} relaxes up to the point of negligibility, but it remains helpful to the final user that wants to limit the memory occupation of the network. In this case, the optimization problem shifts toward the quality of active connections and how much a single connection influences the performance of the ESN.

In this specific case, it is still possible to use *soft-constraints* embedded in the BO process, related to two parameters:

- The overall size of the Reservoir N_r
- The sparsity s .

In fact, when looking for the best configuration on the hyper-parameter space, there may be *flat* regions where notable variations of the configuration lead to negligible changes in the final performance of the system. This idea becomes particularly relevant when considering the amount of R_{conn} of the system chosen. In almost all cases, a sparser and smaller Reservoir is preferred, even at the cost of very small degradation of the performance of the system, since the dimension

Fig. 5. (color online) State update time on different target architectures, with 32-bit or 64-bit floating points, and a detail of configurations with *iso-update* on a Raspberry Pi machine (64-bit). (a) State update time for different targets. (b) Contours with the same update time.

and density of the Reservoir are strongly related to both the amount of physical memory occupied by the Reservoir on the device, and the computational time required for the state update.

Starting from these considerations, in order to drive the optimization phase of the BO toward more compact Reservoir, we added a penalization factor in the cost function, defined as:

$$\hat{fitness}(\mathbf{p}, \mathbf{M}) = (1 - penalty\%) * fitness(\mathbf{p}) + penalty\% * \eta(\mathbf{p}, \mathbf{M}), \quad (9)$$

where *penalty%* is a user defined percentage to perform a weighted average of the two terms, influencing the aggressiveness (<20%), and $\eta(\mathbf{p}, \mathbf{M})$ is an expression of how much memory is used by the configuration $\mathbf{p}$ with respect to the largest Reservoir possible in the parameter space $\mathbf{A}$. η is expressed as follows:

$$\eta(\mathbf{p}, \mathbf{M}) = \frac{Nr_{norm}(\mathbf{p}, \mathbf{M}) + s_{norm}(\mathbf{p}, \mathbf{M})}{2} * 100, \quad (10)$$

where both $Nr_{norm}(\mathbf{p}, \mathbf{M})$ and $s_{norm}(\mathbf{p}, \mathbf{M})$ refer to the actual configuration values of $Nr(\mathbf{p})$ and $s(\mathbf{p})$, normalized between [0, 1] according to their respective upper and lower bounds set during the configuration phase. This allows to avoid any excessive weighting of Nr compared to s that may arise when penalizing the number of active connections R_{conn} or the direct product between $Nr(\mathbf{p})$ and $s(\mathbf{p})$.

Compact topologies that do not depend on the s parameter are excluded by memory optimization, and their extra parameter, namely the number of interconnected edges (SWT) and the length of the jump (CRJ) are optimized independently.

State update time constraints. A single update of the ESN state requires a certain time that is mainly dependent from the target architecture, the Reservoir size, the weight precision (*float32* vs. *double64*) and, partially, the number of input channels (see Figure 5(a)). If the ESN is part of a predictive control system, like the ones presented in Refs [2] or [23], it is likely subject to specified latency requirements that depends on the application, the $T_{sampling}$ of the observed signals, or the dynamics of the controlled systems. In this case, the optimizer requires the setup of hard constraints because each update of the state should be under a certain time threshold deterministically.

In this case, the user has two options: increase the number of threads available to EchoBay, or, if multithreading is not an option, control the Reservoir R_{conn} as $T_{update} \propto R_{conn}$. Compact

topologies are also an option as they require less R_{conn} with respect of random topologies and on equal performance.

As stated in memory optimization paragraph 4.3, the calculation of the benchmark map of a target device also provides a contour of R_{conn} that respects the desired time threshold. Upon this curve (see Figure 5(b)), the user can decide the upper boundaries of N_r and ρ following the aforementioned options: *high* N_r , *high* ρ , and *balanced*. Unfortunately, implementing hard boundaries directly in the BO is non-trivial, because samples that cross the threshold would be misleading in the next approximation of the objective function near the wall. Unfortunately, there is not a general rule to choose the correct boundaries, but the combination of the ESN probe of the system employed and the overall requirements of the problem (speed, memory constraints, number of CPUs), could provide an indication to the user for the choice of boundaries.

Where a tradeoff in numerical precision is acceptable, the library can be also recompiled and the user can easily switch between *double64* and *float32*. Reducing the floating point bit precision allows an increase of the overall Reservoir and a notable increase in computational speed in targets with minimum computational capabilities (see Figures 4 and 5(a)).

Large datasets compensation. As we mentioned before, the readout layer $\mathbf{W}_{out}$ is the only element of ESN that is *trained*, and this is usually done using Ridge Regression. Given the over-terminated system

$$\mathbf{Y}_{target} = \mathbf{W}_{out}\mathbf{X}, \quad (11)$$

where $\mathbf{X}$ is the concatenation of all training states, minus the washout settling after $\mathbf{x}_0$ and $\mathbf{Y}_{target}$ is the supervised target, which could be known classes in classification or a series of values in regression. The least square solution is given by:

$$\mathbf{W}_{out} = \mathbf{Y}_{target}\mathbf{X}^T * (\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}, \quad (12)$$

where λ regularization limits the potential instability of $\mathbf{W}_{out}$.

Both $\mathbf{Y}_{target}$ and $\mathbf{X}$ could be arbitrarily large. However, the memorization of all past states necessary to train the network becomes an issue in case of long datasets, very high N_r , or targets with minimal memory. For this reason, our framework includes another option along with memory optimization to enable the training of ESN without storing all the previous states. Recalling one of the techniques in Ref. [34], both $\mathbf{Y}_{target}\mathbf{X}^T$ and $\mathbf{X}\mathbf{X}^T$ are independent from the length from the training sequence and could be updated incrementally—a portion of the past states is memorized in blocks, which dimension is decided by the user, and then the two matrices are updated. In order to minimize rounding errors, the update is performed using Kahan summation techniques [27].

5 EXPERIMENTAL ANALYSIS

This section will present the results achieved with the proposed methodology. Several tests have been performed in order to evaluate the performance of the EchoBay library in the different settings presented in Section 4. Each methodology in our proposal was designed with specific devices in mind (described below). In particular, the tests performed are:

- A comparison between Grid Search, Random Search, and Bayesian Optimization, in terms of performance and time required to complete the optimization: the aim is to demonstrate that the BO is able to guarantee the same level of performance of grid search with considerable reduction in computational time.
- A comparison between different ESN topologies on standard optimization problems: different topologies present a different balance between computational power and memory occupation, and thus may be suited for different kind of tasks.

Fig. 6. Detail of the SantaFe Laser signal.

- Optimization using hard and soft constraints on the Reservoir memory occupation.
- Optimization using hard constraints on the time required for a single state update.

If not otherwise specified, the topology used for all the tests is the Random ESN, and the Bayesian Search uses 20 bootstrap samples and 40 search samples, while the amount and type of *optimizable* parameter is reported case by case.

5.1 Setup

The test target devices on which we validated our architecture are as follows:

- a standard desktop machine with x86 CPU (Intel i7-6700, 32GB DDR4 2133MHz), on which we run hyper-parameters and topologies comparison.
- a low-cost ARM machine (RaspberryPi 3B+, quad-core CortexA53@1.4GHz, 1GB LPDDR2), on which we tested soft constraints of BO.
- a highly constrained microcontroller (Espressif ESP32 Xtensa LX6@240MHz, 520KB SRAM), which enforced BO hard constraints in both time and memory.

The ESP32 architecture was chosen because it required minimal changes in the codebase, and, at the same time, memory capacity and computational performances are slightly higher than the microcontrollers available on the market. The EchoBay library was compiled with `-O3` (x86 and ARM) and `-Os` (ESP32) optimizations. The compiled library was employed to probe the different target devices and to identify their memory and time constraints, while the proper BO tests were performed on the desktop machine to speed up the optimization phase.

Datasets. We will test our methodology on well-known chaotic time-series prediction tasks: the SantaFe Laser [20] and Mackey-Glass [37] datasets, which provide interesting non-linearities to stress test the network dynamics; the NARMA10 task [3], which forces the Reservoir to embed a 10th order Non-Linear Autoregressive Moving Average; the Lorenz attractor flow, an interesting system of equations that tend to appear in many dynamical physical phenomena (e.g., Refs [20] and [22]), and it could be useful to describe time-critical state updates. The following equations use the standard notation associated to each dataset. Although different from the one used in Section 2, the meaning of each term is easy to extrapolate from the context.

The Laser dataset is a precise recording of a Far-Infrared laser in a chaotic state (an example is shown in Figure 6). Its steep non-linearities represent a good stress test for RNN and ESN networks.

The Mackey-Glass equation in its discretized variant is defined as:

$$x(t+1) = x(t) + \frac{a * x(t-s)}{(1 + x(t-s)^c)} - bx(t) \quad (13)$$

We will use $a = 0.2$, $b = 0.1$, $c = 10$ and a constant bootstrap input $x_0 = 0.1$ and length $s = 17$ samples.

The NARMA10 equation is described as:

$$\hat{y}(t+1) = 0.3\hat{y}(t) + 0.5\hat{y}(t) \left(\sum_{j=0}^9 \hat{y}(t-j) \right) + 1.5u(t-9)u(t) + 0.1, \quad (14)$$

where $u(n)$ is an input randomly drawn in the interval $[0, 0.5]$.

The Lorenz attractor is described by the following equations:

$$\begin{cases} \frac{dx}{dt} = \sigma(y - x) \\ \frac{dy}{dt} = x(\rho - z) - y \\ \frac{dz}{dt} = xy - \beta z \end{cases} \quad (15)$$

We will use *classical* Lorenz solutions with $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$.

In all the tests, the optimization process was repeated five times, and the average values of fitness are reported in tables, along with final configurations. All the tests were also run using a 55-20-25% split ratio in training, validation, and test sets. We used 12,000 points realizations for all datasets, except Laser, which provides only 10,000 points. We will employ the NRMSE loss function to uniform all tests and to be comparable with the rest of the literature available.

For the sake of simplicity, we focused our results on *shallow* ESN with one Reservoir layer. Although the optimization of ESN with multiple deep layers is fully supported by our library, in this article, we wanted to focus on a single set of hyper-parameters, since Deep ESN [16] require further considerations in terms of network stability, computational complexity, and topology constraints.

5.2 Experimental Results

Bayesian Optimization vs. Grid Search. First and foremost, we compared the results of Bayesian Optimization against the same problem treated with Grid Search. We tested the Laser one-step prediction task, with *double* precision and increasing number of hyper-parameters, namely:

- (1) N_r range 100–600,
- (2) s range 0.05–0.3 (fixed 0.2),
- (3) ρ range 0.8–1.2 (fixed 0.9),
- (4) α range 0.05–1 (fixed 0.5),
- (5) ω_{in} range 0.05–1 (fixed 0.2),

where *fixed* refers to the parameter's value when it is not optimized by our library. The topology is random with one single layer and a washout of 100 samples. The Bayesian Optimization employs 20 bootstrap samples and 40 Bayesian samples. Conversely, Grid Search required a tuning of the bins due to memory constraints of the machine: 40 bins for 1 and 2 parameters, 30 for 3.

The Grid Search divides the range uniformly with $(\max_{param} - \min_{param}) / \text{bins}$ spacing, while Random and Bayesian Search traverse the range continuously from 0 to 1 ($value_{norm}$). The value of each parameter is transformed as:

$$value_{full} = value_{norm} * (\max_{param} - \min_{param}) + \min_{param}$$

If the specific hyper-parameter is defined by integer values (e.g., N_r), $value_{full}$ is rounded to nearest integer.

With four and five parameters, it was required to perform multiple runs with 10 bins, and then reduce the range of hyper-parameters manually to increase the search resolution. This is one of

Table 1. Final Performance of Grid Search and BO on One-Step Laser Prediction Task

# params	Grid		Random		EchoBay	
	fitness [%]	time	fitness [%]	time		
1	92.870	5.37s	93.157	25.69s	93.037	13.46s
2	93.079	5m33s	93.376	28.73s	93.058	25.20s
3	93.066	1h33m59s	93.108	2m37s (2 runs)	93.321	17.82s
4	93.677	1h14m9s (3 runs)	93.177	54.87s	93.343	17.056s
5	93.512	2h33m21s (3 runs)	93.647	56s	93.497	15.61s

Random Search on three parameters was run two times to improve the results. Grid Search bins were reduced for four and five hyper-parameters due to machine's memory issues. Fitness is $100 - NRMSE$ (higher is better).

the major drawbacks of Grid Search. The Random Search employs 60 samples plus 60 for each hyper-parameter optimized, one last sample is selected with Bayes using all random samples as an *a priori* model.

The optimization was tested on our x86 machine with three threads available to Eigen and three threads for sampling (both bootstrap and Grid). BO will use only one guess for each configuration.

It is clearly visible from Table 1 that BO obtained a level of performance that matches or it is slightly lower than the one obtained using Grid Search. Furthermore, the BO proved to be competitive in time, with a speedup ranging from 13x to 589x. Conversely, the coupling of an extensive Random Search and a single Bayesian step proved to be competitive both in performance and time, although BO remains 4x faster on average.

The 3 parameters Random Search was run two times to improve a slightly underperforming result (92.658%) in the first run. This inconvenience outlines one of the main problems of Random Search, which does not guarantee that the entire parameter space is explored uniformly if the number of samples is not chosen carefully. The next results will focus on the output of BO, while fine-grained Grid Search will be employed on small examples to provide a view of the loss function's shape (as in Figure 3).

If we compare the final configuration of Table 2, we can observe that each method reaches a different optimum, although the NRMSE was comparable. In terms of Reservoir's stability, both Random and Bayes Search remained under the unity; while from a computational point-of-view, Grid Search obtained the best results in terms of R_{conn} four times out of five. Although R_{conn} is not an important element of comparison in this set of experiments, the aim is to show how naive BO does not give particular importance to the meaning of each parameter, potentially ending up in configurations with high fitness but sub-optimal R_{conn} (four times out of five in these experiments). This issue will be treated in detail in the next results, but since BO requires a time that is extremely lower than Grid Search, it is also possible to increase the number of samples in bootstrap and Bayesian search to get nearer to this global optimum.

Comparison of ESN topologies. The first choice of an ESN designer will be certainly the network topology that better fits the problem. When the knowledge of the domain of interest or a certain guidance are not available, the test of all topologies remains the only choice. On top of this, the optimization of the network does not have a golden rule since each topology responds differently to some hyper-parameters.

EchoBay is helpful in the process, providing an efficient way to test different solutions in a short time. To test these capabilities, we ran Bayesian Optimization on different non-linear regression problems and compared the results. In all tests, we used the same hyper-parameter range: $N_r \in [100, 600]$, $\rho \in [0.8, 1.2]$, $\alpha \in [0.05, 1]$, $\omega_{in} \in [0.05, 1]$. The other hyper-parameters are $s \in [0.05, 0.3]$, **edges** $\in [0, 10]$, **jumps** $\in [0, 10]$, for Random, SWT, and CRJ topologies, respectively.

Table 2. Final Configuration of Grid Search and Bayesian Optimization on One-Step Laser Prediction Task

Method		# hyper-params				
		1	2	3	4	5
Grid	N_r	363	438	413	390	455
	s	—	0.262	0.275	0.214	0.242
	ρ	—	—	0.92	0.87	1.15
	α	—	—	—	0.856	0.911
	ω_{in}	—	—	—	—	0.683
	R_{conn}	26,354	50,263	46,906	32,549	50,100
Random	N_r	402	565	528	593	527
	s	—	0.272	0.216	0.216	0.139
	ρ	—	—	0.89	0.86	0.84
	α	—	—	—	0.848	0.754
	ω_{in}	—	—	—	—	0.273
	R_{conn}	32,627	87,108	60,753	76,534	39,121
EchoBay	N_r	492	515	572	519	230
	s	—	0.3	0.188	0.285	0.119
	ρ	—	—	0.99	0.86	0.99
	α	—	—	—	0.76	1
	ω_{in}	—	—	—	—	0.712
	R_{conn}	48,413	79,568	61,511	76,768	6,295

Values marked with— are fixed (see the text for details). The configuration with the minimum R_{conn} is highlighted.

Table 3. Comparison of Different Topologies on Non-linear Regression Tasks with NRMSE Loss Function (One-Step Prediction, Zero-Step Identification for NARMA10)

Task	fitness[%]			R_{conn}		
	Random	SWT	CRJ	Random	SWT	CRJ
Laser	91.730	76.395	90.028	2,978	5,940	411
Mackey-Glass	99.985	99.931	99.953	28,533	11,950	1,263
NARMA10	90.647	51.134	61.604	45,764	8,514	1,032
Lorenz	98.309	84.524	99.662	9,609	13,778	796

Fitness is $100 - NRMSE$ (higher is better).

In Tables 3 and 4, we can see that different topologies are required to reach the highest performance for the given problem. From our tests, for example, we can see that Small World topology suffers in signals like Laser and Lorenz's x-y channels that present sudden non-linearities (see also Figure 7 for details).

From Figure 7, we can also observe how using different loss function leads the optimization toward configurations with different properties, for example, higher ripples in the error caused by the cusps of Laser flow. NARMA10 identification remains a complex task for all networks considered, but its hidden complexity takes the toll on compact topologies in particular. On the other hand, highly deterministic topologies, such as CRJ, obtain results that are comparable or higher than Random Reservoirs on all other tasks, using a much smaller amount (down to nearly 4%) of active connections.

Fig. 7. Prediction error of Laser flow under different Reservoir topologies.

Table 4. Comparison of Different Topologies on Non-linear Regression Tasks with MSA Loss Function (One-Step Prediction, Zero-Step Identification for NARMA10)

Task	fitness[%]			R_{conn}		
	Random	SWT	CRJ	Random	SWT	CRJ
Laser	98.834	93.776	98.633	83,148	7,409	810
Mackey-Glass	99.746	99.941	99.944	36,157	7,919	1,238
NARMA10	98.101	90.516	95.111	55,520	5,132	886
Lorenz	98.828	97.840	97.822	27,696	6,851	1,004

Fitness is $100 - MSA$ (higher is better).

Table 5. Comparison of Different Hard Constraints for ESP32 Target on Non-Linear Regression Tasks

Task	High N_r				High s				Balanced			
	N_r	s	ρ	fitness[%]	N_r	s	ρ	fitness[%]	N_r	s	ρ	fitness[%]
Laser	150	0.032	1	91.446	65	0.241	0.99	88.437	79	0.109	0.83	90.357
Mackey-Glass	148	0.041	1.05	99.732	78	0.164	0.94	99.856	93	0.088	0.99	99.834
NARMA10	175	0.06	1.12	64.546	82	0.078	1.05	43.586	92	0.127	0.89	37.651
Lorenz	168	0.052	0.98	97.187	83	0.243	0.86	95.397	95	0.144	1.05	94.969

Upper bounds for N_r and s are: {175, 0.06}, {83, 0.264}, {95, 0.21} in High N_r , High s , and Balanced, respectively. Fitness is $100 - NRMSE$ (higher is better).

Memory constraints. First of all, we tested hard-constraints optimizations on different regression tasks, with *double* floating-point precision. Upper bounds were chosen to max-out at a similar level of R_{conn} , and we considered the ESP32 as a target architecture. We optimized N_r , s , and ρ and used the approaches presented in Section 4. The other hyper-parameters were fixed at $\omega_{in} = \alpha = 1$, $\lambda = 0$.

Table 5 describes the obtained results. As before, the NARMA10 identification task requires larger constraints that are outside of the ESP32 capabilities. However, we can observe that when N_r and s are too limited, EchoBay also exploits $\rho > 1$ to get the best performance from the Reservoir. With regards to the stability of the network with $\rho > 1$, the Bayesian Optimization provides some guarantees to the user, because divergent zones of the parameter space lead to a drop in the loss function, and they are consequently excluded by the Bayesian search.

Although the three methods reached different optimal configurations for each task, the Random ESN remains the most performant configuration in terms of accuracy across the tasks. Considering

Table 6. Memory Penalization Effect on the Two Datasets

Dataset	Configuration	N_r	s [%]	R_{conn}	fitness[%]
Narma10	Upper Bound	500	10	25×10^3	—
	0% Penalization	497	6	15×10^3	88.11
	2% Penalization	500	3	7.6×10^3	88.65
	5% Penalization	471	1.4	2.9×10^3	87.76
	10% Penalization	404	1.6	2.6×10^3	86.69
Mackey-Glass 10-step	Upper Bound	150	10	22.5×10^2	—
	0% Penalization	148	7	15.85×10^2	99.55
	2% Penalization	136	3	5.85×10^2	99.44
	5% Penalization	114	2.8	1.14×10^2	98.95
	10% Penalization	93	3.2	0.93×10^2	98.20

Fitness is $100 - NRMSE$ (higher is better).

also the number of active connections, the optimization obtained on average 1,231; 1,054; and 954, respectively.

Considering the *soft constraint* introduced in the cost function of the BO, we tested three penalization levels on two task:

- The NARMA10 task.
- The 10-step ahead prediction using the Mackey-Glass task.

In this case, we wanted to test difficult tasks as to which network will not fit on a small microcontroller, but that can take advantage of a better utilization of available active connections. In both cases, we used a 12,000 points realization. As stated before, the only two *optimizable* parameters in these cases are N_r and s . The other hyper-parameters of the system have been left fixed in order to ignore their influence on loss function shape; in particular, for the NARMA10 task, the values chosen are $\omega^{in} = 0.1$, $\alpha = 1$, $\lambda = 0$, $\rho = 0.9$; for the Mackey-Glass task, the values are $\omega^{in} = 1.2$, $\alpha = 0.73$, $\lambda = 0$, $\rho = 0.9$.

Table 6 reports the results obtained with different penalization factors on the test set. For each factor, the optimization process has been repeated five times, and the values reported are the averaged final configurations and fitness. As shown, the EchoBay optimization is able to significantly reduce the amount of R_{conn} at every level of penalization, with a greater penalization corresponding to a greater reduction in active connections.

The use of a soft-constraint optimization is particularly efficient in this case given the shape of the fitness function, computed on the validation set, and presented in Figure 8. In fact, there is an almost *flat* region of performance, where the BO, if left unconstrained, will move toward the upper-right corner to reduce the error as much as possible, but with negligible variations. In this scenario, the soft constraints will push the optimal solution toward the lower-left corner, forcing BO to find the right compromise of R_{conn} and performance. Another important factor is that there is a clear relationship between the error $NRMSE$ and N_r/s . This is particularly true if compared to the cases presented in Figure 3, where, especially when using the $NRMSE$ cost function, it appears difficult to identify clear boundaries. This may be due also to the type of problem considered. In fact, the one-step ahead Laser task is a typically easy problem to be solved with ESN, where the role of the two hyper-parameters considered, N_r and s , is less relevant compared to the others such as ω^{in} or ρ .

The NARMA10 task, on the other hand, relies more on the Reservoir size, and this is proved also by the tendency to reduce s and not N_r when the penalization increases. Only after the density

Fig. 8. (color online) Effect of different levels of penalization on final configurations for NARMA10 and 10-step ahead Mackey tasks. (a) Parameter Space for NARMA10 Task. (b) Parameter Space for Mackey Task.

Table 7. Optimizer Performance (100 - NRMSE, Higher is Better) under Decreasing Time Constraints

Threshold [us]	Upper bound		Optimal				Fitness [%]
	N_r	R_{conn}	N_r	s	R_{conn}	ρ	
1,000	810	65,610	809	0.095	62176	0.80	99.986
750	720	51,840	719	0.041	21195	0.96	99.969
500	570	32,490	522	0.017	5286	0.85	99.961
350	470	22,090	470	0.099	21869	0.85	99.971
200	330	11,450	330	0.047	5398	0.86	99.931
100	65	4,225	65	0.0328	203	0.93	96.808

Lorenz 10-step task $s_{max} = 0.1$.

reaches almost the minimum, at 5% penalization, the optimization moves toward sacrificing N_r when the penalization reaches 10%. This is in accordance with the shape of the hyper-parameter space in Figure 8(a): left aside, a small interaction between performance and density at very low values of s , the different levels can be roughly approximated as vertical lines that depend only on N_r .

In the 10-step ahead Mackey-Glass prediction task instead, the negative effect of very sparse Reservoirs is much more evident, and, in fact, the minimum density is reached sooner, at 2% penalization, and is much higher compared to the NARMA10 Task. However, also in this case, after a certain level of s , the performance starts to become unrelated to the density of the Reservoir, while depending only on N_r , suggesting that reasonably sparse Reservoirs are able to perform as well as denser ones, with the advantages of reduced memory footprints and computational time.

Time constraints. In the last scenario, we wanted to test a time-constrained setup: the probe measured single-threaded capabilities on a Raspberry Pi, with 64-bit numerical precision. The task is to perform a 10-step ahead prediction of Lorenz's flow, with decreasing latency thresholds. The upper boundary for spectral radius is $\rho = 1.2$, while other parameters are fixed with $\omega_{in} = \alpha = 1$. We opted for quite sparse networks; therefore, the maximum density is $s = 0.1$.

Table 7 shows that with respect to the specific problem, the optimizer tries to maximize the value of N_r , while the sparsity and the spectral radius of the Reservoir are coupled together to store the non-linear information of the Lorenz flow. We can also notice that under the 200us threshold, the

Table 8. Optimizer Performance (100 - NRMSE, Higher is Better) under Decreasing Time Constraints

Threshold [us]	Upper bound		Optimal				Fitness [%]
	N_r	R_{conn}	N_r	s	R_{conn}	ρ	
200	478	11,450	417	0.026	4,998	0.80	99.942
100	281	4,225	275	0.030	2,544	0.86	99.914
50	185	1,717	181	0.042	1,546	0.80	99.873

Lorenz 10-step task $s_{max} = 0.05$.

NRMSE starts to increase fast, but the sparsity remains well under the upper boundary. We can then change our boundaries moving along the curve that have the same number of R_{conn} and decrease s to increase N_r again (results in Table 8).

Not surprisingly, this trick allows the final user to guarantee a similar performance with a lowering latency. Also in this case, a general rule is not available, but since the ESN probe provides also an *iso-update* curve (see Figure 5(b)), the user could exploit this information to choose better boundaries according to the task at hand. The convergence of this approach is the CRJ topology, which has the most sparse Reservoir possible.

6 CONCLUSIONS

In this article, we developed a novel Bayesian Optimization methodology for ESN targeting both embedded and unconstrained computational devices. Moreover, we embedded this methodology inside EchoBay, a framework that is both easy to use, given the easiness of setting everything up for a specific case study, and powerful, since the user has the possibility to fine-tune all the most important aspects relative to the ESN paradigm. These aspects are particularly relevant nowadays, given the necessity to democratize the Machine Learning (ML) approach, thus, making it available also to people with limited knowledge in the ESN, RC field, or in general in code writing. Furthermore, using ML architectures that require less computations and lower training time could be quite beneficial to counteract the high energy consumption caused by regular DNN.

Since our tools are also aimed at *power-users*, which may have very specific needs, in addition to the main objective function of BO that optimizes with respect to accuracy, we added a set of both hard and soft boundaries that help the ESN designer toward a configuration that better exploits memory and computational resources. The system was able to reach optimal configurations under different temporal learning tasks, reducing at the same time the computational cost of the Reservoirs. This is due to the use of ESN as a foundation algorithm for EchoBay; we are able to grant good performance even on a heavily constrained device, a task which is almost impossible to fulfill using other similar techniques such as LSTM or GRU.

Future developments outline the possibility to target hardware accelerators, such as GPUs and FPGA to further improve the performance of ESN or allow larger and deeper Reservoir Architectures. Quantization of the Reservoir weights and/or activation states will be considered to maximize the network compression. Additionally, we expect to also support Reservoirs that are not randomly initialized and novel architectures that exploit more sophisticated readout layers. Regarding time-constrained updates, it would be valuable to introduce a rule for the BO to better explore *iso-update* parameters' space, without leaving the task to the user. Lastly, it may be interesting to explore the possibility to have unsupervised training of the hyper-parameters related to the Reservoir, such that a single Reservoir can be used for different readouts, training multiple $\mathbf{W}_{out}$. Another interesting aspect would be to see how an unsupervised training could be combined

with Deep ESN, and if this approach could lead to an increase of information transmitted from one layer to another.

REFERENCES

- [1] Miquel L. Alomar, Vincent Canals, Nicolas Perez-Mora, Víctor Martínez-Moll, and Josep L. Rosselló. 2016. FPGA-based stochastic echo state networks for time-series forecasting. *Computational Intelligence and Neuroscience* 2016 (2016), 1–14.
- [2] Luca Bugliari Armenio, Enrico Terzi, Marcello Farina, and Riccardo Scattolini. 2019. Echo state networks: Analysis, training and predictive control. *CoRR* abs/1902.01618 (2019). arxiv:1902.01618 <http://arxiv.org/abs/1902.01618>.
- [3] Amir F. Atiya and Alexander G. Parlos. 2000. New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks* 11, 3 (2000), 697–709.
- [4] Davide Bacciu, Paolo Barsocchi, Stefano Chessa, Claudio Gallicchio, and Alessio Micheli. 2014. An experimental characterization of reservoir computing in ambient assisted living applications. *Neural Computing and Applications* 24, 6 (2014), 1451–1464.
- [5] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13 (Feb 2012), 281–305.
- [6] Qingqing Cao, Niranjana Balasubramanian, and Aruna Balasubramanian. 2017. MobiRNN: Efficient recurrent neural network execution on mobile GPU. In *Proceedings of the 1st International Workshop on Deep Learning for Mobile Systems and Applications (EMDL'17)*. ACM, New York, NY, 1–6.
- [7] Luca Cerina, Giuseppe Franco, and Marco Domenico Santambrogio. 2019. Lightweight autonomous bayesian optimization of Echo-State Networks. In *Proceedings of the 2019 European Symposium on Artificial Neural Networks (ESANN'19)*. 637–642.
- [8] Jagmohan Chauhan, Suranga Seneviratne, Yining Hu, Archan Misra, Aruna Seneviratne, and Youngki Lee. 2018. Breathing-based authentication on resource-constrained IoT devices using recurrent neural networks. *Computer* 51, 5 (May 2018), 60–67. DOI: <https://doi.org/10.1109/MC.2018.2381119>
- [9] Naima Chouikhi, Boudour Ammar, Amir Hussain, and Adel M. Alimi. 2019. Bi-level multi-objective evolution of a multi-layered echo-state network autoencoder for data representations. *Neurocomputing* 341 (2019), 195–211.
- [10] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).
- [11] Antoine Cully, Kostantinos Chatzilygeroudis, Federico Allocati, and Jean-Baptiste Mouret. 2018. Limbo: A flexible high-performance library for Gaussian processes modeling and data-efficient optimization. *The Journal of Open Source Software* 3, 26 (2018), 545.
- [12] Haibin Duan and Xiaohua Wang. 2015. Echo state networks with orthogonal pigeon-inspired optimization for image restoration. *IEEE Transactions on Neural Networks and Learning Systems* 27, 11 (2015), 2413–2425.
- [13] Peter I. Frazier. 2018. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811* (2018).
- [14] Claudio Gallicchio. 2019. Chasing the echo state property. *Proceedings of European Symposium on Artificial Neural Networks, ESANN, 2019*.
- [15] Claudio Gallicchio and Alessio Micheli. 2011. Architectural and Markovian factors of echo state networks. *Neural Networks* 24, 5 (2011), 440–456.
- [16] Claudio Gallicchio and Alessio Micheli. 2016. Deep reservoir computing: A critical analysis. *Proceedings of European Symposium on Artificial Neural Networks, ESANN, 2019*.
- [17] Claudio Gallicchio and Alessio Micheli. 2020. Fast and deep graph neural networks. In *Proceedings of AAAI 2020 - arXiv preprint arXiv:1911.08941*.
- [18] Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. 2017. Deep reservoir computing: A critical experimental analysis. *Neurocomputing* 268 (2017), 87–99.
- [19] Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. 2019. Comparison between DeepESNs and gated RNNs on multivariate time-series prediction. *Proceedings of European Symposium on Artificial Neural Networks, ESANN, 2019* (2019), 619–624.
- [20] Neil A. Gershenfeld and Andreas S. Weigend. 1993. *The Future of Time Series*. Technical Report. Xerox Corporation, Palo Alto Research Center.
- [21] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. 2018. Recent advances in convolutional neural networks. *Pattern Recognition* 77 (2018), 354–377. DOI: <https://doi.org/10.1016/j.patcog.2017.10.013>
- [22] Hermann Haken. 1975. Analogy between higher instabilities in fluids and lasers. *Physics Letters A* 53, 1 (1975), 77–78. DOI: [https://doi.org/10.1016/0375-9601\(75\)90353-9](https://doi.org/10.1016/0375-9601(75)90353-9)

- [23] Jian Huang, Jin Qian, Lei Liu, Yongji Wang, Caihua Xiong, and Songhyok Ri. 2016. Echo state network based predictive control with particle swarm optimization for pneumatic muscle actuator. *Journal of the Franklin Institute* 353, 12 (2016), 2761–2782. DOI: <https://doi.org/10.1016/j.jfranklin.2016.05.004>
- [24] Herbert Jaeger. 2001. The “echo state” approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* 148, 34 (2001), 1–12.
- [25] Herbert Jaeger and Harald Haas. 2004. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science* 304, 5667 (2004), 78–80.
- [26] Herbert Jaeger, Mantas Lukoševičius, Dan Popovici, and Udo Siewert. 2007. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Networks* 20, 3 (2007), 335–352.
- [27] William Kahan. 1965. Pracniques: Further remarks on reducing truncation errors. *Commun. ACM* 8, 1 (1965), 40.
- [28] Yuji Kawai, Jihoon Park, and Minoru Asada. 2019. A small-world topology enhances the echo state property and signal propagation in reservoir computing. *Neural Networks* 112 (2019), 15–23. DOI: <https://doi.org/10.1016/j.neunet.2019.01.002>
- [29] John F. Kolen and Stefan C. Kremer. 2001. *A Field Guide to Dynamical Recurrent Networks*. John Wiley & Sons.
- [30] Petia Koprinkova-Hristova, Donka Angelova, Denitsa Borisova, and Georgi Jelev. 2013. Clustering of spectral images using Echo state networks. In *2013 IEEE INISTA*. IEEE, 1–5.
- [31] Aditya Kusupati, Manish Singh, Kush Bhatia, Ashish Kumar, Prateek Jain, and Manik Varma. 2018. Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In *Advances in Neural Information Processing Systems*. 9017–9028.
- [32] Laurent Larger, Antonio Baylón-Fuentes, Romain Martinenghi, Vladimir S. Udaltsov, Yanne K. Chembo, and Maxime Jacquot. 2017. High-speed photonic reservoir computing using a time-delay-based architecture: Million words per second classification. *Physical Review X* 7, 1 (2017), 011015.
- [33] Jie Lin, Wei Yu, Nan Zhang, Xinyu Yang, Hanlin Zhang, and Wei Zhao. 2017. A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal* 4, 5 (2017), 1125–1142.
- [34] Mantas Lukoševičius. 2012. A practical guide to applying echo state networks. In *Neural Networks: Tricks of the Trade*. Springer, 659–686.
- [35] Mantas Lukoševičius and Herbert Jaeger. 2009. Reservoir computing approaches to recurrent neural network training. *Computer Science Review* 3, 3 (2009), 127–149.
- [36] Wolfgang Maass and Henry Markram. 2004. On the computational power of circuits of spiking neurons. *Journal of Computer and System Sciences* 69, 4 (2004), 593–616.
- [37] Michael C. Mackey and Leon Glass. 1977. Oscillation and chaos in physiological control systems. *Science* 197, 4300 (1977), 287–289.
- [38] Sharan Narang, Gregory F. Diamos, Shubho Sengupta, and Erich Elsen. 2017. Exploring sparsity in recurrent neural networks. *CoRR* abs/1704.05119 (2017). arxiv:1704.05119 <http://arxiv.org/abs/1704.05119>
- [39] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*. 1310–1318.
- [40] Ali Rodan and Peter Tino. 2011. Minimum complexity echo state network. *IEEE Transactions on Neural Networks* 22, 1 (Jan. 2011), 131–144.
- [41] Ali Rodan and Peter Tiño. 2012. Simple deterministically constructed cycle reservoirs with regular jumps. *Neural Computation* 24, 7 (2012), 1822–1852.
- [42] Gouhei Tanaka, Toshiyuki Yamane, Jean Benoit Héroux, Ryosho Nakane, Naoki Kanazawa, Seiji Takeda, Hidetoshi Numata, Daiju Nakano, and Akira Hirose. 2019. Recent advances in physical reservoir computing: A review. *Neural Networks* (2019).
- [43] Luca Anthony Thiede and Ulrich Parlitz. 2019. Gradient based hyperparameter optimization in Echo State Networks. *Neural Networks* 115 (2019), 23–29. DOI: <https://doi.org/10.1016/j.neunet.2019.02.001>
- [44] Peter Tino and Georg Dorffner. 2001. Predicting the future of discrete sequences from fractal representations of the past. *Machine Learning* 45, 2 (2001), 187–217.
- [45] Peter Tiño, Barbara Hammer, and Mikael Bodén. 2007. Markovian bias of neural-based architectures with feedback connections. In *Perspectives of Neural-symbolic Integration*. Springer, 95–133.
- [46] David Verstraeten, Benjamin Schrauwen, Michiel d’Haene, and Dirk Stroobandt. 2007. An experimental unification of reservoir computing methods. *Neural Networks* 20, 3 (2007), 391–403.
- [47] Heshan Wang and Xuefeng Yan. 2015. Optimizing the echo state network with a binary particle swarm optimization algorithm. *Knowledge-Based Systems* 86 (2015), 182–193.
- [48] Shiping Wen, Rui Hu, Yin Yang, Tingwen Huang, Zhigang Zeng, and Yong-Duan Song. 2018. Memristor-based echo state network with online least mean square. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 99 (2018), 1–10.

- [49] Dongming Xu, Jing Lan, and José C. Principe. 2005. Direct adaptive control: An echo state network and genetic algorithm approach. In *Proceedings of the 2005 IEEE International Joint Conference on Neural Networks, 2005*, Vol. 3. IEEE, 1483–1486.
- [50] Izzet B. Yildiz, Herbert Jaeger, and Stefan J. Kiebel. 2012. Re-visiting the echo state property. *Neural Networks* 35 (2012), 1–9.

Received October 2019; revised February 2020; accepted June 2020