
Preprint:
Manneschi, L., Lin, A.C. and Vasilaki, E. (Submitted: 2019) SpaRCe : sparse reservoir computing. [Preprint - arXiv] (Submitted)

https://doi.org/10.48550/arXiv.1912.08124

SpaRCe: Sparse reservoir computing

Luca Manneschi[1] manneschi1@sheffield.ac.uk
Andrew C. Lin[2] andrew.lin@sheffield.ac.uk
Eleni Vasilaki[1] e.vasilaki@sheffield.ac.uk

[1]Department of Computer Science, The University of Sheffield
[2]Department of Biomedical Science, The University of Sheffield

December 18, 2019

"Sparse" neural networks, in which relatively few neurons or connections are active, are common in both machine learning and neuroscience. Whereas in machine learning, "sparseness" is related to a penalty term which effectively leads to some connecting weights becoming small or zero, in biological brains, sparseness is often created when high spiking thresholds prevent neuronal activity. Inspired by neuroscience, here we introduce sparseness into a reservoir computing network via neuron-specific learnable thresholds of activity, allowing neurons with low thresholds to give output but silencing outputs from neurons with high thresholds. This approach, which we term "SpaRCe", optimises the sparseness level of the reservoir and applies the threshold mechanism to the information received by the read-out weights. Both the read-out weights and the thresholds are learned by a standard online gradient rule that minimises an error function on the outputs of the network. Threshold learning occurs by the balance of two opposing forces: reducing inter-neuronal correlations in the reservoir by deactivating redundant neurons, while increasing the activity of neurons participating in correct decisions. We test SpaRCe in a set of classification problems and find that introducing threshold learning improves performance compared to standard reservoir computing networks.

1 Introduction

Function of artificial neural networks is often improved by adopting "sparse" representations or connectivity, in which relatively few neurons or connections are active. Previous research has studied the role of sparse connectivity in terms of memory of Hopfield networks through the application of statistical mechanics, demonstrating how sparse connectivity leads to an increased storage capacity [1] [2] [3] [4]. In this context, memory retrieval and associative learning have been studied as 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] and a reduced computational cost [7], and can help solve overfitting problems [8]. Sparseness is typically introduced in machine learning networks through regularisation, in which a penalty term tends to reduce connection weight. In this regard, the work in [7] demonstrated how structured sparceness can have benefits in terms of 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 trade-off between pure classification accuracy and reproducibility.

Sparseness 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 sparseness 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 high selectivity of neurons, as with the Kenyon cells (KCs), the principal neurons of the insect mushroom body, which fire sparsely in response to odor stimuli [13] [14] [15] [16]. In the fruit fly Drosophila, this sparse odor coding enhances learned discrimination of similar odors [12]. More-

over, 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].
Here we applied the concept of adaptable spiking thresholds to machine learning to create SpaRCe, a Sparse Reservoir Computing network with learnable thresholds for each reservoir neuron. Our network is a reservoir of leaky integrators [20]. The connectivity between the nodes is represented through a random sparse fixed adjacency matrix; the recurrent activity created by this connectivity exhibits a multitude of characteristic timescales and allows the network to learn not only single stimuli but also sequences of stimuli. This complex connectivity is consistent with experimental reports of chemical [21] [22] and electrical [23] synapses between Kenyon cells in Drosophila, although the physiological function of KC-KC synapses remains unknown. Analogously to the concept of firing thresholds, SpaRCe exploits learnable thresholds to optimize the level of sparsity inside the network. Both the learnable thresholds and the read-out weights (but not the recurrent connections within the reservoir) are optimised by minimising an error function without exploiting any normalization term. We analysed the learning rule derived from this error minimisation 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 (Fig. 3). The novelty of the proposed approach lies in the fact that a sparsity level is reached due to the presence of firing thresholds, rather than to regularization [24] [6] [25].

2 Methods

The reservoir under consideration is a network of leaky integrators described by the following equation

$$\mathbf{V}(t+\delta t) = (1-\alpha)\mathbf{V}(t) + \alpha f\big[W_{in}\mathbf{s} + \rho W\mathbf{V}(t)\big] \quad (1)$$

where $\alpha = \frac{\delta t}{\tau}$ defines the temporal scale of the neuron and $\mathbf{V}(t)$ is the activity vector of the integrators [1]. W is the fixed sparse random matrix that describes the recurrency of the reservoir, and $\mathbf{s}$ is the input signal. The rescaling factor ρ is chosen in order to constrain the eigenvalues of the associated dynamic system inside the unit circle of the imaginary plane, a necessary condition for the Echo State property of the network. It is possible to control a priori the range of timescales that the reservoir exhibits by choosing appropriately α and ρ as described in the methodology reported in *Supplementary materials*. The specific form of the input matrix W_{in} and the activation function f is task dependent and will be specified in sections 3.1 and 3.2.
The values of the hyperparameters of the model adopted are reported in section 5.

2.1 SpaRCe

In contrast to previous models [26] [27] [20] that define the output of the neural network through a read-out of the $\mathbf{V}$ vector, we introduced another variable $\mathbf{x}(t)$, defined as follows

$$\mathbf{x}(t) = relu\big[\mathbf{V}(t) - \boldsymbol{\theta}\big] \quad (2)$$

where *relu* stands for rectified linear unit, and $\boldsymbol{\theta}$ is a vector of thresholds that enables $\mathbf{x}$ to be sparse. Thus, the variable $x_i(t)$ is zero if the variable $V_i(t)$ is lower than the corresponding threshold θ_i. The additional complexity consequent to Eq. 2 is summarized by Fig. 1, which depicts the difference between the read-out of a standard Echo-State network and our formulation. Panel $\boldsymbol{B}$ of figure 1 shows how Eq. 2 can be thought of as an additional layer that is connected to the reservoir through a fixed adjacency matrix that is equal to the identity matrix $\mathbb{1}_N$, where N is the number of nodes in the network. In this comparison, the vector of thresholds $\boldsymbol{\theta}$ would correspond to the bias of the additional layer $\mathbf{b}$. Indeed, considering that $\mathbf{x}$ is the activity of an additional layer

$$\mathbf{x}(t) = relu\big[W^h\mathbf{V}(t) + \mathbf{b}\big] = \\ = relu\big[\mathbb{1}_N\mathbf{V}(t) - \boldsymbol{\theta}\big] = relu\big[\mathbf{V}(t) - \boldsymbol{\theta}\big]$$

where W^h is the adjacency matrix among the reservoir and the additional layer, and we used the constraint $W^h = \mathbb{1}_N$ and the notation $\boldsymbol{\theta} = -\mathbf{b}$.

[1] It is called $\mathbf{V}$ to resemble the voltage of a neuron.

This specific formulation permits the model to use local information to learn the threshold values, to focus on the concept of learnable bias to introduce and optimize sparse representations, and not to rely on backpropagation through time. We note also that the addition of the thresholds through Eq. 2 does not affect the timescales of the network and thus preserves the idea behind reservoir computing as a fixed, dynamically rich, representation.
The training procedure minimizes a measure of the distance $E(t)$ between the output $\mathbf{y}(t) = W^{out}\mathbf{x}(t)$ of the neural network and the desired value $\tilde{\mathbf{y}}(t)$. Theoretically,

$$E = dist\Big(\tilde{\mathbf{y}}, \mathbf{y}\Big) \tag{3}$$

We will now apply a gradient based optimization on an example cost function, and show how the resulting learning rule for the thresholds can be interpreted.

Gradient on θ, Mean Square Error (MSE)

Let us consider the mean square cost function, given by

$$\begin{aligned} E = dist\Big(\tilde{\mathbf{y}}, \mathbf{y}\Big) = \\ = \sum_j \Big[\tilde{y}_j - y_j\Big]^2 = \\ = \sum_j \Big[\tilde{y}_j - \sum_i W^{out}_{ji} relu\Big(\mathbf{V}_i(t) - \theta_i\Big)\Big]^2 \end{aligned} \tag{4}$$

where we have used a read-out of Eq. 2 to define the output of the neural network. A gradient based approach that minimizes E leads to the following learning rule on the output weights

$$\begin{aligned} \Delta W^{out}_{lk} = -\eta_W \frac{\partial E}{\partial W_{lk}} = \\ = \eta_W \Big[\tilde{y}_l - y_l\Big] relu\Big(x_k(t)\Big) \end{aligned}$$

and to the following learning rule for the thresholds

$$\begin{aligned} \Delta\theta_k = -\eta_\theta \frac{\partial E}{\partial \theta_k} = \\ \eta_\theta \sum_{j=1}^{N_{class}} \big[\tilde{y}_j - y_j(t)\big] \frac{\partial}{\partial \theta_k} \Big\{ \sum_i W^{out}_{ji} relu\big[V_i(t) - \theta_i\big] \Big\} = \\ = -\eta_\theta \sum_{j=1}^{N_{class}} \big[\tilde{y}_j - y_j(t)\big] W^{out}_{jk} H\big(x_k(t)\big) = \\ = -\eta_\theta \sum_{j=1}^{N_{class}} \tilde{y}_j W^{out}_{jk} H\big(x_k(t)\big) + \\ + \sum_{j=1}^{N_{class}} y_j(t) W^{out}_{jk} H\big(x_k(t)\big) \end{aligned} \tag{5}$$

By taking into account the specific case of a classification task where $\tilde{y}_j$ is positive for j that corresponds to the desired class and zero otherwise, it is possible to manipulate Eq. 5 and to separate it in two terms to uncover the meaning of the learning on the thresholds.

$$\begin{aligned} \Delta\theta_k = -\eta_\theta \beta W^{out}_{\tilde{j}k} H\big(x_k(t)\big) + \\ + \sum_{j=1}^{N_{class}} y_j(t) W^{out}_{jk} H\big(x_k(t)\big) \\ = -\eta_\theta \beta W^{out}_{\tilde{j}k} H\big(x_k(t)\big) + \\ + \eta_\theta \sum_{j=1}^{N_{class}} \sum_{l=1}^{N} W^{out}_{jl} W^{out}_{jk} x_l(t) H\big(x_k(t)\big) \end{aligned} \tag{6}$$

where $\tilde{j}$ indicates the correct class for the considered input, and β is the positive quantity equal to the correct desired output value $\tilde{y}_{\tilde{j}}$. Eq. 6 contains two clearly interpretable factors:

$$\Delta_+\theta = \sum_{j=1}^{N_{class}} \sum_{l=1}^{N} W^{out}_{jl} W^{out}_{jk} x_l(t) H\big(x_k(t)\big) \tag{7}$$

$$\Delta_-\theta = -\beta W^{out}_{\tilde{j}k} H\big(x_k(t)\big) \tag{8}$$

The factor $\Delta_-\theta$ is decreasing (increasing) the threshold value of nodes with $W^{out}_{\tilde{j}k} > 0$ ($W^{out}_{\tilde{j}k} < 0$) that are contributing to reach the right (wrong) classification. Thus, $\Delta_-\theta$ is driven by the output weight between the considered node (if it is active)

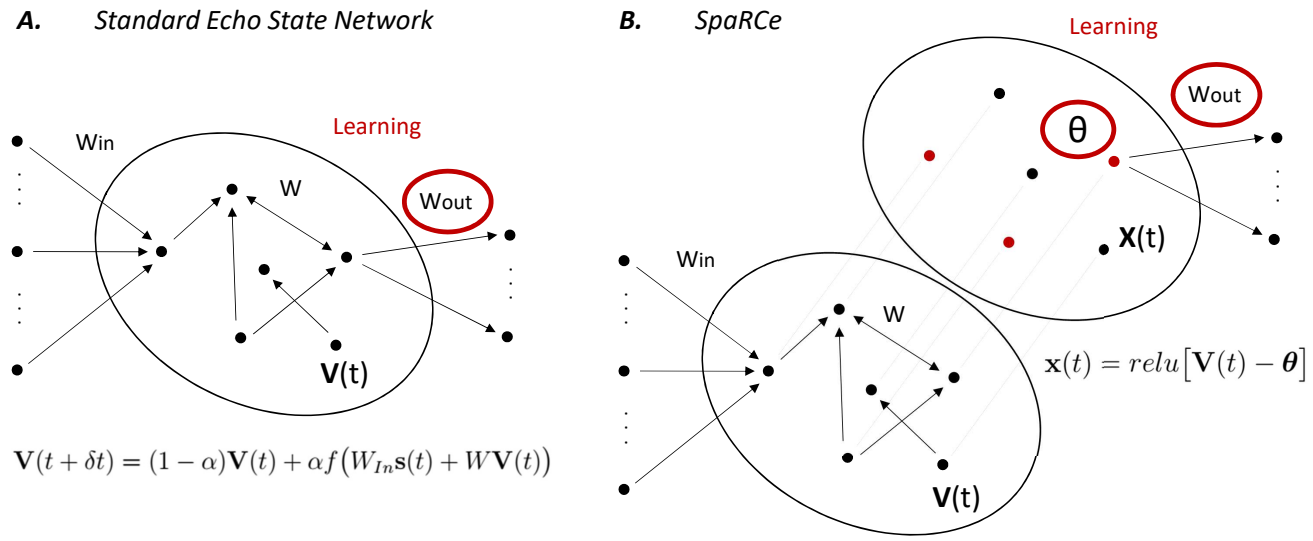

Figure 1: The SpaRCe model is equivalent to an additional layer with a constrained connectivity. **Left**: Echo-state network. The learning is applied on the output weights. **Right**: SpaRCe model. The network scheme is a representation of equations 1 and 2; while the first describes the dynamic of the reservoir, the latter can be thought as an additional layer with a connectivity matrix that is constrained to be an identity matrix. The SpaRCe algorithm leads to N parameters, corresponding to $\boldsymbol{\theta}$, to be trained in addition to the output weights.

and the desired class. $\Delta_+\theta$ is, instead, a measure of correlation of activities between different nodes in the reservoir and is increasing the thresholds of neurons that have coherent synapses and that are simultaneously active. The results of section 2 will demonstrate how the antagonist nature of these two forces drive the learning on the thresholds to modulate the sparse representation. A similar analysis of the learning rule for a cross entropy cost function is reported in paragraph 5.3, *Supplementary materials*.

We note that eq.5 is structurally analogous to the learning rule for the bias of an additional hidden layer, reported below

$$\begin{aligned}\Delta b_k = -\eta_b \frac{\partial E}{\partial b_k} = \\ \eta_b \sum_{j=1}^{N_{class}} \big[\tilde{y}_j - y_j(t)\big] W^{out}_{jk} H\big(x_k(t)\big) = \\ = \eta_b \beta W^{out}_{\tilde{j}k} H\big(x_k(t)\big) - \\ -\eta_b \sum_{j=1}^{N_{class}} \sum_{l=1}^{N} W^{out}_{jl} W^{out}_{jk} x_l(t) H\big(x_k(t)\big) \end{aligned} \quad (9)$$

where $b_k = -\theta_k$ and W^h is the connectivity matrix between the reservoir and the additional hidden layer. More specifically our model is the special case for which $W^h_{k'k} = 1$ for $k' = k$ and $W^h_{k'k} = 0$ for $k' \neq k$. In our case, the analysis of the update rule for the thresholds revealed a simple interpretation at the level of the reservoir since $x_l = relu\big(V_l(t) - \theta_l\big)$, while in the case of full connectivity $x_l = relu\big(\sum_{m=1}^{N} W^h_{lm} V_m(t) + b_l\big)$. This difference has two consequences: (i) Sparseness, if achievable, will be achieved not among the reservoir neurons $V_l(t)$ but among the neurons of the hidden layer, i.e. linear combinations of reservoir neurons. (ii) To deactivate a neuron in the fully connected hidden layer $\sum_{m=1}^{N} W^h_{lm} V_m(t) < -b_l$, implying that a good initialisation value for $b_l \approx -\sum_{m=1}^{N} W^h_{lm}$. By initialising b_l using the same distribution as for W^h_{lm} makes likely that the initial condition may be far off from a value that can deactivate the neuron and it is then possible that the learning process might be trapped to local minima. In fact, we will later show that if we compare our method with learning W^h, we achieve equivalent or better performance, likely due to the smaller parameter space.

Having analysed the meaning of the learning rule on the $\boldsymbol{\theta}$ vector, we now focus on the importance of the initialization. A specific initialization of $\boldsymbol{\theta}$ can correspond to a particular initial sparsity level of the network, and this can affect the per-

formance of the model as will be shown in Fig. 4. Thus, we have formulated a systematic procedure to overcome the dependence of the model on the starting condition.
By definition, we notice that an optimal sparse representation should avoid the existence of totally active or inactive neurons. Consequently, a threshold θ_i of a neuron i should be defined within the range of the distribution of activities of such a node. In particular, the initial value of the threshold θ_i is defined as the n-th percentile $P_{i,n}$ of the distribution of activity of the node i on the training set. If we assume that all the threshold values start from the same percentile of the distributions of the nodes, that is $\theta_i = P_{i,n}, \forall i$ or in short notation $\boldsymbol{\theta} = \mathbf{P}_n$ [2], the problem of selecting the starting correct initial condition becomes the problem of choosing the right percentile. Trivially, selecting the percentile number n leads to a staring sparsity level of $1 - n/100$.We defined two approaches to overcome the choice of selecting the starting percentile:

- A simple grid search over n. This can be done by running on parallel N_P reservoirs for a number of time steps that is about ten percent of the total number of the training instance, and then to select and to train the best performing reservoir. From our results, the utilization of a small fraction of the training time is enough to choose the starting condition without any loss in the performance.
- To select the sparse representation that leads to the highest value of specificity, a measure of the quality of the sparse representations that is defined below.

Specificity

The measure of *specificity* (Sp) reflects how a level of sparsity can facilitate the learning process in a classification task. The assumption behind the following formulation is that for a good sparse representation the ensembles of active nodes for different classes should overlap as little as possible. Let us consider two classes j and k and a neuron i. The node is specific if there is an asymmetry in the number of times it is active for one class with respect to the other. Generalizing this idea it is possible to build a measure $spec_{ijk}$ for a node i defined as

$$spec_{ijk} = |\frac{N_{ij}}{M_j} - \frac{N_{ik}}{M_k}| \tag{10}$$

where N_{ij} (N_{ik}) are the number of times the neuron i was active after the presentation of a stimulus of class j (k) and M_j is the total number of presentations of the stimuli belonging to class j. Since the denominator of Eq. 10 contains the total number of presentations, $spec_{ijk}$ does not simply increase with the level of sparsity introduced. Let us focus on the particular case where $M_j \approx M_k$. A too high level of sparsity would lead the node to be almost silent, with a consequent poor specificity value due to N_{ij} and N_{ik} being both close to zero. On the contrary, a too low sparsity level would lead the neuron to be excessively responsive, and $spec_{ijk}$ would be poor because $N_{ij} \approx N_{ik}$ even if N_{ij} and N_{ik} are both high.
Given $spec_{ijk}$ it is possible to compute a measure of specificity for each single neuron as

$$Sp_i = \frac{2}{N_{class}(N_{class}-1)} \sum_j \sum_{k>j} spec_{ijk} \tag{11}$$

where we considered only the upper triangular part of $spec_{ijk}$ because of the symmetry of the latter tensor.
It is possible to select the starting initial values of the thresholds as the n-th percentile of the distribution $\mathbf{V}$ that leads to the highest specificity measure. Figure 4 shows how the best performing sparse representation corresponds approximately to the maximum value of the average specificity across neurons $Sp = \frac{1}{N}\sum_i^N Sp_i$.

3 Results

3.1 Odor Sequence Learning

We evaluated the performance of the models in classifying an ensemble of N_{In} dimensional sequences $\{\mathbf{S}_i\}_{i=1,..,N_{seq}}$ of three successive stimuli. Each stimulus of a sequence is derived from the simulated response of $N_{In} = 24$ projection neurons (PNs, second-order neurons in the fly olfactory system) to 110 different odors, based on physiological recordings of olfactory receptor neurons (ORNs)

[2] where $\mathbf{P}_n$ is the vector corresponding to the n-th percentile of all the activity distributions

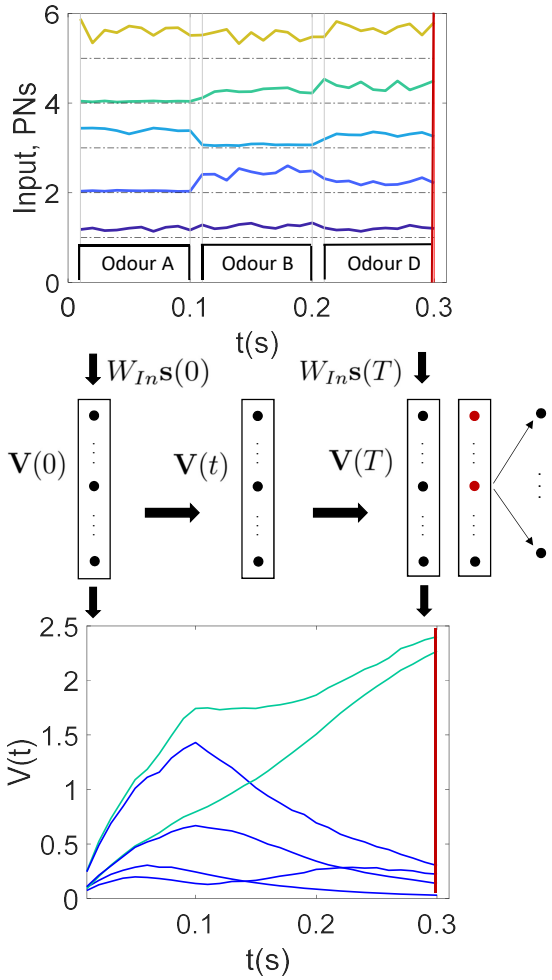

Figure 2: Scheme of the task on the biological data. **Top:** Input example, succession of three stimuli of time duration $\Delta t = 0.1s$ each. Coloured lines are associated to five examples of input neurons activity. The red vertical line corresponds to the final time step of the sequence when the classification process happens. **Middle:** Scheme of the evolution of the reservoir across time. Each black box marked with the letter $\mathbf{V}$ corresponds to the activity of a reservoir at a specific point in time. The vertical arrows represent the input to the reservoir, while the horizontal arrows reflect the evolution across time. The final black depicts the application of the SpaRCe model on the final temporal layer. The red dots correspond to a schematic representation of the active nodes and emphasized the sparse representation achieved through Eq. 2. **Bottom:** Example of $V_i(t)$ across time for six nodes in the reservoir.

and known characteristics of the ORN-PN synapse [28] [29]. This simulated activity, which we call $\mathbf{s}^{HO}$ (HO for Hallem-Olsen), has previously been used in computational analyses of fly olfaction [30] [31] [32].

The procedure for building different sequences from single stimuli is described in detail in section 5.4. Each of the three stimuli in a sequence is presented for a time interval $\Delta t = 0.1s$ in order to allow the network to integrate the information, and the total duration of an input corresponds to $T = 0.3s$. Given a sequence i $\mathbf{s}_i(t)$ built following the procedure in 5.4, we added multiplicative white noise to each separate dimension to make the task more realistic and complex. Thus, the i-th dimension of the final sequence $\mathbf{S}_i(t)$ to be classified is $S_i(t) = s_i(t) + \sigma_s \xi(t) s_i(t)$, where $\xi(t)$ is a Gaussian distributed random variable with zero mean and unitary variance.
For this specific task, the activation function f of Eq. 1 is a rectified linear unit and the connections of the input adjacency matrix W_{In} follow a lognormal distribution, where each node in the reservoir is connected on average to six input nodes and the number of connections is inversely proportional to the connections strength. This particular form of W_{In} is inspired by the biological results in [13] [33] [34]. In the machine learning task faced in section 3.2 we will use a more common form for W_{In}.

First, we want to confirm the theoretical analysis in section 2.1 and our expectations on the interpretability of the learning rule on the thresholds. The left panel of Fig. 3 shows the two factors $\Delta_+\theta$ and $\Delta_-\theta$ across learning for an example of sequence classification and for different starting conditions of the thresholds. The positive y-axis reports a running average of the correlation factor $\Delta_+\theta$ with solid lines, and the negative y-axis reports a running average of $\Delta_-\theta$. Colours correspond to diverse initial sparsity levels, where the thresholds starting values are defined as the n-th percentile $\mathbf{P}_n$ of the $\mathbf{V}$ distributions. It is clear that the positive correlation term is increasing the threshold values and deactivating neurons, while $\Delta_-\theta$ is recruiting nodes by activating neurons whose weights to the desired output node are positive. The two forces are almost symmetric, and their slight imbalance provides the direction to change the threshold values. Indeed, the right panel of Fig. 3 shows the cumulative average change per threshold for the three cases analysed. If the starting sparsity level is high (P_{70} in the figure) the total force is negative and the factor $\Delta_-\theta$ dominates, while if the sparsity level is low (P_{10}) the correlation term $\Delta_+\theta$ wins and the thresholds increase on average. From the left panel it is also evident that the magnitudes of the forces vary for the two cases; this result is understandable by realizing that input stimuli are represented by less overlapping clusters when the sparsity level is higher. Eq. 7 is a measure of correlation of nodes with coherent output weights and it increases with the sparsity level. Thus, higher sparsity leads to less overlap in the representations, which leads to more coherent output weights of the nodes belonging to a cluster toward a specific class, which makes the two forces ($\Delta_-\theta$ and $\Delta_+\theta$) stronger.

If Fig. 3 confirmed our expectations on the learning rule introduced with Eq. 5, Fig. 4 shows the model dependence on the initial condition and demonstrates the existence of an optimal sparsity level for the task. The left panel of Fig. 4 reports the mean square error as a function of sparsity during the learning process. The various percentages of active nodes are obtained through the initialization procedure, that is thresholds values defined as various percentiles of the $\mathbf{V}$ distribution. Specifically, the different percentiles, and of course the initial percent of sparsity, are $n = [10, 20, 30, 40, 50, 60, 70, 80, 90]$. Colours report different training instance, and the results corresponding to the same training time are fitted through a second degree polynomial. It is clear that there is an optimal sparsity level of about 50% where the error is minimized for all the training instance. Furthermore, the change in the threshold values obtained through the learning rule is highlighted by the black dashed lines that connect dots of training instances from the top to the bottom of the graph. All these lines tend approximately toward the optimal representation, showing how the learning rule appropriately modulates the percentage of active nodes. The right panel of Fig. 4 focuses on a single training instance and reports the error as a function of sparsity and specificity. The specificity measure has its peak where the error is smallest, and thus it provides a systematic way to choose the starting condition of the network. Indeed, it is possible to select the thresholds as the percentile value that corresponds to the higher specificity measure.

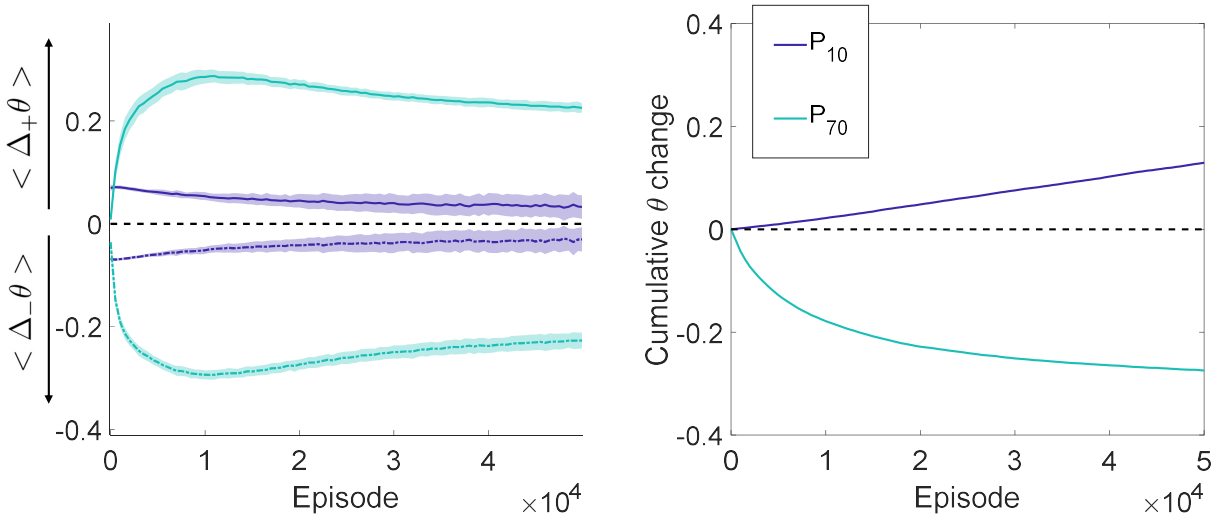

Figure 3: The learning rule for the thresholds is driven by the unbalance between two antagonist forces. **Left**: Analysis of the two forces $\Delta_+\theta$ and $\Delta_-\theta$ involved in the learning rule for the thresholds. The positive y-axis shows a running average of $\Delta_+\theta$ with solid lines, while the negative y-axis shows a running average of $\Delta_-\theta$ with dashed lines. $<>$ indicates averaging across all neurons. $\Delta_+\theta$ increases the threshold values proportionally to the correlation of the activities of the nodes. $\Delta_-\theta$ decreases the threshold values thanks to the positive contribute of the output weights that are connected to the correct output. Colours correspond to initial conditions. **Right**: Average cumulative change of a threshold. If the starting level of sparsity is low (high) the average threshold change is positive (negative).

There is no need to excessively fine-tune the initialization, since the learning rule will optimize the thresholds values anyway. We note also that this simulation is performed through a simple gradient descent algorithm, and that the dependence on the initial conditions of the model can be ameliorated through the utilization of more complex optimizers, as will be shown in section 3.2.

Finally, we compared the performance of the SpaRCe model with:

i) *Standard* Echo-State network, where the same on-line learning is applied to the output weights W^{out} only. We note that the algorithm SpaRCe learns N more parameters (the thresholds) in comparison to the standard Echo-state network.

ii) *Hidden layer*, where we added a full hidden layer of N_h nodes on the top of the reservoir. This approach learns an additional connectivity matrix between the reservoir and the hidden layer, dramatically increasing the number of parameters by a factor of about $N_h N$.

iii) A standard Echo-State network with L_1 or L_2 normalization terms on the output weights.

The top panels of Fig. 5 show the classification accuracy and the root mean square error for the algorithms analysed on a case where the number of sequences to be classified is 192. The SpaRCe model outperforms the standard Echo-state network with or without the penalization terms, which report the worst performance on this specific task. Furthermore, the model showed comparable results to the addition of a full hidden layer with $N_h = 100$ nodes, which increases dramatically the number of parameters to be learned. We note that this surprising result is also due to the specific nature of the task analysed, in which the memorization capacity of the model is the most important factor. In comparison to the addition of a hidden layer, the model SpaRCe provides a cheap formulation to achieve an optimal and reliable sparsity level.

The bottom panel of Fig. 5 shows that the performance of SpaRCe continues to match or exceed the performance of the hidden layer model as the number of stimuli to be classified increases. We can conclude that the SpaRCe model improved considerably the performance and the convergence time of a reservoir on this biologically inspired task. The next section is dedicated to the results achieved by the model on a more concrete machine learning ap-

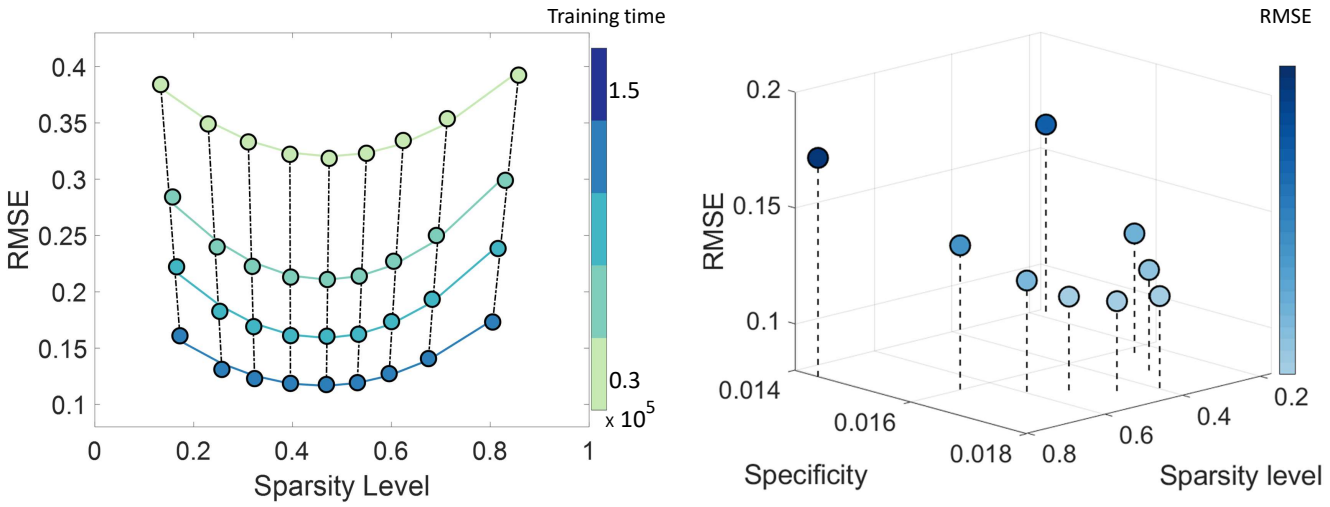

Figure 4: The learning process modulates the sparsity level in the network toward an optimal level of percentage of active nodes. **Left**: Performance as a function of sparsity for different training instance of the model (a color represents a specific training time, which increases from top to bottom). For each instance the results are fitted with a second degree polynomial that has clearly a minimum around 0.5 on the x-axis, demonstrating the existence of an optimal percentage of active nodes. The dashed line connecting the results for diverse training time highlights the change in the sparsity level achieved through the learning rule 5. **Right**: Performance as a function of sparsity and specificity. The best performance correspond to the highest specificity value, demonstrating the interpretability of the model.

plication.

$$\mathbf{V}(t+1) = (1-\alpha)\mathbf{V}(t) + \alpha f\big[W_{in}\mathbf{s} + \rho W\mathbf{V}(t)\big] \tag{12}$$

$$\mathbf{x}(t) = sign\Big(\mathbf{V}(t)\Big) relu\Big(|\mathbf{V}(t)| - \boldsymbol{\theta}_t\Big) \tag{13}$$

3.2 MNIST Database

In this section we faced a classification task on the MNIST dataset. Each image is fed into the network sequentially one column at a time to make the task temporally dependent. Thus, one written digit corresponds to a sequence of 28 time steps of a 28 dimensional input. The application of Echo state networks on this specific task was previously analysed in [35], in which the original dataset was preprocessed and augmented by resizing and deforming the original images. Without such a preprocessing, the Echo state network could not clearly outperform a simple perceptron [35]. To increase the capacity of the reservoir we concatenated all the temporal states $\mathbf{V}(t)_{t=1,...,T}$ into a matrix $\mathbf{V} = [\mathbf{V}(1), ..., \mathbf{V}(t), ..., \mathbf{V}(T)]$. In order to take into account for the temporal variability and dynamic of the $\mathbf{V}$ variable across time, we optimized a vector of thresholds for each time step of the network. Eq. 2 can be rewritten as

where the activation function is a hyperbolic tangent and the W_{In} matrix is full and its elements are drawn from a Gaussian distribution with zero mean and unitary variance. This choice of the setting of the reservoir is probably the most common and historically known. In Eq. 13 the time dependent thresholds $\boldsymbol{\theta}_t$ are applied on the absolute value of the $\mathbf{V}$ variable to accommodate the negative values that $\mathbf{V}$ can assume. For each separate time step of the input sequence the starting values of the thresholds $\boldsymbol{\theta}$ are computed as percentiles of the absolute values $|\mathbf{V}(t)|$. In practice, this is done by feeding the training data set into the network once, by computing the distribution $|V_i(t)|$ for all nodes and all t, and finally by setting each starting value of θ_i as the n-th percentile $P_{n,i}(t)$ of the distribution $|V_i(t)|$. Once $\mathbf{x}(t)$ is computed, we concatenated all the sparse representations obtained for all time steps defining a vector $\mathbf{X} = \big[\mathbf{x}(0), ..., \mathbf{x}(t), ..., \mathbf{x}(T)\big]$

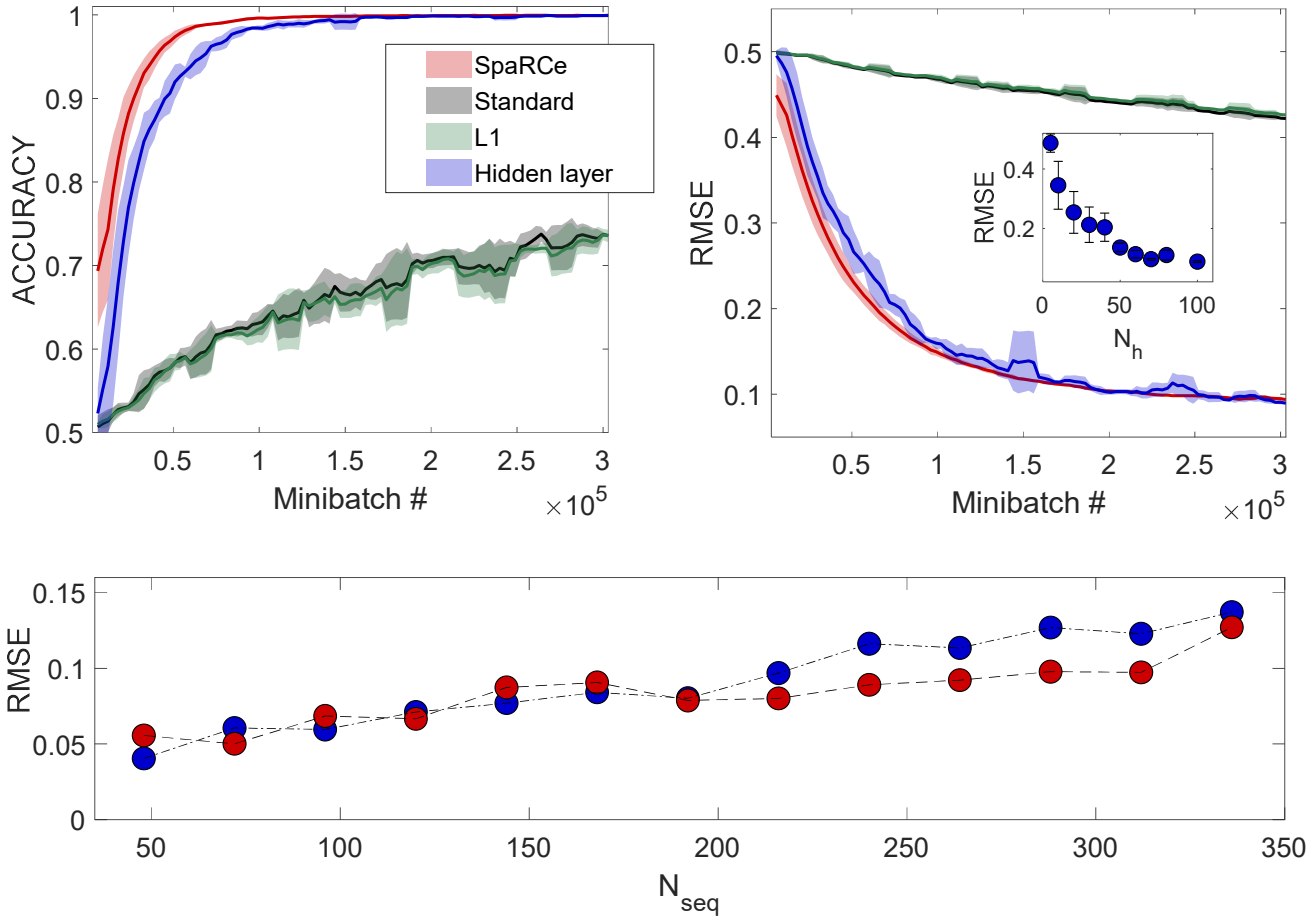

Figure 5: The SpaRCe algorithm reports comparable performance to the addition of a hidden layer with backpropagation. **Top**: Classification accuracy and root mean square error of the models for a case where the number of sequences to be classified is 192. Each minibatch corresponds to the presentation of 20 training samples. The inset on the right shows the performance of the hidden layer as the number of nodes in the hidden representation varies. **Bottom**: Performance as the number of inputs to be classified increases, for the hidden layer model (blue) and SpaRCe (red).

from which we train the output weights. A scheme of the procedure described above is depicted in the left panel of Fig. 6, while the right panel of Fig. 6 shows an example of the network dynamics and an average of the starting thresholds over nodes across time. The cost function adopted for this task is a sigmoidal cross entropy

$$E = -\sum_j \tilde{y}_j log\Big(\sigma(y_j)\Big) \tag{14}$$

whose application on the model is analysed in 5.4. The optimizer used is Adam.

Fig. 7 shows the results of the SpaRCe model from various initial sparsity levels (colours). Independently of the starting conditions and considering that SpaRCe exploits local learning rules, the model reaches performance levels comparable to those achieved through the utilization of two-layer or three-layer neural networks with backpropagation [36]. We note that convolutional neural networks are the best performing networks for this task and for images classification problems in general. The best performance corresponds to an error rate of 0.21% on the MNIST dataset and it is achieved through a pool of five convolutional neural networks [36]. However, the approach faced here is not specific for visual data and the task is more complex because of the sequential nature of the input. The minimum error achieved by SpaRCe is

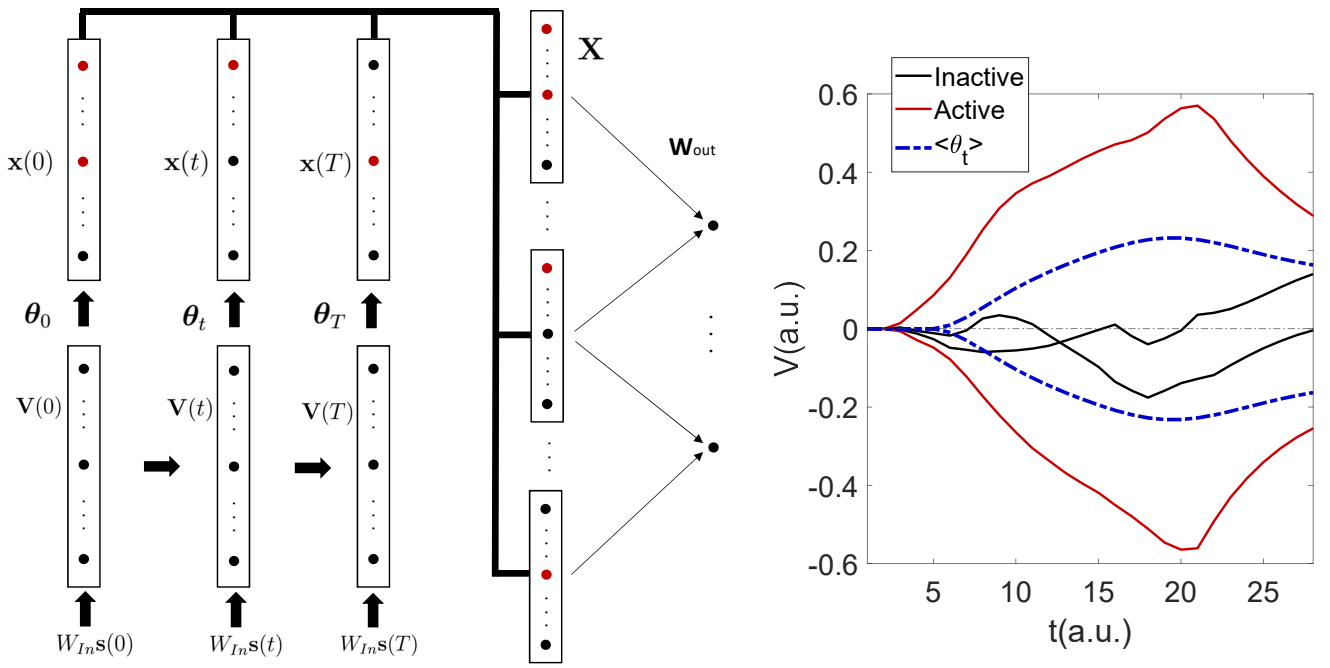

Figure 6: We increased the dimensionality of the representation by concatenating previous temporal activities, which are rectified through dynamic thresholds. **Left**: Scheme of the model. The learnable thresholds $\boldsymbol{\theta}_t$ are time dependent. Each sparse representation $\mathbf{x}(t)$ is concatenated to form the output layer from which the output weights are learned. The concatenation of previous time steps activities enrich the representation and permits the model to exploit the trajectory of the dynamic system (the network) to perform the classification task. **Right:** Example of $\mathbf{V}$ activities of active nodes ($|V_i(t)| > \theta_{i,t}$, red lines) and inactive nodes ($|V_j(t)| < \theta_{j,t}$, black lines), and the average value of the thresholds across time (dashed blue lines).

1.7% [3]. The sizes of the dots reflect the percentage of active nodes in the network. It is evident that, regardless of the initial condition, the final level of sparsity and the performance shown by the trends are similar.

4 Discussion

Typically in Machine Learning sparseness is introduced via regularisation: a penalty term is added to the error function of the network, which leads to increased error proportional to the usage of the weight parameters. This is a technique related to overfitting rather than directly to performance. From the point of the view of neuroscience, sparseness is related to the percentage of neurons that are active per stimulus, suggesting constraints not necessarily on the weights but rather directly on the neuronal activity. In this work, we take the latter approach. We learn a threshold per neuron via the minimisation of standard error functions, linking therefore sparseness to the performance of the network. We also demonstrate theoretically that such a rule effectively reduces the number of reservoir neurons with correlated activities that have a similar contribution to the output neurons, i.e. are connected to the same output neuron with weights of the same sign.

Because we formulated the recurrent network by having one observable variable per neuron (the thresholded activity) and one hidden variable (the activity before the threshold), learning does not disrupt the dynamics of neurons in the reservoir that interact via the hidden variable. A biological interpretation of this structure could be that the recurrently connected neurons signal to each other based on subthreshold depolarization rather than action potentials. Such signalling could occur through dendro-dendritic synapses, which have been observed in the fly mushroom body [22], the structure that inspired the task in section 3.1.

Threshold learning also leads to higher neuronal specialisation, i.e. neurons preferably fire for one

[3] The values of the hyperparameters adopted can be found in the table in section 5.

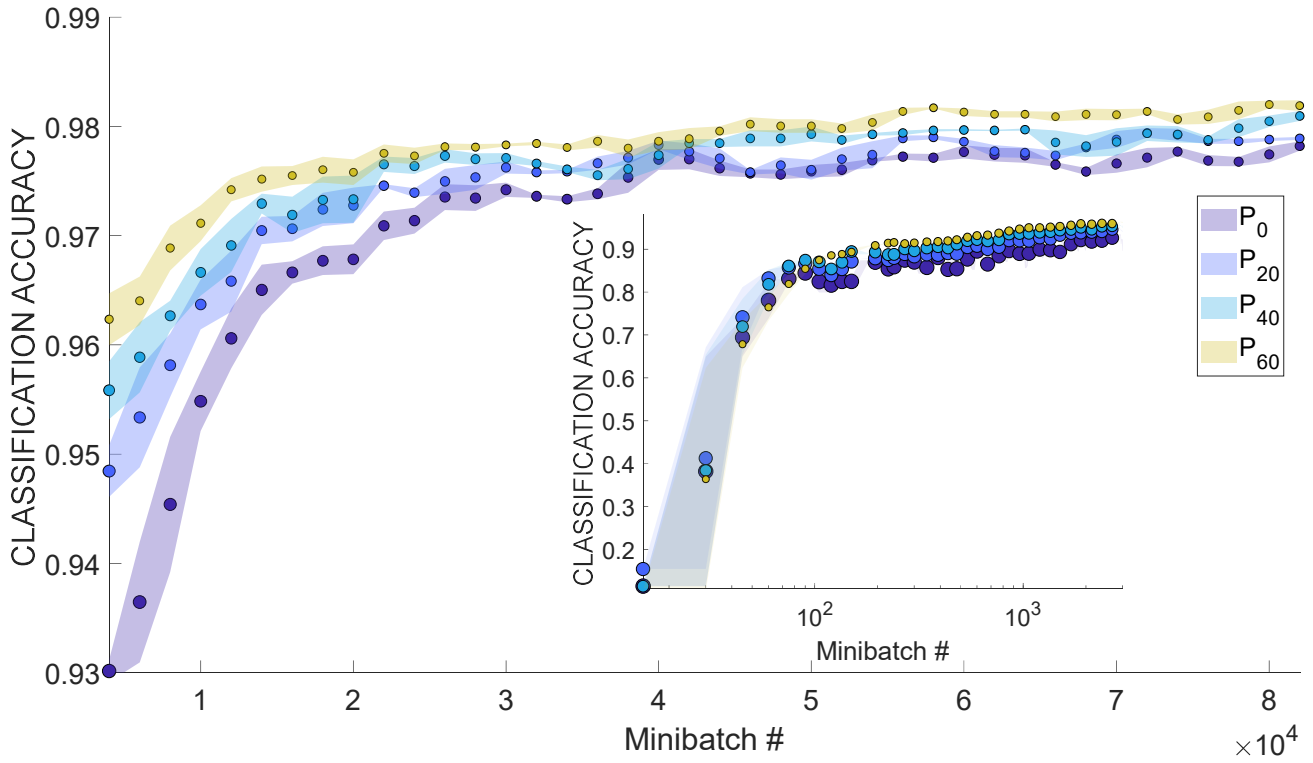

Figure 7: The SpaRCe model shows comparable performance to a 2/3 hidden layers neural network on the MNIST dataset. The sizes of the dots reflect the percentage of active nodes in the network. Each minibatch corresponds to the presentation of 20 training samples. The abscissa of the inset figure is scaled logarithmically.

class vs the other, which we demonstrate experimentally by defining a measure of specificity and showing its relationship to performance in simulations. These results hold also for multi-class problems. We also compare learning with and without thresholds on the same network, and find that there are advantages for both the speed and the accuracy of learning. While this might not be particularly surprising, as we have introduced additional N parameters, one per reservoir neuron, the performance is still equal or better than in the case where we introduce a hidden layer, whose parameters are learned via backpropagation.

In the case of the hidden layer, we learn additional parameters $N \times N_h + N$, and performance only becomes similar to the threshold model for $N_h = 100$. A possible interpretation of why SpaRCe performs so well is the following: threshold learning reduces redundancies in the reservoir, and therefore the network has to learn the read-out weights in a smaller weight space. It is worthy noting that this is a two-way interaction: the threshold changes depend proportionally to the size of the weights. Statistically an increase of specificity follows large weight changes.

Most interestingly, the learning rule we derive is identical to the update rule for threshold via backpropagation, since our formulation is effectively equivalent to adding a hidden layer with a very specific architecture, a one to one connectivity with the neurons of the reservoir. Hence the advantage in relation to training a fully connected network learning comes from the architecture, as demonstrated via a simple mathematical analysis. Another advantage of the threshold learning is that it helps stabilise the network if a large learning rate has been selected. In simulations, learning rate values that lead to instabilities in the learning in the non-threshold model allow for excellent performance in the threshold model: the thresholds act as a stabilisation mechanism, by quickly decreasing the activity of the network through a faster deactivation of neurons. It is possible that this also a contributing factor to the faster learning observed in the simula-

tion. A key ingredient to the rule and its remarkably consistent performance regardless of the exact initialisation conditions is the threshold initialisation process. The gradient rule for the threshold cannot activate silent neurons. Therefore, if the initialisation 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. To prevent this issue, the whole input 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, making sure that each neuron is active for a pre-decided percentage of time, across all stimulus presentations. In fact, one doesn't need to use the exact input of the network, but any signal(s) with the same statistics as the actual input. Similarly, it turns out that while some initial values may be better in terms of performance, in practice all that is needed is to give the same chance to all neurons to be active during the stimulus presentation.

Finally, reservoir computing has an increasing interest for the neuromorphic computing community, particularly those who aim to use material dynamics for computation, for instance the spintronic community. As the reservoir is present to only serve as a spatiotemporal kernel [37], increasing therefore the dimensionality of the input signal and allowing for a linear model (a perceptron) to separate the classes, it can also be replaced with any system that transforms its input to an output of appropriate richness, so that separation by a perceptron will be allowed. Such proof of concept systems can be found for instance in [38] [39]. Our algorithm does not impose any modification to the reservoir itself, which allows its use even when the recurrent network is replaced with a physical material.

5 Supplementary materials

5.1 Reservoir initialization

The equation describing the dynamic of reservoir of leaky integrators is

$$\mathbf{V}(t+1) = (1-\alpha)\mathbf{V}(t) + \alpha f\big[W_{in}\mathbf{s} + \rho W\mathbf{V}(t)\big] \quad (15)$$

where W is a connectivity matrix whose eigenvalues are uniformly distributed inside the unit circle of the imaginary plane, and $\rho < 1$ is a constant. Given the eigenvalues λ_W of W, the eigenvalues λ of the linearised dynamic system associated to Eq. 15 are

$$\lambda = \mathbb{1}_N(1-\alpha) + \alpha\rho\mathbb{1}_N\lambda(W) \quad (16)$$

and thus λ_W are compressed by a factor α and translated by a factor $1-\alpha$ in the imaginary plane. As a consequence, λ follows the probability distribution

$$p(x,y) = \begin{cases} \dfrac{1}{\pi\alpha^2\rho^2}, & \text{if } \big[x-(1-\alpha)\big]^2 + y^2 \leq \alpha^2\rho^2 \\ 0, & \text{otherwise} \end{cases} \quad (17)$$

where $x = Re(\lambda)$ and $y = Im(\lambda)$ for simplicity of notation. Since the real part of the eigenvalues is associated to the timescales τ of the dynamic system as $Re(\lambda) = \exp(-\dfrac{\delta t}{\tau}) \approx 1 - \dfrac{\delta t}{\tau}$, it is possible to compute the marginal distribution over x of $p(x,y)$ for the real part, and then compute the distribution of timescales. Fig. 8 shows the result of this procedure.

A simple strategy to choose α and ρ by knowing the range of the timescales $[\tau_m, \tau_M]$ that the network should exhibit is to notice how the fastest (slowest) timescale τ_m (τ_M) is given by the minimum (maximum) real eigenvalue of the dynamic system. Calling $\lambda_m = min\big\{Re(\lambda)\big\}$ and $\lambda_M = max\big\{Re(\lambda)\big\}$, we have

$$\begin{aligned} \lambda_m = 1-\alpha-\alpha\rho \approx 1-2\alpha = \\ = \exp(-2\alpha) = \exp(-\delta t/\tau_m) \rightarrow \\ \rightarrow \alpha = \frac{\delta t}{2\tau_m} \end{aligned}$$

and

$$\begin{aligned} \lambda_M = 1-\alpha-\alpha\rho = 1-(\alpha-\alpha\rho) \approx \\ = \exp(-\alpha-\alpha\rho) = \exp(-\frac{\delta t}{\tau_M}) \rightarrow \\ \rightarrow \rho = 1 - 2\frac{\tau_m}{\tau_M} \end{aligned}$$

that are relations between α, ρ and the minimum and maximum timescales that the model can exhibit. In this way, it is possible to choose the hyperparameters α and ρ by selecting a priori the more interpretable parameters τ_m and τ_M and by considering that the timescales would approximately follow the distribution in Fig. 8. We want to emphasize that this procedure does not guarantee an optimal choice of the hyperparameters, but it can guide the research and it assures a good choice in terms of temporal memory of the reservoir.

5.2 Thresholds initialization

Fig. 9 shows the $\mathbf{V}$ distribution for two example neurons and the corresponding starting threshold values.

5.3 Cross entropy loss

The error function has the form

$$E = -\sum_j \tilde{y}_j log\Big(\sigma(y_j)\Big) \quad (18)$$

The learning rule for the thresholds is

$$\begin{aligned} \Delta\theta_k = -\eta_\theta \sum_j \tilde{y}_j\big(1-\sigma(y_j)\big)W_{jk}H(V_k-\theta_k) = \\ = -\eta_\theta \sum_j \tilde{y}_j W_{jk}H(V_k-\theta_k) + \\ +\eta_\theta \sum_j \tilde{y}_j\sigma(y_j)W_{jk}H(V_k-\theta_k) \end{aligned} \quad (19)$$

The two terms in Eq. 19 have comparable meaning to $\Delta_-\theta$ and $\Delta_+\theta$ of Eq. 7 and 8 computed for the mean square error, To demonstrate this, we can consider the case of a classification task where $y_j^{true} = 1$ for the correct class and zero otherwise. Furthermore, considering that the neural network output is not in the saturating regime of the sigmoid function when the majority of the learning happens, we

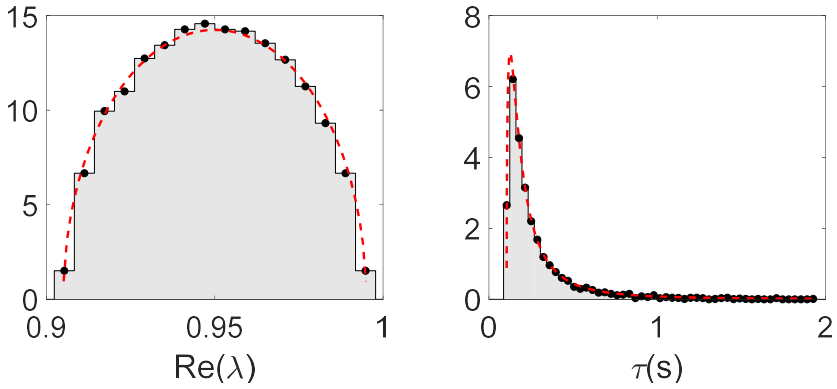

Figure 8: **Left**: Real part of the eigenvalues and theoretical distribution. **Right**: Timescales and theoretical distribution.

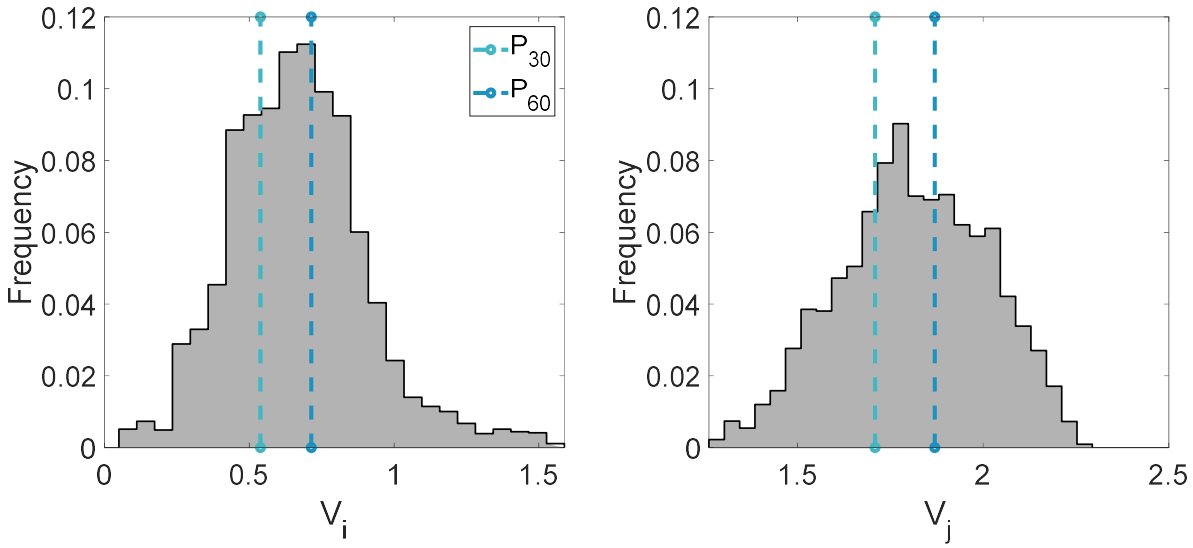

Figure 9: The initialization of the thresholds is defined in the range of the activity distribution to permit every node to be selectively active. **Left** and **Right**: Distributions of V and corresponding percentiles for two example neurons.

can use the dominant first term of the Taylor series of the sigmoid and approximate the second term of Eq. 19

$$\Delta\theta_k = -\eta_\theta W_{\tilde{j}k} H(V_k - \theta_k) + \tag{20}$$
$$+\eta_\theta \sigma(y_{\tilde{j}}) W_{\tilde{j}k} H(V_k - \theta_k) =$$
$$= -\eta_\theta W_{\tilde{j}k} H(V_k - \theta_k) +$$
$$+\eta_\theta \big[\frac{1}{2} + \frac{1}{2} y_{\tilde{j}}\big] W_{\tilde{j}k} H(V_k - \theta_k) =$$
$$= -\eta_\theta W_{\tilde{j}k} H(V_k - \theta_k) + \tag{21}$$
$$+\eta_\theta \sum_l W_{\tilde{j}l} W_{\tilde{j}k} relu(V_l - \theta_l) H(V_k - \theta_k) \tag{22}$$

that have the exact same form as Eq. 7 and 8 considering only the correct output class $\tilde{j}$.

5.4 Procedure for building sequences

Given an ensemble of elements $\mathcal{E} = \{A, B, C, ...\}$, we formulated a systematic procedure to build successions of N_t [4] elements from $\mathcal{E}$. Considering an N_{class} classification task, we selected $N_t N_{class} N_{basis}$ random elements from $\mathcal{E}$ without repetitions and composed sequences of length N_t, where N_{basis} is a free parameter that is proportional to the number of sequences that this procedure will define. Then, we divided the $N_{class} N_{basis}$ sequences into N_{class} subgroups of N_{basis} successions; we call each of these

[4] In the case analysed, $N_t = 3$

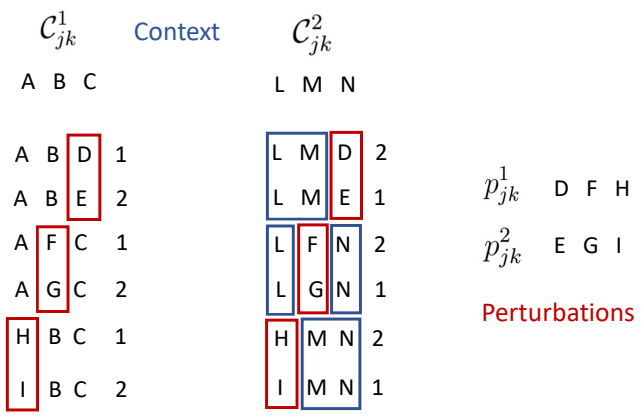

Figure 10: Scheme of the the procedure to define sequences.

subgroups a context group $\mathcal{C}^i_{jk}$. A context group can be represented in the matrix notation

$$\mathcal{C}^i_{jk} = \begin{pmatrix} c^i_{11} & \dots & c^i_{1k} & \dots & c^i_{1N_t} \\ \vdots & & \vdots & & \vdots \\ c^i_{j1} & \dots & c^i_{jk} & \dots & c^i_{jN_t} \\ \vdots & & \vdots & & \vdots \\ c^i_{N_{basis}1} & \dots & c^i_{N_{basis}k} & \dots & c^i_{N_{basis}N_t} \end{pmatrix}$$

We selected other $N_t N_{class} N_{basis}$ random elements without repetitions from $\mathcal{E}$; we call these elements perturbations p^l_{jk}, where $l = 1, ..., N_{class}$, $j = 1, ..., N_{basis}$ and $k = 1, ..., N_t$. Considering the sequence $C^i_j = [c^i_{j1}, ..., c^i_{jk}, ..., c^i_{jN_t}]$, that is the j-th row of $\mathcal{C}^i_{jk}$, we substituted each element k of the sequence C^i_j with N_{class} different perturbations p^l_{jk}, with $l = 1, ..., N_{class}$, once at a time to obtain

$$C^i_j \to \tilde{s}^i_j = \begin{pmatrix} p^1_{j1} & \dots & c^i_{jk} & \dots & c^i_{jN_t} \\ & & \dots & & \\ p^{N_{class}}_{j1} & \dots & c^i_{jk} & \dots & c^i_{jN_t} \\ \vdots & & \vdots & & \vdots \\ c^i_{jk} & \dots & p^1_{jk} & \dots & c^i_{jN_t} \\ & & \dots & & \\ c^i_{jk} & \dots & p^{N_{class}}_{jk} & \dots & c^i_{jN_t} \\ \vdots & & \vdots & & \vdots \\ c^i_{jk} & \dots & c^i_{jk} & \dots & p^1_{jN_t} \\ & & \dots & & \\ c^i_{jk} & \dots & c^i_{jk} & \dots & p^{N_{class}}_{jN_t} \end{pmatrix}$$

and associated each new sequence with the class l, which is the apex of the perturbation p^l_{jk}. The matrix corresponding to the ensemble of perturbations applied to a sequence C^i_j is called s^i_j. We iterated this procedure for each row of the context group and applied the same perturbations to different context groups. We define the perturbed context group i as

$$s^i_{jk} = \begin{pmatrix} \tilde{s}^i_1 \\ \vdots \\ \tilde{s}^i_j \\ \vdots \\ \tilde{s}^i_{N_{basis}} \end{pmatrix}$$

The meaning of this procedure is understandable from Fig. 10, which depicts this procedure to generate sequences for a simple case of $N_{basis} = 1$ and $N_{class} = 2$. If the task was restricted to one perturbed context group s^i_{jk}, the memorization of the perturbation elements (capitals inside the red boxes of Fig. 10) in the sequence would be enough to achieve a perfect classification accuracy. However, the repetitions of the perturbations over multiple context elements force the algorithm to consider the whole pattern of elements to achieve a high classification accuracy.

Parameters	Values
Bioinspired Task	
σ	0.3
Δ_t	$0.1s$
T	$0.3s$
Network, Bioinspired/ML task	
δt	$0.01s/0.01s$
α	0.1/0.17
ρ	0.95/0.97
N	1000/1000
Model, Bioinspired/ML task	
η_W	0.002/0.002
η_θ	$10^{-1}\eta_W/10^{-1}\eta_W$
minibatch size	20/20

Acknowledgements

We thank Paolo Del Giudice and Guido Gigante for their input on the analysis of the timescales in the reservoir model. EV and AL would like to acknowledge support from a Google Deepmind Award.

References

[1] Mikhail V Tsodyks and Mikhail V Feigel'man. The enhanced storage capacity in neural networks with low activity level. *EPL (Europhysics Letters)*, 6(2):101, 1988.

[2] MV Tsodyks. Associative memory in asymmetric diluted network with low level of activity. *EPL (Europhysics Letters)*, 7(3):203, 1988.

[3] Bernard Derrida, Elizabeth Gardner, and Anne Zippelius. An exactly solvable asymmetric neural network model. *EPL (Europhysics Letters)*, 4(2):167, 1987.

[4] Daniel J Amit, Hanoch Gutfreund, and Haim Sompolinsky. Storing infinite numbers of patterns in a spin-glass model of neural networks. *Physical Review Letters*, 55(14):1530, 1985.

[5] Sandro Romani, Itai Pinkoviezky, Alon Rubin, and Misha Tsodyks. Scaling laws of associative memory retrieval. *Neural computation*, 25(10):2523–2544, 2013.

[6] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical learning with sparsity: the lasso and generalizations*. Chapman and Hall/CRC, 2015.

[7] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2074–2082. Curran Associates, Inc., 2016.

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[9] Peter M Rasmussen, Lars K Hansen, Kristoffer H Madsen, Nathan W Churchill, and Stephen C Strother. Model sparsity and brain pattern interpretation of classification models in neuroimaging. *Pattern Recognition*, 45(6):2085–2100, 2012.

[10] Edmund T Rolls and Martin J Tovee. Sparseness of the neuronal representation of stimuli in the primate temporal visual cortex. *Journal of neurophysiology*, 73(2):713–726, 1995.

[11] Kyle S Honegger, Robert AA Campbell, and Glenn C Turner. Cellular-resolution population imaging reveals robust sparse coding in the drosophila mushroom body. *Journal of Neuroscience*, 31(33):11772–11785, 2011.

[12] Andrew C Lin, Alexei M Bygrave, Alix De Calignon, Tzumin Lee, and Gero Miesenböck. Sparse, decorrelated odor coding in the mushroom body enhances learned odor discrimination. *Nature neuroscience*, 17(4):559, 2014.

[13] Glenn C Turner, Maxim Bazhenov, and Gilles Laurent. Olfactory representations by drosophila mushroom body neurons. *Journal of neurophysiology*, 99(2):734–746, 2008.

[14] Eyal Gruntman and Glenn C Turner. Integration of the olfactory code across dendritic claws of single mushroom body neurons. *Nature neuroscience*, 16(12):1821, 2013.

[15] Hao Li, Yiming Li, Zhengchang Lei, Kaiyu Wang, and Aike Guo. Transformation of odor selectivity from projection neurons to single mushroom body neurons mapped with dual-color calcium imaging. *Proceedings of the National Academy of Sciences*, 110(29):12084–12089, 2013.

[16] Javier Perez-Orive, Ofer Mazor, Glenn C Turner, Stijn Cassenaer, Rachel I Wilson, and Gilles Laurent. Oscillations and sparsening of odor representations in the mushroom body. *Science*, 297(5580):359–365, 2002.

[17] James M Jeanne and Rachel I Wilson. Convergence, divergence, and reconvergence in a feedforward network improves neural speed and accuracy. *Neuron*, 88(5):1014–1026, 2015.

[18] Rony Azouz and Charles M Gray. Dynamic spike threshold reveals a mechanism for synaptic coincidence detection in cortical neurons in vivo. *Proceedings of the National Academy of Sciences*, 97(14):8110–8115, 2000.

[19] Matthew S Grubb and Juan Burrone. Activity-dependent relocation of the axon initial segment fine-tunes neuronal excitability. *Nature*, 465(7301):1070, 2010.

[20] Herbert Jaeger, Mantas Lukoševičius, Dan Popovici, and Udo Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural networks*, 20(3):335–352, 2007.

[21] Shin-ya Takemura, Yoshinori Aso, Toshihide Hige, Allan Wong, Zhiyuan Lu, C Shan Xu, Patricia K Rivlin, Harald Hess, Ting Zhao, Toufiq Parag, et al. A connectome of a learning and memory center in the adult drosophila brain. *Elife*, 6:e26975, 2017.

[22] Zhihao Zheng, J Scott Lauritzen, Eric Perlman, Camenzind G Robinson, Matthew Nichols, Daniel Milkie, Omar Torrens, John Price, Corey B Fisher, Nadiya Sharifi, Steven A Calle-Schuler, Lucia Kmecova, Iqbal J Ali, Bill Karsh, Eric T Trautman, John A Bogovic, Philipp Hanslovsky, Gregory S X E Jefferis, Michael Kazhdan, Khaled Khairy, Stephan Saalfeld, Richard D Fetter, and Davi D Bock. A Complete Electron Microscopy Volume of the Brain of Adult Drosophila melanogaster. *Cell*, 174(3):730–743.e22, July 2018.

[23] Qingqing Liu, Xing Yang, Jingsong Tian, Zhongbao Gao, Meng Wang, Yan Li, and Aike Guo. Gap junction networks in mushroom bodies participate in visual learning and memory in drosophila. *Elife*, 5:e13238, 2016.

[24] Junzhou Huang, Tong Zhang, and Dimitris Metaxas. Learning with structured sparsity. *Journal of Machine Learning Research*, 12(Nov):3371–3412, 2011.

[25] Emmanuel J Candes, Michael B Wakin, and Stephen P Boyd. Enhancing sparsity by reweighted l 1 minimization. *Journal of Fourier analysis and applications*, 14(5-6):877–905, 2008.

[26] Herbert Jaeger. 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):13, 2001.

[27] Herbert Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*, volume 5. GMD-Forschungszentrum Informationstechnik Bonn, 2002.

[28] Elissa A Hallem and John R Carlson. Coding of odors by a receptor repertoire. *Cell*, 125(1):143–160, 2006.

[29] Shawn R Olsen, Vikas Bhandawat, and Rachel I Wilson. Divisive normalization in olfactory population codes. *Neuron*, 66(2):287–299, 2010.

[30] Sean X Luo, Richard Axel, and LF Abbott. Generating sparse and selective third-order responses in the olfactory system of the fly. *Proceedings of the National Academy of Sciences*, 107(23):10713–10718, 2010.

[31] Moshe Parnas, Andrew C Lin, Wolf Huetteroth, and Gero Miesenböck. Odor discrimination in drosophila: from neural population codes to behavior. *Neuron*, 79(5):932–944, 2013.

[32] Kamesh Krishnamurthy, Ann M Hermundstad, Thierry Mora, Aleksandra M Walczak, and Vijay Balasubramanian. Disorder and the neural representation of complex odors: smelling in the real world. *arXiv preprint arXiv:1707.01962*, 2017.

[33] Sophie JC Caron, Vanessa Ruta, LF Abbott, and Richard Axel. Random convergence of olfactory inputs in the drosophila mushroom body. *Nature*, 497(7447):113, 2013.

[34] Sen Song, Per Jesper Sjöström, Markus Reigl, Sacha Nelson, and Dmitri B Chklovskii. Highly nonrandom features of synaptic connectivity in local cortical circuits. *PLoS Biology*, 3(3):e68, March 2005.

[35] Nils Schaetti, Michel Salomon, and Raphaël Couturier. Echo state networks-based reservoir computing for mnist handwritten digits recognition. In *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, pages 484–491. IEEE, 2016.

[36] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[37] Michiel Hermans and Benjamin Schrauwen. Recurrent kernel machines: Computing with infinite echo state networks. *Neural Computation*, 24(1):104–133, 2012.

[38] Danijela Marković, Nathan Leroux, Mathieu Riou, Flavio Abreu Araujo, Jacob Torrejon, Damien Querlioz, Akio Fukushima, Shinji Yuasa, Juan Trastoy, Paolo Bortolotti, et al. Reservoir computing with the frequency, phase, and amplitude of spin-torque nano-oscillators. *Applied Physics Letters*, 114(1):012409, 2019.

[39] Miguel Romera, Philippe Talatchian, Sumito Tsunegi, Flavio Abreu Araujo, Vincent Cros, Paolo Bortolotti, Juan Trastoy, Kay Yakushiji, Akio Fukushima, Hitoshi Kubota, et al. Vowel recognition with four coupled spin-torque nano-oscillators. *Nature*, 563(7730):230, 2018.

[40] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

[41] Justin Werfel, Xiaohui Xie, and H Sebastian Seung. Learning curves for stochastic gradient descent in linear feedforward networks. In *Advances in neural information processing systems*, pages 1197–1204, 2004.

[42] Scott Waddell. Reinforcement signalling in drosophila; dopamine does it all after all. *Current opinion in neurobiology*, 23(3):324–329, 2013.

[43] Ke Huang and Selin Aviyente. Sparse representation for signal classification. In *Advances in neural information processing systems*, pages 609–616, 2007.