Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

RESEARCH OUTPUTS / RÉSULTATS DE RECHERCHE

Machine learning in spectral domain

Giambagli, Lorenzo; Buffoni, Lorenzo; Carletti, Timoteo; Nocentini, Walter; Fanelli, Duccio

Published in:
Nature Communications

DOI:
10.1038/s41467-021-21481-0

Publication date:
2021

Document Version
Publisher's PDF, also known as Version of record

Link to publication

Citation for pulished version (HARVARD):
Giambagli, L, Buffoni, L, Carletti, T, Nocentini, W & Fanelli, D 2021, 'Machine learning in spectral domain', *Nature Communications*, vol. 12, no. 1, 1330. https://doi.org/10.1038/s41467-021-21481-0

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 04. Jul. 2025

nature COMMUNICATIONS

ARTICLE

https://doi.org/10.1038/s41467-021-21481-0 OPEN

Machine learning in spectral domain

Lorenzo Giambagli[1], Lorenzo Buffoni[1,2], Timoteo Carletti[3], Walter Nocentini[1] & Duccio Fanelli[1]✉

Deep neural networks are usually trained in the space of the nodes, by adjusting the weights of existing links via suitable optimization protocols. We here propose a radically new approach which anchors the learning process to reciprocal space. Specifically, the training acts on the spectral domain and seeks to modify the eigenvalues and eigenvectors of transfer operators in direct space. The proposed method is ductile and can be tailored to return either linear or non-linear classifiers. Adjusting the eigenvalues, when freezing the eigenvectors entries, yields performances that are superior to those attained with standard methods restricted to operate with an identical number of free parameters. To recover a feed-forward architecture in direct space, we have postulated a nested indentation of the eigenvectors. Different non-orthogonal basis could be employed to export the spectral learning to other frameworks, as e.g. reservoir computing.

[1] Università degli Studi di Firenze, Dipartimento di Fisica e Astronomia, CSDC and INFN, Sesto Fiorentino, Italy. [2] Dipartimento di Ingegneria dell'Informazione, Università di Firenze, Florence, Italy. [3] naXys, Namur Institute for Complex Systems, University of Namur, Namur, Belgium. ✉email: duccio.fanelli@unifi.it

Machine learning (ML)[1–4] refers to a broad field of study, with multifaceted applications of cross-disciplinary breadth. ML is a subset of artificial intelligence (AI) which ultimately aims at developing computer algorithms that improve automatically through experience. The core idea is that systems can learn from data, so as to identify distinctive patterns and make consequent decisions, with minimal human intervention. The range of applications of ML methodologies is extremely vast[5–8], and still growing at a steady pace due to the pressing need to cope with the efficiently handling of big data[9]. Biomimetic approaches to sub-symbolic AI[10] inspired the design of powerful algorithms. These latter sought to reproduce the unconscious process underlying fast perception, the neurological paths for rapid decision making, as e.g. employed for faces[11] or spoken words[12] recognition.

An early example of a sub-symbolic brain-inspired AI was the perceptron[13], the influential ancestor of deep neural networks (NN)[14,15]. The perceptron is indeed an algorithm for supervised learning of binary classifiers. It is a linear classifier, meaning that its forecasts are based on a linear prediction function which combines a set of weights with the feature vector. Analogous to neurons, the perceptron adds up its input: if the resulting sum is above a given threshold the perceptron fires (returns the output value one) otherwise it does not (and the output equals zero). Modern multilayer perceptrons account for multiple hidden layers with non-linear activation functions. The learning is achieved via minimising the classification error. Single or multilayered perceptrons should be trained by examples[14,16,17]. Supervised learning requires indeed a large set of positive and negative examples, the training set, labelled with their reference category.

The perceptrons' acquired ability to perform classification is eventually stored in a finite collection of numbers, the weights and thresholds that were learned during the successive epochs of the supervised training. To date, it is not clear how such a huge collection of numbers (hundred-millions of weights in state of the art ML applications) are synergistically interlaced for the deep networks to execute the assigned tasks, with an exceptional degree of robustness and accuracy[18–20].

Starting from these premises, the aims of this paper are multifold. On the one side, we develop a novel learning scheme which is anchored on reciprocal space. Instead of iteratively adjusting the weights of the edges that define the connection among nodes, we modify the spectra of a collection of suitably engineered matrices that bridge adjacent layers. To eventually recover a multilayered feedforward architecture in direct space, we postulate a nested indentation of the associated eigenvectors. These latter act as the effective gears of a processing device operated in reciprocal space. The directed indentation between stacks of adjacent eigenvectors yield a compression of the activation pattern, which is eventually delivered to the detection nodes. As a starting point, assume eigenvectors are frozen to a reference setting which fulfils the prescribed conditions. The learning is hence solely restricted to the eigenvalues, a choice which amounts to performing a global training, targeted to identifying key collective modes, the selected eigen-directions, for carrying out the assigned classification task. The idea of conducting a global training on a subset of parameters has been also proposed in other works[21,22]. This is at odds with the usual approach to machine learning where local adjustments of pairwise weights are implemented in direct space. As we shall prove, by tuning the eigenvalues, while freezing the eigenvectors, yields performances superior to those reached with usual (local) techniques bound to operate with an identical number of free parameters, within an equivalent network architecture. Eigenvalues are therefore identified as key targets of the learning process, proving more fundamental than any other (randomly selected) set of identical cardinality, allocated in direct space. Remarkably, the distribution of weights obtained when applying the spectral learning technique restricted to the eigenvalues is close to that recovered when training the neural network in direct space, with no restrictions on the parameters to be adjusted. In this respect, spectral learning bound to the eigenvalues could provide a viable strategy for pre-training of deep neural networks. Further, the set of trainable eigenvalues can be expanded at will by inserting linear processing units between the adjacent layers of a non-linear multilayered perceptron. Added linear layers act as veritable booms of a telescopic neural network, which can be extracted during the learning phase and retracted in operational mode, yielding compact networks with improved classification skills. The effect of the linear expansion is instead negligible, if applied to neural learning of standard conception. The entries of the indented eigenvectors can be also trained resulting in enhanced performance, as compared to the setting where eigenvalues are exclusively modulated by the learning algorithm. To demonstrate the principles which underly spectral training, we employ the MNIST database, a collection of handwritten digits to be classified. The examined problem is relatively simple: a modest number of tunable parameters is indeed necessary for achieving remarkable success rates. When allowing for the simultaneous training of the eigenvalues and (a limited fraction of) eigenvectors, the neural networks quickly saturates to accuracy scores which are indistinguishable from those obtained via conventional approaches to supervised learning. More challenging tasks should be probably faced to fully appreciate the role played by a progressive optimisation of the eigenmodes, the collective directions in reciprocal space where information flows (see e.g. an application to Fashion NMIST database as reported in the Supplementary Material). As remarked above, the eigenvectors are here constructed so as to yield a feedforward multilayered architecture in direct space. By relaxing this assumption, comes to altering the network topology and thus exporting the spectral learning strategy to other frameworks, as e.g. reservoir computing. In general terms, working in the spectral domain corresponds to optimising a set of non-orthogonal directions (in the high dimensional space of the nodes) and associated weights (the eigenvalues), a global outlook which could contribute to shed novel light on the theoretical foundations of supervised learning.

Results

To introduce and test the proposed method we will consider a special task, i.e. recognition of handwritten digits. To this end, we will make use of the MNIST database[23] which has a training set of 60,000 examples, and a test set of 10,000 examples. Each image is made of $N_1 = 28 \times 28$ pixels and each pixel bears an 8-bit numerical intensity value, see Fig. 1. A deep neural network can be trained using standard backpropagation[14] algorithms to assign the weights that link the nodes (or perceptrons) belonging to consecutive layers. The first layer has N_1 nodes and the input is set to the corresponding pixel's intensity. The highest error rate reported on the original website of the database[23] is 12%, which is achieved using a simple linear classifier, with no preprocessing. In early 2020, researchers announced 0.16% error[24] with a deep neural network made of branching and merging convolutional networks. Our goal here is to contribute to the analysis with a radically different approach to the learning, rather than joining the efforts to break current limit in terms of performance and classification accuracy. More specifically, and referring to the MNIST database as a benchmark application, we will assemble a network made of N nodes, organized in successive ℓ layers, tying the training to reciprocal space.

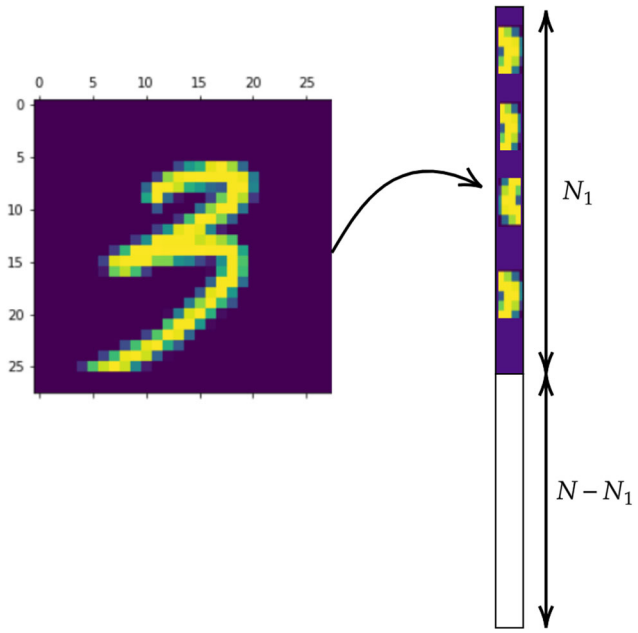

Fig. 1 From images to vectors. Each image of the training set is mapped into a column vector $\vec{n}_1$, of size N, whose first $N_1 = 28 \times 28$ entries are the intensities displayed on the pixels of the image.

Directed connections between nodes belonging to consecutive layers are encoded in a set of $\ell - 1$, $N \times N$ adjacency matrices. The eigenvectors of these latter matrices are engineered so as to favour the information transfer from the reading frame to the output layer, upon proper encoding. The associated eigenvalues represent the primary target of the novel learning scheme. In the following we will set up the method, both with reference to its linear and non-linear versions. Tests performed on the MNIST database are discussed in the next section. In the annexed Supplementary Material we report the results of the analysis carried out for Fashion NMIST database.

Linear spectral learning: single-layer perceptron trained in reciprocal space. Assume N_i to label the nodes assigned to layer i, and define $N = \sum_{i=1}^{\ell} N_i$. For the specific case here inspected the output layer is composed of ten nodes ($N_\ell = 10$), where recognition takes eventually place. Select one image from the training set and be $n_1(=0, 1, 2..., 9)$ the generic number therein displayed. We then construct a column vector $\vec{n}_1$, of size N, whose first N_1 entries are the intensities displayed on the pixels of the selected image (from the top-left to the bottom-right, moving horizontally), as illustrated in Fig. 1. All other entries are initially set to zero. As we shall explain in the following, our goal is to transform the input $\vec{n}_1$ into an output vector with the same dimension. The last N_ℓ elements of this latter vector represent the output nodes where reading is eventually performed.

To set the stage, we begin by reporting on a simplified scenario that, as we shall prove in the following, yields a single-layer perceptron. The extension to multilayered architectures will be discussed right after.

Consider the entry layer made of N_1 nodes and the outer one composed of N_2 elements. In this case $N = N_1 + N_2$. The input vector $\vec{n}_1$ undergoes a linear transformation to yield $\vec{n}_2 = \mathbf{A}_1 \vec{n}_1$ where $\mathbf{A}_1$ is a $N \times N$ matrix that we shall characterise in the following. Introduce matrix Φ_1: this is the identity matrix $\mathbb{1}_{N\times N}$ modified by the inclusion of a sub-diagonal block $N_2 \times N_1$, e.g. filled with uniformly distributed random numbers, defined in a bounded interval, see Fig. 2. The columns of Φ_1, hereafter $\left(\vec{\phi}_1\right)_k$ with $k = 1, ..., N$, define a basis of the N-dimensional space to which $\vec{n}_1$ and $\vec{n}_2$ belong. Then, we introduce the diagonal matrix Λ_1. The entries of Λ_1 are set to random (uniform) numbers spanning a suitable interval. A straightforward calculation returns $(\Phi_1)^{-1} = 2\mathbb{1}_{N\times N} - \Phi_1$. We hence define $\mathbf{A}_1 = \Phi_1 \Lambda_1 (2\mathbb{1}_{N\times N} - \Phi_1)$ as the matrix that transforms $\vec{n}_1$ into $\vec{n}_2$. Because of the specific structure of the input vector, and owing to the nature of $\mathbf{A}_1$, the information stored in the first N_1 elements of $\vec{n}_1$ is passed to the N_2 successive entries of $\vec{n}_2$, in a compact form which reflects both the imposed eigenvectors' indentation and the chosen non-trivial eigenvalues.

To see this more clearly, expand the N-dimensional input vector $\vec{n}_1$ on the basis made of $\left(\vec{\phi}_1\right)_k$ to yield $\vec{n}_1 = \sum_{k=1}^{N} c_k \left(\vec{\phi}_1\right)_k$ where c_k stands for the coefficients of the expansion. The first N_1 vectors are necessarily engaged to explain the non-zero content of $\vec{n}_1$ and, because of the imposed indentation, rebound on the successive N_2 elements of the basis. These latter need to adjust their associated weights c_k to compensate for the echoed perturbation. The action of matrix $\mathbf{A}_1$ on the input vector $\vec{n}_1$ can be exemplified as follows:

$$\vec{n}_2 = \mathbf{A}_1 \vec{n}_1 = \mathbf{A}_1 \sum_{k=1}^{N} c_k \left(\vec{\phi}_1\right)_k = \sum_{k=1}^{N_1+N_2} c_k (\Lambda_1)_k \left(\vec{\phi}_1\right)_k \quad (1)$$

where $(\Lambda_1)_k$ are the element of matrix Λ_1. In short, the entries of $\vec{n}_2$ from position $N_1 + 1$ to position $N_1 + N_2$ represent a compressed (if $N_2 < N_1$) rendering of the supplied input signal, the key to decipher the folding of the message being stored in the $N_2 \times N_1$ sub-diagonal block of Φ_1 (i.e. the eigenvector indentation) and in the first set of $N = N_1 + N_2$ eigenvalues $(\Lambda_1)_k$. The key idea is to propagate this message passing scheme, from the input to the output in a multilayer setting, and adjust (a subset of) the spectral parameters involved so as to optimise the encoding of the information.

To this end, we introduce the $N \times N$ matrix operator Φ_k, for $k = 2, ..., \ell - 1$. In analogy with the above, Φ_k is the identity matrix $\mathbb{1}_{N\times N}$ modified with a sub-diagonal block $N_{k+1} \times N_k$, which extends from rows N_k to $N_k + N_{k+1}$, and touches tangentially the diagonal, as schematically illustrated in Fig. 2a. Similarly, we introduce Λ_k, for $k = 2, ..., \ell - 1$, which is obtained from the identity matrix $\mathbb{1}_{N\times N}$ upon mutating to uniformly distributed random entries the diagonal elements that range from $\sum_{i=1}^{k} N_i$ (not included) to $\sum_{i=1}^{k+1} N_i$ (included). Finally, we define $\mathbf{A}_k = \Phi_k \Lambda_k (2\mathbb{1}_{N\times N} - \Phi_k)$, as the matrix that transforms $\vec{n}_k$ into $\vec{n}_{k+1}$, with $k = 2, ..., \ell - 1$. In principle, both non-trivial eigenvalues' and eigenvectors' input can be self-consistently adjusted by the envisaged learning strategy. The input signal $\vec{n}_1$ is hence transformed into an output vector $\vec{n}_\ell$ following a cascade of linear transformations implemented via matrices $\mathbf{A}_k$. In formulae:

$$\vec{n}_\ell = \mathbf{A}_{\ell-1} ... \mathbf{A}_1 \vec{n}_1 = \left(\Pi_{k=1}^{\ell-1} \Phi_k \Lambda_k (2\mathbb{1}_{N\times N} - \Phi_k)\right) \vec{n}_1 \quad (2)$$

where in the last step we made use of the representation of $\mathbf{A}_k$ in dual space. The generic vector $\vec{n}_{k+1}$, for $k = 1,..., \ell - 1$ is obtained by applying matrix $\mathbf{A}_k$ to $\vec{n}_k$. The first $N_1 + N_2 + ... + N_k$ components of $\vec{n}_{k+1}$ coincide with the corresponding entries of $\vec{n}_k$, namely $\left[\vec{n}_{k+1}\right]_m \equiv \left[\vec{n}_k\right]_m$ for $m < N_1 + N_2 + ... + N_k$. Here, $[(\vec{\cdot})]_m$ identifies the m-th component of the vector $(\vec{\cdot})$. Recall that, by construction, $\left[\vec{n}_k\right]_m = 0$ for $m > N_1 + N_2 + ... + N_k$. On the contrary, the components $\left[\vec{n}_{k+1}\right]_m$ with $N_1 + N_2 + ... + N_k + 1 <= m <= N_1 + N_2 + ... + N_k + N_{k+1}$ are

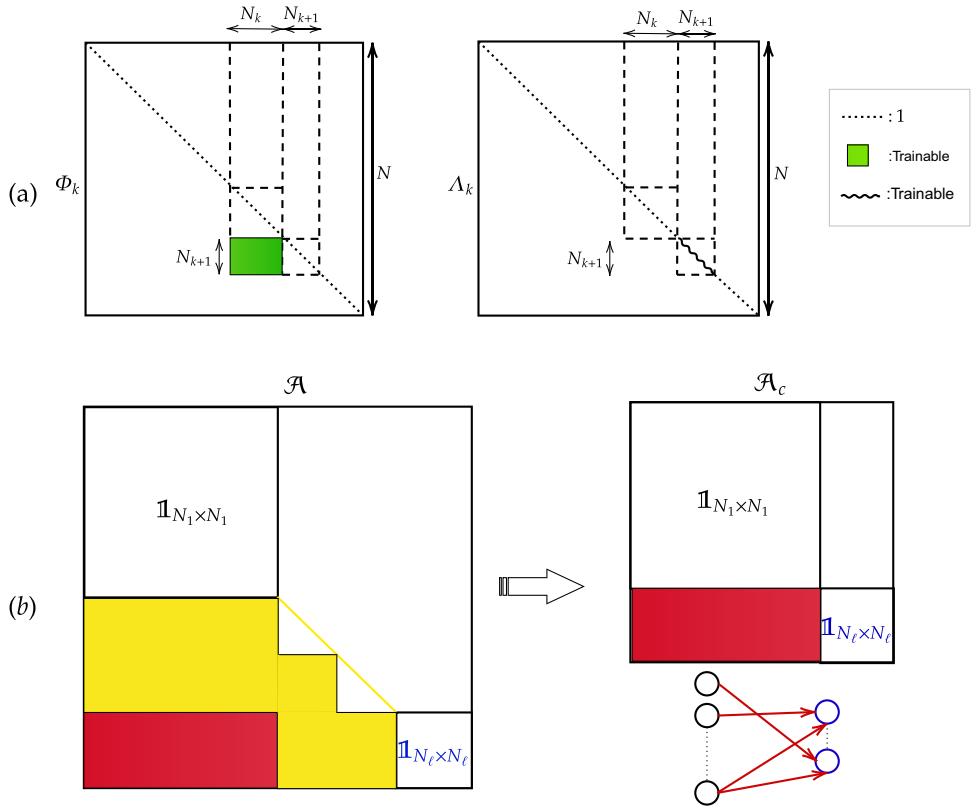

Fig. 2 The choice of the eigenvectors. a The structure of matrix Φ_k is schematically depicted. The diagonal entries of Φ_k are unities. The sub-diagonal block of size $N_{k+1} \times N_k$ for $k = 1, \ell - 1$ is filled with uniform random numbers in $[a, b]$, with $a, b \in \mathbb{R}$. These blocks yield an effective indentation between successive stacks of linearly independent eigenvectors. The diagonal matrix of the eigenvalues Λ_k is also represented. The sub-portions of Φ_k and Λ_k that get modified by the training performed in spectral domain are highlighted (see legend). In the experiments reported in this paper, the initial eigenvectors entries are uniform random variables distributed in $[-0.5, 0.5]$. The eigenvalues are uniform random numbers distributed in the interval $[-0.01, 0.01]$. Optimising the range to which the initial guesses belong (for both eigenvalues and eigenvectors) is an open problem that we have not tackled. **b** A $(N_1 + N_\ell) \times (N_1 + N_\ell)$ matrix $\mathcal{A}_c$ can be obtained from $\mathcal{A} = \left(\Pi_{k=1}^{\ell-1}\mathbf{A}_k\right)$, which provides the weights for a single-layer perceptron, that maps the input into the output, in direct space.

populated by non-trivial values which reflect the eigenvectors indentation, as well as the associated eigenvalues. This observation can be mathematically proven as follows. Write $\vec{n}_k$ on the basis formed by the eigenvectors $\left(\vec{\phi}_k\right)_l$ to eventually get:

$$\vec{n}_k = \sum_{l=1}^{N_1+N_2+\ldots+N_{k+1}} c_l\left(\vec{\phi}_k\right)_l \equiv \sum_{l=1}^{N_1+N_2+\ldots+N_k} c_l\,\vec{e}_l \qquad (3)$$

where $\left(\vec{e}_1, \vec{e}_2 \ldots\right)$ stand for the canonical basis and the last inequality follows the specific structure of the eigenvectors (remark that the leftmost sum in the above equation includes N_{k+1} more elements than the second). By definition:

$$\vec{n}_{k+1} = \mathbf{A}_k\vec{n}_k = \sum_{l=1}^{N_1+N_2..+N_{k+1}} c_l(\Lambda_k)_l\left(\vec{\phi}_k\right)_l \qquad (4)$$

From the above relation, one gets for $m \le N_1 + N_2 + \ldots + N_k$

$$\left[\vec{n}_{k+1}\right]_m = \sum_{l=1}^{N_1+N_2..+N_k} c_l\left[\vec{e}_l\right]_m \equiv \left[\vec{n}_k\right]_m \qquad (5)$$

where the first equality sign follows from the observation that $\left(\vec{\phi}_k\right)_l$ coincides with $\vec{e}_l$ and $(\Lambda_k)_l = 1$, over the explored range of m. For $N_1 + N_2 + \ldots + N_k + 1 \le m \le N_1 + N_2 + \ldots + N_k + N_{k+1}$, we obtain instead:

$$\left[\vec{n}_{k+1}\right]_m = \sum_{l=N_1+N_2..+N_{k-1}+1}^{N_1+N_2..+N_{k+1}} c_l(\Lambda_k)_l\left[\left(\vec{\phi}_k\right)_l\right]_m \qquad (6)$$

Finally, it is immediate to show that $\left[\vec{n}_{k+1}\right]_m = 0$ for $m > N_1 + N_2 + \ldots + N_k + N_{k+1}$, because of the specific form of the employed eigenvectors. In short, the information contained in the last non-trivial N_k entries of $\vec{n}_k$ rebound on the successive N_{k+1} elements of $\vec{n}_{k+1}$, funnelling the information downstream from the input to the output. The successive information processing relies on the indented (non-orthogonal) eigenvectors and the associated eigenvalues, which hence define the target of the training in reciprocal space.

To carry out the learning procedure one needs to introduce a loss function $L(\vec{n}_1)$. For illustrative purposes this latter can be written as:

$$L(\vec{n}_1) = ||l(\vec{n}_1) - \sigma\left[\left(\Pi_{k=1}^{\ell}\Phi_k\Lambda_k(2\mathbb{1}_{N\times N} - \Phi_k)\right)\vec{n}_1\right]||^2 \qquad (7)$$

where $\sigma(\cdot)$ is the softmax operation applied to the last entries of the ℓ-th image of the input vector $\vec{n}_1$. In the above expression, $l(\vec{n}_1)$ stands for the label attached to $\vec{n}_1$ depending on its category. More into details, the k-th entry of $l(\vec{n}_1)$ is equal unit (and the rest identically equal to zero) if the number supplied as an input is identical to k, with $k = 0, 1, \ldots, 9$. The loss function can be minimized by acting on the free parameters of the learning

scheme. Specifically, the learning can be restricted to the set of N non-trivial eigenvalues, split in ℓ distinct groups, each referred to one of the $\mathbf{A}_k$ matrices (i.e. N_1+N_2 eigenvalues of $\mathbf{A}_1$, N_3 eigenvalues of $\mathbf{A}_2$, ..., N_ℓ eigenvalues of $\mathbf{A}_{\ell-1}$). In addition, the sub-diagonal block entries of Φ_k, the elements of the basis which dictate the successive indentation between adjacent layers, can be adjusted as follows the training scheme. In the following section we will report about the performance of the method, implemented in its different modalities, against those obtained with a classical approach to the learning anchored in direct space. In the actual implementation we have chosen to deal with a categorical cross-entropy loss function.

Before ending this section a few remarks are mandatory. Introduce $\mathcal{A}=\Pi_{k=1}^{\ell}\mathbf{A}_k$. The linear transformation that links the input vector $\overrightarrow{n}_1$ to the generated output $\overrightarrow{n}_\ell$, can be compactly expressed as $\overrightarrow{n}_\ell=\mathcal{A}\overrightarrow{n}_1$. Then, recall that the classification relies on examining the last N_ℓ entries of $\overrightarrow{n}_\ell$. Hence, for the specific setting here examined, where the mapping is obtained as a cascade of linear transformations, one can imagine recasting the whole procedure in a space of reduced dimensionality. Be $\overrightarrow{z}$ a column vector made of N_1+N_ℓ elements. The first N_1 entries of $\overrightarrow{z}$ are the intensities on the pixels of the selected image, as for the homologous $\overrightarrow{n}_1$ quantity. The other elements are set to zero. Then, consider the $(N_1+N_\ell)\times(N_1+N_\ell)$ matrix $\mathcal{A}_c$ (the label c stands for compact), constructed from $\mathcal{A}$ by trimming out all the information that pertain to the intermediate layers, as introduced in the reciprocal space (see Fig. 2b). Stated differently, matrix $\mathcal{A}_c$ provides the weighted links that feed from the input to the output layer in direct space, via the linear transformation $\mathcal{A}_c\overrightarrow{z}$: this is a single-layer perceptron, shown in Fig. 2b, which was trained by endowing reciprocal space with an arbitrary number of additional dimensions, the intermediate stacks responsible for the sequential embedding of the information. Intermediate layers can be literally extracted, during the training phase, and subsequently retracted in operational mode. The importance of allowing for additional layers, and so provide the neural network of a telescopic attribute, will be assessed in the forthcoming sections.

From the algorithmic point of view the process outlined above can be rephrased in simpler, although equivalent terms. For all practical purposes, one could take the (column) input vector $\overrightarrow{n}_1$ to have N_1+N_2 elements. Following the scheme depicted above, the first N_1 entries are the intensities on the pixels of the selected image, while the remaining N_2 elements are set to zero. We now introduce a $(N_1+N_2)\times(N_1+N_2)$ matrix $\mathbf{A}_1$. This is the identity matrix $\mathbb{1}_{(N_1+N_2)\times(N_1+N_2)}$ with the inclusion of a sub-diagonal block $N_2\times N_1$, which handles the information processing that will populate the second N_2 elements of the output vector $\overrightarrow{n}_2=\mathbf{A}_1\overrightarrow{n}_1$. Then, we formally replace the (N_1+N_2) column vector $\overrightarrow{n}_2$ with a column vector made of (N_2+N_3) elements, termed $\overrightarrow{n}_{2t}$, whose first N_2 elements are the final entries of $\overrightarrow{n}_2$. The remaining N_3 elements of $\overrightarrow{n}_{2t}$ are set to zero. Now, rename $\overrightarrow{n}_{2t}$ as $\overrightarrow{n}_2$ and presents it as the input of a $(N_2+N_3)\times(N_2+N_3)$ matrix $\mathbf{A}_2$, with a non-trivial sub-diagonal $N_3\times N_2$ block. This latter maps the first N_2 elements of the input vector, into the successive N_3 of the output one, by completing the second step of an algorithmic scheme which can be iteratively repeated. In analogy with the above, each $(N_k+N_{k+1})\times(N_k+N_{k+1})$ matrix $\mathbf{A}_k$ can be written as $\mathbf{A}_k=\Phi_k\Lambda_k\left(2\mathbb{1}_{(N_k+N_{k+1})\times(N_k+N_{k+1})}-\Phi_k\right)$, where now the column vectors of Φ_k are the eigeventors of $\mathbf{A}_k$ and form a non-orthogonal basis of the (N_k+N_{k+1}) space where input and output vectors belong. Λ_k is a diagonal matrix of the eigenvalues: the first N_k are set to one, while the other N_{k+1} are non-trivial entries to be adjusted self-consistently via the learning scheme. Framing the process in the augmented space of N dimensions, as done earlier, allows us to avoid adapting the dimensions of the involved vectors at each iteration. On the contrary, this is a convenient procedure to be followed when aiming at a numerical implementation of the envisaged scheme. Notice that to discuss the algorithmic variant of the method, we made use of the same symbols employed earlier. The notation clash is however solely confined to this paragraph.

In the following, we will discuss how these ideas extend to the more general setting of non-linear multilayered neural networks.

Training non-linear multilayered neural networks in the spectral domain. In analogy with the above, the image to be processed is again organised in a $N\times 1$ column vector $\overrightarrow{n}_1$. This latter is transformed into $\overrightarrow{n}_2=\mathbf{A}_1\overrightarrow{n}_1$, where matrix $N\times N$ matrix $\mathbf{A}_1$ is recovered from its spectral properties, respectively encoded in Φ_1 and Λ_1. The output vector $\overrightarrow{n}_2$ is now filtered via a suitable non-linear function $f(\cdot)$. This step marks a distinction between, respectively, the linear and non-linear versions of the learning schemes. For the applications here reported we have chosen to work with a rectified linear unit (ReLU) $f(\cdot)=\max(0,\cdot)$. Another possibility is to set $f(\cdot,\beta_1)=\tanh[\beta_1(\cdot)]$, where β_1 is a control parameter which could be in principle self-consistently adjusted all along the learning procedure. We are now in a position to iterate the same reasoning carried out in the preceding section, adapted to the case at hand. More specifically, we introduce the generic $N\times N$ matrix $\mathbf{A}_k=\Phi_k\Lambda_k\left(2\mathbb{1}_{N\times N}-\Phi_k\right)$ which transforms $\overrightarrow{n}_k$ into $\overrightarrow{n}_{k+1}$, with $k=2,\ldots,\ell-1$. The outcome of this linear transformation goes through the non-linear filter. The loss function $L(\overrightarrow{n})$ generalises to:

$$L(\overrightarrow{n})=||l(\overrightarrow{n}_1)-\sigma\left(f\left(\mathbf{A}_{\ell-1}\ldots f\left(\mathbf{A}_2 f\left(\mathbf{A}_1\overrightarrow{n}_1,\beta_1\right),\beta_2\right),\beta_{\ell-1}\right)\right)||^2 \tag{8}$$

with an obvious meaning of the involved symbols. In the set of experiments reported below we assume, in analogy with the above, a categorical cross-entropy loss function. The loss function is minimised upon adjusting the free parameters of the learning scheme: the $\ell-1$ blocks of tunable eigenvalues, the elements that define the successive indentation of the nested basis which commands the transfer of the information (and e.g. the quantities β_k, if the sigmoidal hyperbolic function is chosen as a non-linear filter). This eventually yields a fully trained network, in direct space, which can be unfolded into a layered architecture to perform pattern recognition (see Fig. 3). Remarkably, self-loop links are also present. The limit of a linear single-layer perceptron is recovered when silencing the non-linearities: a $(N_1+N_\ell)\times(N_1+N_\ell)$ matrix $\mathcal{A}_c$ can be generated from the $N\times N$ matrices $\mathbf{A}_k$, following the same strategy outlined above. A sequence of linear layers can be also interposed between two consecutive non-linear stacks. The interposed layers allow to enlarge the space of parameters employed in the learning scheme, and can be retracted when operating the deep neural network after completion of the learning stage. Their role is de facto encapsulated in the entries of the linear operator that bridges the gap between the adjacent non-linear stacks, as explained above when referring to the telescopic operational modality.

Testing the performance of the method. To build and train the aforementioned models we used TensorFlow and created a custom spectral layer matrix that could be integrated into virtually every TensorFlow or Keras model. That allowed us to leverage on the automatic differentiation capabilities and the built-in optimisers of TensorFlow. Recall that we aim at training just a

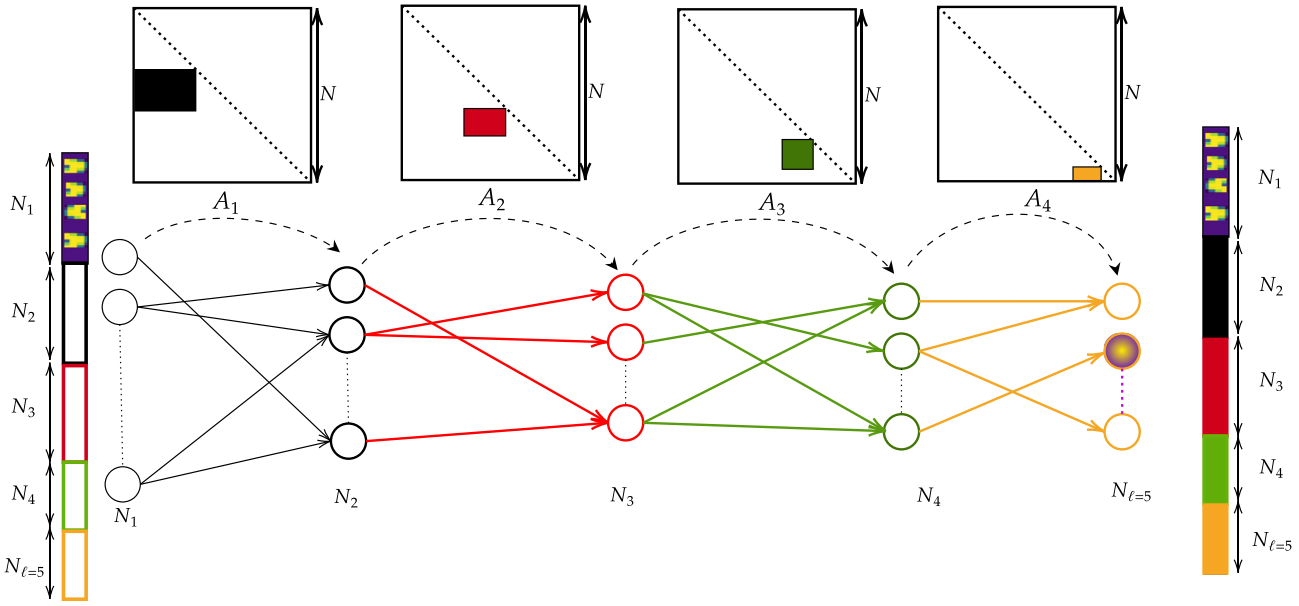

Fig. 3 The non-linear setting. The non-linear version of the training scheme returns a multilayered architecture with self-loops links in direct space. Linear and non-linear transformation can be combined at will, matrices $\mathbf{A}_k$ providing the connection between successive layers. Linear layers can be retracted in operational mode, following a straightforward variant of the compactification procedure described in the main body of the paper.

portion of the diagonal of Λ_k and a block of Φ_k. To reach this goal we generated two fully trainable matrices, for each layer in the spectral domain, and applied a suitably designed mask to filter out the sub-parts of the matrices to be excluded from the training. This is easy to implement and, although improvable from the point of view of computational efficiency, it works perfectly, given the size of the problem to be handled. We then trained all our models with the AdaMax optimizer[25] by using a learning rate of 0.03 for the linear case and 0.01 for the non-linear one. The training proceeded for about 20 epochs and during each epoch the network was fed with batches of images of different size, ranging from 300 to 800. These hyperparameters have been chosen so as to improve on GPU efficiency, accuracy and stability. However, we did not perform a systematic study to look for the optimal setting. All our models have been trained on a virtual machine hosted by Google Colaboratory. Standard neural networks have been trained on the same machine using identical software and hyperparameters, for a fair comparison. Further details about the implementation, as well as a notebook to reproduce our results, can be found in the public repository of this project (https://github.com/Buffoni/spectral_learning).

We shall start by reporting on the performance of the linear scheme. The simplest setting is that of a perceptron made of two layers: the input layer with $N_1 = 28 \times 28 = 784$ nodes and the output one made of $N_2 = 10$ elements. The perceptron can be trained in the spectral domain by e.g. tuning the $N = N_1 + N_2 = 794$ eigenvalues of $\mathbf{A}_1$, the matrix that links the input ($\vec{n}_1$) and output ($\vec{n}_2$) vectors. The learning restricted to the eigenvalues returns a perceptron which performs the sought classification task with an accuracy (the fraction of correctly recognised images in the test set) of $(82 \pm 2)\%$ (averaging over 5 independent runs). This figure is to be confronted with the accuracy of a perceptron trained with standard techniques in direct space. For a fair comparison, the number of adjustable weights should be limited to N. To this aim, we randomly select a subset of weights to be trained and carry out the optimisation on these later (constrained neural network (NN)). The process is repeated a few (5 in this case) times and, for each realisation, the associated accuracy computed. Combining the results yields an average performance of $(79 \pm 3)\%$, i.e. a slightly

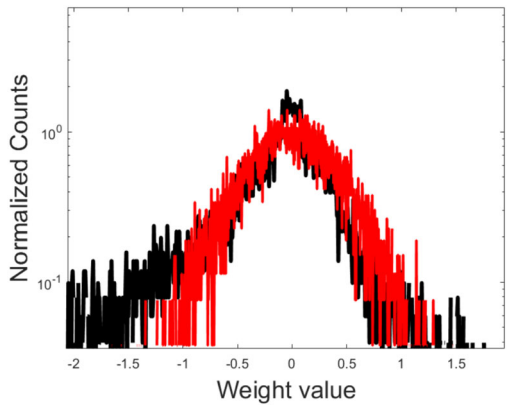

Fig. 4 The distribution of weights. Distribution of the weights of a perceptron. The red line follows the spectral training limited the $N_1 + N_2$ eigenvalues. The black line follows the training in direct space. Here, $N_1 \times N_2$ parameters are adjusted in the space of the nodes. The distribution is very similar, but the spectral learning employs about 10% of the parameters used in direct space. The distributions obtained when forcing the training in direct space to operate on a subset of $N_1 + N_2$ weights are very different from the one displayed (for every choice of the randomly selected family of weights to be trained).

smaller score (although compatible within error precision) than that achieved when the learning takes place in the spectral domain. When the training extends to all the $N_1 \times N_2$ weights (plus $N_1 + N_2$ bias), conventional learning yields a final accuracy of $(92.7 \pm 0.1)\%$. This is practically identical to the score obtained in the spectral domain, specifically $(92.5 \pm 0.2)\%$, when the sub-diagonal entries of the eigenvectors matrix are also optimised (for a total of $N_1 + N_2 + N_1 \times N_2$ free parameters). The remarkable observation is however that the distribution of the weights as obtained when the learning is restricted on the eigenvalues (i.e. using about the 10% of the parameters employed for full training in direct space) matches quite closely that retrieved by means of conventional learning schemes, see Fig. 4. This is not the case when the learning in direct space acts on a subset of N, randomly selected, weights (data not

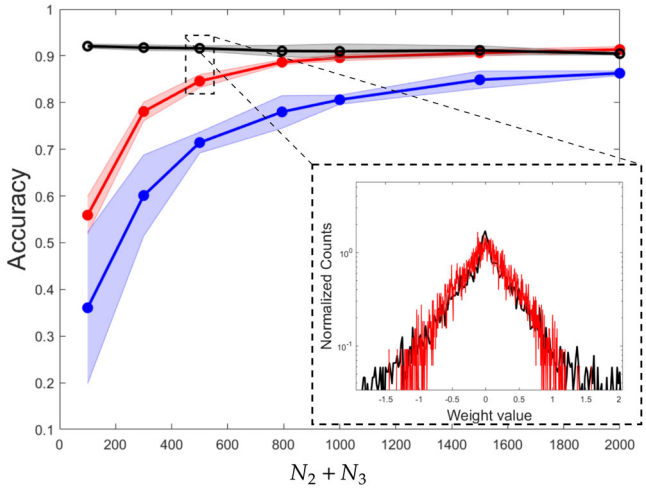

Fig. 5 Wide linear scheme. A three layers neural network is considered. The accuracy of the neural network is plotted as a function of the number of parameters that we chose to train with the spectral algorithm, $N_2 + N_3$. The red line reports on the performance of the spectral training. The blue line refers to the neural network trained in direct space: the optimisation runs on $N_2 + N_3$ parameters, a subset of the total number of adjustable weights $N_1N_2 + N_2N_3$. The black line stands for the accuracy of the linear neural network when training the full set of $N_1N_2 + N_2N_3$ parameters. Notice that the reported accuracy is comparable to that obtained for a standard two-layers perceptron. Inset: the distribution of the entries of the equivalent perceptrons are plotted. The red curve refer to the spectral learning restricted to operate on the eigenvalues; the black profile to the neural network trained in direct space, employing the full set of adjustable parameters. In both cases, the weights refer to the two-layers configuration obtained by retracting the intermediate linear layer employed during the learning stage.

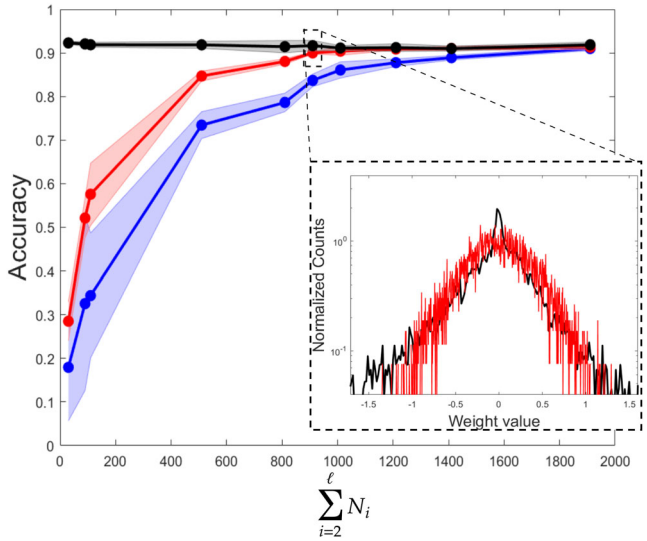

Fig. 6 Deep linear scheme. The performance of the spectral algorithm is tested for a multilayered linear configuration. Symbols are chosen in analogy to Fig. 5. In all cases, the input layer is made of $N_1 = 784$ elements and the output layer has $N_\ell = 10$ nodes. The first five points, from left to right in each of the curves depicted in the main panel, refer to a three layers (linear) neural network. The size of the intermediate layer is progressively increased, as $N_2 = 20, 80, 100, 500, 800$. The total number of trained eigenvalues is $N_2 + N_3$. The subsequent four points are obtained by considering a four layers architecture. In particular, $N_2 = 800$ while N_3 takes values in the interval $(100, 200, 400, 600)$. The training acts on $N_2 + N_3 + N_4$ eigenvalues. The final point in each curve is obtained with a four layers deep neural network. Here, $N_2 = 800$, $N_3 = 600$ and $N_3 = 500$, for a total of $N_2 + N_3 + N_4 + N_5$ tunable parameters in the spectral setting. Inset: the distribution of the entries of the equivalent perceptrons are displayed, with the same colour code adopted in Fig. 5. Also, in this case, the weights refer to the two-layers configuration obtained by retracting the intermediate linear layers employed in the learning stage.

shown). Based on the above, it can be therefore surmised that optimising the eigenvalues constitutes a rather effective pre-training strategy, which engages a modest computational load.

To further elaborate on the potentiality of the proposed technique, we modify the simple two-layers perceptron, with the inclusion of supplementary computing layers. As explained above the newly added layers plays an active role during the learning stage, but can be retracted in operating mode so as to return a two-layers perceptron. The weights of this latter bear however an imprint of the training carried out for the linear network in the expanded configuration. Two alternative strategies will be in particular contemplated. On the one side, we will consider a sole additional layer, endowed with N_2 nodes, interposed between the input and output layers made of, respectively, $N_1 = 784$ and $N_\ell \equiv N_3 = 10$ nodes. We will refer to this as to the wide linear configuration. The performance of the method can be tested by letting N_2 to progressively grow. On the other side, the deep linear configuration is obtained when interposing a sequence of successive (linear) stacks between the input ($N_1 = 784$) and the output ($N_\ell = 10$) layers.

In Fig. 5, we report on the performance of the wide learning scheme as a function of $N_2 + N_3$. As we shall clarify, this latter stands for the number of trained parameters for (i) the spectral learning acted on a subset of the tunable eigenvalues and for (ii) the conventional learning in direct space restricted to operate on a limited portion of the weights. The red line in the main panel of Fig. 5 refers to the simplified scheme where a subset of the eigenvalues are solely tuned (while leaving the eigenvectors fixed at the random realisation set by the initial condition). We have in particular chosen to train the second bunch of N_2 eigenvalues of the transfer matrix $\mathbf{A}_1$ and the $N_3 = 10$ non-trivial eigenvalues of matrix $\mathbf{A}_2$, in line with the prescriptions reported in the preceding section. The blue line reports on the accuracy of the neural network trained in direct space: the target of the optimisation is a subset of cardinality $N_2 + N_3$ of the $N_1N_2 + N_2N_3$ weights which could be in principle adjusted in the space of the nodes. The performance of the spectral method proves clearly superior, as it can be readily appreciated by visual inspection of Fig. 5. The black line displays the accuracy of the linear neural network when the optimisation acts on the full set of $N_1N_2 + N_2N_3$ trainable parameters (unconstrained neural network (NN)). No improvement is detectable when increasing the size of the intermediate layer: the displayed accuracy is substantially identical to that obtained for the basic perceptron trained with $N_1N_2 = 7840$ parameters. The spectral learning allows to reach comparable performance already at $N_2 = 1000$ (13% of the parameters used for the standard two-layers perceptron with $N_1 \times N_2$ parameters, as discussed above). In the inset of Fig. 5, the distribution of the entries of matrix $\mathcal{A}_c$, the equivalent perceptron, is depicted in red for the setting highlighted in the zoom. The black line refers to the two-layers equivalent of the neural network trained in direct space, employing the full set of trainable parameters (black dot enclosed in the top-left dashed rectangle drawn in the main panel of Fig. 5). The two distributions look remarkably close, despite the considerable reduction in terms of training parameters, as implemented in the spectral domain (for the case highlighted, 0.13% of the parameters employed under the standard training). Similarly to the above, the distribution obtained when forcing the training in direct space to act on a subset of $N_1 + N_2$ weights are

just a modest modulation of the initially assigned profile, owing to the local nature of the learning in the space of the nodes.

In Fig. 6, we report the results of the tests performed when operating under the deep linear configuration. Symbols are analogous to those employed in Fig. 5. In all inspected cases, the entry layer is made of $N_1 = 784$ elements and the output one has $N_\ell = 10$ nodes. The first five points, from left to right, refer to a three layers (linear) neural network. Hence, $\ell = 3$ and the size of the intermediate layer is progressively increased, $N_2 = 20, 80, 100, 500, 800$. The total number of trained eigenvalues is $N_2 + N_3$, and gets therefore larger as the size of the intermediate layer grows. The successive four points of the collections are obtained by setting $\ell = 4$. Here, $N_2 = 800$ while N_3 is varied (=100, 200, 400, 600). The training impacts on $N_2 + N_3 + N_4$ parameters. Finally, the last point in each displayed curve is obtained by working with a five layers deep neural network, $\ell = 5$. In particular, $N_2 = 800$, $N_3 = 600$ and $N_4 = 500$, for a total of $N_2 + N_3 + N_4 + N_5$ tunable parameters. Also in this case, the spectral algorithm performs better than conventional learning schemes constrained to operate with an identical number of free parameters. Similarly, the distribution of the weights of an equivalent perceptron trained in reciprocal space matches that obtained when operating in the space of the nodes and resting on a considerably larger number of training parameters. To sum up, eigenvalues are parameters of key importance for neural networks training, way more strategic than any other set of equivalent cardinality in the space of the nodes. As such, they allow for a global approach to the learning, with significant reflexes of fundamental and applied interest. In all cases here considered, the learning can extend to the eigenvectors: an optimised indentation of the eigen-directions contribute to enhance the overall performance of the trained device.

We now turn to considering a non-linear architecture. More specifically, we will assume a four layers network with, respectively, $N_1 = 784$, N_2, $N_3 = 120$, $N_4 = 10$. The non-linear ReLU filter acts on the third layer of the collection, while the second is a linear processing unit. As in the spirit of the wide network configuration evoked above, we set at testing the performance of the neural network for increasing N_2. For every choice of N_2, the linear layer can be retracted yielding a three-layered effective non-linear configurations. We recall however that training the network in the enlarged space where the linear unit is present leaves a non-trivial imprint in the weights that set the strength of the links in direct space.

In Fig. 7, we plot the computed accuracy as a function of N_2, the size of the linear layer. In analogy with the above analysis, the red curve refers to the training restricted to $N_2 + N_3 + N_4$ eigenvalues; the blue profile is obtained when the deep neural network is trained in direct space by adjusting an identical number of inter-nodes weights. As for the case of a fully linear architecture, by adjusting the eigenvalues yields better classification performances. The black line shows the accuracy of the neural network when the full set of $N_1N_2 + N_2N_3 + N_3N_4$ is optimised in direct space. The green line refers instead to the spectral learning when the eigenvalues and eigenvectors are trained simultaneously. The accuracies estimated for these two latter settings agree within statistical error, even if the spectral scheme seems more robust to overfitting (the black circles declines slightly when increasing N_2, while the collection of green points appears rather stable). To sum up, we enclose Table 1 where the obtained scores are compared. We display in particular the best (average) performance of the different training schemes, applied to the aforementioned distinct architectures, and over the explored range of sizes.

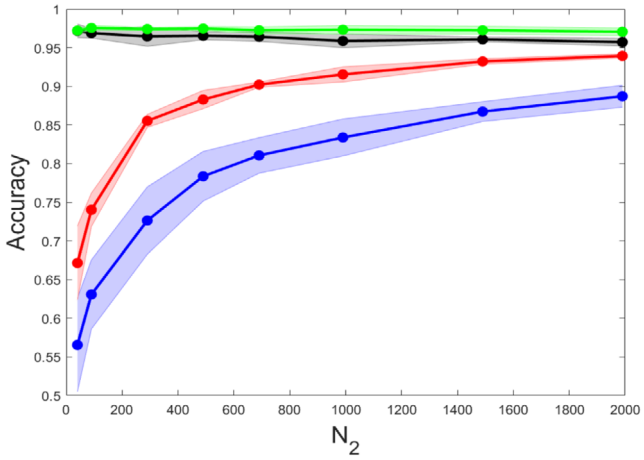

Fig. 7 Deep non-linear scheme. The accuracy of the non-linear deep neural network is tested. We assume a four layers network with, respectively, $N_1 = 784$, N_2, $N_3 = 120$, $N_4 = 10$; N_2 is changed so as to enlarge the set of parameters to be trained. The red line refers to the spectral training, with $N_2 + N_3 + N_4$ adjusted eigenvalues. The blue line stands for a neural network trained in direct space, the target of the optimisation being a subset made of $N_2 + N_3 + N_4$ weights, randomly selected from the available pool of $N_1N_2 + N_2N_3 + N_3N_4$ tunable parameters. The black line reports the accuracy of the linear neural network when training the full set of $N_1N_2 + N_2N_3 + N_3N_4$ weights. The green line refers to the spectral learning when eigenvalues and eigenvectors are simultaneously trained.

Table 1 Average performance.

MNIST best mean accuracy over the explored range	Linear wide	Linear deep	Non-linear
Constrained NN	87.8%	90.8%	88.9%
Train Λ	90.9%	91.5%	93.5%
Unconstrained NN	91.0%	91.2%	96.4%
Train Λ, Φ	92.2%	91.8%	97.3%

The best (average) performance of the different training schemes is displayed.

Discussion

Summing up, we have here proposed a novel approach to the training of deep neural networks which is bound to the spectral, hence reciprocal, domain. The eigenvalues and eigenvectors of the adjacency matrices that connects consecutive layers via directed feedforward links are trained, instead of adjusting the weights that bridge each pair of nodes of the collection, as it is customarily done in the framework of conventional ML approaches.

The first conclusion of our analysis is that optimising the eigenvalues, when freezing the eigenvectors, yields performances which are superior to those attained with conventional methods restricted to operate with an identical number of free parameters. It is therefore surmised that eigenvalues are key target parameters for neural networks training, in that they allow for a global handling of the learning. This is at variance with conventional approaches which seek at modulating the weights of the links among mutually connected nodes. Secondly, the spectral learning restricted to the eigenvalues yields a distribution of the weights which resembles quite closely that obtained with conventional algorithms bound to operate in direct space. For this reason, the proposed method could be used in combination with existing ML algorithms for an effective (and computationally advantageous) pre-training of deep neural networks. We have also shown that linear processing units inserted in between consecutive, non-linearly activated layers produce an enlargement of the learning parameters space, with beneficial effects in terms of performance of the trained device. Extending the learning so as to optimise the eigenvectors enhances the ability of the network to operate the

sought classification. In the proposed implementation, and to recover a feedforward architecture in direct space, we have assumed a nested indentation of the eigenvectors. Entangling the eigenvectors referred to successive stacks is the key for a recursive processing of the data, from the input to the output layer. Employing other non-orthogonal basis could eventually allow to challenge different topologies in direct space and shed novel light on the surprising ability of deep networks to cope with the assigned tasks.

In future perspective, it would be interesting to characterise the solutions attained with the spectral method, following the strategy outlined in ref. [26]. Further, it could be interesting to combine the spectral approach to other existing schemes which have been devised to improve the computational performance of deep neural networks, without significant loss in final recognition accuracy[21,27]. As a side remark, we notice that the authors of ref. [22] propose a single value decomposition of the rectangular matrix of the weights that links two adjacent layers. In our setting, the problem is reformulated in a space of extended dimensions which enables one to deal with square matrices and so invoke the spectral theorem. The eigenvalues are therefore stemming from the adjacency matrix that sets the architecture of the underlying neural network. The similitude of the two approaches solely resides in the idea of highlighting a (different) subset of parameters to be optimized. Our analysis is specifically aimed at implementing and testing an efficient learning scheme anchored to reciprocal space (which we have challenged against standard approaches to the learning). The focus of ref. [22] is on testing the behaviour of entropies and mutual information throughout learning.

Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Code availability

The codes employed, as well as a notebook to reproduce our results, can be found in the public repository of this project (https://github.com/Buffoni/spectral_learning).

Received: 5 June 2020; Accepted: 21 January 2021;
Published online: 26 February 2021

References

1. Bishop, C. M. *Pattern Recognition and Machine Learning* 1st edn (Springer, 2011).
2. Cover, T. M. & Thomas, J. A. *Elements of Information Theory*. Wiley series in telecommunications (Wiley, 1991).
3. Hastie, T., Tibshirani, R. & Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (Springer Science & Business Media, 2009).
4. Burkov, A. *The hundred-page machine learning book* **1** (Andriy Burkov Canada, 2019) http://themlbook.com/.
5. Sutton, R. S. & Barto, A. G. *Reinforcement Learning: an Introduction* (MIT Press, 2018).
6. Graves, A., Mohamed, A.-r. & Hinton, G. Speech recognition with deep recurrent neural networks. in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 6645–6649 (IEEE, 2013).
7. Sebe, N., Cohen, I., Garg, A. & Huang, T. S. *Machine Learning in Computer Vision*, Vol. 29 (Springer Science & Business Media, 2005).
8. Grigorescu, S., Trasnea, B., Cocias, T. & Macesanu, G. A survey of deep learning techniques for autonomous driving. *J. Field Robot.* **37**, 362–386 (2020).
9. Chen, M., Mao, S. & Liu, Y. Big data: a survey. *Mob. Netw. Appl.* **19**, 171–209 (2014).
10. Rosenblatt, F. *Principles of Neurodynamics, Perceptrons and the Theory of Brain Mechanisms* (Cornell Aeronautical Lab Inc., 1961).
11. Meyers, E. & Wolf, L. Using biologically inspired features for face processing. *Int. J. Comput. Vis.* **76**, 93–104 (2008).
12. Caponetti, L., Buscicchio, C. A. & Castellano, G. Biologically inspired emotion recognition from speech. *EURASIP J. Adv. Signal Process.* **2011**, 24 (2011).
13. Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychol. Rev.* **65**, 386 (1958).
14. Bengio, Y., Lamblin, P., Popovici, D. & Larochelle, H. Greedy layer-wise training of deep networks. *Adv. Neural Inform. Process. Syst.* **19**, 153–160 (2007).
15. Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* (MIT Press, 2016).
16. Hinton, G. E., Osindero, S. & Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural Comput.* **18**, 1527–1554 (2006).
17. Rumelhart, D. E. et al. Learning representations by back-propagating errors. *Cognitive Modeling* **5**, 1 (1988).
18. Xie, N., Ras, G., van Gerven, M. & Doran, D. Explainable deep learning: a field guide for the uninitiated. Preprint at https://arxiv.org/abs/2004.14545 (2020).
19. Hinton, G., Vinyals, O. & Dean, J. Distilling the knowledge in a neural network. Preprint at https://arxiv.org/abs/1503.02531 (2015).
20. Erhan, D., Courville, A. & Bengio, Y. Understanding representations learned in deep architectures. *Department dInformatique et Recherche Operationnelle, University of Montreal, QC, Canada, Tech. Rep 1355, no. 1* (2010).
21. Frankle, J., Schwab, D. J. & Morcos, A. S. Training BatchNorm and only BatchNorm: on the expressive power of random features in CNNs. Preprint at https://arxiv.org/abs/2003.00152 (2020).
22. Gabrié, M. et al. Entropy and mutual information in models of deep neural networks. *J. Stat. Mech. Theory Exp.* **2019**, 124014 (2019).
23. LeCun, Y. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/ (1998).
24. Byerly, A., Kalganova, T. & Dear, I. A branching and merging convolutional network with homogeneous filter capsules. Preprint at https://arxiv.org/abs/2001.09136 (2020).
25. Kingma, D. & Ba, J. Adam: a method for stochastic optimization. Preprint at https://arxiv.org/abs/1412.6980 (2014).
26. Feizi, S., Javadi, H., Zhang, J. & Tse, D. Porcupine neural networks: (almost) all local optima are global. Preprint at https://arxiv.org/abs/1710.02196 (2017).
27. Sainath, T. N., Kingsbury, B., Sindhwani, V., Arisoy, E. & Ramabhadran, B. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 6655–6659 (2013).

Author contributions

L.G. and L.B. carried out the numerical work. L.G. and D.F designed the study. L.B., T.C., D.F., L.G. and W.N. contributed to the interpretation of the results and to the editing of the manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s41467-021-21481-0.

Correspondence and requests for materials should be addressed to D.F.

Peer review information *Nature Communications* thanks Adam Byerly, and the other, anonymous, reviewer(s) for their contribution to the peer review of this work. Peer reviewer reports are available.

Reprints and permission information is available at http://www.nature.com/reprints

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/.

© The Author(s) 2021