Chaos, Solitons and Fractals 168 (2023) 113128

Contents lists available at ScienceDirect

Chaos, Solitons and Fractals

journal homepage: www.elsevier.com/locate/chaos

Recurrent Spectral Network (RSN): Shaping a discrete map to reach automated classification

Lorenzo Chicchi [a], Duccio Fanelli [a,*], Lorenzo Giambagli [a,b], Lorenzo Buffoni [a,c], Timoteo Carletti [b]

[a] Dipartimento di Fisica e Astronomia, Università di Firenze, INFN and CSDC, Via Sansone 1, 50019 Sesto Fiorentino, Firenze, Italy
[b] naXys, Namur Center for Complex Systems, University of Namur, rue Grafé 2, B 5000 Namur, Belgium
[c] Physics of Information and Quantum Technologies Group, Instituto de Telecomunicacoes, University of Lisbon, Portugal

ARTICLE INFO

Keywords:
Machine learning
Dynamical systems
Spectral learning

ABSTRACT

A novel strategy to automated classification is introduced which exploits a fully trained dynamical system to steer items belonging to different categories towards distinct asymptotic target destinations. These latter are incorporated into the model by taking advantage of the spectral decomposition of the operator that rules the linear evolution across the processing network. Non-linear terms act for a transient and allow to disentangle the data supplied as initial condition to the discrete dynamical system. The system effectively aligns along assigned directions, which reflect the specificity of the provided input and that are encoded in the loss function via suitable spectral projections. The network can be equipped with several memory kernels which can be sequentially activated for serial datasets handling. Our novel approach to classification, that we here term Recurrent Spectral Network (RSN), is successfully challenged against a simple test-bed model, created for illustrative purposes, as well as a standard dataset for image processing training.

1. Introduction

Machine learning (ML) technologies are becoming increasingly popular due to their inherent degree of transversal adaptability, which transcends different realms of applications [1–7]. Multi-layered feed-forward neural networks, notably the most basic architecture schemes, are routinely employed to cope with a large plethora of case studies and test-bed models. ML seeks at solving an optimization problem, upon minimization of a suitably defined loss function which confronts the expected target to the output produced at the exit layer, after a nested sequence of linear (across layers) and non-linear (punctually localized on the nodes) manipulations of the data supplied as an entry. The target of the optimization are the weights of the links that connect pair of nodes belonging to adjacent stacks of the multi-layered arrangement, in a fully coupled setting. An alternative training scheme has been recently proposed which anchors the learning to reciprocal domain [8,9]: the eigenvalues and the eigenvectors of the transfer operators get adjusted by the optimization. Spectral learning, so far engineered to deal with a feedforward organization, identifies key collective variables, the eigenvalues, which are more fundamental than any other (randomly selected) set of identical cardinality, allocated in direct space. Extending the learning to include the eigenvectors enhances the ability of the network to carry out the assigned task.

Delving into the principles of the spectral methodology, we here propose a radically novel approach to computational machine learning which is deeply rooted into the theory of discrete dynamical systems. In a nutshell, the incoming signal is processed by successive iterations across the very same constellation of nodes. The links, and thus the topology of the ensuing network, are fixed and shaped under the spectral paradigm, upon optimization at a given number of iterations. Non-linearities acting on the nodes are imposed a priori or, conversely, learned self-consistently via an apposite deep neural network, which is embedded into the cost function. In either settings, non-linear terms acting in real space at the nodes locations, are forced to vanish asymptotically, iteration after iteration, in such a way that the dynamics eventually turns purely linear. The linear operator, mirroring the processing network, possesses a high dimensional attracting linear manifold spanned by the eigenvectors associated to the eigenvalues equal to one. These latter come in a number that matches the classes to be eventually categorized. A suitable non linear spectral filter is enforced in the loss function to project the ensuing direction along a given eigenvector, assumed as the destination target of a class of homologous entities and selected from those displaying unitary eigenvalues [10]. Stated differently, the classification is accomplished when the processed output - approximately - aligns along a specific direction in dual space, instead of turning active a single node in direct space,

* Corresponding author.
E-mail address: duccio.fanelli@gmail.com (D. Fanelli).

https://doi.org/10.1016/j.chaos.2023.113128
Received 18 July 2022; Received in revised form 19 December 2022; Accepted 7 January 2023
Available online 17 January 2023

as customarily done. This formulation yields a rather natural interpretation of the classifier operational mode: non-linearities, acting at the early stages of the dynamical evolution, drive the discrete dynamical system towards distinct effective stationary equilibria, self-consistently sculpted across the learning scheme and associated to different classes of supplied items. Delineating the non-trivial contours that separate the inspected classes in the input space constitutes the tangible outcome of the learning scheme. Remarkably, the trained dynamical system can be iterated forward in time, beyond the limited horizon of the learning procedure: the ability of classifying stays unchanged. The eigenvectors associated to eigenvalues equal to one, are hence veritable memory kernels where the information is kept stored. We name Recurrent Spectral Network (RSN) our novel approach to automated classification via sculpting the attracting invariant subspace of a discrete dynamical map.

Points of connections are found with the framework of reservoir computing. In this latter case, input signals are mapped into higher dimensional computational spaces through the dynamics of a fixed, non-linear system termed reservoir [11–13]. Within the RSN, the bulk model is not fixed but self-consistently tailored to the assigned task.

A straightforward variant of the RSN recipe, which accounts for quasi-orthogonal eigen-directions for each processed task, can be also introduced. This latter enables for the sequential handling of different datasets. In simple terms, an artificial computing unit can be assembled which keeps memory of a task, for which it was initially trained, while being exposed to another training session, with an independent dataset to be processed. This is at present arduous with standard approaches to machine learning, as the second learning stage causes the so-called catastrophic forgetting taking over any form of digital consciousness inherited from the first [14–16]. Few attempts have been so far reported which aim at overcoming this limitation [17–19].

The paper is organized as follows. In the next Section we introduce the mathematical notation and the relevant model setting. Then, in the subsequent Section, we will turn to considering a simple example of a dataset defined in $\mathbb{R}^2$ that will prove useful for clarifying the essence of the proposed methodology. In particular we will show, that the system can effectively trace the boundaries that non-linearly separate different classes within a given datasets. Each class is evolved towards a distinct target, that we identify with a specific direction of the attracting subspace possessed by the underlying linear system. Further, we will proceed by applying the proposed technique to the celebrated MNIST dataset [20]. We will also show that the RSN can also handle multiple datasets with a modest drop in the peak accuracy, and following sequential stages of learning. Finally, we will sum up and draw our conclusions.

2. The mathematical foundation

Consider N isolated nodes. Our aim is to assign weighted links among the latter, in such a way that the ensuing network can cope with the assigned task, as e.g., classification of different items in distinct categories. Here, N can coincide with the number of input variables (e.g., the pixels of a supplied image): in this case, the nodes where reading is performed match the units where calculations are carried out. This is at variance with usual feedforward deep neural networks, where the information to be processed flows from the input to the output, the collection of computing neurons growing with the number of layers that define the underlying architecture [5–7]. Working within the proposed framework, the topology of the network will unfold as an emerging byproduct of the optimization procedure. As we shall discuss, N can be larger than the characteristic dimension of the input data, a setting that we will specifically assume when dealing with the problem of sequential learning, with dedicated memory kernels.

Denote by $\vec{x}^{(0)}$ the input vector, made of N entries organized in a column. The idea that we shall hereafter develop is to set up a recursive scheme, the Recurrent Spectral Network (RSN), that takes $\vec{x}^{(0)}$ as the initial condition and transforms it via successive iterations into a stationary stable output. This latter should somehow reflect the specific traits of the input items, as identified self-consistently upon dedicated training sessions. Different objects should eventually align along distinct directions of the attracting manifold, depending on the category of specific pertinence. Stated differently, the multidimensional space where the examined objects belong to gets partitioned in mutually exclusive portions, as tailored by suited non-linearities, each associated to a definite asymptotic destination. In the following, we shall label with n the number of independent target directions, namely the number of independent classes in which the inspected dataset can be eventually partitioned.

Assume $\vec{x}^{(k)}$ to represent the image of the input vector $\vec{x}^{(1)}$ after k application of the iterative scheme. Then:

$$\vec{x}^{(k+1)} = \vec{f}_k\left(\mathbf{A}\vec{x}^{(k)}\right) \tag{1}$$

where $\mathbf{A}$ is a $N \times N$ weighed adjacency matrix that defines the patterns of interactions among nodes; $\vec{f}_k(\cdot)$ is a non-linear (N- dimensional) function that depends on the iteration parameter k and which acts at the level of individual nodes. We require in particular $\lim_{k\to\infty} \vec{f}_k \to \vec{\mathbb{1}} \equiv (1, 1, \dots, 1)^T$, in such a way that, for large enough k, the system approximately follows a linear update rule. This is achieved by setting:

$$\vec{f}_k(\cdot) = \vec{\mathbb{1}} + \frac{\vec{g}(\cdot)}{k^\gamma} \tag{2}$$

where $\vec{g}(\cdot)$ is a non-linear function which can be imposed a priori or determined self-consistently via a neural network regression model and γ is a parameter that can be freely adjusted (here we chose to set $\gamma = 1.5$). Focus now on the linear component of the dynamics, as encapsulated in matrix $\mathbf{A}$, which takes over for sufficiently large k. We cast in particular $\mathbf{A} = \mathbf{\Phi}\mathbf{\Lambda}(\mathbf{\Phi})^{-1}$, by invoking spectral decomposition. Here, $\mathbf{\Lambda}$ is the diagonal matrix of the eigenvalues $(\lambda_1, \lambda_2, \dots, \lambda_N)$. Working in the spectral domain enables us to enforce n-dimensional attracting subspace. To this end, we impose $\lambda_1 = \lambda_2 = \cdots = \lambda_n = 1$, and assume $|\lambda_i| < 1$ for $i > n$. These latter $N - n$ quantities are among the target of the optimization scheme. Moreover, we assume $\vec{\phi}_1$, $\vec{\phi}_2$, $\dots$, $\vec{\phi}_n$, namely the eigenvectors relative to the eigenvalues identically equal to one, to identify frozen linearly independent directions of the embedding N-dimensional space. The remaining eigenvectors ($\vec{\phi}_i$, with $i > n$, relative to eigenvalues λ_i) can be freely adjusted, so contributing with a total of $(N - n) \times N$ tunable parameters to the optimization scheme. When $k \gg 1$, non-linear terms fade away and the iterative scheme converges to a linear map, $\vec{x}^{(k+1)} \simeq \mathbf{A}\vec{x}^{(k)}$.

By definition, $\vec{\phi}_i$, with $i \leq n$ are stationary solutions of the above system. This latter is hence associated with a high dimensional attracting invariant manifold: any linear combination of $\vec{\phi}_i$ with $i \leq n$ is in fact a stationary solution of the linear dynamics that is approached by the examined non linear system, for large enough iterations k. By acting on the collection of tunable spectral parameters, which ultimately echo on the topology of the network made of N computing nodes, and exploiting the non-linearities that act over a finite transient, we aim at steering different input objects towards distinct target solutions, which can be stably maintained beyond the limited horizon of the performed training. To rephrase in words, we postulate that any generically complex classification task is eventually amenable to a multi-dimensional linear problem, with properly tuned interactions strengths and provided non-linearities, imposed or self-consistently learned, are made to initially deform the features landscape.

To implement the learning scheme on these basis, we consider $\vec{x}^{(\bar{k})}$, the image on the output layer of the input vector $\vec{x}^{(0)}$ after $\bar{k}$ iterations of the iterative algorithm, where $\bar{k}$ is sufficiently large for the linear approximation to hold true. Then, we calculate $\vec{c}_{\bar{k}} = (\mathbf{\Phi})^{-1}\vec{x}^{(\bar{k})}$: the ith element $\left(\vec{c}_{\bar{k}}\right)_i$ represents the projection of $\vec{x}^{(\bar{k})}$ along the eigen-direction $\vec{\phi}_i$. Each element of the training set is associated to a label $\ell \leq n$ to identify the category to which $\vec{x}^{(0)}$ belongs to. Then, an optimization is carried out which seeks at minimizing the squared distance of $\vec{c}_{\bar{k}}$

(that implicitly depend on the training parameters) with a target n-dimensional column vector $\vec{c}_\ell$, made by zeros except for the element in position ℓ which is set to unit. In such a way, we require that after sufficiently many iterations the dynamical map aligns (as much as possible) along the direction $\vec{\phi}_\ell$, where ℓ identifies the class to which the supplied entry refers. Different initial conditions, decorated with their reference labels pointing to one of the n classes, are forced (by a proper use of the non-linearities, as vehiculated by the network arrangement) to yield different asymptotic equilibria, which approximately align along distinct directions in reciprocal space. A perfect alignment along the eigen-modes that flag distinct classes can be eventually forced by performing a projection along the most represented direction, at the end of the iterative update.

Operatively, we begin by initializing the trainable portion of the eigenvectors matrix $\mathbf{\Phi}$ with a random uniform distribution of the assigned entries. Similarly, for the $N - n$ trainable eigenvalues that enter the definition of matrix $\mathbf{\Lambda}$. Then, we define a global model (via Tensorflow [21]) that implements a chain of successive applications of the linear mapping $\mathbf{A}$. Each linear transfer is followed by the application of the non-linear filter as specified by Eq. (2), which acts at the nodes location. Matrix $\mathbf{A}$ is written in terms of its spectral decomposition by composing together the three matrices $(\mathbf{\Phi})^{-1}$, $\mathbf{\Lambda}$ and $\mathbf{\Phi}$, as introduced above. The number of iterations is set to $\bar{k}$, a parameter supplied as an input. After iteration $\bar{k}$, we apply one more time matrix $(\mathbf{\Phi})^{-1}$ to obtain the coefficients $\vec{c}_{\bar{k}}$ that enter the definition of the loss function. The trainable weights of the model are updated according to the gradient descent rule, the loss function gradients being estimated via a standard backpropagation algorithm.

In the following Section, to challenge the effectiveness of the proposed recipe, we set to study a simple dataset defined in $\mathbb{R}^2$, which bears pedagogical interest. We will then turn, in a subsequent Section, to examining the ability of the RSN methodologies to cope with a standard datasets of image.

3. Testing RSN: A simple dataset in $\mathbb{R}^2$

As mentioned above, we aim at testing the RSN as outlined above against a simple dataset, created for this specific purpose. The goals are twofolds. On the one side, we wish to provide the first consistent implementation of the procedure, by showing that a dynamical system can be trained which preserves its ability to discern beyond the horizon of the training (as instead it is the case for conventional recurrent neural networks). This is an indirect mark of the imposed convergence towards an asymptotic equilibrium, inherent to the dynamical scheme, which flags the class to be identified. Then, we shall convincingly demonstrate that classification by RSN amounts to segmenting the space of the initial conditions in disconnected domains, each pointing to a distinct asymptotic direction, within the invariant attracting manifold. Indeed, the trained map will make a single target mode, representative of the processed class, to stand out as compared to the other. The degree of alignment as observed empirically improves with the complexity of the explored dataset, as we shall remark in the following section. A perfect alignment can be forced by means of a suitable non-linearity that implements a punctual projection along the most represented direction, at final iteration.

The dataset that we shall here consider as a proof of concept is composed by two sets of points, laying on the plane. The points falling inside the unitary circle, centered at the origin, define the first class (displayed in yellow, in Fig. 1). Those situated outside the circle and inside a square domain of linear width $L = \sqrt{2\pi}$, contribute to the second reservoir of datapoints (shown in blue, in Fig. 1). The size of the square has been chosen in such a way that the surface of the two regions where the dataset insists is equal. The two sets are divided by a non-linear boundary that coincides with the perimeter of the unitary circle. Our objective is to train a RSN, following the prescriptions of the preceding Section, so as to associate any given point — randomly

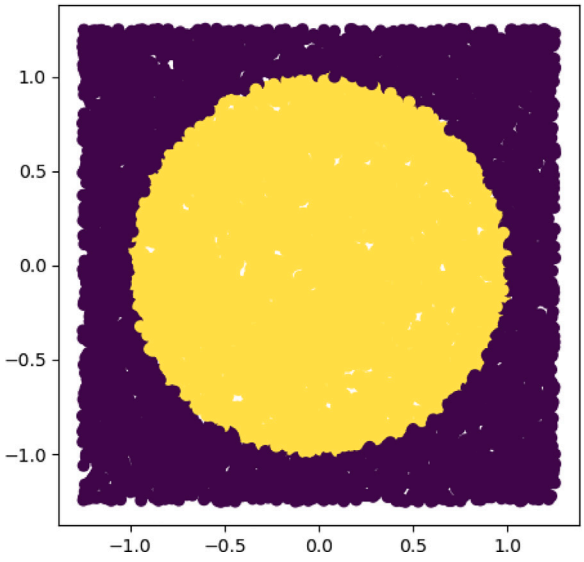

Fig. 1. The dataset used as a validation test for the RSN scheme. Points populate two different regions, of equal relevance, separated by a sharp non-linear boundary, which we identify as the unitary circle. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

generated to belong to the square domain of width L - to its reference portion, as introduced above.

For the sake of definiteness we cast $N = 10$. Every point of coordinates (x, y) (constrained so as to fall inside the square of linear size L) yields an initial condition for the RSN that we wish at training, i.e., $\vec{x}^{(1)} = (x, y, 0, 0, 0, 0, 0, 0, 0, 0)$. During the training stage, we generate a sufficiently large reservoir of (M) points, each complemented with a scalar label that specifies the class, or domain, where the corresponding point falls. The first two eigenvalues of $\mathbf{\Lambda}$ are set to unit and the corresponding eigenvectors, respectively $\vec{\phi}_1$ and $\vec{\phi}_2$, are fixed and identify randomly selected (linearly independent) directions in $\mathbb{R}^{10}$. The eigenvalues λ_i, as well and the entries of the vectors $\vec{\phi}_i$, for $i > 2$, contribute to the pool of parameters that one can freely adjust during optimization. Moreover, and to test the method in its general formulation, we do not impose a priori the non-linear function $g(\cdot)$ (the very same function for each node of the RSN). Rather, we represent $g(\cdot)$ as a two layered neural network, whose parameters are to be self-consistently adjusted during optimization. Each of these latter layers is made of 30 neurons and nodes are entitled with a tanh activation function. We label with $\bar{k}$ the number of iterations of the RSN, assumed during training. Recall that we will also be interested in assessing the behavior of the fully trained systems for $k > \bar{k}$. In the following $\bar{k} = 60$. The number of epochs is set to 200 and an early stopping technique has been employed.

In Fig. 2, the test-accuracy and the corresponding loss are plotted for $k < \bar{k}$ and for $\bar{k} < k < 100$. As it can be visually appreciated, the accuracy (and the loss) is stable for $k > \bar{k}$, i.e., when extending the RSN beyond the iteration number assumed for training.

The trained RSN classifies points $(x, y) \in \mathbb{R}^2$, provided as an input, by generating a late time output in $\mathbb{R}^{10}$ which tentatively aligns along different target directions: points in the plane contained within the unitary circle with center in the origin, should predominantly activate the spectral mode $\vec{\phi}_1$. In this case, c_1 is thus expected to stand out, as compared to all others coefficients, after sufficiently many iterations. At variance, points falling outside the unitary circle are dynamically driven towards a final equilibrium which selectively favors the eigen-direction $\vec{\phi}_2$. The coefficient c_2 should therefore prevail over the others. This scenario is confirmed by inspection of Fig. 4, where c_1 and c_2 are plotted against the iteration number for data points falling respectively

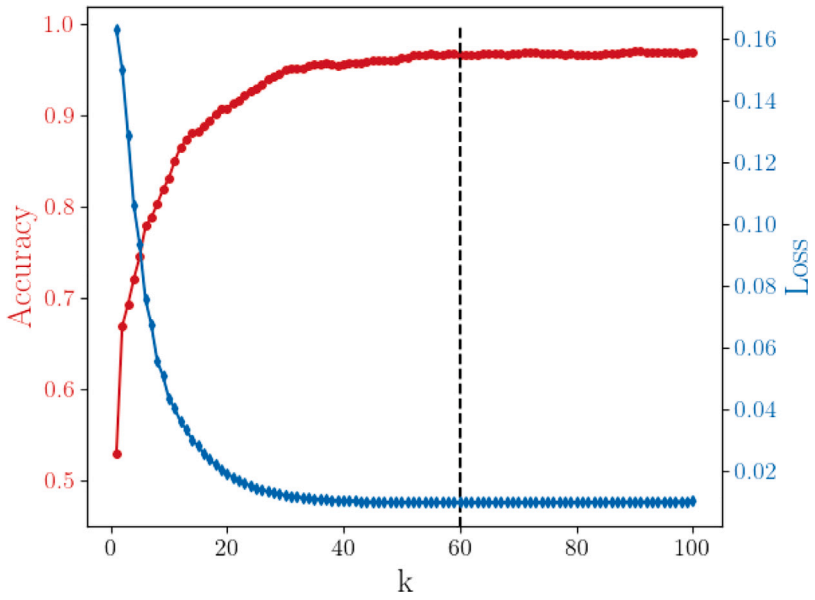

Fig. 2. Accuracy (in red) and loss (in blue) against the iteration k, for a trained RSN with $\bar{k} = 60$ (vertical dashed line). Data refer to just one realization of the training procedure. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

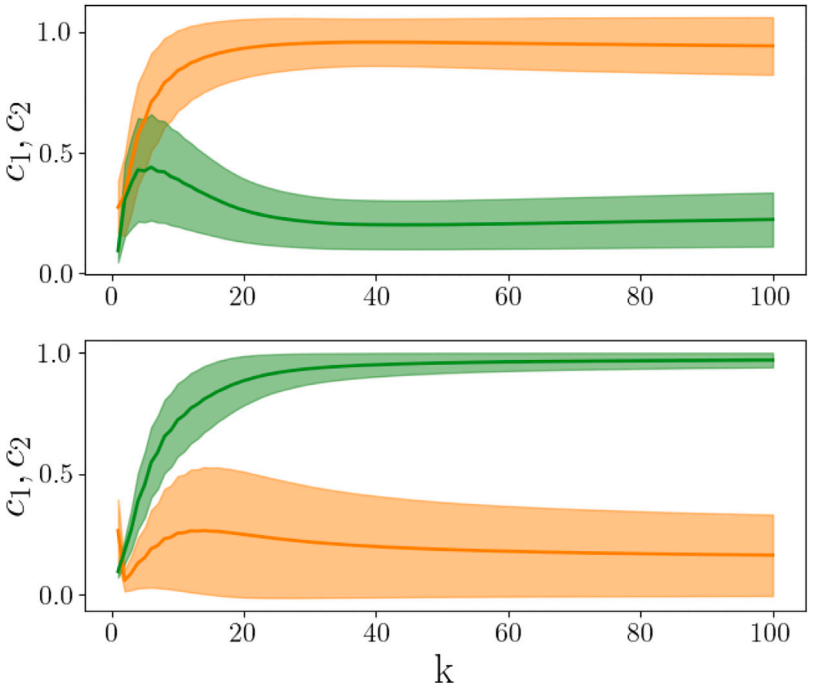

Fig. 3. The evolution of the coefficients c_1 (orange) and c_2 (green) is plotted for points of the test set positioned respectively inside (top panel) and outside (lower panel) the unitary circle. The shadowed region points to the standard deviation of the collected signal when averaging over the population of supplied input, organized in groups which reflect their domain of pertinence. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

inside (top panel) and outside (lower panel) the unitary circle. Different classes are hence flagging distinct solutions, as stipulated a priori. It is worth recalling that any direction obtained as a linear combination of $\vec{\phi}_i$ with $i = 1, 2$, is also, by construction, a stationary solution of the RSN. This is why a residual activation of the other modes – those relative to eigenvalues one but different from that identified as the target for the class under scrutiny – can in principle manifest when the RSN is challenged against the test-set. A projection along the most represented eigen-mode would enforce a perfect alignment along the sought target direction, with no impact on the performance of the trained device (see Fig. 3).

The above analysis carried out for a simple benchmark model allowed us to grasp some intuition on the decision making scheme as implemented via the dynamical RSN. Classification is here synonym of convergence towards a specific direction of the attracting manifold. This latter direction is flagged as the destination target of the dynamics, for a homogeneous ensemble of input items. Different classes are hence associated by the RSN to the eigen-directions of $\mathbf{A}$ associated to eigenvalues equal to one. For the case at hand, the separatrix between the domains in $\mathbb{R}^2$ which defines the two classes to be eventually identified matches the unitary circle. To show that the RSN is able to correctly spot out the non-linear separation between the two contiguous domain in $\mathbb{R}^2$, and so resolve the distinctive features of the dataset

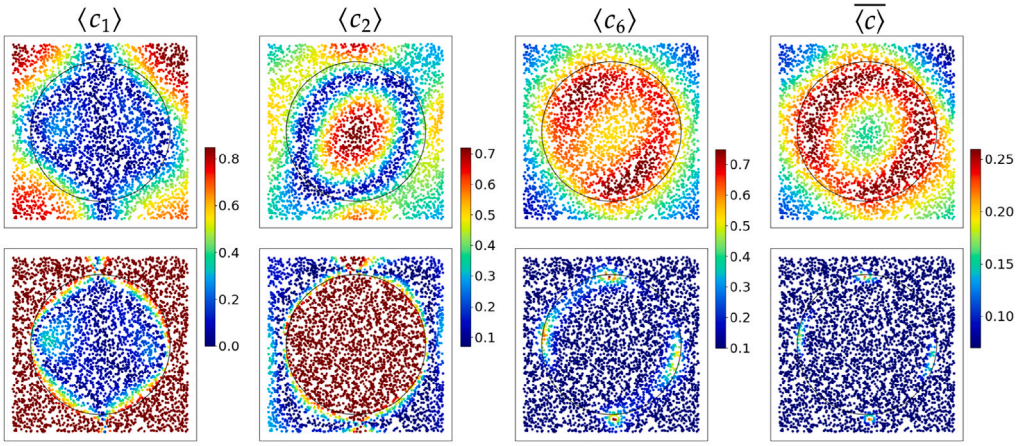

Fig. 4. The quantities $\langle c_i \rangle$ for $i = 1, 2, 6$ and $\overline{\langle c \rangle}$ are plotted, for different (x, y), i.e moving on the plane of the initial condition. Top panels refer to $k_F = 5, k_I = 1$. Lower panel to $k_F = 50, k_I = 40$. The separatrix between the two considered classes (which coincides with the unitary circle centered at the origin) is sensed, at short times, by the transient directions. The projections of the generated output along these latter directions fade asymptotically away and the existence of the two classes, as well as the relative domain of definition, leave an imperishable trace in $\langle c_1 \rangle$ and $\langle c_2 \rangle$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

under exam, we consider $\langle c_i \rangle$, the average of the ith coefficient, across successive phases of the RSN evolution and for different input choices $(x, y) \in \mathbb{R}^2$. More specifically, $\langle c_i \rangle(x, y) = \frac{1}{k_F - k_I} \sum_{k=k_I}^{k_F} (c_k)_i(x, y)$, for all specific coefficients i - including those which will fade away after a transient -and as function of the departure point. In Fig. 4 the computed coefficients are displayed in the reference plane (x, y), with an apposite colorcode and for different choices of (k_I, k_F). The panels on the top refers to the initial stages of the evolution ($k_F = 5, k_I = 1$): the separation between the two classes here considered leaves a clear imprint in the distribution of the $\langle c_i \rangle$ (in particular those with $i > 2$) across (x, y) (in Fig. 4 we plot $\langle c_6 \rangle$, as an illustrative example as well as $\overline{\langle c \rangle} = \left(\sum_{i=3}^{10} \langle c_i \rangle \right)/8$). An abrupt transition is indeed observed for $\langle c_i \rangle$, with $i > 2$, when crossing the unitary circle, namely the separatrix between the two adjacent classes that defines our test model. For small k_F (see top panels), the aforementioned coefficients are in fact remarkably different inside and outside the separatrix. On the other hand, for large k_F, they are spatially uniformly vanishing (see lower panels, referred to $k_F = 50$, $k_I = 40$). The patterns associated to $\langle c_1 \rangle$ and $\langle c_2 \rangle$ are less clear, at short times, but become evidently distinct when the iterations number is made to increase (see lower panels). Transient modes (those associated to eigenvalues with magnitude smaller than unit) are employed for an early assessment of the examined dataset and get progressively disengaged, at later times. The processed information is in fact passed over the stationary directions, where it is eventually crystallized for classification purposes. Averaged projection coefficients can be employed to trace out, in direct space, key distinctive features that form the basis of decision making. It is here speculated that this is a general attribute of the RSN that can be exported to other, more complex, settings for an a posteriori understanding of the principles that guide artificial reasoning. As a side complement, in Fig. 5 we depict the non-linear function $g(\cdot)$ self-consistently obtained via the regression neural model accommodated for in the RSN. In this specific case, it looks like an inverted ReLu (a rectified linear unit) with an additional offset.

Building on these preliminary observations, we will turn in the next Section to considering the application of RSN to MNIST dataset.

4. Applying RSN to the MNIST dataset

As a further step in the analysis, we apply the RSN to the celebrated MNIST dataset [20]. This is a collection of handwritten digits: the training set consists of 60,000 examples, and a test set of 10,000 examples. Each image is made of $N = 28 \times 28 = 784$ pixels and each pixel bears an 8-bit numerical intensity value. The images are to be classified in 10 distinct groups (the numbers from 0 to 9). Each element of the training set is associated with an integer label to point to the class to which the selected image belongs to. In the following we will set to train a RSN made by $N = 784$ nodes: the nodes that receive the information as an input are the very same nodes that carry out the classification, through a dynamical segmentation that originates from the underlying RSN. The network of excitatory (positive weight) or inhibitory (negative weight) interactions is shaped by the optimization scheme which seeks at adjusting the non trivial eigenvalues and eigenvectors of matrix $\mathbf{\Phi}$. The first 10 eigenvalues are set to unit, as in the spirit of the above, and refer to the eigen-directions employed for discrimination. These latter eigenvectors are a priori fixed and can be engineered so as to return evocative patterns in the space of the inspected images, as we shall demonstrate in the following. Further, we assume $g(\cdot) = \tanh(\cdot)$, for the sake of simplicity. Summing up, we can count on a total of $N \times (N - 10) + (N - 10)$ adjustable parameters to yield a fully trained RSN which can efficiently classify MNIST images.

In Fig. 6, we challenge the ability of the trained RSN to discern images of the test set that respectively corresponds to four (top panel) and five (lower panel). In the former case, as expected, c_4 (depicted in orange) sticks out as the only residual coefficients after sufficiently many iterations of the RSN machinery. All other coefficients (including c_5, plotted in green) are eventually bound to almost disappear, thus implying that all items belonging to the very same reservoir of images align along a specific direction that can be here traced back to one individual eigen-mode. Remarkably, all coefficients – except for the one that stands for the selected direction – become rapidly negligible. The system is hence directed towards the chosen asymptotic state, without forcing the projection. The shadowed regions that are associated to each average curve refer to the degree of variability inherent to the examined gallery of images. The lower plot in Fig. 6 shows the response of the RSN when the images displaying a number five are read as an input, and the interpretation is in line with the above. In both cases, the training is performed by arresting the RSN at iteration $\bar{k} = 10$: the outcome is however stably maintained well beyond the training horizon, with a modest, although significative in terms of its philosophical implications, improvements in terms of confidence of the assessment. When it comes to the overall performance, the accuracy on the train set is of about 98%, while on the test set the RSN scores 97%, in line with what usually reported when using conventional approaches to machine learning.

Fig. 7 illustrates the progressive convergence of the scheme, for two distinct exemplaries of input images. The RSN converges asymptotically

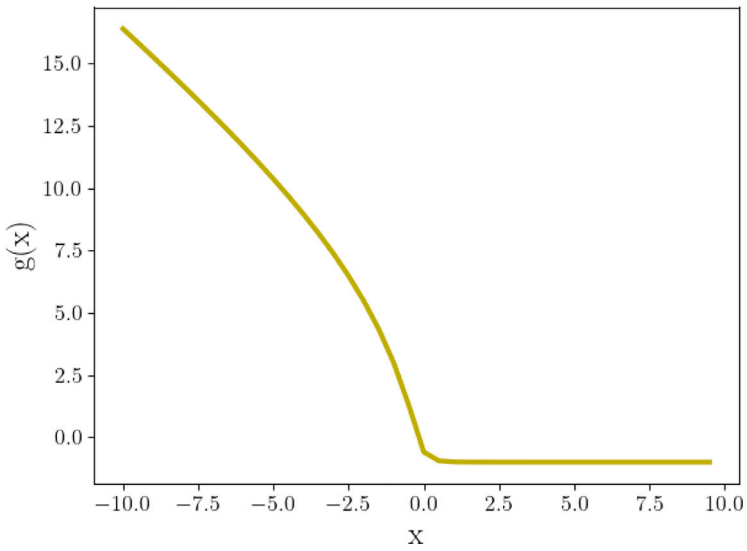

Fig. 5. The non-linear function $g(x)$ as obtained from the regression neural model that is associated to each computing neuron of the RSN.

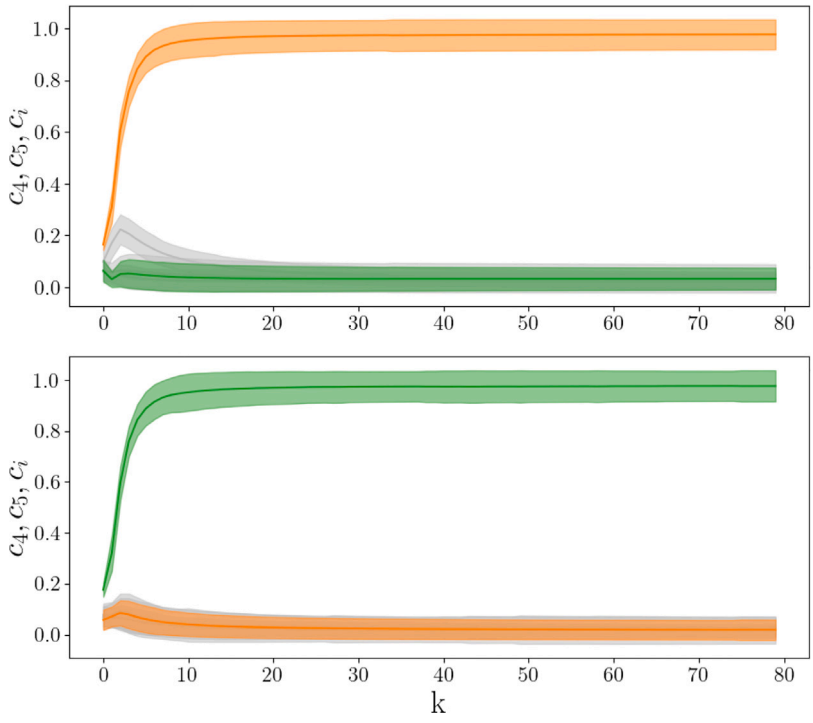

Fig. 6. Top panel: the full set of handwritten four available in the test set is provided as an input to the trained RSN and the response monitored in terms of the obtained c_i, with $i = 1, \dots, 10$. As expected, c_4 (orange) emerges and converges to unit, for $k > 10$ ($\bar{k} = 10$ being the maximum iteration number set during training). All other coefficients, including c_5 (green) disappear. Lower panel: the situation is analogous to that analyzed in the top panel with the notable exception that now handwritten five are analyzed by the RSN. Hence, c_5 (green) converges to unit while, c_i with $i \neq 5$ (including c_4, in orange) fade away. In both cases, the shadowed regions reflect the variability of the images, within any given class of the test set. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

to the deputed solutions, which respectively correspond to eigenvectors $\vec{\phi}_4$ (left) and $\vec{\phi}_5$ (right). The entries of these latter eigenvectors are shaped so as to return a stylized version of the digits that define the categories in which the dataset is partitioned. The outcome of the analysis is hence a stationary stable image, the plastic modulation of the input that is dynamically steered towards a final destination shaped at will by the operator. It is worth stressing that the performances of the method are not affected by the specificity of the target eigenvectors. Stated differently there is no need for them to align with the category that we aim at identifying. Any random eigenvectors would equivalently serve the scope.

As mentioned earlier, a specific advantage of the RSN model is the ability to keep memory of the final state for $k > \bar{k}$. This is a byproduct of the fact that, for sufficiently large times, the non-linear activation terms are virtually silenced and the update rule converges to a simple linear scheme. The dynamics aligns by construction towards stationary directions of the linear mapping, and this makes it possible to operate the RSN for any k larger than the training horizon $\bar{k}$. As a benchmark model, we consider a standard Recurrent Neural Network (RNN) trained in direct space [22–24]. The RNN in its simplest version is conceived as a single transfer layer between two adjacent stacks made of $N = 784$ nodes, iterated k times (recognition is performed on the first

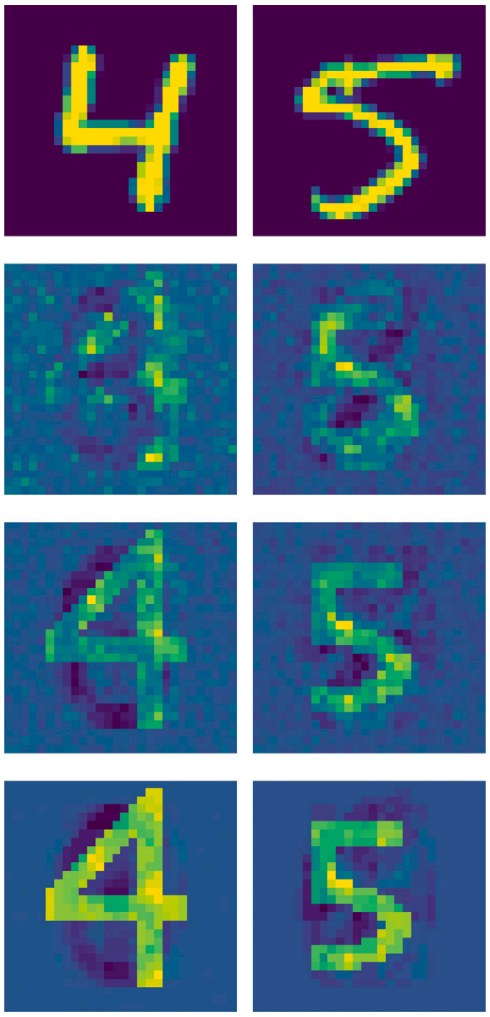

Fig. 7. In each row we plot the activity on each node of the RSN, at different iterations and for two input numbers that belong to two distinct categories, respectively a four (left) and a five (right), see top panels. After a few iterations the RSN converges asymptotically to the eigenvectors $\vec{\phi}_4$ (left) and $\vec{\phi}_5$ (right) that are triggered by the provided input. Note that the asymptotic solutions can be shaped to manifest as a stylized version of the number to be classified. The more yellow the pixels, the more intense the activity on the associated nodes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

10 nodes of the final layer). The number of trainable parameters is thus $N \times N$, comparable to the number of parameters adjusted by the RSN model. In Fig. 8, we compare the accuracy measured for the MNIST dataset, for both the RSN and the RNN trained upon completion of iteration $\bar{k}$. The accuracy recorded for the RSN (red symbols) converges rapidly and the achieved score is stably maintained for $k > \bar{k}$ (here $\bar{k} = 5$). Conversely, the RNN (blue symbols) returns its largest accuracy (basically identically to that obtained with the RSN) only for $k = \bar{k}$. By taking just one step further (i.e. adding one additional layer to the RNN) is enough to lose predictive power.

As also shown for the case of the simple model discussed in the preceding session, there is a progressive tendency to crystallize the final output along the eigen-directions, where recognition is eventually performed. This observation can be made quantitative — see Fig. 9 - by monitoring the evolution of the coefficients c_i, as computed from the state vector across successive iterations. In particular, three sets of c_i are identified: each group clusters together the coefficients associated to eigen-directions relative to eigenvalues that approximately share the same magnitude (a set relative to small eigenvalues, a set relative to larger eigenvalues and the final set of eigenvalues equal to one, i.e., those associated to the eigen-directions where recognition takes place). We evaluate the three sets of coefficients for each image in the test set displaying a four and a five and compute the average distance (square norm) between each set of coefficients, against k, the iteration of the RSN. The coefficients stemming from the transient modes single out the differences between the analyzed samples, before converging to zero when the stationary eigen-modes, inactive at first, get eventually approached

In the next Section we will turn to considering a variant of the RSN which is constructed to yield sequential handling of different datasets, with a long term memory effect. To demonstrate our findings, and as a preliminary proof of concept, we will split MNIST into two distinct, though perfectly balanced, datasets, the first formed by digits from zero to four, and the other populated with the remaining elements, ranging from five to nine.

5. Sequential learning: spectral quasi-orthogonality and the memory effects

In this Section we will discuss a generalization of the RNS which allows to keep track, to some extent, of a learned task, while dealing with an independent session of training, on a distinct dataset. To elaborate along these lines, and with the sole aim of providing a preliminary proof of concept of the basic implementation, we shall split the MNIST into two distinct, though balanced datasets. The first will be composed by handwritten digits ranging from zero to four. The remaining images, displaying numbers from five to nine, constitute the second reservoir. We will then train the RSN to classify the images belonging to the first dataset. Then, the obtained RSN undergoes a second round of training focusing on the images that define the complementary dataset. By assuming sets of quasi-orthogonal eigenvectors with associated memory kernels, yields a fully coupled network, the backbone of the RSN, which is capable to efficiently handle novel tasks while preserving notion of past knowledge. This is at variance of conventional schemes, based on standard deep learning architectures or RNN, which tend to eradicate former imprints by overwriting existing memory slots, as we shall hereafter demonstrate [14–16].

MNIST images are read as an input by a layer made of $N_0 = 28 \times 28$. This information is passed to the N nodes of the RSN via an all-to-all linear transformation encoded by a $N_0 \times N$ matrix $\mathbf{A}_0$, see Fig. 10. Here, $\mathbf{A}_0$ is fixed. As such, the entries of $\mathbf{A}_0$ do not take active part to the optimization process which is instead focused on the RSN component of the dynamics. Further, N (assumed even, with no loss of generality) can be larger of smaller than N_0 without any limitation whatsoever. We then postulate the following form for matrix $\mathbf{\Phi}$:

$$\mathbf{\Phi} = \left(\begin{array}{c|c} \mathbf{\Phi}_{11} & \epsilon\mathbf{\Phi}_{12} \\ \hline \epsilon\mathbf{\Phi}_{21} & \mathbf{\Phi}_{22} \end{array}\right) \tag{3}$$

The four blocks $\mathbf{\Phi}_{ij}$, with $i, j = 1, 2$ have dimensions $N/2 \times N/2$, and comparable norms. The parameter ϵ sets the importance of the off-diagonal blocks as compared to those that define the block diagonal terms. In the limiting case $\epsilon = 0$ the matrix of the eigenvectors is block diagonal. The eigenvectors are hence organized into two distinct ensemble, mutually orthogonal and the corresponding network splits into two disconnected parts. When $\epsilon \neq 0$ instead the two subparts of the ensuing network get mutually entangled and virtually indistinguishable for a sufficiently large magnitude of the coupling parameter ϵ. For $\epsilon \neq 0$, though relatively small as we shall assume in the following, the eigenvectors form two quasi-orthogonal blocks. Focus now on the diagonal matrix of the eigenvalues. These are also split into two groups of identical cardinality, which will be eventually structured as follows $(1, 1, 1, 1, 1, \lambda_6, \ldots, \lambda_{N/2})$ and $(1, 1, 1, 1, 1, \lambda_{N/2+6}, \ldots, \lambda_N)$. Trivial eigenvalues are associated to specific eigen-directions, the target of the RSN, which stay put across optimization. In practice, each eigenvalue equal to unit points to a specific memory slot which can be filled and, at least partially, preserved, across multiple learning stages. Starting from this setting we proceed as follows:

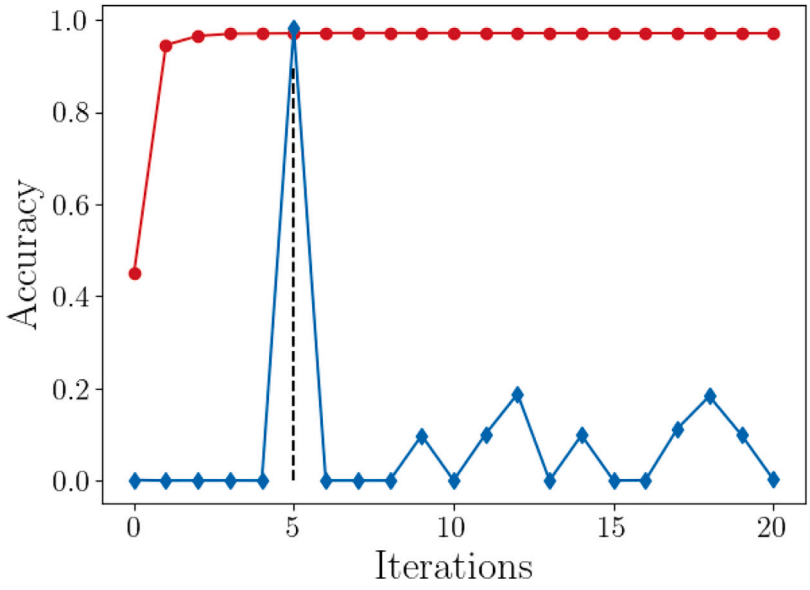

Fig. 8. Evolution of the accuracy as computed on the test set of the MNIST dataset. Red symbols stand for a RSN model, trained at $\bar{k} = 5$ (black dashed line); blue symbols refer to a RNN, with $\bar{k}+1$ consecutive layers, i.e. with $\bar{k}$ nested applications of the same $N \times N$ transfer operator. The RSN quickly converges to the best accuracy, which stays constant for $k > \bar{k}$. At variance, the RNN is capable to correctly discriminating the items provided as input entries only punctually, at $\bar{k} = 5$. It loses any predictive power for $k > \bar{k}$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

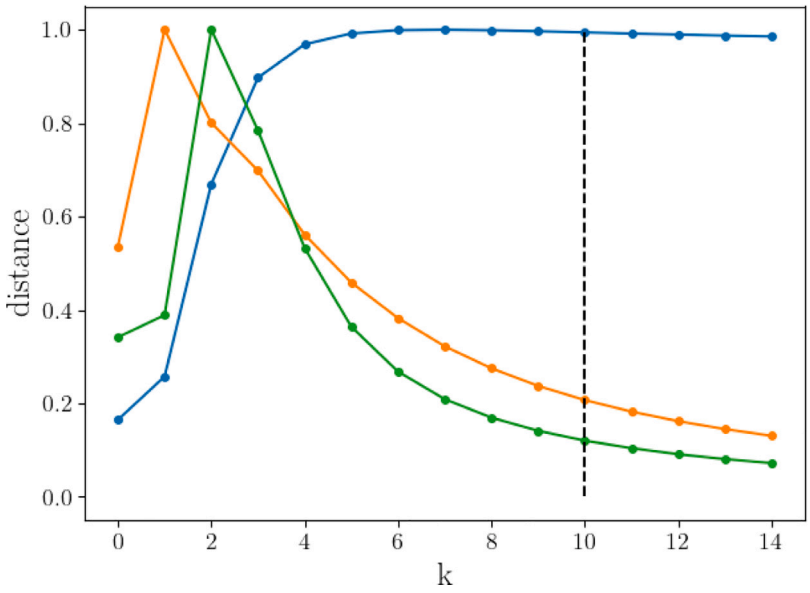

Fig. 9. Euclidean distance (normalized to its maximum value) between the three sets of coefficients as described in the main body of the paper and respectively referred to fours and fives, against the iteration index k. The orange curve refers to (10) coefficients, associated to modes with small magnitude, as obtained after the training. The green curve is computed by considering the projections along (10) modes with eigenvalues bearing larger absolute values, though smaller than one. The curve depicted in blue refers to the values of the coefficients of the 10 eigen-directions relative to eigenvalues one, where classification is eventually performed. The peak travels horizontally suggesting that the information crawls from the transient towards the stationary modes. The fact that the orange curve seems more persistent that the green at larger k is just a consequence of the imposed normalization. The vertical dashed line is set at $\bar{k}$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

- We set at first to zero the first five eigenvalues belonging to the second group, as identified above. In doing so, we seek at protecting a specific set of memory slots, which should not be contaminated during the first round of training
- We then train the RSN to recognize and correctly classify the first reservoir made of handwritten digits from zero to four, as outlined above. During this operation, the optimization acts on $\lambda_6 ... \lambda_{N/2}$ and on (the full set or a limited sub-portion of) the entries of the eigenvectors associated to these latter eigenvalues. Here, $\epsilon \neq 0$, which in turn implies that by modulating the entries of the eigenvectors belonging to the first of the two sets, yields an indirect signature on all the inter-nodes weights in direct space. At the end of the optimization, the RSN is capable to correctly classifying analogous images belonging to the test set.
- We then turn to the second round of training by providing to the above RSN (namely, the RSN that has been trained to cope with the first dataset) the elements belonging to the second reservoir of images, those depicting digits ranging from five to nine. The second set of memory slots is turned on, by setting to unit the eigenvalues initialized to be zero: the corresponding eigenvectors define the asymptotic solutions that the trained system should eventually approach. The eigenvalues that identify the target

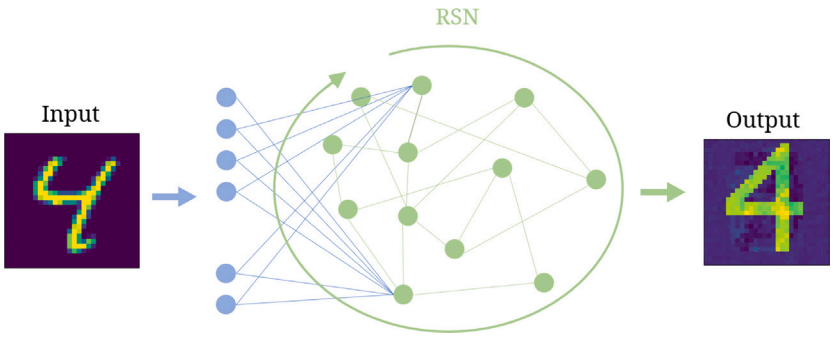

Fig. 10. A schematic layout of the architecture employed to handle sequential learning. The information stemming from the image presented as an input are passed to the RSN, and therein iteratively elaborated until convergence to the deputed stationary solution (here exemplified as a stylized version of the input number).

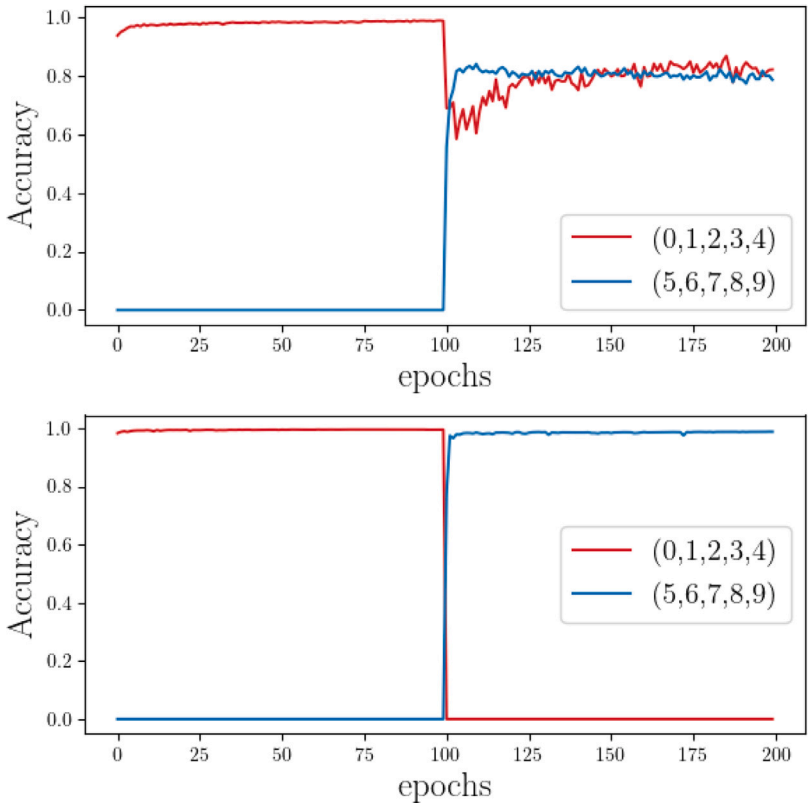

Fig. 11. Top panel: accuracy against the epochs number for the RSN. The first 100 epochs refer to the RSN confronted with the task of classifying the images of the dataset made of digits from zero to four. Then, the second range of epochs, refers to the RSN while learning to classify numbers from five to nine, after having completed the first stage of training. The accuracy drops but the RSN keeps still memory of the first task, while learning to cope with the second with an almost identical score of reported success. In this specific example, the elements of the off diagonal blocks Φ_{12} and Φ_{21} are kept fixed, during optimization. Lower panel: sequential learning is ineffective with usual RNN (and standard feedforward deep neural networks, data not shown), since any form of pre-installed knowledge gets washed out during a subsequent, independent, training stage. Here $\bar{k} = 10$, $\epsilon = 0.25$ and $N = 1000$.

eigen-directions from the preceding training are instead set to zero.

After completion of the optimization, one can check the performance of the RSN, which has been trained across two successive stages, referred to two distinct datasets. To this end we turn on all possible memory slots (trained as follows the above, two steps, procedure): in practice we set to one the eigenvalues relative to the (10) eigen-directions where information is asymptotically conveyed. In Fig. 11 (top panel), the performance of the RSN, as measured by the reported accuracy, is tested against the epochs of the optimization scheme. The optimization is carried out by assuming $\bar{k} = 10$ in the RSN, and assuming 100 of epochs for each of the two nested stages of learning. Already after a few epochs the RSN returns a very high accuracy against images of the test set which display digits ranging from zero to four. When the RSN gets also trained on the complementary reservoir of handwritten digits, as follows the sequential scheme highlighted above, it quickly manages to handle the novel task with an adequate success rate, while, at the same time, manifesting a relatively modest drop in performance as referred to the former. Notice that the images, differently from other methods, are supplied as an input with no extra markings, or alert flags, to point to the relevant group of destination patterns. To grasp the interest of the proposed scheme we report in Fig. 11 the results obtained for a RNN with a number of layers equal to $\bar{k}+1$. As immediately confirmed by visual inspection, any knowledge coming from the first round of training is – almost instantaneously – lost, when the network becomes

acquainted with the second task. Similar conclusions (data not shown) are obtained when dealing with a deep neural network, with a standard feedforward architecture [14–16]. Summing up, working with a quasi-orthogonal basis, with a set of (almost) mutually exclusive blocks equal to the number of tasks to be eventually handled, yields a RSN which can be sequentially trained, while keeping memory of the previous training sessions. A drop in the recorded accuracy is however found which could be possibly mitigated for increasing RSN size and/or addressing ad hoc solutions that require further investigations, beyond the scope of a mere proof of concept. It is also remarkable that the accuracy displayed against the first dataset, and after the initial sudden jump that follows the second training round, ramps again, epoch after epoch, to align to that refereed to the second dataset.

6. Conclusions

In this paper we have introduced and tested a novel approach to automated learning, which is rooted in reciprocal space and exploits foundational elements of the theory of discrete dynamical systems. The information under scrutiny is read by a collection of nodes, typically (but not necessarily) the pixels of the image provided as an entry, and further processed by the very same nodes, as follows an iterative update scheme which alternates linear mapping and non-linear filters. Depending on the characteristics of the signal provided as an input, the ensuing dynamics is steered (as a byproduct of the training) towards different asymptotic solutions for the subsequent recognition to take eventually place. The convergence to the asymptotic state is stable by construction, and the alignment along the selected direction that we demonstrate empirically for a classical benchmark model is guaranteed also when pushing the iterations beyond the limited horizon of the optimization. We have referred to the proposed methodology as to Recurrent Spectral Network RSN, to signify the dynamical nature of the process which is formulated in reciprocal domain.

Neural networks are sometimes called black boxes because it is not immediate to understand how or why they work as well as they do. At variance, the operational mode of a RSN is absolutely transparent and, as such, it could help unveiling the blanked of mystery that surrounds machine learning applications. Indeed, the RSN asymptotically aligns along different directions within the attracting manifold of an underlying – linear – discrete dynamical system. Learning to classify within the RSN amounts to partitioning the high dimensional input space into separated domains, each pointing to a specific stationary eigen-mode of the underlying dynamical system, in its linear approximation. Non-linearities act over a transient and fades eventually away, when the non trivial classification problem has been de facto turned into a linear one.

A variant of the RSN has been also considered which accounts for quasi-orthogonal eigen-directions to carry out a sequential handling of different datasets. In practice, a RSN can be assembled which keeps memory of an initial task, while being subject to another session of training on an independent dataset.

Several directions for further investigations can be outlined. One interesting possibility is to modify the loss function by forcing the contribution at iteration k to be smaller than that at iteration $k+1$. Preliminary checks shows that the RSN tunes self-consistently its convergence rate, which is hence not a priori imposed as it is here done. It is also tempting to speculate that proceeding along these lines, one could eventually generate a RSN which is capable of improving its accuracy score by iterating further beyond the specific window of training. Another possibility is to introduce apposite frustration mechanisms, which tend to disfavor the accidental convergence towards directions that have been already exploited, when operating with the sequential learning protocol. Also, it would be extremely important to devise other possible strategies, alternative to the one here employed, to structure the eigenvectors matrix for multiple datasets handling.

CRediT authorship contribution statement

Lorenzo Chicchi: Conceived the study and carried out the analytical work, Carried out the numerical work. **Duccio Fanelli:** Conceived the study and carried out the analytical work. **Lorenzo Giambagli:** Conceived the study and carried out the analytical work, Carried out the numerical work. **Lorenzo Buffoni:** Carried out the numerical work.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Duccio Fanelli reports a relationship with University of Florence that includes: employment.

Data availability

Data will be made available on request.

Acknowledgment

D.F. acknowledges financial support from the project PE-MNESYS funded by the Italian Ministry of Research (MUR) under the program PNRR.

All authors contributed to interpret the results and to write the manuscript' to Acknowledgment section.

References

[1] He Y, Lin J, Liu Z, Wang H, Li L-J, Han S. Amc: Automl for model compression and acceleration on mobile devices. In: Proceedings of the European conference on computer vision. ECCV, 2018, p. 784–800.
[2] Sutton RS, Barto AG. Reinforcement learning: an introduction. MIT Press; 2018.
[3] Grigorescu S, Trasnea B, Cocias T, Macesanu G. A survey of deep learning techniques for autonomous driving. J Field Robotics 2020;37(3):362–86.
[4] Biancalani T, Scalia G, Buffoni L, Avasthi R, Lu Z, Sanger A, Tokcan N, Vanderburg CR, Segerstolpe Å, Zhang M, et al. Deep learning and alignment of spatially resolved single-cell transcriptomes with Tangram. Nature Methods 2021;18(11):1352–62.
[5] Goodfellow I, Bengio Y, Courville A. Deep learning. MIT Press; 2016, http://www.deeplearningbook.org.
[6] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature 2015;521(7553):436–44.
[7] Deng L, Yu D. Deep learning: methods and applications. Found Trends Signal Process 2014;7(3–4):197–387.
[8] Giambagli L, Buffoni L, Carletti T, Nocentini W, Fanelli D. Machine learning in spectral domain. Nature Commun 2021;12(1):1–9.
[9] Chicchi L, Giambagli L, Buffoni L, Carletti T, Ciavarella M, Fanelli D. Training of sparse and dense deep neural networks: Fewer parameters, same performance. Phys Rev E 2021;104(5):054312.
[10] In principle, the system could eventually align along any direction in the manifold spanned by the eigenvectors (of the linear operator) relative to unit eigenvalues. Indeed the learning process, as encoded in the chosen loss function, forces the system to align (as much as possible) along a specific direction - a given eigenvectors selected from those that are associated to eigenvalues identically equal to one. The effectiveness of the procedure is confirmed by a posteriori inspection, as we shall discuss in the following. The proposed method proves indeed remarkably successfully beyond the toy model setting investigated for pedagogical reasons and against classical benchmark datasets. The approximate alignment along the target direction can be made exact by a non linear projection filter that singles out the most prominent among residual directions, in reciprocal space at the time of decision.
[11] Gauthier DJ, Bollt E, Griffith A, Barbosa WA. Next generation reservoir computing. Nature Commun 2021;12(1):1–8.
[12] Tanaka G, Yamane T, Héroux JB, Nakane R, Kanazawa N, Takeda S, Numata H, Nakano D, Hirose A. Recent advances in physical reservoir computing: A review. Neural Netw 2019;115:100–23.
[13] Maass W, Natschläger T, Markram H. Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural Comput 2002;14(11):2531–60.
[14] McCloskey M, Cohen NJ. Catastrophic interference in connectionist networks: The sequential learning problem. In: Psychology of learning and motivation, Vol. 24. Elsevier; 1989, p. 109–65.
[15] Lewandowsky S, Li S-C. Catastrophic interference in neural networks: Causes, solutions, and data. In: Interference and inhibition in cognition. Elsevier; 1995, p. 329–61.

[16] Kemker R, McClure M, Abitino A, Hayes T, Kanan C. Measuring catastrophic forgetting in neural networks. In: Proceedings of the AAAI conference on artificial intelligence, Vol. 32. 2018.
[17] Kirkpatrick J, Pascanu R, Rabinowitz N, Veness, et al. Overcoming catastrophic forgetting in neural networks. Proc Natl Acad Sci 2017;114(13):3521–6.
[18] Goodfellow IJ, Mirza M, Xiao D, Courville A, Bengio Y. An empirical investigation of catastrophic forgetting in gradient-based neural networks. 2013, arXiv preprint arXiv:1312.6211.
[19] Li X, Zhou Y, Wu T, Socher R, Xiong C. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In: International conference on machine learning. PMLR; 2019, p. 3925–34.
[20] LeCun Y. The MNIST database of handwritten digits. 1998, http://yann.lecun.com/exdb/mnist/.
[21] Chollet F, et al. Keras. 2015, https://keras.io.
[22] Sherstinsky A. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. Physica D 2020;404:132306.
[23] Goldberg Y. Neural network methods for natural language processing. Synth Lect Hum Lang Technol 2017;10(1):1–309.
[24] Medsker LR, Jain L. Recurrent neural networks. Des Appl 2001;5:64–7.