
Article:
Manneschi, L., Lin, A.C. and Vasilaki, E. (2023) SpaRCe: Improved learning of reservoir computing systems through sparse representations. IEEE Transactions on Neural Networks and Learning Systems, 34 (2). pp. 824-838. ISSN: 2162-237X

https://doi.org/10.1109/tnnls.2021.3102378

SpaRCe: Improved Learning of Reservoir Computing Systems Through Sparse Representations

Luca Manneschi, Andrew C. Lin, and Eleni Vasilaki

***Abstract*— "Sparse" neural networks, in which relatively few neurons or connections are active, are common in both machine learning and neuroscience. While, in machine learning, "sparsity" is related to a penalty term that leads to some connecting weights becoming small or zero, in biological brains, sparsity is often created when high spiking thresholds prevent neuronal activity. Here, we introduce sparsity into a reservoir computing network via neuron-specific learnable thresholds of activity, allowing neurons with low thresholds to contribute to decision-making but suppressing information from neurons with high thresholds. This approach, which we term "SpaRCe," optimizes the sparsity level of the reservoir without affecting the reservoir dynamics. The read-out weights and the thresholds are learned by an online gradient rule that minimizes an error function on the outputs of the network. Threshold learning occurs by the balance of two opposing forces: reducing interneuronal correlations in the reservoir by deactivating redundant neurons, while increasing the activity of neurons participating in correct decisions. We test SpaRCe on classification problems and find that threshold learning improves performance compared to standard reservoir computing. SpaRCe alleviates the problem of catastrophic forgetting, a problem most evident in standard echo state networks (ESNs) and recurrent neural networks in general, due to increasing the number of task-specialized neurons that are included in the network decisions.**

***Index Terms*— Catastrophic forgetting, echo state networks (ESNs), machine learning, online learning, reservoir computing, sparsity.**

I. Introduction

THE performance of artificial neural networks is often improved by adopting "sparse" representations, in which relatively few neurons or connections are active. Previous research has studied the role of sparse connectivity, in terms of memory, in Hopfield networks, demonstrating how sparse connectivity increases storage capacity [1]–[4]. Memory retrieval and associative learning have been studied in the context of neural network attractors, and the work in [5] has provided an abstract mathematical analysis of retrieval capacity. From the machine learning perspective, adopting sparse connectivity can lead to more interpretable models [6], a reduced computational cost [7], and can help solve overfitting problems [8]. Sparsity in machine learning is typically introduced to artificial networks through regularization, in which a penalty term leads to the reduction of the connection weights. In this regard, the work in [7] demonstrated how structured sparsity can improve computational speed and accuracy in a convolutional neural network. Rasmussen *et al.* [9] showed how the choice of regularization parameters of the model can impact the interpretability and the reproducibility of a classifier of neuroimaging data and showed the existence of a tradeoff between pure classification accuracy and reproducibility.

Sparsity is also a well-known concept in neuroscience: biological neurons are highly selective in systems ranging from mammalian sensory cortex [10] to the insect mushroom body [11], [12]. However, unlike in typical machine learning approaches, biological sparsity is introduced not only by reducing connection weights between neurons but also by the fact that neurons have spiking thresholds: they only fire when their summed inputs exceed a certain threshold. High spiking thresholds relative to the size of synaptic inputs can often contribute to the high selectivity of neurons, as with Kenyon cells (KCs), the principal neurons of the insect mushroom body, which fire sparsely in response to odor stimuli [13]–[16]. In the fruit fly *Drosophila*, this sparse odor coding enhances learned discrimination of similar odors [12]. Moreover, spiking thresholds vary across neurons [13] and over time for the same neuron [17], [18], and spiking thresholds for different neurons are adapted to neurons' particular input statistics [17] and past activity [19].

Previous efforts that introduce sparsity inspired by neuroscience findings are based on the BCM learning rule [20] and intrinsic plasticity [21]. Both these concepts introduce a "soft" regulation of network activity and have been applied to reservoir computing [22]–[24]. Here, we are taking a different approach to regulate sparsity in reservoir computing: we introduce hard, adaptable activity thresholds to create SpaRCe.

Manuscript received 7 January 2021; revised 13 May 2021; accepted 17 July 2021. Date of publication 16 August 2021; date of current version 6 February 2023. The work of Andrew C. Lin was supported in part by the Google Deepmind Faculty Research Awards Program, in part by the European Research Council under Grant 639489, and in part by the Biotechnology and Biological Sciences Research Council under Grant BB/S016031/1. The work of Eleni Vasilaki was supported in part by the Google Deepmind Faculty Research Awards Program; and in part by the Engineering and Physical Sciences Research Council under Grant EP/P006094/1, Grant EP/S030964/1, Grant EP/S009647/1, and Grant EP/V006339/1. *(Corresponding author: Eleni Vasilaki.)*

Luca Manneschi and Eleni Vasilaki are with the Department of Computer Science, The University of Sheffield, Sheffield S14DP, U.K. (e-mail: e.vasilaki@sheffield.ac.uk).

Andrew C. Lin is with the Department of Biomedical Science, The University of Sheffield, Sheffield S10 2TN, U.K.

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2021.3102378.

Digital Object Identifier 10.1109/TNNLS.2021.3102378

Reservoir computing takes inspiration from the complex nonlinear behavior of recurrent neural networks and their abilities to process temporal information. Such a computing paradigm exploits the inherent complexity of a dynamical system, which is called the reservoir and that can be virtually or physically defined, to represent a stream of data into a high-dimensional space useful for learning. In the classical learning framework of such systems, training occurs on the read-out connections only, which connects the activities of the nonlinear reservoir components to the output neurons. The advantages of the reservoir computing framework are: first, the system can be physically defined as in [25]–[29], an aspect that can dramatically reduce the computational and energetic cost [30]; second, the response of the reservoir can exhibit memory over a range of timescales without training the internal dynamic of the system. In the latter aspect, the reservoir computing framework contrasts with that of standard recurrent neural networks, whose connectivity is trained through the computationally expensive backpropagation through time (BPTT) algorithm.

In this work, we model the reservoir as a recurrent network of leaky integrators [31] known as an echo state network (ESN). The connectivity between the nodes is represented through a random sparse fixed adjacency matrix, whose spectrum of eigenvalues allows the system to exhibit a wide range of timescales and efficiently represent temporal information. Previous works have explored alternatives to the typical Erdos–Renyi connectivity of the reservoir, imposing a regular structure as a delay line [32], where the nodes are connected unidirectionally composing a circular graph, or defining the graph connectivity through more complex topologies [33] as scale-free or small world. Furthermore, complex structures composed of multiple, interconnected ESNs have been recently studied, and the works [34], [35] demonstrate how the use of different hyperparameters for different ESNs can expand the range of timescales that are accessible by the system and, consequently, increase the quality of the reservoir representation for temporal tasks with complex temporal dynamic. In this regard, the work [35] gives insights into the understanding of hierarchical systems from the point of view of accessible timescales, offering a theoretical and practical analysis on how to tune the hyperparameters of the system to a considered task.

While ESNs are most traditionally adopted by the machine learning community for time-series prediction and forecasting [31], more recent works have studied their behavior for time-series classification [36]. In particular, [36] studies and compares different methods to exploit the representation of the reservoir as input to more complex machine learning algorithms as multilayer perceptrons or SVM. In this work, we formulate a different approach to improve the classification ability of reservoir systems in general while maintaining the low cost of computation that is intrinsic to the concept of reservoir computing. Analogously to the concept of firing thresholds, SpaRCe exploits learnable thresholds to optimize the level of sparsity inside the network, without disturbing the underlying network dynamics. Both the learnable thresholds and the read-out weights (but not the recurrent connections within the reservoir) are optimized by minimizing an error function that does not include any normalization term. We analyzed the learning rule derived from this error minimization and found that learning occurs by two antagonist factors: the first raises the thresholds proportionally to the correlated activity of the nodes (thus, silencing nodes that are correlated and, therefore, redundant), while the second lowers the thresholds of nodes that contribute to the correct classification. The novelty of our work lies in the fact that a sparsity level is reached due to the presence of online learnable firing thresholds, rather than to penalty terms [6], [37], [38], as well as in the detailed analysis on how such thresholds enable ESNs to perform competitively in tasks where their performance was lacking.

II. Methods

A. Standard Echo State Network

An ESN is composed of randomly connected neurons. The activity of such a system is commonly defined through the following equation [31]:

$$\mathbf{V}(t+\delta t) = (1-\alpha)\mathbf{V}(t) + \alpha f\left[\gamma\,\mathbf{W}_{\text{in}}\mathbf{s}(t) + \rho\mathbf{W}\mathbf{V}(t)\right] \quad (1)$$

where $\alpha = (\delta t/\tau)$ is the leakage term, τ defines the temporal scale of integration of the nodes, $\mathbf{V}(t)$ is the activity vector of the integrators, and $\mathbf{s}(t)$ is the input signal. $\mathbf{W}$ is the fixed sparse random matrix that describes the recurrency of the reservoir. The matrix $\mathbf{W}$ is random and sparse, corresponding to an Erdos–Renyi graph where the probability of a connection is p_{ER}. The eigenvalues of $\mathbf{W}$ are rescaled to be confined inside the unit circle of the imaginary plane, the necessary condition for the echo state property. Considering the parameterization of the network in (1), this condition is equivalent to the one proposed in [34] and guarantees that the eigenvalues of the associated, linearized dynamic system are also confined in the unit circle of the imaginary plane. The hyperparameter ρ (between 0 and 1) controls further the radius of the spectrum of the system's eigenvalues. Finally, γ is a gain factor of the input signal. The specific form of the input matrix $\mathbf{W}_{\text{in}}$ and the activation function f is task-dependent and will be specified in Sections III-A and III-B.

It is possible to control *a priori* the range of timescales that the reservoir exhibits by appropriately choosing α and ρ, as described in the methodology reported in Appendix A.1 and our earlier work [35]. In the standard ESN learning protocol, learning occurs exclusively on the read-out defined from a vector $\tilde{\mathbf{V}}$ that represents an ensemble of the reservoir activities $\mathbf{V}$. The output of the system is then computed through

$$\mathbf{y} = \mathbf{W}^o\tilde{\mathbf{V}} \quad (2)$$

$$E = \frac{1}{2}\left[\tilde{\mathbf{y}} - \mathbf{y}\right]^2 \quad (3)$$

where $\tilde{\mathbf{y}}$ denotes the desired output values and the output connectivity $\mathbf{W}^o$ is optimized to minimize the cost function E through ridge regression [31] or through gradient descent methods [39]. We note that, depending on the task and the learning framework, the $\tilde{\mathbf{V}}$ vector can be defined from different ensembles of activities of the reservoir across time. The idea to

exploit the dynamics of the ESNs, instead of using its activities at one step only, is particularly useful for classification tasks. Previous works have introduced different techniques to define a $\tilde{\mathbf{V}}$ vector from the ESN's dynamic: adopting PCA on the multidimensional response of the reservoir across time [36], introducing a learnable temporal kernel to define the read-out [40], or concatenating the past activities of the reservoir to define a higher dimensional vector $\tilde{\mathbf{V}}$ [39], [41]. In this work, we will exploit the latter approach, while other techniques, such as PCA, which reduces the dimensionality of the ESN's representation, are more desirable in the case of overfitting. In this regard, while the most spontaneous choice for time-series prediction would be $\tilde{\mathbf{V}} = \mathbf{V}(t)$, corresponding to the activity of the reservoir at the current time, it is also possible to expand the dimensionality of the system by including previous reservoir representations at times t_l and defining a vector $\tilde{\mathbf{V}} = \mathcal{C}(\{\mathbf{V}(t_l)\}_{t_l \varepsilon \mathcal{T}})$, where $\mathcal{C}$ denotes the concatenation operator and $\mathcal{T}$ denotes the ensemble of time steps t_l considered. This latter approach has been adopted also in physical reservoir models through the use of virtual nodes [42], [43]. Of course, such dimensionality expansion leads to an artificial increase of the memory of the system. However, the concatenation of previous activities does not guarantee the understanding of the dependencies among events that are distant in time because the memory of a past input signal could have faded away when a new stimulus $\mathbf{s}(t)$ comes. A scheme of this procedure for time-series classification can be found in Fig. 1(a).

B. Sparse Reservoir Computing

In contrast to previous models that define the output of the neural network through a read-out of the $\tilde{\mathbf{V}}$ vector, i.e., the activities of the reservoir considered for the learning process, we introduced another variable x_i for each dimension of $\tilde{\mathbf{V}}$, defined as follows:

$$x_i = \text{sign}\big(\tilde{V}_i\big)\text{ReLU}\big\{\big|\tilde{V}_i\big| - \theta_i\big\} \tag{4}$$

$$\theta_i = P_n\big(\big|\tilde{\mathbf{V}}_i\big|\big) + \tilde{\theta}_i \tag{5}$$

where ReLU stands for rectified linear unit, sign is the sign function (1 if $\tilde{V}_i > 0$, -1 if $\tilde{V}_i < 0$, and 0 if $\tilde{V}_i = 0$), and θ_i is a threshold that enables x_i to be sparse. Thus, the variable x_i is zero if the absolute value of the variable V_i is lower than the corresponding threshold. We term this variant SpaRCe for sparse reservoir computing. Of course, $\mathbf{x} = \tilde{\mathbf{V}}$ if $\boldsymbol{\theta} = 0$, which is the case where the formulated learning procedure coincides with the standard ESN paradigm. Considering (5), each threshold is composed by two factors.

1) $P_n\big(\big|\tilde{\mathbf{V}}_i\big|\big)$, defined as the percentile n of the distribution of activity of the ith component of $\big|\tilde{\mathbf{V}}\big|$ across a dataset, where the same value of n is applied to all dimensions of $\tilde{\mathbf{V}}$. Given that the response $\tilde{\mathbf{V}}$ of the reservoir to a specific input remains unchanged across learning, this term can be computed over the training dataset and maintained constant throughout the simulation. Thus, the role of $P_n\big(\big|\tilde{\mathbf{V}}_i\big|\big)$ is to initialize the thresholds from an initial condition, where the sparsity level of all dimensions of $\mathbf{x}$ is $n/100$. The thresholding mechanism introduced is symmetric with respect to zero, as is visible from the examples in Fig. 1(d) and (e). While panel D highlights the portions of the distributions that are positive after normalization, panel E shows the resulting distributions after application of (θ_i), but before learning, that is when $\tilde{\theta}_i \approx 0$. In this formulation, the value of n needs to be considered as an additional, interpretable hyperparameter. However, we will show that all the sensible choices of n will correspond to faster convergence and better performance of the formulated algorithm in comparison to the standard ESN learning paradigm across all tasks considered. We note that, even for P_0, SpaRCe and a vanilla ESN would differ because the threshold values for P_0 are nonzero, and the activities of the reservoir would be shifted. More specifically, choosing P_0 will lead to a shift to the distributions of $\big|\tilde{\mathbf{V}}\big|$, whose minimum value will now become approximately zero. Throughout this article, the same values of $P_n\big(\big|\tilde{\mathbf{V}}_i\big|\big)$ computed over the training dataset will be used for the validation and test datasets.
2) An adaptable component $\tilde{\theta}_i$ optimized through gradient descent learning rules. This parameter adapts the sparsity level for the considered task and can rediscover the standard learning paradigm in the case, where $\tilde{\theta}_i = -P_n\big(\big|\tilde{\mathbf{V}}_i\big|\big)\ \forall i$.

The additional complexity arising from (4) is summarized by Fig. 1(b), which depicts the difference between the read-out of a standard echo-state network and our formulation. Equation (4) acts as a normalization operator that directly controls sparsity in the reservoir representation and that, due to the learnable components $\tilde{\boldsymbol{\theta}}$, works as a feature-selection mechanism. In the latter sense, we can now demonstrate the interpretability of the updating equations on $\boldsymbol{\theta}$ derived from a gradient descent approach. For the case of a mean squared error function, the learning rule can be factorized in two terms

$$\Delta\theta_k = \eta_\theta\Big[\Delta\theta_k^{(1)} + \Delta\theta_k^{(2)}\Big] \tag{6}$$

$$\Delta\theta_k^{(1)} = \sum_{j=1}^{N_{\text{class}}} y_j W_{jk}^o \text{sign}(x_k) = \sum_{j=1}^{N_{\text{class}}}\sum_{l=1}^{N}\big[W_{jl}^o x_l\big]\big[W_{jk}^o \text{sign}(x_k)\big] \tag{7}$$

$$\Delta\theta_k^{(2)} = -W_{\tilde{j}k}^o \text{sign}(x_k) \tag{8}$$

where η_θ is the learning rate on the thresholds, $\tilde{j}$ refers to the correct output class for the sample considered, and $\mathbf{W^o}$ is the output connectivity matrix. Equations (7) and (8) are derived by assuming one-hot encoding (see Appendix B.2). Considering $x_k > 0$ (analogous considerations are valid for $x_k < 0$), the factor $\Delta\theta^{(2)}$ decreases (increases) the threshold value of nodes with $W_{\tilde{j}k}^o > 0$ $\big(W_{\tilde{j}k}^o < 0\big)$ that help (hinder) the network to reach the right classification. Thus, $\Delta\theta^{(2)}$ is driven by the output weight between the considered node (if it is active) and the desired class. In contrast, $\Delta\theta^{(1)}$ is a measure of the correlation of weighted activities between different nodes in the reservoir and increases (decreases) the thresholds of

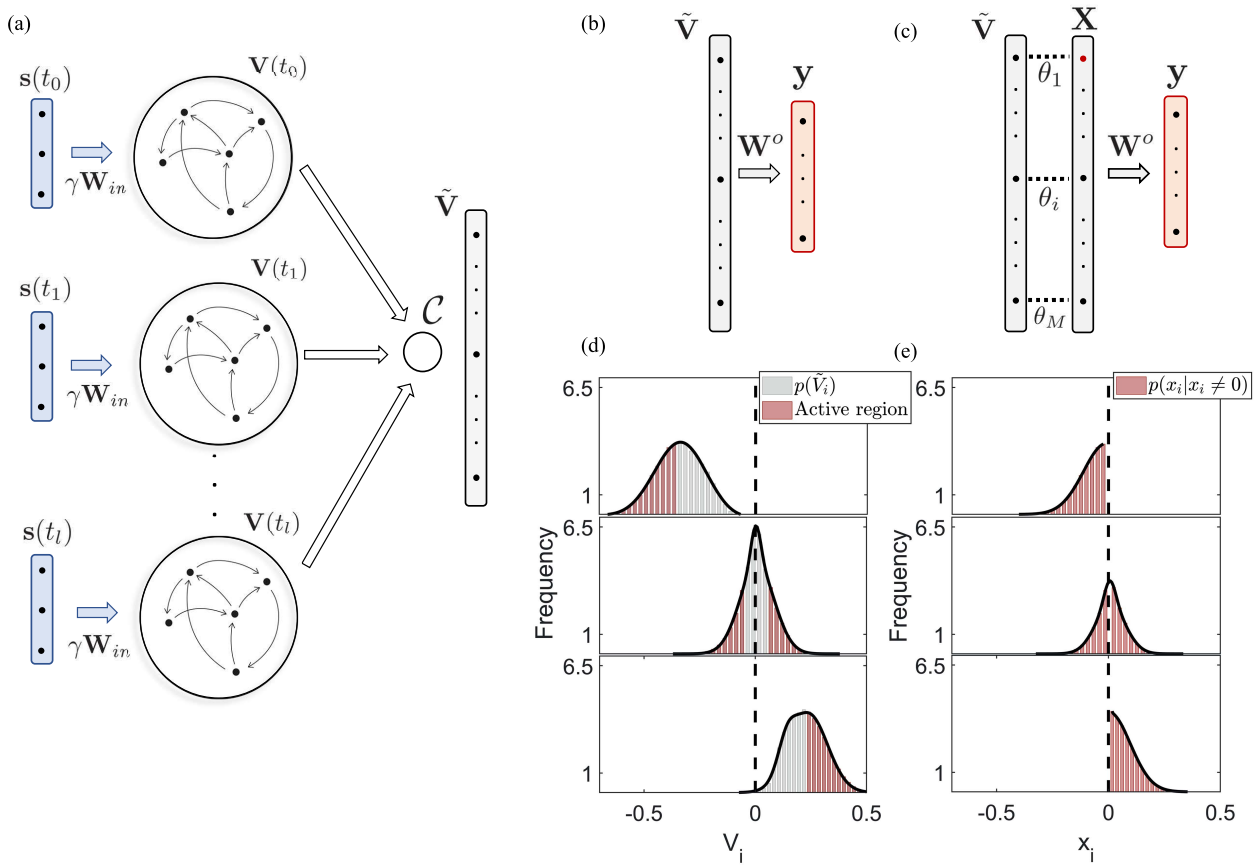


Fig. 1. Comparison between the basic reservoir computing framework and SpaRCe, which exploits the concepts of adaptable thresholds to introduce sparsity in the representation of the ESN. (a) Typical reservoir computing sampling paradigm for a time-series classification task. Time flows from top to bottom. The input signal $\mathbf{s}(t)$ is fed into the ESN through the input connectivity matrix $\gamma\,\mathbf{W}_{\text{in}}$. An ensemble of the activities $\mathbf{V}(t)$ across time is selected and concatenated to compose a vector $\tilde{\mathbf{V}}$, which is then used to define the read-out. Typical choices of ESN activities used to define the read-out are $\tilde{\mathbf{V}} = \mathbf{V}(T)$, where T denotes the final temporal step of the input sequence, or $\tilde{\mathbf{V}} = \mathcal{C}(\{\mathbf{V}(t)\}_{\forall t})$, denoting the concatenation of all the reservoir dynamic across the temporal length of $\mathbf{s}(t)$. Of course, while the first approach relies on the ESN intrinsic memory capacity, the latter case corresponds to a dimensionality expansion that contributes artificially to the memory of the system. Both these approaches and intermediate cases, i.e., where the dynamic of the reservoir across time is sampled with low frequency to compose the read-out, will be exploited in the tasks studied. (b) and (c) Comparison between the typical reservoir computing read-out (b) and SpaRCe model. Considering that $\tilde{\mathbf{V}}$ is used for the read-out, we define a threshold for each dimension of $\tilde{\mathbf{V}}$. The result of this approach is the definition of different thresholds across time in the case where the representation is enriched through concatenation of the history of activities of the reservoir. In this way, the approach defined is general and can be applied regardless of the technique used to define the representation $\tilde{\mathbf{V}}$ for the read-out of the system. Furthermore, the initialization of different thresholds across time can be helpful to the case where there is temporal drift in the dataset, leading the activity $\mathbf{x}$ of the reservoir to exhibit a more stationary behavior across time. (b) ESN output $\mathbf{y} = \mathbf{W}^o\tilde{\mathbf{V}}$ is responsible for the classification process of the example sequence $\mathbf{s}(t)$. In this paradigm, learning occurs exclusively on $\mathbf{W}^o$. (c) Scheme of the SpaRCe model. Thresholds are introduced at the level of the $\tilde{\mathbf{V}}$ vector, leaving unaffected the dynamic of the reservoir and making the approach applicable to any physical or virtual reservoir model. Each threshold value is composed by a normalization term $\mathbf{P}_n(|\tilde{\mathbf{V}}_i|)$, defined as the nth percentile of the activity distribution of the ith component across the data, plus an adaptable term $\tilde{\theta}_i$ (see text for more details). (d) and (e) Activity distributions of $\tilde{\mathbf{V}}$ and $\mathbf{x}$ correspondingly before the training process, where $\tilde{\theta} \approx 0$, for three example nodes. Each distribution is fit through two Gaussians for clarity purposes. The highlighted red region in (d) corresponds to the values for which the nodes would be active if the normalization mechanism proposed in (4) would be applied (percentile P_{50} in this case). From (e), it is clear that (4) also shifts the activity distributions acting as a normalization mechanism.

neurons that have coherent (opposite sign) contributions and, therefore, reduces redundancy in the reservoir.

We examined the two factors $\Delta\theta^{(1)}$ and $\Delta\theta^{(2)}$ across learning for an example of sequence classification and for different initial sparsity levels. These are plotted in Fig. 2(c), with $\Delta\theta^{(2)}$ on the positive y-axis and $\Delta\theta^{(1)}$ on the negative y-axis. The two forces are almost symmetric, but their slight imbalance provides the direction to change the threshold values. Indeed, if the starting sparsity level is high, the total force is negative, and the factor $\Delta\theta^{(1)}$ dominates, while, if the sparsity level is low, the correlation term $\Delta\theta^{(2)}$ wins, and the thresholds increase on average [compare P_{60} and P_{95} in Fig. 2(d)], which shows the effect of learning the thresholds in terms of average percentile (sparsity) change across the network. The reason that the magnitudes of the forces are larger for a higher starting sparsity [see Fig. 2(c)] is that stimulus representations overlap less when the sparsity level is higher and neurons are more specialized, i.e., preferably fire for one pattern over the others. This leads to a more coherent sign of the output weights of the nodes belonging to a cluster toward a specific class, which increases $\Delta\theta^{(1)}$ according to (7). A similar analysis of the learning rule for a cross-entropy cost function is reported in Appendix B.2. Fig. 2(a) and (b) describes the benefits due to the application of the proposed normalization mechanism: first,

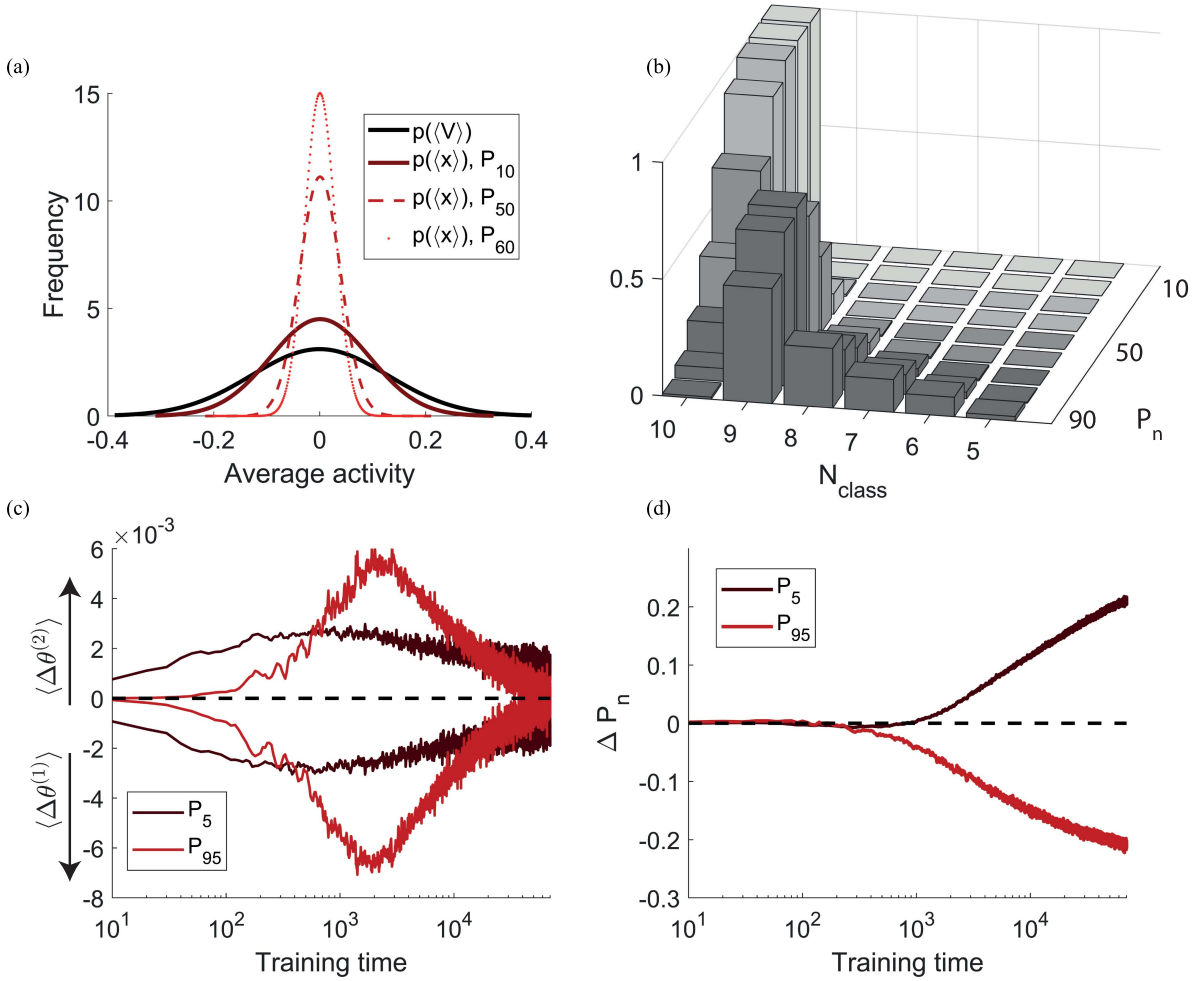


Fig. 2. SpaRCe read-out proposed leads to decreased variability in the ESN representation, to specialized responses, and to a interpretable learning rule that acts as feature selection mechanism. The learning rule for the thresholds is driven by the imbalance between two antagonist forces. (a) Distributions of the average activities of nodes across an example dataset (MNIST, Section II-C) for the standard ESN and for SpaRCe as the percentile of the normalization mechanism changes at the beginning of training $\left(\text{when } \tilde{\theta} = 0\right)$. SpaRCe decreases the variability of the activity distributions, acting as a normalization mechanism. (b) Frequency of active nodes for different starting sparsity levels and different number of classes for a classification task with ten classes (MNIST, Section II-C) before training $\left(\tilde{\theta} = 0\right)$. We note that N_{class} refers to the total number of classes to which a node responds. In particular, nodes that are active for $N_{\text{class}} = 10$ are responding to all ten classes, while nodes that react for $N_{\text{class}} = 6$ are responding to six classes only. As sparsity (P_n) increases, nodes respond to a smaller number of classes becoming more specialized. (c) Analysis of the two forces $\Delta\theta^{(1)}$ and $\Delta\theta^{(2)}$ involved in the learning rule for the thresholds. The positive y-axis shows a running average of $\Delta\theta^{(2)}$, while the negative y-axis shows a running average of $\Delta\theta^{(1)}$. $\langle\rangle$ indicates averaging across all neurons. $\Delta\theta^{(2)}$ increases (decreases) the threshold values for nodes that are equally (differently) contributing to the classification process. $\Delta\theta^{(1)}$ decreases (increases) the threshold values due to the positive (negative) contribution of the output weights connected to the correct output. Colors correspond to initial conditions (P_{60}, P_{95}). (d) Average cumulative change of a threshold in terms of percentile change due to adaptation of $\tilde{\theta}$. If the starting level of sparsity is suboptimal and low (high), the average percentile change is positive (negative).

it directly controls sparsity and introduces specialized neurons [see Fig. 2(b)]; second, it shifts the activity distributions $\tilde{\mathbf{V}}$ of the network into a more stereotyped response $\mathbf{x}$, avoiding the possibility that learning could be dominated by the nodes with highest activities [see Fig. 2(a)]. Finally, it stabilizes the learning process for a wide range of possible learning rates that would not be accessible without thresholds (see Appendix B.2, Fig. 2). This specific formulation allows the model to use local information to learn the threshold values and optimize sparse representations. We note also how (4) does not affect the timescales of the network and, consequently, preserves the idea of reservoir computing as a fixed, dynamically rich representation. Furthermore, the method formulated constitutes a computationally inexpensive procedure that is easily applicable to any type of reservoir, virtually or physically defined.

C. Benchmarks

We tested the proposed model on the following benchmarks, comparing its performance to the standard read-out of an ESN.

1) A biologically inspired task to test the storage capacity of the system in the memorization of associations between sequences and desired output values. In this case, the model is tested on the same associations used for training but with different realizations of multiplicative white noise inserted on the sequences. The dependence of SpaRCe on the starting sparsity level is

analyzed along with comparisons of the performance achieved with other methods for reading out from the ESN representation, including deep feedforward networks.

2) Three variants of classification tasks based on the Modified National Institute of Standards and Technology database (MNIST), where data are presented to the system as temporal varying sequences. In the first task, we studied the performance of the model when each image, composed by 28×28 pixels, is presented column by column as a 28-D sequence of 28 time steps (MNIST [41]). In the second, the input is processed in the same way, but a random permutation of the pixels is applied to all data to make the task more challenging and to randomize the structures of the images (pMNIST). In the third, we applied a random permutation of the data as in the second paradigm, but each image is presented pixel by pixel as a 1-D sequence of 784 temporal steps (permuted sequential MNIST (psMNIST) [44]). When the initial conditions of the model are studied, the performance is computed and shown on the test set as P_n varies for comparison, but, to select the best P_n value and report the highest performance, we used a validation dataset. Of course, we selected the hyperparameters corresponding to the highest accuracy on the validation set and then computed the performance on the test dataset.
3) Two tasks that involve sequential learning. In the first case, we applied ten different permutations to the MNIST data and trained the model processing the tasks one after the other [45], [46]. In the second, the network is trained on the MNIST data belonging to different classes sequentially [45], [46]. In both cases, the network can access a specific dataset (corresponding to a permutation or a class) only once during training. As before, a validation dataset is adopted when selecting the best hyperparameters (in particular P_n) of the proposed model.

III. Results

A. Threshold Learning Increases Storage Capacity

We evaluated the performance of a standard ESN and SpaRCe in classifying an ensemble of sequences $\{\mathbf{S}_i\}_{i=1,\ldots,N_{\text{seq}}}$ of three successive stimuli, where the dimensionality of the signal is $N_{\text{In}} = 24$. Each stimulus of a sequence is derived from the simulated response of $N_{\text{In}} = 24$ projection neurons (PNs, in the fly olfactory system) to 110 different odors [47], [48]. This simulated activity, which we call $\mathbf{s}^{\text{HO}}$ (HO for Hallem-Olsen), has previously been used in computational analyses of fly olfaction [49]–[51].

Sequences are generated to test the storage capacity of the system in memorizing associations between input signals and desired output values. The procedure for building different sequences from single stimuli is described in the Appendix (see Appendix C.1, Fig. 1), but, essentially, it guarantees that each sequence has to be classified as independently as possible from other sequences, and that there are no correlations of elements among different inputs that can inform the classification process. Thus, the system can only memorize the associations among a specific succession of elements and the corresponding output value.

There are three stimuli in a sequence; each of them is presented for a time interval $\Delta t = 0.1$ s in order to allow the network to integrate the information. The total duration of an input sequence is $T = 0.3$ s. Given a sequence $\mathbf{s}_i(t)$, we inserted multiplicative white noise to each separate dimension to make the task more complex. Thus, the ith dimension of the final sequence $\mathbf{S}_i(t)$ to be classified is $S_i(t) = s_i(t) + \sigma_s \xi_i(t) s_i(t)$, where $\xi_i(t)$ is a Gaussian distributed random variable with zero mean and unitary variance, and σ_s is the noise variance.

For this specific task, the activation function f of (1) is a rectified linear unit, and the connections of the input adjacency matrix $\mathbf{W}_{\text{in}}$ follow a lognormal distribution. This particular form of $\mathbf{W}_{\text{in}}$ is inspired by the biological results in [13], [52], and [53]. In this case, $\tilde{\mathbf{V}} = \mathbf{V}(T)$, meaning that only the activities at the last temporal step of a sequence are adopted for the read-out, and the memory of past events is left to the internal dynamic of the ESN. A schematic representation of the task can be found in the Appendix.

We first investigated how the model depends on initial sparsity and found an optimal sparsity level for the task. We initialized the network with different initial sparsity percentiles (i.e., thresholds at different percentiles of the $\mathbf{V}$ distribution, $n = [10, 20, 30, 40, 50, 60, 70, 80, 90]$), and tracked the mean square error over the learning process [see Fig. 3(a)]. Errors decreased as learning proceeded, but, at each time point, the lowest error occurred for an initial sparsity of about 50%. Furthermore, models initialized with sparsity values other than 50% (an explanation of why 50% can be found in the Appendix) converged toward 50% as training proceeded, as shown by the black dashed lines connecting dots of training instances from the top to the bottom of the graph [see Fig. 3(a)]. This shows how the learning rule pushes the percentage of active nodes toward the optimal sparsity level.

Notably, the error is smallest when specialization is highest (for a definition of specialization, see Appendix A.2 (A5); conceptually, specialization represents to what extent a node is active for stimuli of one class, not stimuli of other classes), as shown in Fig. 3(b), which reports the error as a function of sparsity and specialization. For all training instances analyzed, the lowest error corresponds to the highest specialization value. Thus, specialization provides a systematic way to choose the starting condition of the network. Indeed, it is possible to select the thresholds using the percentile value that yields the highest specialization measure. However, there is no need to excessively fine-tune the initialization since the learning rule will optimize the threshold values anyway. We note also that this simulation is performed through a simple gradient descent algorithm and that the model's dependence on initial conditions can be ameliorated by using more complex optimizers, as will be shown in Section III-B.

Finally, we compared the performance of the SpaRCe model with the following.

1) ESN without thresholds, where the same online learning is applied to the output weights W^{out} only. We note

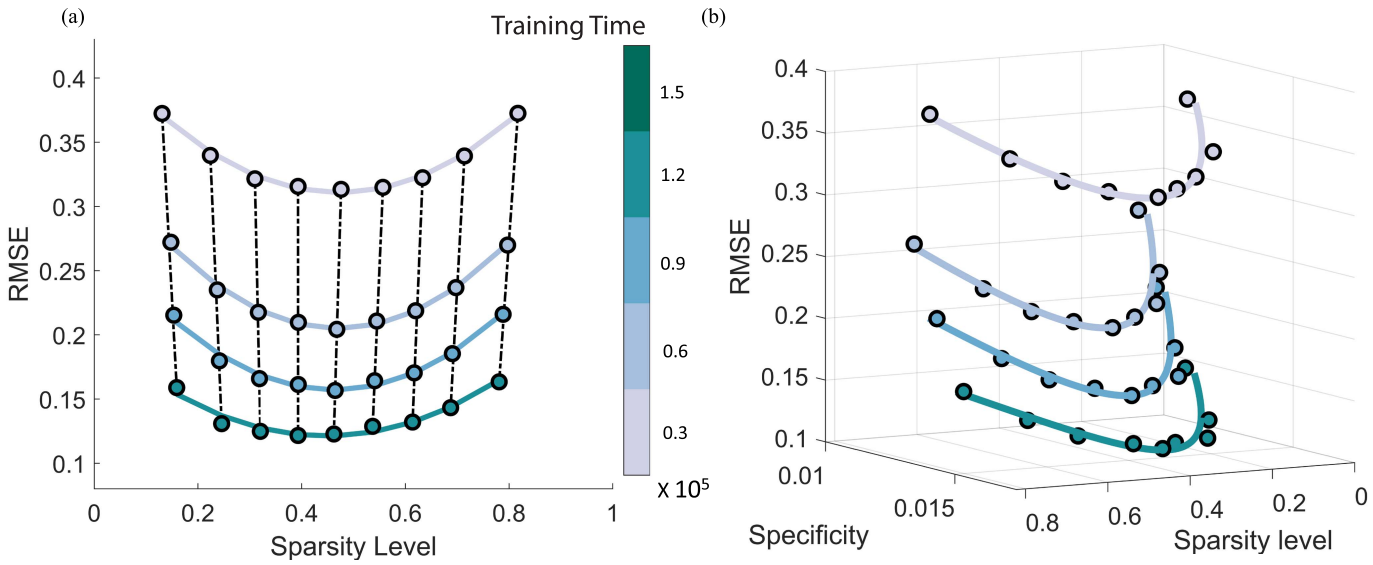


Fig. 3. Learning process modulates the network's sparsity level toward an optimal percentage of active nodes. (a) Performance as a function of sparsity for different training instances of the model (a color represents a specific training time, which increases from top to bottom). For each instance, the results are fit with a second degree polynomial $\left(\chi^2 = 3 \times 10^{-4},\ R^2 = 0.98\right)$, demonstrating the existence of an optimal percentage of active nodes, which is around 50% for the sequence classification task (note that the optimal sparsity level is task-dependent). The dashed line connecting the results for diverse training time highlights the change in the sparsity level achieved through the learning rule. (b) Performance as a function of sparsity and specialization. The best performance corresponds to the highest specialization values for all training instances, demonstrating the interpretability of the model.

that the algorithm SpaRCe learns N more parameters (the thresholds) in comparison to the ESN without thresholds.

2) Hidden layer (HL), where we added a full HL of N_h nodes on the top of the reservoir. This approach learns an additional connectivity matrix between the reservoir and the HL, dramatically increasing the number of parameters by a factor of approximately $N_h N$.
3) ESN with online learning and L_1 or L_2 regularization terms on the output weights.

The SpaRCe model outperforms the standard ESN with or without the regularization terms, based on classification accuracy and root mean square error [see Fig. 4(a), (c), and (d)]. This advantage is consistent across different levels of external noise (σ_s) and different numbers of stimuli [see Fig. 4(c) and (d)]. Furthermore, SpaRCe performs comparably to a network with an additional full HL with $N_h \approx 100$ nodes even though the HL dramatically increases the number of learnable parameters compared to SpaRCe [see Fig. 4(a), (b), and (d)]. In comparison to the addition of an HL, the model SpaRCe provides a cheap formulation to achieve an optimal and reliable sparsity level [see Fig. 4(b), where the number of learnable parameters for the models is reported with the corresponding performance].

In general, when introducing an HL of N_h neurons trained with backpropagation, for a network of N_o output neurons, we would learn a number of parameters equal to $N \times N_h + N_h + N_h \times N_o + N_o$ (weights + biases + output weights + biases), which can be approximated by $N_h \times N$ (assuming $N_o \ll N$, i.e., $1000 \times 100 + 100 + 100 \times 2 + 2 \approx 10^5$ for the case with $N_h = 100$), while, for a classical reservoir, we learn $N \times N_o + N_o \approx N \times N_o$ $\left(1000 \times 2 + 2 \approx 2 \times 10^3\right)$, and for SpaRCe, $N \times N_o + N + N_o \approx (N_o + 1) \times N$ $\left(1000 \times 2 + 1000 + 2 \approx 3 \times 10^3\right)$. While adding an HL goes against the principle of ESN of exploiting the network dynamics while using simple learning methods, it remains an interesting comparison for quantifying the efficiency of the proposed method. With $N_h = 100$ and while $N_o \ll N_h$, our proposed thresholded architecture is efficient in terms of numbers of learnable parameters in comparison to adding a fully trained HL to an ESN.

We conclude that the SpaRCe model considerably improved the performance and the convergence time of a reservoir on this biologically inspired benchmark task, with the relatively small overhead of N additional parameters, one per reservoir node. Finally, we investigated the cost of the additional complexity of SpaRCe, composed by an initialization procedure (the quantile operation over the training data) and threshold learning, in terms of computational time. Considering a training example for this task, SpaRCe required approximately 1234.6 s, while an ESN required 1233.98 s. The values of the hyperparameters adopted for the task and more details about the evaluation of the computational time required for different models, with the specifics of the computer used for the simulations, can be found in Appendix E.

B. Threshold Learning Increases Performance

In this section, we faced three variants of classification tasks using the MNIST dataset. Each image is fed into the network sequentially either one column at a time or one pixel at a time to make the task temporally dependent. Thus, one written digit corresponds to a sequence of 28 time steps of a 28-D input in the column by column paradigm or to a sequence of 728 time steps of a 1-D input (psMNIST).

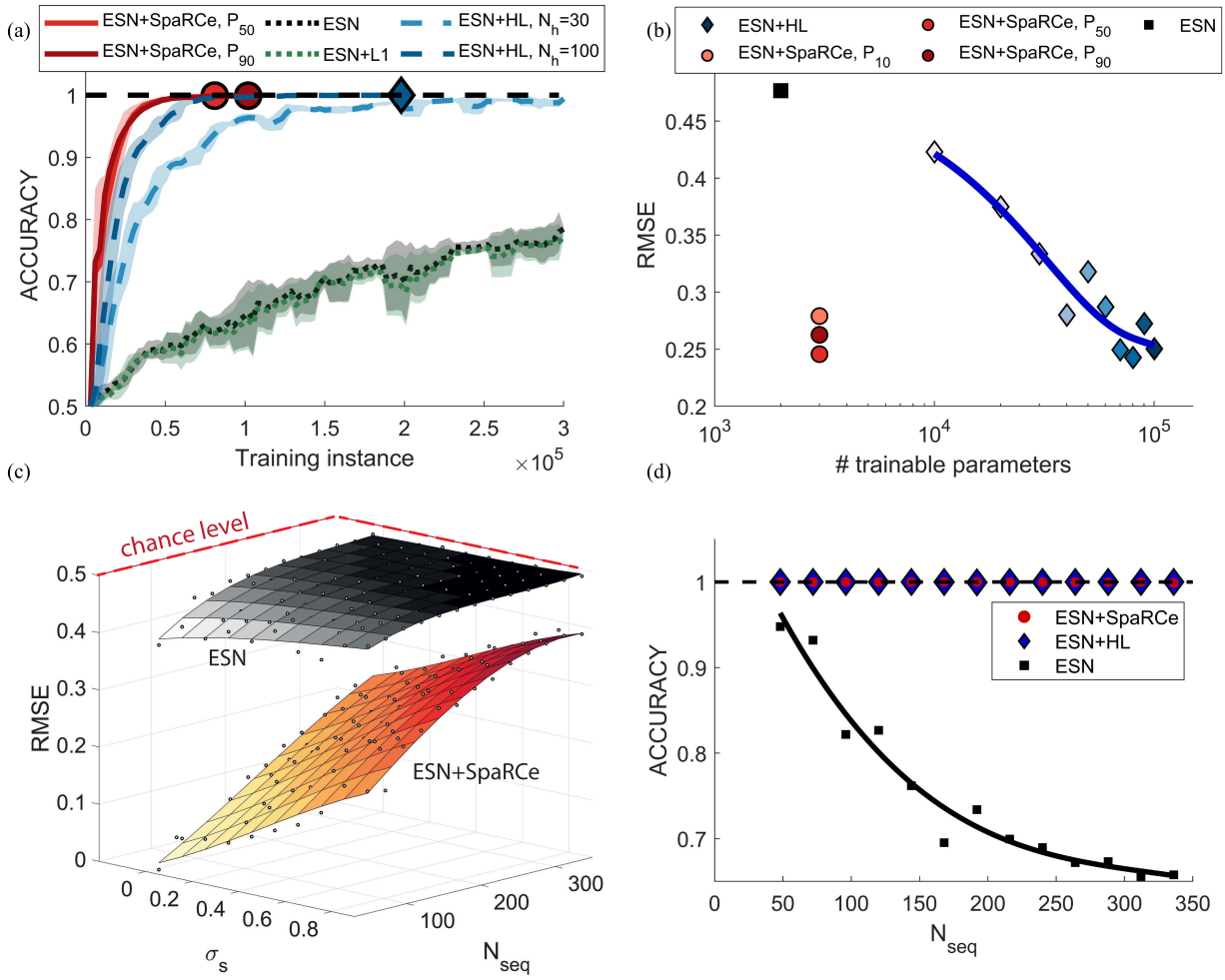


Fig. 4. SpaRCe algorithm increases the memory capacity of the ESN and the stability of the found solution. (a) Classification accuracy and root mean square error of the models for a case where the number of sequences to be classified is 192. Each minibatch corresponds to the presentation of 20 training samples. In this case, the performance of SpaRCe is related to a starting $P_n = 50$, while the HL has $N_h = 100$ nodes. Dots correspond to the training instances in which the considered models solve the task (for the models that can solve the task), thus showing the training speed of the algorithms analyzed. (b) Performance of SpaRCe for three different starting sparsity levels (red and orange), for a standard ESN (black) and of HL (blue) for diverse N_h nodes. The x-axis reports the number of trainable parameters and is shown on a logarithmic scale. The graph shows how SpaRCe is able to reach good performance while maintaining a low number of trainable parameters. (c) Comparison of the root mean square error for the ESN and the ESN with thresholds as the external noise σ_s and the number of stimuli vary. The introduction of thresholds leads to a robust result. (d) Performance as the number of inputs to be classified increases, for the HL model with $N_h = 100$ (blue), SpaRCe (red) with $P_n = 50$, and a standard ESN (black). SpaRCe and HL solve the tasks considered, but the latter uses a number of trainable parameters (reported along with the performance with the color scheme that reflects the referred model) that are two magnitudes higher than the first. The accuracy of the standard ESN drops considerably as the number of sequences increases. The inset shows the root mean square error for SpaRCe and HL, varying the starting condition P_n and the number of nodes N_h, respectively. The errors shown correspond to the training instance in which the fastest model reaches perfect classification accuracy. Numbers reflect the number of trainable parameters for example cases.

The tasks considered require representations over multiple timescales to achieve competitive performance and, thus, serve as benchmarks to study the ability of the network to discover temporal dependencies. In this regard, the psMNIST problem constitutes a challenge and a benchmark for recurrent neural networks and has been often considered as a metaphor for practical applications where long temporal dependencies are present. Prime examples of such applications are in the fields of language, decision-making, and reinforcement learning in partially observable Markov processes. We want to emphasize how all the tasks considered, even the MNIST and pMNIST, are processed by the network temporally. For this reason, the performance reported of static networks, as multilayer perceptrons (MLPs) and convolutional neural networks, serves only as a baseline since these networks could not solve the temporally dependent task faced. The activation function f used for these tasks is a hyperbolic tangent.

1) MNIST, Column by Column: The application of ESNs on this specific task was previously analyzed in [41], in which the original dataset was preprocessed and augmented by resizing and deforming the original images. Without such a preprocessing, the ESN could not outperform a simple perceptron [41]. In contrast, in this work, we use the original dataset, without any additional transformations. In this experiment, the pixels of each image are fed to the ESN sequentially column by column; thus, the input signal $\mathbf{s}(t)$ corresponding to an example image is a 28-D sequence of temporal length 28. In this case, $\tilde{\mathbf{V}} = \mathcal{C}(\{\mathbf{V}(t)\}_{\forall t})$, meaning that all the dynamics of the

recurrent network across time are used to define the read-out of the system. The cost function adopted is a sigmoidal cross entropy

$$E = -\left[\sum_j \tilde{y}_j \log\big(\sigma\big(y_j\big)\big) + \big(1-\tilde{y}_j\big)\log\big(1-\sigma\big(y_j\big)\big)\right] \quad (9)$$

which is analyzed in Appendix B.2. The optimizer used is Adam [54]. We first tested the SpaRCe model with various initial sparsity levels [see Fig. 5(a), different colors]. Regardless of the initial condition, the final performance was similar, as was the final level of sparsity [the size of dots in Fig. 5(a) shows the percentage of active nodes in the network]. The error achieved by SpaRCe is 1.9%. The hyperparameter values are given in Appendix E. The model reaches performance levels comparable to those achieved with a three-layer neural network with backpropagation [55]. However, the task faced here with SpaRCe (column-by-column MNIST) is more challenging than the common approach used to train neural networks on the MNIST dataset, in which the whole image is fed into the network at once. The performances of convolutional neural networks and MLPs are reported in Table I as a benchmark for the results obtained with ESN with SpaRCe. Convolutional neural networks are best suited for image classification problems, and nevertheless, SpaRCe is only 0.2% from the lower reported performance. In this comparison, we have not matched the number of parameters between SpaRCe and the other models; we have used a reservoir with 1000 neurons. To demonstrate the importance of thresholds in the SpaRCe model, we compared the performance of SpaRCe to that of an ESN without thresholds trained online with the same optimizer, using a learning rate optimized through grid search. SpaRCe outperformed the ESN in both classification accuracy and convergence time [see Fig. 5(b), "MNIST"]. We also found that the normalization mechanism introduced due to SpaRCe stabilizes the learning process for relatively higher learning rates versus standard ESNs. For this simulation, the total computational time required by SpaRCe is 579.81 s while, for a standard ESN, is 474.51 s [as before, see Appendix E for more information]. As a second task, we applied the same model to the MNIST dataset where the pixels of each image are reordered through a permutation of the data. Each image is again fed one column at a time. Again, SpaRCe outperformed the ESN without thresholds (see Fig. 5(b), "pMNIST").

2) Permuted Sequential MNIST (psMNIST): We next analyzed the performance of the SpaRCe model and ESNs, in general, on the psMNIST task, which has become a standard benchmark for recurrent neural networks [44]. In this case, each image is reordered through a fixed permutation and fed into the recurrent network one pixel at a time. The task is particularly challenging because temporal dependencies must be learned between widely separated time steps of an image. Since each sequence is a succession of 784 pixels, we decided to sample the dynamic of the reservoir across time at constant steps multiples of 28, defining $\tilde{V} = \mathcal{C}(\{\mathbf{V}(t) : 28|t\})$, where $28|t$ indicates that t has to be divisible by 28. This sampling procedure at constant temporal time steps could be suboptimal compared to an approach where the most informative time steps are selected. However, an analysis of optimal sampling procedures goes beyond the scope of this article.

TABLE I

ASTERISKS * ** INDICATE THAT THE RESULTS ARE TAKEN FROM [55] AND [44], RESPECTIVELY. MLP STANDS FOR MULTILAYER PERCEPTRON (THREE LAYERS AND WITHOUT DISTORTIONS), CONV. FOR SINGLE CONVOLUTIONAL NETWORKS (WITHOUT DISTORTIONS), NRU FOR NONGATED RECURRENT UNIT, AND LSTM FOR LONG SHORT TERM MEMORY. THE SUPERSCRIPTS IN ESN^2 AND SPARCE^2 INDICATE THAT EACH NETWORK WAS COMPOSED BY TWO RESERVOIRS (SEE TEXT). THE PARAMETER "RESERVOIR SIZE" CORRESPONDS TO THE TOTAL NUMBER OF NEURONS IN BOTH RESERVOIRS (SEE ALSO APPENDIX E)

Results			
MNIST			
ESN	95.2		
$SpaRCe$	98.1		
MLP	97.0|98.5*		
$Conv.$	98.3|99.6*		
psMNIST			
Reservoir size	500	1000	1500
ESN^2	95.6	95.9	96.3
$SpaRCe^2$	96.2	96.6	96.9
$LSTM$	89.9**		
NRU	95.4**		

Our first attempts to solve the task with a single ESN gave performance comparable to or worse than a standard perceptron model trained on the whole image because the ESN could not simultaneously discover long-time dependencies and quickly adapt to new inputs. Indeed, in any network of randomly connected low pass filters, such as ours, it can be difficult to associate events that are distant in time, as it may be impossible to find a workable balance in the tradeoff between keeping a memory of past events (which requires each node's activity to have a long time constant, i.e., slow decay) versus allowing the network to evolve dynamically over time (which requires a short time constant, i.e., fast decay).

To overcome this difficulty, we used an architecture composed of two reservoirs. The first has 200 nodes with fast time constants, and the second has 300 nodes with slower time constants (the parameters for the two reservoirs are reported in Table II). The faster reservoir signals unidirectionally to the slower one, and nodes of both reservoirs are used for the read-out. This type of structure, where the timescales of the first network are faster than the second, was found to be optimal in previous works [35]. The sparsity level in the connectivity of the faster reservoir is important: it regulates a tradeoff between too much connectivity (the relation between the input information to the second reservoir and the input signal becomes too complex) versus not enough connectivity. The best performance arises when the network has the shortest path lengths between two nodes that could permit a sufficient amount of memory.

Using this architecture, SpaRCe outperforms published methods despite using a much simpler training algorithm. As with MNIST and pMNIST, on psMNIST, SpaRCe reaches higher accuracy and converges faster than the threshold-less ESN [see Fig. 5(b)]. To compare SpaRCe with published models, two ESNs with a total of $N = 500$ nodes (thus, 154×10^3 parameters) were adopted. The model gave an accuracy > 0.96, higher than the best, more complex recurrent neural networks that exploit BPTT, whose performances are 0.954 (NRU) and 0.899 (LSTM) [44] (see Table I) with a comparable number of parameters $\left(\approx 167 \times 10^3\right)$. While BPTT needs to unroll all the dependencies of the neural network activities across time, ESNs have the advantage to train a much simpler perceptron on top of the reservoir representation. In particular, considering that the dynamic of a reservoir across datasets can be computed once only and then used to train a high-dimensional perceptron, the computational cost of ESNs is lower than the computational cost of RNNs. We emphasize that the procedure of concatenating previous temporal representations is not simply a shortcut, but it is necessary to increase the dimensionality of the representation in order to solve complex machine learning tasks through a reservoir computing approach. Indeed, the idea behind reservoir computing is to exploit the temporal dynamic of a system as a fixed and higher dimensional representation that allows it to separate the classes of a classification task through a hyperplane. This approach contrasts with the learning process of a recurrent neural network with BPTT, which trains the dynamics of the system and draws a nonlinear manifold to solve the classification task. Furthermore, the concatenating procedure does not guarantee any understanding of the long temporal dependencies among pixels that are necessary to effectively solve the problem. It is, therefore, of interest that ESNs using SpaRCe could perform comparably to state-of-the-art recurrent networks, whose parameters are trained via a far more complex algorithm, BPTT. Finally, we repeated the experiment by changing the total number of nodes in the network, therefore increasing the dimensionality of the representation and the number of trainable parameters. In such a way, it is possible to understand if the performance difference between SpaRCe and vanilla ESN can be alleviated by simply increasing the network size of the latter. The results of this analysis are reported in Table I for three different numbers of nodes ($N = 500, 1000, 1500$). Since the model is composed of two reservoirs, in this case, N indicates the total number of nodes in the network (see Appendix E for more details). The standard ESN reaches the accuracy of SpaRCe only when its number of trainable parameters is approximately tripled (similar performance of SpaRCe^2 with $N = 500$ and ESN^2 with $N = 1500$), while the performance difference remains unaltered if we increase the network size for both models. The difficulty found in the attempt to compensate for the performance improvement of SpaRCe permits us to evaluate the relatively small difference between the two models (ESN and SpaRCe) and demonstrate to demonstrate how the impact of the proposed model is meaningful even on the psMNIST. This result, and the small computational time required by SpaRCe (reported and compared to ESN gradient descent training in the Appendix), shows how the proposed model constitutes a relatively inexpensive procedure considering its impact on terms of performance benefit.

C. SpaRCe Alleviates Catastrophic Forgetting

Catastrophic forgetting refers to the inability of standard neural networks to learn different tasks sequentially. If a neural network is trained on a specific dataset and then retrained to perform a novel task, it will probably forget what it has learned before. This unsolved problem [45] is critical for the future development of neural networks in general. Previous research formulated models that mitigate catastrophic forgetting, using a variety of techniques categorized by Kemker *et al.* [45]. These techniques include regularization [impose a cost to changing the weights that contribute to previous tasks, as in elastic weight consolidation (EWC)], rehearsal (replaying previously learned data during subsequent training, as in GeppNet), and sparse coding (reducing the fraction of active nodes, as in the fixed expansion layer (FEL) model, and hard attention to the task (HAT) [46]). Sparse coding is also the approach that we use here, with SpaRCe. However, in contrast to these techniques that exploit additional information, such as model awareness of the task identity, and the computation of *ad hoc* quantities, such as the importance of specific parameters (EWC) or nodes (HAT) for a given task, we will demonstrate how the application of (4) alone with a high starting sparsity level can alleviate catastrophic forgetting. Notably, the same methodology applied in the previous simulations, to improve convergence time and performance of reservoir computing, can improve the ability of an ESN to learn different tasks sequentially. The difference between the application of SpaRCe in a single task and a sequential tasks paradigm is the tuning of the hyperparameters of the proposed model, in particular, the percentile value n and the learning rates η_W and η_θ. We will demonstrate how sparsity regulates a tradeoff between initial learning speed versus memory retention because high sparsity decreases overlap among representations, which prevents new learning from disrupting old memories, whereas low sparsity means more nodes are active, allowing memories to be formed faster on new tasks.

This tradeoff between initial learning versus preventing forgetting is studied in detail for two paradigms that are commonly used to measure catastrophic forgetting in neural networks.

1) *Sequential Data Permutations:* A different permutation is applied to the considered dataset N_{task} times. These new N_{task} reshuffled datasets are then learned sequentially by the system. In our simulation, $N_{\text{task}} = 10$. In such a scenario, the complexity of the different datasets is the same, and we trained SpaRCe for approximately two epochs for each task.
2) *Sequential Classes:* The first task is composed of the data corresponding to half of the possible classes, while the other classes are trained sequentially. Since, in the dataset considered, there will be ten classes, the number of tasks that the network has to learn sequentially is six. In this case, we trained the model for approximately one epoch for each task.

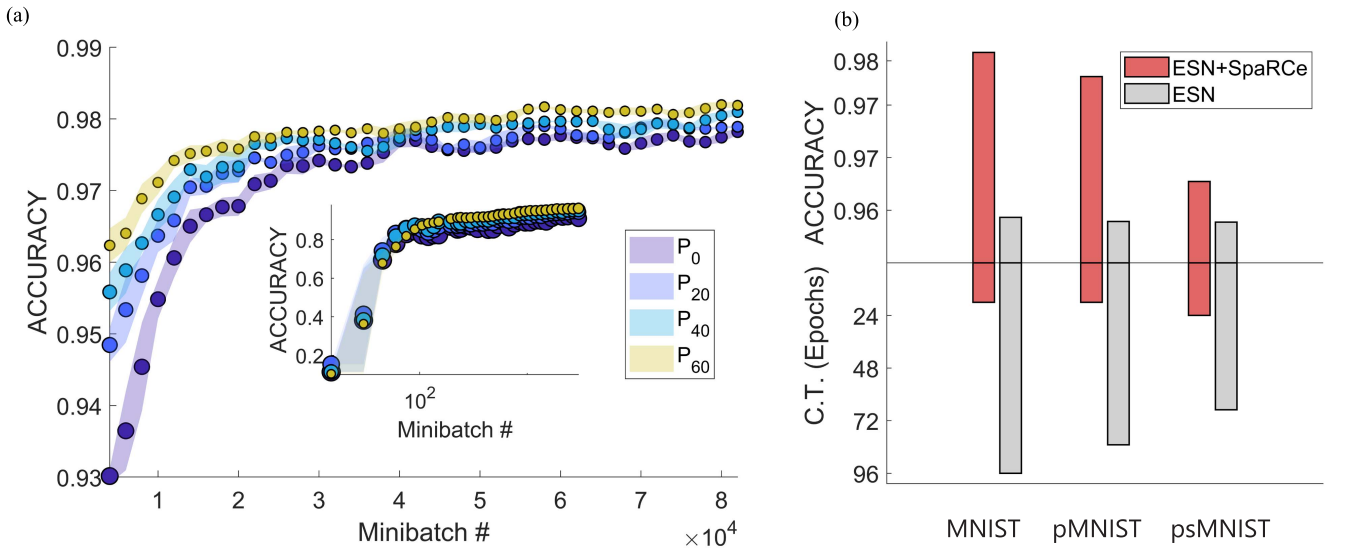


Fig. 5. SpaRCe model shows comparable performance to a 2/3 HL neural network on the MNIST dataset and accuracy comparable to more complex RNNs trained with BPTT on the psMNIST task. On the contrary, the standard ESN with online training leads to lesser performance. (a) Sizes of the dots reflect the percentage of active nodes (sparser network = smaller dots) in the network. Each minibatch corresponds to the presentation of 20 training samples. The abscissa of the inset figure is scaled logarithmically. (b) Performance of ESN with (red) and without (gray) threshold learning on the three tasks analyzed, measured by accuracy and convergence time (C.T.). The network with the SpaRCe model outperforms the standard ESN read-out on all the benchmarks, but the contribution of the thresholds decreases as the task becomes more complex. This can be understood by considering that the increasing complexity of the tasks from left to right of the graph arises from a greater demand for the network's ability to understand long-term dependencies. This aspect depends on the system dynamics and is not strongly related to threshold learning. Furthermore, the SpaRCe model converges about five times faster than an ESN without thresholds.

The dataset used is MNIST, where the reservoir processed every image column by column as in Section III-B1, adopting $\tilde{\mathbf{V}} = \mathcal{C}(\{\mathbf{V}(t)\}_{\forall t})$ as before. Of course, the model is able to learn from the data corresponding to a task only once. We learn from the training dataset and compute the accuracy on the testing dataset by varying the initial starting sparsity levels through grid search of the value of n in (4) and as the number of tasks considered varies [see Fig. 6(c) and (d)]. These performances are used for comparison and demonstration only, and a validation dataset will be used to select the best hyperparameters setting. In comparison to previous simulations, the learning rate for the thresholds is smaller (see also Table III) because it was crucial to prevent a dramatic change of sparsity levels during learning, as such an abrupt change in the percentage of active nodes would alter stimulus representations and, thereby, affect previously learned tasks. In both tasks, lower sparsity levels allowed better initial learning on novel data [accuracy across $N_{\text{task}} = 1$; see Fig. 6(c) and (d)], while higher sparsity levels alleviated forgetting of previous tasks during subsequent training. In other words, low sparsity allows good performance when the number of tasks is low, but the accuracy decreases quickly when N_{task} increases. On the contrary, at optimal sparsity levels, accuracy remains high even as the number of tasks increases [highlighted path on the surface plot in Fig. 6(c) and (d)]. These two conflicting trends combined make the total accuracy across all datasets an inverted U when plotted against sparsity level [see Fig. 6(a) and (b)].

Moreover, we computed the following quantities to measure the alleviation of catastrophic forgetting in more detail:

$$\alpha_{\text{Overall}} = \frac{1}{N_{\text{task}} - 1} \sum_{n=2}^{N_{\text{task}}} \frac{\text{acc}_n}{\text{acc}(1,1)} \tag{10}$$

$$\alpha_{\text{Memory}} = \frac{1}{N_{\text{task}}} \sum_{n=1}^{N_{\text{task}}} [\text{acc}(n, N_{\text{task}}) - \text{acc}(n,n)] \tag{11}$$

$$\alpha_{\text{New}} = \frac{1}{N_{\text{task}}} \sum_{n=1}^{N_{\text{task}}} \text{acc}(n,n) \tag{12}$$

where acc_n is the accuracy computed on the datasets seen until task number n (included), and $\text{acc}(1,1)$ is the accuracy of the first dataset immediately after its learning, which is considered as the ideal baseline. In general, we denote with $\text{acc}(n,m)$ the accuracy of the nth dataset after the presentation of m datasets. The performance metric α_{Memory} measures model's ability to remember previous tasks, i.e., to alleviate catastrophic forgetting, while the metric α_{New} measures the model's ability to learn new tasks. In other terms, α_{Memory} is the average difference over tasks between the performance obtained after the whole training and immediately after the presentation of a specific dataset, while α_{New} is the average performance of a dataset after its presentation. These metrics and α_{Overall} are taken from [45], where the performance of various models that alleviate catastrophic forgetting in MLPs is compared.

TABLE II

SPARCE PERFORMS COMPARABLY TO THE BEST OF THE *Ad Hoc* MODELS, WHICH TENDS TO PERFORM WELL ONLY ON ONE OF THE TWO TASKS ANALYZED. THE SYMBOL $^+$ INDICATES THAT THE RESULTS WERE TAKEN FROM [45]

Results, $\alpha_{overall}$			
Sequential classes		Sequential permutations	
ESN	0.1	ESN	0.1
$SpaRCe$	0.870	$SpaRCe$	0.897
EWC	0.133^+	EWC	0.746^+
FEL	0.439^+	FEL	0.279^+
$GeppNet$	0.922^+	$GeppNet$	0.364^+

Finally, we repeated the simulation ten times for each of the two paradigms considered, averaged over different possible permutations (sequential data permutations) and different ways of dividing the data into separate classes and corresponding tasks (sequential classes), selected the best performing algorithm based on the validation dataset, and computed its performance α_{Overall} on the test dataset. The performance obtained is reported in Table II together with the results obtained by a standard ESN and published models that use a variety of strategies to prevent catastrophic forgetting in MLPs. Of course, we do not expect to compete with newer deep learning methods that exploit additional information and many more parameters. The results reported are obtained following, to the best of our knowledge, the same methodology as [45], and differ only in the necessity of an ESN to process the data temporally, rather than feeding the whole image to the network at once. In this aspect, the temporal processing of the data can make the task only more challenging for our model since catastrophic forgetting appears to constitute an even more difficult problem to recurrent neural networks [56]. We need to highlight, however, a drawback of our model: the tuning of an additional hyperparameter which is the starting sparsity level P_n regulated by the proposed normalization mechanism. However, we note that all other methods that alleviate catastrophic forgetting in neural networks usually have additional hyperparameters (set through heuristics or grid search), such as the learning rate of the penalty term in EWC.

If the results on MLPs can be considered exclusively as a useful baseline, we notice from Table II the complete inability of an ESN to solve sequential learning. Indeed, the training of the read-out weights of an ESN in the standard reservoir computing paradigm causes a complete forgetting of previous tasks, resulting in an overall performance that corresponds to the learning of the last task processed by the model.

The success of SpaRCe on these catastrophic forgetting tasks arises from both the initial sparsity and from threshold learning. We analyzed the relative importance of the starting sparsity level versus the online threshold adaptation introduced by the learning rule in the Appendix.

IV. DISCUSSION

It is customary in machine learning to introduce sparsity via regularization: an *ad hoc* penalty term is added to the network's error function to penalize the use of the weight parameters, leading to solutions with smaller (or sparser) weight values. In this work, inspired by the insect mushroom body, we propose a simple and elegant modification on the standard ESN that leads to sparse representations. We introduce a threshold per reservoir neuron, adaptable via online gradient learning, thereby associating sparsity directly to network performance. This results in active neurons that preferentially fire for one class versus another, which, in turn, leads to improved performance without disturbing the reservoir dynamics. In our setup, threshold learning and weight learning are a two-way interaction: the threshold changes are proportional to the weight values. In practice, we have found that, in most cases, an increase in neuronal specialization follows large weight changes.

The threshold learning rule is structurally identical to the update rule for the bias in backpropagation. This is not surprising as our formulation is equivalent to the introduction of an HL with as many neurons as the reservoir and a one-to-one fixed connectivity. The only learnable parameters are, therefore, the neuronal thresholds in the HL. However, we demonstrated that it is advantageous to only learn these thresholds in comparison to the full HL. In our simulations, an HL with approximately 10^5 learnable parameters was required to catch up with SpaRCe, which required only 3×10^3 learnable parameters, and training was twice slower. Our technique is also conceptually similar to the extreme learning machines, which has been also applied in reservoir computing [57]. Nevertheless, allowing for threshold learning increases the memory capacity of the network, and we have demonstrated that it can discover the optimal sparsity level.

We also found that threshold learning helps stabilize the network in the case that a large learning rate has been selected. We have demonstrated (see Appendix B.2) that high learning rates that lead to instabilities in the nonthreshold model are appropriate for the threshold model: the thresholds act as a stabilization mechanism, by decreasing the activity of the network through neuronal deactivation. This allows for larger parameter areas and reduces the requirement of fine-tuning for the learning rates.

Notably, our model competes with feedforward and recurrent networks on standard benchmark problems (MNIST, sequential MNIST, and permuted sequential MNIST) and is always best or close to the best alternative algorithm. This is an impressive result given the simplicity of our model. While, in general, an ESN is unlikely to compete with more complex networks, SpaRCe permits ESNs to achieve performance levels that were not possible before, at least not without augmentation of the dataset or in combination with other algorithms (e.g., [58]).

Perhaps less obviously, threshold initialization is key to achieving consistent performances. If the initialization is entirely random, neurons with excessively high initial thresholds would never fire during the stimulus presentation. Effectively, such neurons would be removed from the network for the whole duration of the simulation. To prevent this issue, the training dataset is first presented to the recurrent network, and we observe the operational activity range of each neuron. This allows us to set up a threshold within this regime in

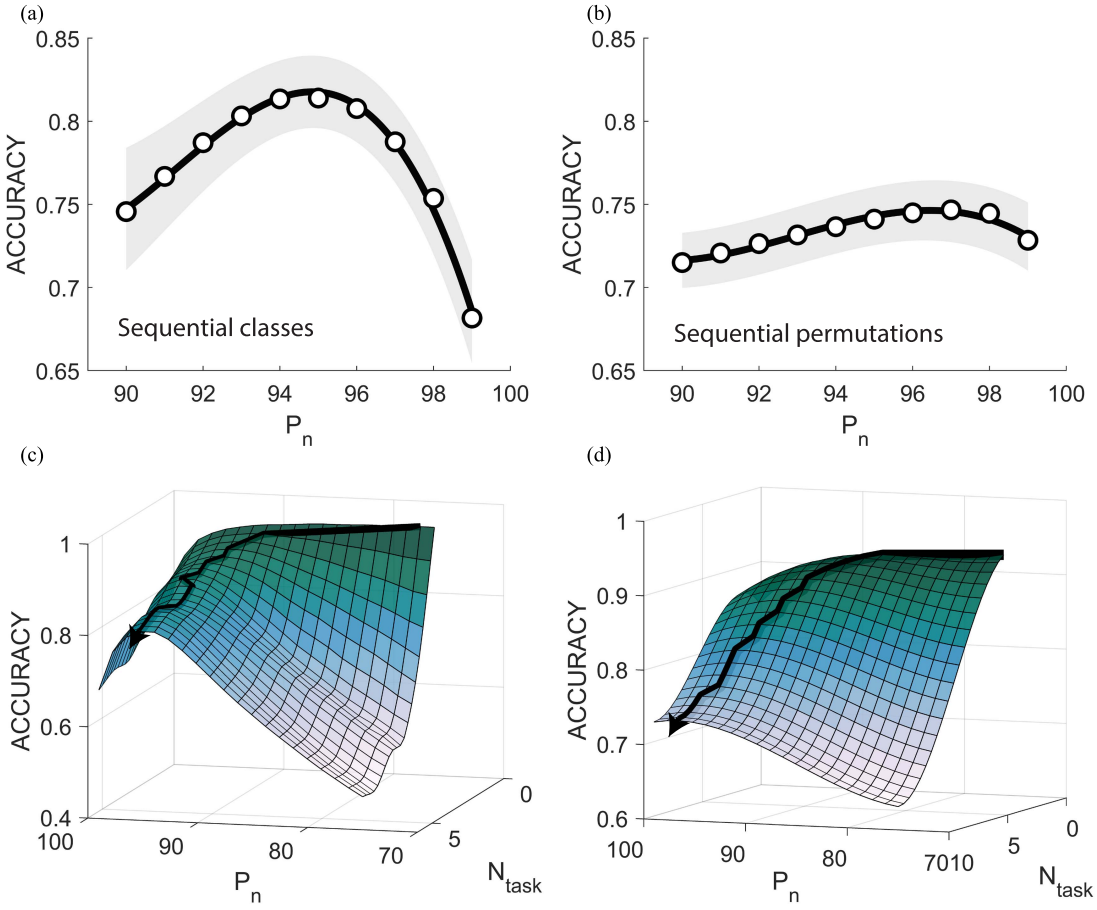


Fig. 6. SpaRCe helps to prevent catastrophic forgetting on the two analyzed benchmarks. Different data points correspond to diverse repetitions of the experiment. (a) Results on the MNIST dataset in the sequential classes paradigm as the starting fraction of active nodes varies (see text). (b) Same as (a), but for the permuted datasets paradigm. (c) Performance as a function of the starting percentage of active nodes and the number of tasks that are learned in the catastrophic forgetting simulations for the sequential classes task. (d) Same as (c), but for the permuted datasets task. (c) and (d) The surface is a cubic interpolation of the accuracy as the number of datasets and the starting sparsity level vary. The path shows the best performing sparsity levels across various numbers of tasks; its movement from right to left demonstrates the necessity of adopting an increasing level of sparsity as the number of datasets increases and the memory of previous tasks becomes more relevant.

a "fair" way, making sure that each neuron is active for a predecided percentage of the time, across all stimulus presentations. This process has a relatively small computational overhead over the ESN; however, it is possible to learn the activity percentile entirely online (see Appendix B.3) without significant performance loss.

Our model competes with published models across two standard tests for catastrophic forgetting. In fact, sparsity alone, without threshold learning, significantly helps in the case of catastrophic forgetting, but threshold learning adds to the ability to better learn newer sets. In general, however, we do not expect to outperform complex new methods that exploit additional information: our model, following the principle of ESN and reservoir computing in general, uses inherent properties of the network (e.g., dynamics and sparsity) to boost performance in classification tasks and catastrophic forgetting.

Our work may lead to a reinterpretation of the role of thresholds in neural networks. We have shown that by having a layer where learning takes place via threshold adaptation only and by disentangling the learning of the thresholds from the learning of the weights, via different learning rates, we were able to achieve sparse solutions. We were also able to demonstrate mathematically that sparsity is shaped by effectively "removing" redundant neurons from the reservoir. We believe that this work might be applicable to network structures beyond ESNs.

Finally, reservoir computing is of increasing interest to the neuromorphic computing community, particularly to those who aim to use material dynamics for computation. For instance, in the spintronic community, magnetic devices are proposed as reservoir replacements, and more complex methods, such as deep learning, could not be implemented at the material level. In the context of the ESN, the reservoir serves only as a spatiotemporal kernel [59], [60], i.e., it increases the dimensionality of the input signal in order to allow a linear model (a perceptron) to separate the classes. Therefore, it can be replaced by any highly nonlinear but nonchaotic system able to transform its input to an appropriate higher dimensional space. Such proof of concept systems can be

found, for instance, in [61] and [62]. Our algorithm does not impose any modification to the reservoir itself, which allows its use even when the recurrent network is replaced by a physical material.

ACKNOWLEDGMENT

The authors would like to thank Paolo Del Giudice and Guido Gigante for their input on the analysis of the timescales in the reservoir model, Jelmer Borst for suggesting the use of their method on catastrophic forgetting, and Herbert Jaeger for feedback on a preliminary version of this work.

REFERENCES

[1] M. V. Tsodyks and M. V. Feigel'man, "The enhanced storage capacity in neural networks with low activity level," *EPL*, vol. 6, no. 2, p. 101, 1988.

[2] M. Tsodyks, "Associative memory in asymmetric diluted network with low level of activity," *EPL*, vol. 7, no. 3, p. 203, 1988.

[3] B. Derrida, E. Gardner, and A. Zippelius, "An exactly solvable asymmetric neural network model," *EPL*, vol. 4, no. 2, p. 167, 1987.

[4] D. J. Amit, H. Gutfreund, and H. Sompolinsky, "Storing infinite numbers of patterns in a spin-glass model of neural networks," *Phys. Rev. Lett.*, vol. 55, no. 14, p. 1530, 1985.

[5] S. Romani, I. Pinkoviezky, A. Rubin, and M. Tsodyks, "Scaling laws of associative memory retrieval," *Neural Comput.*, vol. 25, no. 10, pp. 2523–2544, Oct. 2013.

[6] T. Hastie, R. Tibshirani, and M. Wainwright, *Statistical Learning With Sparsity: The Lasso and Generalizations*. Boca Raton, FL, USA: CRC Press, 2015.

[7] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2016, pp. 2074–2082. [Online]. Available: http://papers.nips.cc/paper/6504-learning-structured-sparsity-in-deep-neural-networks.pdf

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[9] P. M. Rasmussen, L. K. Hansen, K. H. Madsen, N. W. Churchill, and S. C. Strother, "Model sparsity and brain pattern interpretation of classification models in neuroimaging," *Pattern Recognit.*, vol. 45, no. 6, pp. 2085–2100, Jun. 2012.

[10] E. T. Rolls and M. J. Tovee, "Sparseness of the neuronal representation of stimuli in the primate temporal visual cortex," *J. Neurophysiol.*, vol. 73, no. 2, pp. 713–726, 1995.

[11] K. S. Honegger, R. A. A. Campbell, and G. C. Turner, "Cellular-resolution population imaging reveals robust sparse coding in the drosophila mushroom body," *J. Neurosci.*, vol. 31, no. 33, pp. 11772–11785, Aug. 2011.

[12] A. C. Lin, A. M. Bygrave, A. De Calignon, T. Lee, and G. Miesenböck, "Sparse, decorrelated odor coding in the mushroom body enhances learned odor discrimination," *Nature Neurosci.*, vol. 17, no. 4, p. 559, 2014.

[13] G. C. Turner, M. Bazhenov, and G. Laurent, "Olfactory representations by Drosophila mushroom body neurons," *J. Neurophysiol.*, vol. 99, no. 2, pp. 734–746, Feb. 2008.

[14] E. Gruntman and G. C. Turner, "Integration of the olfactory code across dendritic claws of single mushroom body neurons," *Nature Neurosci.*, vol. 16, no. 12, p. 1821, 2013.

[15] H. Li, Y. Li, Z. Lei, K. Wang, and A. Guo, "Transformation of odor selectivity from projection neurons to single mushroom body neurons mapped with dual-color calcium imaging," *Proc. Nat. Acad. Sci. USA*, vol. 110, no. 29, pp. 12084–12089, Jul. 2013.

[16] J. Perez-Orive, O. Mazor, G. C. Turner, S. Cassenaer, R. I. Wilson, and G. Laurent, "Oscillations and sparsening of odor representations in the mushroom body," *Science*, vol. 297, no. 5580, pp. 359–365, Jul. 2002.

[17] J. M. Jeanne and R. I. Wilson, "Convergence, divergence, and reconvergence in a feedforward network improves neural speed and accuracy," *Neuron*, vol. 88, no. 5, pp. 1014–1026, Dec. 2015.

[18] R. Azouz and C. M. Gray, "Dynamic spike threshold reveals a mechanism for synaptic coincidence detection in cortical neurons *in vivo*," *Proc. Nat. Acad. Sci. USA*, vol. 97, no. 14, pp. 8110–8115, Jul. 2000.

[19] M. S. Grubb and J. Burrone, "Activity-dependent relocation of the axon initial segment fine-tunes neuronal excitability," *Nature*, vol. 465, no. 7301, p. 1070, 2010.

[20] N. Intrator and L. N. Cooper, "Objective function formulation of the BCM theory of visual cortical plasticity: Statistical connections, stability conditions," *Neural Netw.*, vol. 5, no. 1, pp. 3–17, 1992. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608005800036

[21] J. Triesch, "A gradient rule for the plasticity of a neuron's intrinsic excitability," in *Artificial Neural Networks: Biological Inspirations—ICANN 2005*, W. Duch, J. Kacprzyk, E. Oja, and S. Zadrożny, Eds. Berlin, Germany: Springer, 2005, pp. 65–70.

[22] M. H. Yusoff, J. Chrol-Cannon, and Y. Jin, "Modeling neural plasticity in echo state networks for classification and regression," *Inf. Sci.*, vols. 364–365, pp. 184–196, Oct. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025515008270

[23] J. J. Steil, "Online reservoir adaptation by intrinsic plasticity for backpropagation–decorrelation and echo state learning," *Neural Netw.*, vol. 20, no. 3, pp. 353–364, Apr. 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608007000317

[24] B. Schrauwen, M. Wardermann, D. Verstraeten, J. J. Steil, and D. Stroobandt, "Improving reservoirs using intrinsic plasticity," *Neurocomputing*, vol. 71, nos. 7–9, pp. 1159–1171, Mar. 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231208000519

[25] C. Du, F. Cai, M. A. Zidan, W. Ma, S. H. Lee, and W. D. Lu, "Reservoir computing using dynamic memristors for temporal information processing," *Nature Commun.*, vol. 8, no. 1, pp. 1–10, Dec. 2017.

[26] M. S. Kulkarni and C. Teuscher, "Memristor-based reservoir computing," in *Proc. IEEE/ACM Int. Symp. Nanosc. Archit. (NANOARCH)*, Jul. 2012, pp. 226–232.

[27] X. Zhu, Q. Wang, and W. D. Lu, "Memristor networks for real-time neural activity analysis," *Nature Commun.*, vol. 11, no. 1, pp. 1–9, Dec. 2020.

[28] K. Vandoorne *et al.*, "Experimental demonstration of reservoir computing on a silicon photonics chip," *Nature Commun.*, vol. 5, no. 1, pp. 1–6, May 2014.

[29] Y. Paquot *et al.*, "Optoelectronic reservoir computing," *Sci. Rep.*, vol. 2, no. 287, p. 287, Feb. 2012.

[30] K. Nakajima, "Physical reservoir computing—An introductory perspective," *Jpn. J. Appl. Phys.*, vol. 59, no. 6, Jun. 2020, Art. no. 060501.

[31] H. Jaeger, M. Lukoševičius, D. Popovici, and U. Siewert, "Optimization and applications of echo state networks with leaky-integrator neurons," *Neural Netw.*, vol. 20, no. 3, pp. 335–352, Apr. 2007.

[32] A. Rodan and P. Tino, "Minimum complexity echo state network," *IEEE Trans. Neural Netw.*, vol. 22, no. 1, pp. 131–144, Jan. 2011.

[33] H. Cui, X. Liu, and L. Li, "The architecture of dynamic reservoir in the echo state network," *Chaos, Interdiscipl. J. Nonlinear Sci.*, vol. 22, no. 3, 2012, Art. no. 033127.

[34] C. Gallicchio, A. Micheli, and L. Pedrelli, "Deep reservoir computing: A critical experimental analysis," *Neurocomputing*, vol. 268, pp. 87–99, Dec. 2017.

[35] L. Manneschi, M. O. A. Ellis, G. Gigante, A. C. Lin, P. Del Giudice, and E. Vasilaki, "Exploiting multiple timescales in hierarchical echo state networks," *Frontiers Appl. Math. Statist.*, vol. 6, p. 76, Feb. 2021.

[36] F. M. Bianchi, S. Scardapane, S. Løkse, and R. Jenssen, "Reservoir computing approaches for representation and classification of multivariate time series," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 5, pp. 2169–2179, May 2021.

[37] J. Huang, T. Zhang, and D. Metaxas, "Learning with structured sparsity," *J. Mach. Learn. Res.*, vol. 12, pp. 3371–3412, Jan. 2011.

[38] E. J. Candes, M. B. Wakin, and S. P. Boyd, "Enhancing sparsity by reweighted ℓ_1 minimization," *J. Fourier Anal. Appl.*, vol. 14, nos. 5–6, pp. 877–905, 2008.

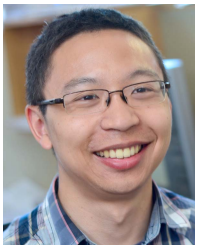
[39] M. Lukoševičius, "A practical guide to applying echo state networks," in *Neural Networks: Tricks of the Trade*. Berlin, Germany: Springer, 2012, pp. 659–686.

[40] Q. Ma, L. Shen, W. Chen, J. Wang, J. Wei, and Z. Yu, "Functional echo state network for time series classification," *Inf. Sci.*, vol. 373, pp. 1–20, Dec. 2016.

[41] N. Schaetti, M. Salomon, and R. Couturier, "Echo state networks-based reservoir computing for MNIST handwritten digits recognition," in *Proc. IEEE Int. Conf. Comput. Sci. Eng. (CSE), IEEE Int. Conf. Embedded Ubiquitous Comput. (EUC), 15th Int. Symp. Distrib. Comput. Appl. Bus. Eng. (DCABES)*, Aug. 2016, pp. 484–491.

[42] J. Torrejon *et al.*, "Neuromorphic computing with nanoscale spintronic oscillators," *Nature*, vol. 547, pp. 428–431, Jul. 2017.
[43] G. Tanaka *et al.*, "Recent advances in physical reservoir computing: A review," *Neural Netw.*, vol. 115, pp. 100–123, Jul. 2019.
[44] S. Chandar, C. Sankar, E. Vorontsov, S. E. Kahou, and Y. Bengio, "Towards non-saturating recurrent units for modelling long-term dependencies," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 3280–3287.
[45] R. Kemker, M. McClure, A. Abitino, T. L. Hayes, and C. Kanan, "Measuring catastrophic forgetting in neural networks," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 1–9.
[46] J. Serra, D. Suris, M. Miron, and A. Karatzoglou, "Overcoming catastrophic forgetting with hard attention to the task," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80, Jul. 2018, pp. 4548–4557.
[47] E. A. Hallem and J. R. Carlson, "Coding of odors by a receptor repertoire," *Cell*, vol. 125, no. 1, pp. 143–160, 2006.
[48] S. R. Olsen, V. Bhandawat, and R. I. Wilson, "Divisive normalization in olfactory population codes," *Neuron*, vol. 66, no. 2, pp. 287–299, 2010.
[49] S. X. Luo, R. Axel, and L. F. Abbott, "Generating sparse and selective third-order responses in the olfactory system of the fly," *Proc. Nat. Acad. Sci. USA*, vol. 107, no. 23, pp. 10713–10718, 2010.
[50] M. Parnas, A. C. Lin, W. Huetteroth, and G. Miesenböck, "Odor discrimination in drosophila: From neural population codes to behavior," *Neuron*, vol. 79, no. 5, pp. 932–944, Sep. 2013.
[51] K. Krishnamurthy, A. M. Hermundstad, T. Mora, A. M. Walczak, and V. Balasubramanian, "Disorder and the neural representation of complex odors: Smelling in the real world," 2017, *arXiv:1707.01962*. [Online]. Available: http://arxiv.org/abs/1707.01962
[52] S. J. Caron, V. Ruta, L. Abbott, and R. Axel, "Random convergence of olfactory inputs in the Drosophila mushroom body," *Nature*, vol. 497, no. 7447, p. 113, 2013.
[53] S. Song, P. J. Sjöström, M. Reigl, S. Nelson, and D. B. Chklovskii, "Highly nonrandom features of synaptic connectivity in local cortical circuits," *PLoS Biol.*, vol. 3, no. 3, p. e68, Mar. 2005.
[54] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980
[55] L. Deng, "The MNIST database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, Nov. 2012.
[56] G. Arora, A. Rahimi, and T. Baldwin, "Does an LSTM forget more than a CNN? An empirical study of catastrophic forgetting in NLP," in *Proc. 17th Annu. Workshop Australas. Lang. Technol. Assoc.*, 2019, pp. 77–86.
[57] J. B. Butcher, D. Verstraeten, B. Schrauwen, C. R. Day, and P. W. Haycock, "Reservoir computing and extreme learning machines for non-linear time-series data analysis," *Neural Netw.*, vol. 38, pp. 76–89, Feb. 2013.
[58] L. Boccato, A. Lopes, R. Attux, and F. J. Von Zuben, "An extended echo state network using volterra filtering and principal component analysis," *Neural Netw.*, vol. 32, pp. 292–302, Aug. 2012.
[59] M. Hermans and B. Schrauwen, "Recurrent kernel machines: Computing with infinite echo state networks," *Neural Comput.*, vol. 24, no. 1, pp. 104–133, 2012.
[60] J. Dong, R. Ohana, M. Rafayelyan, and F. Krzakala, "Reservoir computing meets recurrent kernels and structured transforms," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 16785–16796. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/c348616cd8a86ee661c7c98800678fad-Abstract.html
[61] D. Marković *et al.*, "Reservoir computing with the frequency, phase, and amplitude of spin-torque nano-oscillators," *Appl. Phys. Lett.*, vol. 114, no. 1, Jan. 2019, Art. no. 012409.
[62] M. Romera *et al.*, "Vowel recognition with four coupled spin-torque nano-oscillators," *Nature*, vol. 563, no. 7730, p. 230, 2018.

Luca Manneschi received the bachelor's degree in physics from the University of Padua, Padua, Italy, in 2014, and the master's degree from Sapienza University of Rome, Italy, in 2017. He is currently pursuing the Ph.D. degree with The University of Sheffield, U.K.



Andrew C. Lin received the A.B. degree from Harvard University, Cambridge, MA, USA, in 2004, and the Ph.D. degree from the University of Cambridge, U.K., in 2009.

He is currently a Lecturer and a Group Leader with the Department of Biomedical Science, The University of Sheffield, U.K. His research interests include olfaction, sensory coding, homeostatic plasticity, and learning and memory.

Eleni Vasilaki received the bachelor's degree in informatics and telecommunications in 1996, the master's degree in microelectronics in 2002, and the D.Phil. (Ph.D.) degree in computer science and artificial intelligence from the University of Sussex, U.K., in 2004.

In 2009, she joined the Department of Computer Science, The University of Sheffield, U.K., as a Lecturer, where she has been a Professor since 2016.