UNIVERSITÀ DEGLI STUDI FIRENZE

PhD in
Physics and Astronomy

CYCLE XXXVI

On the foundation of artificial intelligence: spectral formulation and bioinspired models

Academic Discipline (SSD) FIS/02

Doctoral Candidate
Dr. Lorenzo Chicchi

[illegible]

Supervisors
Prof. Duccio Fanelli,
Prof. Franco Bagnoli

[illegible]

[illegible]

Coordinator
Prof. Giovanni Modugno

[illegible]

Contents

Introduction

Artificial intelligence (AI) has revolutionized numerous sectors, but its theoretical foundations remain a subject of intense research and debate. The extreme efficiency of this new technology too often tends to overshadow the importance of providing a more complete understanding of the mechanisms behind it. Understanding the fundamentals of AI is crucial both because it allows us to develop more efficient and interpretable models and also because it enables us to accompany the use of such technologies with an appropriate error theory that can quantify the associated degree of reliability. In this context, physics offers a rigorous and systematic approach. In fact, physicists, accustomed to studying the laws and principles that govern the physical world, can bring a unique perspective to the analysis of the foundations of AI.

One of the central algorithm of AI that has catalyzed scholars' attention are Artificial Neural Networks (ANNs), fascinating computational models inspired by the human brain. These networks, consisting of an interconnected set of artificial neurons, are capable of learning from data and adapting to changes in the environment.

The theory of networks, with its solid conceptual structure, can provide a coherent framework for describing the complex structures of artificial neural networks. In particular, the physics of complex systems has dedicated years of study to network theory, exploring the properties of dynamics evolving on networks, trying to understand how the interactions between individual components can give rise to collective behaviors and complex phenomena. During the years, scientists have revealed general principles that can be applied to a wide range of systems evolving on networks: from information propagation in social networks to the spread of diseases in a network of contacts.

The application of concepts and tools typical of network theory therefore contribute to shed novel light on the foundational aspect of AI and artificial neural networks. This multidisciplinary approach opens up new possibilities for understanding the functioning of neural networks, exploring their learning capabilities, analyzing the mechanisms of generalization and studying the emerging properties that characterize their behavior. In addition, network theory offers a fertile ground for developing new models of artificial neural networks, capable of overcoming current challenges and improving performance in various application areas.

Another highly promising research direction is to bridge the gap between artificial neural network models and biological neural networks. Neuroscience has made important progress in developing models of biological neural networks that try to

describe the complex functioning of the human brain. The goal is to better understand the mechanisms of learning, information processing and communication dynamics that characterize biological neural networks.
The integration of knowledge and techniques from both the field of artificial intelligence and neuroscience could open up new perspectives for a deeper understanding of artificial neural networks and for the advancement of neuroscience itself. Identifying and exploring correspondences between artificial neural network models and their biological counterparts can help defining general principles that guide the functioning of neural networks in both contexts. This synergy between artificial neural networks and neuroscience could lead to important discoveries and applications in both fields. A better understanding of biological neural networks could provide inspiration for the development of new models of artificial neural networks capable of mimicking the cognitive abilities of the human brain. At the same time, models of artificial neural networks can serve as tools for exploring and testing hypotheses about the functioning of biological neural networks, allowing for advances in understanding their complex dynamics.
During my doctoral project, we explored the points of contact between artificial neural networks, network theory and neuroscience. Particular attention was given to a novel way of parameterizing artificial neural networks known as Spectral Learning [1]. In this formulation, new trainable parameters are identified that can be interpreted, under appropriate assumptions, as the eigenvalues and eigenvectors of matrices that perform linear transformations between adjacent layers of a feedforward network.
In the first chapter of this thesis, we will illustrate the spectral method and discuss the relevant mathematical background. An improvement of the original version of the spectral parametrization will be also introduced, where additional decompositions (SVD and QR) are performed in synergy with spectral decomposition. By doing so, we achieve two distinct objectives: 1) obtain a level of accuracy comparable to that obtained when working with standard feedforward networks by training a significantly reduced number of parameters, 2) define a strategy whereby sparse networks can be obtained while maintaining high performance.
In the second chapter, we will introduce a pruning technique based on ranking the nodes of a feed forward neural network according to the eigenvalues obtained after the training. This strategy, which is achievable only within the framework of spectral training, seems to be particularly effective in generating highly compact networks in terms of the number of nodes, a characteristic in high demand in various application domains.
With the third chapter of this thesis, we will begin to bridge the gap between artificial neural networks and neuroscience models by making use of concepts derived from the new spectral formulation. In particular, in this chapter, we will present a new model of recurrent neural network called the Recurrent Spectral Network (RSN). This model represents a first attempt to introduce dynamics in the evaluative steps of neural networks. The learning process in RSN involves modeling the attraction basins of a discrete map in such a way that the dynamics associated with this map cause different inputs belonging to the same

class to converge to the same final state, thereby accomplishing classification tasks. Thanks to the unique characteristics of this model, a strategy has been defined through which multiple datasets can be successively processed without encountering the catastrophic forgetting phenomenon, a typical problem that arises when attempting to address sequential learning with standard feedforward networks.
In chapter four, we will introduce an extended and generalized version of the RSN model, referred to as Complex Recurrent Spectral Network ($\mathbb{C}$-RSN). In this updated model, the network's final state becomes localized and exhibits oscillatory behavior over time. This improved version addresses certain critical issues that exist in the original RSN model. Additionally, the trained model has the capability to sequentially process different inputs and to keep memory of multiple classifications made at subsequent moments.
Finally, in the fifth and sixth chapters, we will delve into two practical applications of neural networks within distinct research domains: epidemiology and geothermobarometry. In the first case we will use a particular neural network architecture (LSTM) to predict the trend of Covid19 cases starting from past time series of health and mobility indicators. In the second case, we will employ neural networks to derive insights into the internal morphology of volcanoes based on the chemical compositions of minerals collected after an eruption.
This thesis is strongly based on some of the papers published during my PhD. In particular, the papers used are: [2] for the first chapter, [3] for the second chapter, [4] for the third chapter an [5], [6] for fifth and sixth chapters respectively.

Chapter 1

Introducing and improving the Spectral Learning

Machine learning (ML) technologies nowadays play a pivotal role in systematizing complex data via self-consistent elucidation of intertwined data correlations. To this end, ML models learn from data by identifying distinctive features, which form their basis for decision-making. This task is accomplished by solving an optimisation problem that seeks to minimise a suitably defined loss function which compares expected and actual output. When operating with deep learning architectures with a feedforward structure, data supplied as input are processed via a repeated sequence of linear and non-linear transformations. These latter reflect the spatial sorting of the chosen reservoir of computing neurons. The linear transformation, indeed, maps the signal from any given layer to its immediate analogue, following the assigned neural network arrangement. Customarily, the matrix elements of the linear transformation (weights) are the target of the optimisation (i.e. loss minimization). In [1] a radically new approach to this learning scheme, which anchors the process to reciprocal space, was first proposed. The training there seeks to modify the eigenvectors and eigenvalues of transfer operators in direct space. By acting on the spectral coordinates no increase in terms of computational and complexity load is produced, as compared to conventional algorithms. In this chapter we will introduce such parametrization and discuss further improvement of the technology.

1.1 Introduction

By reformulating the learning in reciprocal space enables one to shape key collective modes, the eigenvectors, which are implicated in the process of progressive embedding, from the input layer to the detection point. Even more interestingly, one can assume the eigenmodes of the inter-layer transfer operator to align along suitable random directions and identify the associated eigenvalues as target for the learning scheme. This results in a dramatic compression of the training parameters space, yielding accuracies which are superior to those attained with conventional methods restricted to operate with an identical number of tunable parameters. Nonetheless, neural networks trained in the space of nodes with no restrictions on the set of adjusted weights, achieve better classification scores, as compared to their spectral homologues with quenched eigendirections. In the former case, the number of free parameters grows as the product of the sizes of adjacent layer pairs, thus quadratically in terms of hosted neurons. In the latter, the number of free parameters increases linearly with the size of the layers (hence with the number of neurons), when the eigenvalues are solely allowed to change. Also training the eigenvectors amounts to dealing with a set of free parameters equivalent to that employed when the learning is carried out in direct space: in this case, the two methods yield performances which are therefore comparable.

Starting from this setting, we begin by discussing a straightforward generalisation of the spectral learning scheme presented in [1], which proves however effective in securing a significant improvement on the recorded classification scores, while still optimising a number of parameters which scales linearly with the size of the network. The proposed generalisation paves the way to a biomimetic interpretation of the spectral training scheme. The eigenvalues can be tuned so as to magnify/damp the contribution associated to the input nodes. At the same time, they modulate the excitability of the receiving nodes, as a function of the local field. This latter effect is reminiscent of the homeostatic plasticity [7] as displayed by living neurons. Further, we will show that the residual gap between conventional and spectral trainings methods can be eventually filled by resorting to apt decompositions of the non trivial block of the eigenvectors matrix, which place the emphasis on a limited set of collective variables. Finally, we will prove that working in reciprocal space turns out to be by far more performant, when aiming at training sparse neural networks. Because of the improvement in terms of computational load, and due to the advantage of operating with collective target variables as we will make clear in the following, it is surmised that modified spectral learning of the type here discussed should be considered as a viable standard for deep neural networks training in artificial intelligence applications.

Stated differently, the results reported in this chapter provide evidence that neural networks can be efficiently trained with substantially lower computational cost while maintaining comparable accuracy. The quest for innovative neural network schemes beyond state-of-the-art technology constitutes a rather fertile field of investigation which can be tackled via diverse strategies [3], [8], [9]. Compression and acceleration techniques are routinely employed to suit the scope. These are customarily divided into four distinct categories (parameter pruning and quantization, low-rank factorization, transferred or compact convolutional filters, and knowledge distillation) as thoroughly reviewed in the comprehensive survey [10]. We here face the problem from a different angle by aiming at reducing the number of trainable parameters rather than compactifying the underlying network as a whole. This is also the spirit of the methods put forward in [11], [12].

1.2 The spectral learning

To test the effectiveness of the proposed method we will consider classification tasks operated on three distinct database of images. The first is the renowned MNIST [13], composed by greyscale images of dimension 28×28 pixels. The second is Fashion-MNIST (F-MNIST) [14] which are still depicted with a greyscale with dimension 28×28 but display an enhanced degree of inherent complexity for what concerns the type of classification requiredas compared to the basic MNIST benchmark model (more complex shapes, patterns on items). The last one is CIFAR-10 [15] a richer dataset composed by 32×32 RGB images of complex real-world objects divided in 10 classes (airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks). In all considered cases, use can be made of a deep neural network to perform the sought classification, namely to automatically assign the image supplied as an input to the class it belongs to. The neural network is customarily trained via standard backpropagation algorithms to tune the weights that connect consecutive stacks of the multi-layered architecture. The assigned weights, target of the optimisation procedure, bear the information needed to allocate the examined images to their reference category.

Consider a deep feedforward network made of ℓ distinct layers and label each layer with the progressive index i $(= 1, ..., \ell)$. Denote by N_i the number of computing units, the neurons, that belong to layer i. The total number of parameters that one seeks to optimise in a dense neural network setting (all neurons of any given layer with $i < \ell - 1$ are linked to every neurons of the adjacent layer) equals $\sum_{i=1}^{\ell-1} N_i N_{i+1}$, when omitting additional bias. As we shall prove in the following, impressive performance can be also achieved by pursuing a markedly different procedure, which requires acting on just $N_1 + N_\ell + 2\sum_{i=2}^{\ell-1} N_i$ free parameters (not including bias). To this end, let us begin by reviewing the essence of spectral learning method as set forth in [1].

Introduce $N = \sum_{i=1}^{\ell} N_i$ and create a column vector $\vec{n}_1$, of size N, whose first N_1 entries are the intensities (from the top-left to the bottom-right, moving horizontally) as displayed on the pixels of the input image. All other entries of $\vec{n}_1$ are set to zero. The ultimate goal is to transform $\vec{n}_1$ into an output vector $\vec{n}_\ell$, of size N, whose last N_ℓ elements reflect the intensities of the output nodes where reading takes eventually place. This is achieved with a nested sequence of linear transformations, as exemplified in the following. Consider the generic vector $\vec{n}_k$, with $k = 1, ..., \ell - 1$, as obtained after k compositions of the above procedure. This latter vector undergoes a linear transformation to yield $\vec{n}_{k+1} = \mathbf{A}^{(k)}\vec{n}_k$. Further, $\vec{n}_{k+1}$ is processed via a suitably defined non-linear function, denoted by $f(\cdot, \beta_k)$, where β_k stands for an optional bias. Focus now on $\mathbf{A}^{(k)}$, a $N \times N$ matrix with a rather specific structure, as we will highlight hereafter. Posit $\mathbf{A}^{(k)} = \mathbf{\Phi}^{(k)}\mathbf{\Lambda}^{(k)}\left(\mathbf{\Phi}^{(k)}\right)^{-1}$, by invoking a spectral decomposition. $\mathbf{\Lambda}^{(k)}$ is the diagonal matrix of the eigenvalues of $\mathbf{A}^{(k)}$. By construction we impose, $\left(\mathbf{\Lambda}^{(k)}\right)_{jj} = 1$ for $j < \sum_{i=1}^{k} N_i$ and $j > \sum_{i=1}^{k+1} N_i$. The remaining N_k elements are initially set to random numbers, e.g. extracted from a uniform distribution, and define the target of the learning scheme [1]. Back to the spectral decomposition of $\mathbf{A}^{(k)}$, $\mathbf{\Phi}^{(k)}$ is assumed to be the identity matrix $\mathbb{I}_{N\times N}$, with the inclusion of a sub-diagonal rectangular block $\boldsymbol{\phi}^{(k)}$ of size $N_{k+1}\times N_k$. This choice corresponds to dealing with a feedforward arrangements of nested layers. A straightforward calculation returns $\left(\mathbf{\Phi}^{(k)}\right)^{-1} = 2\mathbb{I}_{N\times N} - \mathbf{\Phi}^{(k)}$, which readily yields $\mathbf{A}^{(k)} = \mathbf{\Phi}^{(k)}\mathbf{\Lambda}^{(k)}\left(2\mathbb{I}_{N\times N} - \mathbf{\Phi}^{(k)}\right)$. In the simplest setting that we shall inspect in the following, the off-diagonal elements of matrix $\mathbf{\Phi}^{(k)}$ are frozen to nominal values, selected at random from a given distribution. In this minimal version, the spectral decomposition of the transfer operators $\mathbf{A}^{(k)}$ enables one to isolate a total of $N=\sum_{i=1}^{\ell} N_i$ adjustable parameters, the full collection of non trivial eigenvalues, which can be self-consistently trained. To implement the learning scheme on these premises, we consider $\vec{n}_\ell$, the image on the output layer of the input vector $\vec{n}_1$:

$$\vec{n}_\ell = f\left(\mathbf{A}^{(\ell-1)}...f\left(\mathbf{A}^{(2)}f\left(\mathbf{A}^{(1)}\vec{n}_1,\beta_1\right),\beta_2\right),\beta_{\ell-1}\right) \tag{1.1}$$

and calculate $\vec{z}=\sigma(\vec{n}_\ell)$ where $\sigma(\cdot)$ stands for the softmax operation. We then introduce the categorical cross-entropy loss function $\mathrm{CE}(l(\vec{n}_1),\vec{z})$ where the quantity $l(\vec{n}_1)$ identifies the label attached to $\vec{n}_1$ reflecting the category to which it belongs via one-hot encoding [16]. More specifically, the k-th elements of vector $l(\vec{n}_1)$ is equal to unit (the other entries being identically equal to zero) if the image supplied as an input is associated to the class of items grouped under label k.

The loss function can be minimized by acting on a limited set of free parameters, the collection of N non trivial eigenvalues of matrices $\mathbf{A}_k$ (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 principle, the sub-diagonal blocks $\boldsymbol{\phi}^{(k)}$ (the non orthogonal entries of the basis that diagonalises $\mathbf{A}^{(k)}$) can be optimised in parallel, but this choice nullifies the gain in terms of parameters containment, as achieved via spectral decomposition, when the eigenvalues get solely modulated. The remaining part of this chapter is entirely devoted to overcoming this limitation, while securing the decisive enhancement of the neural network's performance.

1.3 Improving the spectral scheme and testing it

The first idea, as effective as it is simple, is to extend the set of trainable eigenvalues. When mapping layer k into layer $k+1$, we can in principle act on N_k+N_{k+1} eigenvalues, without restricting the training to the N_{k+1} elements, which were identified as the sole target of the spectral method in its original conception (except for the first mapping, from the input layer to its adjacent counterpart). As we shall clarify in the following, the eigenvalues can be trained twice, depending on whether they originate from incoming or outcoming nodes, along the successive arrangement of nested layers. The global number of trainable

[1]The only noticeable exception is when $k=1$. In this case, the first N_1 diagonal elements of $\mathbf{\Lambda}^{(1)}$ take part to the training.

parameters is hence $N_1 + N_\ell + 2\sum_{i=2}^{\ell-1} N_i$, as anticipated above. A straightforward calculation, carried out in the appendix (A), returns a closed analytical expression for $w_{ij}^{(k)}$, the weights of the edges linking nodes i and j in direct space, as a function of the underlying spectral quantities. In formulae:

$$w_{ij}^{(k)} = \left(\lambda_{m(j)}^{(k)} - \lambda_{l(i)}^{(k)}\right) \Phi_{l(i),m(j)}^{(k)} \tag{1.2}$$

where $l(i) = \sum_{s=1}^{k} N_s + i$ and $m(j) = \sum_{s=1}^{k-1} N_s + j$, with $i \in (1, ..., N_{k+1})$ and $j \in (1, ..., N_k)$. More specifically, j runs on the nodes at the departure layer (k), whereas i identifies those sitting at destination (layer $k+1$). In the above expression, $\lambda_{m(j)}^{(k)}$ stand for the first N_k eigenvalues of $\mathbf{\Lambda}^{(k)}$. The remaining N_{k+1} eigenvalues are labelled $\lambda_{l(i)}^{(k)}$. To help comprehension denote by $x_j^{(k)}$ the activity on nodes j. Then,

$$x_i^{(k+1)} = \sum_{j=1}^{N_k} \left(\lambda_{m(j)}^{(k)} \Phi_{l(i),m(j)}^{(k)} x_j^{(k)}\right) - \lambda_{l(i)}^{(k)} \sum_{j=1}^{N_k} \left(\Phi_{l(i),m(j)}^{(k)} x_j^{(k)}\right). \tag{1.3}$$

The eigenvalues $\lambda_{m(j)}^{(k)}$ modulate the density at the origin, while $\lambda_{l(i)}^{(k)}$ appears to regulate the local node's excitability relative to the network activity in its neighbourhood. This is the artificial analogue of the *homeostatic plasticity*, the strategy implemented by living neurons to maintain the synaptic basis for learning, respiration, and locomotion [7].

To illustrate the effectiveness of the proposed methodology we make reference to Fig. 1.1, which summarises a first set of results obtained for MNIST. To keep the analysis as simple as possible we have here chosen to deal with $\ell = 3$. The sizes of the input (N_1) and output (N_3) layers are set by the specificity of the considered dataset. Conversely, the size of the intermediate layer (N_2) can be changed at will. We then monitor the relative accuracy, i.e. the accuracy displayed by the deep neural networks trained according to different strategies, normalised to the accuracy achieved with an identical network trained with conventional methods. In the upper panel of Fig. 1.1, the performance of the neural networks trained via the modified spectral strategy (referred to as to *Spectral*) is displayed in blue (triangles). The recorded accuracy is satisfactory (about 90% of that obtained with usual means and few percent more than that obtained with the spectral method of original conception [1]), despite the modest number of trained parameters. To exemplify this, in the bottom panel of Fig. 1.1 we plot the relative ratio of the number of tuned parameters (*Spectral*) vs. conventional one) against N_2 (blue triangles) : the reduction in the number of parameters as follows the modified spectral method is staggering. Working with the other employed dataset, respectively F-MNIST and CIFAR-10, yields analogous conclusions (see appendix (A)).

One further improvement can be achieved by replacing $\boldsymbol{\phi}^{(k)}$ with its equivalent singular value decomposition (SVD), a factorization that generalizes the eigen-decomposition to rectangular (in this framework, $N_{k+1} \times N_k$) matrices (see [17] for an application to neural networks). In formulae, this amounts to postulate

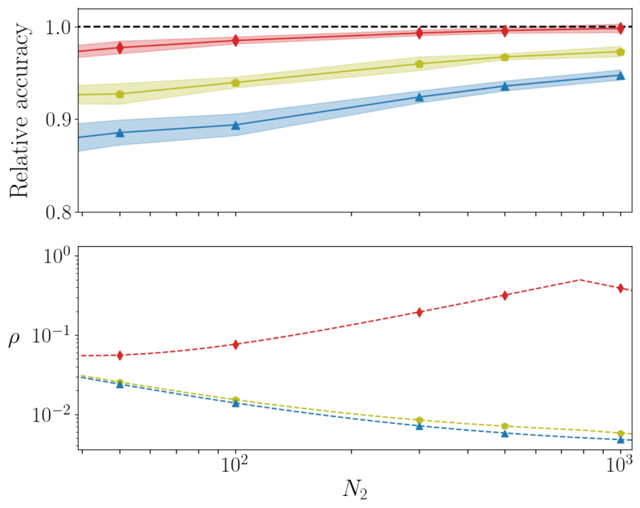


Figure 1.1: **The case of MNIST.** Upper panel: the accuracy of the different learning strategies, normalised to the accuracy obtained for an identical deep neural network trained in direct space, as a function of the size of the intermediate layer, N_2. Triangles stand for the the relative accuracy obtained when employing the spectral method (*Spectral*). Pentagons refer to the setting which extends the training to the eigenvectors' blocks via a SVD decomposition. Specifically, matrices $\mathbf{U}_k$ and $\mathbf{V}_k$ are randomly generated (with a uniform distribution of the entries) and stay unchanged during optimisation. The singular values are instead adjusted together with the eigenvalues which stem from the spectral method (this configuration is labelled S-SVD). Diamonds are instead obtained when the eigenvalues and the elements of the triangular matrix $\mathbf{R}$ (as follows a QR decomposition of the eigenvectors' blocks) are simultaneously adjusted S-QR). Here, $\mathbf{Q}$ is not taking part to the optimisation process (its entries are random number extracted from a uniform distribution). Errors are computed after 10 independent realisations of the respective procedures. Lower panel: ρ the ratio of the number of tuned parameters (modified spectral, S-SVD, S-QR methods vs. conventional one) is plotted against N_2. In calculating ρ the contribution of the bias is properly acknowledged. As a reference, the best accuracy obtained over the explored range for the deep network trained with conventional means is 98%.

$\boldsymbol{\phi}^{(k)} = \mathbf{U}_k \boldsymbol{\Sigma}_k \mathbf{V}_k^T$ where $\mathbf{V}_k$ and $\mathbf{U}_k$ are, respectively, $N_k \times N_k$ and $N_{k+1} \times N_{k+1}$ real orthogonal matrices. On the contrary, $\boldsymbol{\Sigma}_k$ is a $N_{k+1} \times N_k$ rectangular diagonal matrix, with non-negative real numbers on the diagonal. The diagonal entries of $\boldsymbol{\Sigma}_k$ are the singular values of $\boldsymbol{\phi}_k$. The symbol $(\cdot)^T$, stands for the transpose operation. The learning scheme can be hence reformulated as follows. For each k, generate two orthogonal random matrices $\mathbf{U}_k$ and $\mathbf{V}_k$. These latter are not updated during the successive stages of the learning process. At variance, the $M_{k+1} = \min(N_k, N_{k+1})$ non trivial elements of $\boldsymbol{\Sigma}_k$ take active part to the optimisation process. For each k, $M_{k+1} + N_k + N_{k+1}$ parameters can be thus modulated to optimize the information transfer, from layer k to layer $k+1$. Stated differently, M_{k+1} free parameters adds up to the $N_k + N_{k+1}$ eigenvalues that get modulated under the original spectral approach. One can hence count on a larger set of parameters as compared to that made available via the spectral method, restricted to operate with the eigenvalues. Nonetheless, the total number of parameters scales still with the linear size N of the deep neural network, and not quadratically, as for a standard training carried out in direct space. This addition (referred to as the *S-SVD* scheme) yields an increase of the recorded classification score, as compared to the setting where the *Spectral* method is solely employed, which is however not sufficient to fill the gap with conventional schemes (see Fig. 1.1). Similar scenarios are found for F-MNIST and CIFAR-10 (see appendix (A)), with varying degree of improvement, which reflects the specificity of the considered dataset.

A decisive leap forward is however accomplished by employing a QR factorization of matrix $\boldsymbol{\phi}^{(k)}$. For $N_{k+1} > N_k$, this corresponds to writing the $N_{k+1} \times N_k$ matrix $\boldsymbol{\phi}_k$ as the product of an orthogonal $N_{k+1} \times N_k$ matrix $\mathbf{Q}_k$ and an upper triangular $N_k \times N_k$ matrix $\mathbf{R}_k$. Conversely, when $N_{k+1} < N_k$, we factorize $\boldsymbol{\phi}_k^T$, in such a way that the square matrix $\mathbf{R}_k$ has linear dimension N_{k+1}. In both cases, matrix $\mathbf{Q}_k$ is randomly generated and stays frozen during gradient descent optimisation. The $M_{k+1}(M_{k+1}+1)/2$ entries of the $M_{k+1} \times M_{k+1}$ matrix $\mathbf{R}_k$ can be adjusted so as to improve the classification ability of the trained network (this strategy of training, integrated to the *Spectral* method, is termed *S-QR*). Results are depicted in Fig. 1.1 with (red) diamonds. The achieved performance is practically equivalent to that obtained with a conventional approach to learning. Also in this case $\rho < 1$, the gain in parameter reduction being noticeable when N_1 is substantially different (smaller or larger) than N_2, for the case at hand. Interestingly enough, for a chief improvement of the performance, over the SVD reference case, it is sufficient to train a portion of the off diagonal elements of $\mathbf{R}$. In the appendix (A), we report the recorded accuracy against p, the probability to train the entries that populate the non null triangular part of $\mathbf{R}_k$. The value of the accuracy attained with conventional strategies to the training is indeed approached, already at values of p which are significantly different from unit.

1.4 Imposing sparsity with spectral method

The quest for a limited subset of key parameters which define the target of a global approach to the training is also important for its indirect implications, besides the obvious reduction in terms of algorithmic complexity. As a key application to exemplify this point, we shall consider the problem of performing the classification tasks considered above, by training a neural network with a prescribed degree of imposed sparsity. This can be achieved by applying a non linear filter on each individual weight w_{ij}. The non linear mask is devised so as to return zero (no link present) when $|w_{ij}| < C$. Here, C is an adaptive cut-off which can be freely adjusted to allow for the trained network to match the requested amount of sparsity. This latter is measured by a scalar quantity, spanning the interval $[0, 1]$: when the degree of sparsity is set to zero, the network is dense. At the opposite limit, when the sparsity equals one, the nodes of the network are uncoupled and the information cannot be transported across layers. Working with the usual approach to the training, which seeks to modulate individual weights in direct space, one has to face an obvious problem. When the weight of a given link is turned into zero, then it gets excluded by the subsequent stages of the optimisation process. Consequently, a weight that has been silenced cannot regain an active role in the classification handling. This is not the case when operating under the spectral approach to learning, also when complemented by the supplemental features tested above. The target of the optimisation, the spectral attributes of the transfer operators, are not biased by any filtering masks: as a consequence, acting on them, one can rescue from oblivion weights that are deemed useless at a given iteration (and, as such, silenced), but which might prove of help, at later stages of the training. In Fig. 1.2, the effect of the imposed sparsity on the classification accuracy is represented for conventional vs. *S-QR* method. The latter is definitely more performant in terms of displayed accuracy, when the degree of sparsity gets more pronounced. The drop in accuracy as exhibited by the sparse network trained with the *S-QR* modality is clearly less pronounced, than that reported for an equivalent network optimised in direct space. Deviations between the two proposed methodologies become indeed appreciable in the very sparse limit, i.e. when the residual active links are too few for a proper functioning of the direct scheme. In fact, edges which could prove central to the classification, but that are set silent at the beginning, cannot come back to active. At variance, the method anchored to reciprocal space can identify an optimal pool of links (still constraint to the total allowed for) reversing to the active state, those that were initially set to null. Interestingly, it can be shown that a few hubs emerge in the intermediate layer, which collect and process the information delivered from the input stack.

1.5 Conclusion

Taken altogether, it should be unequivocally concluded that a large body of free parameters, usually trained in machine learning applications, is *de facto*

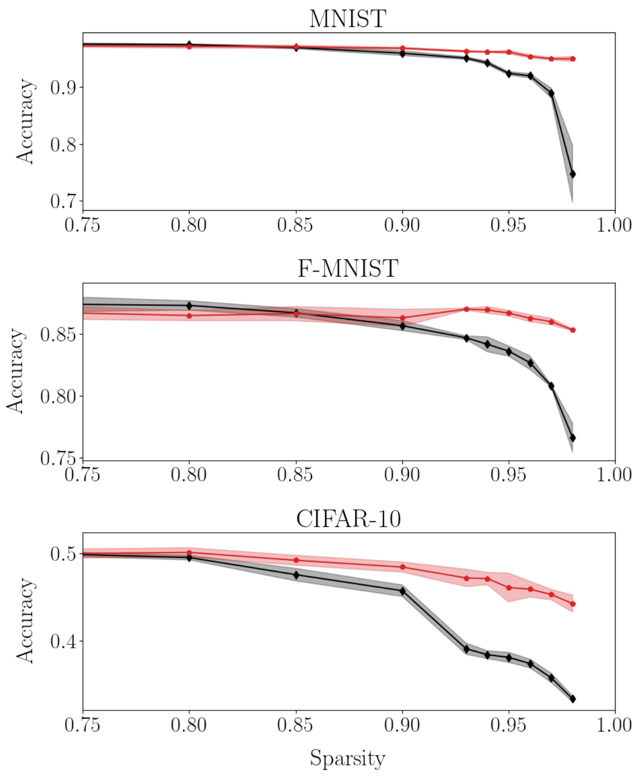


Figure 1.2: **Training sparse networks.** The accuracy of the trained network against the degree of imposed sparsity. Black diamonds refer to the usual training in direct space, while red pentagons refer to the *S-QR* method. From top to bottom: results are reported for MNIST, F-MNIST and CIFAR-10, respectively. In all cases, $\ell = 3$.

unessential. The spectral learning scheme, supplemented with a QR training of the non trivial portion of eigenvectors' matrix, enabled us to identify a limited subset of key parameters which prove central to the learning procedure, and reflect back with a global impact on the computed weights in direct space. This observation could materialise in a drastic simplification of current machine learning technologies, a challenge at reach via algorithmic optimisation carried out in dual space. Quite remarkably, working in reciprocal space yields trained networks with better classification scores, when operating at a given degree of imposed sparsity. This finding suggests that shifting the training to the spectral domain might prove beneficial for a wide gallery of deep neural networks applications.

In the next chapter, we will show that the spectral method can be employed to devise an effective pruning strategy that ranks nodes based on their associated eigenvalues as obtained during training

Chapter 2

Spectral Pruning

As seen in the previous chapter, training of neural networks can be reformulated in spectral space, by allowing eigenvalues and eigenvectors of the network to act as target of the optimization instead of the individual weights. Working in this setting, we show that the eigenvalues can be used to rank the nodes' importance within the ensemble. Indeed, we will prove that sorting the nodes based on their associated eigenvalues, enables effective pre- and post-processing pruning strategies to yield massively compacted networks (in terms of the number of composing neurons) with virtually unchanged performance. The proposed methods are tested for different architectures, with just a single or multiple hidden layers, and against distinct classification tasks of general interest.

2.1 Introduction

In this chapter we will discuss a relevant byproduct of the spectral learning scheme. More specifically, we will argue that the eigenvalues do provide a reliable ranking of the nodes, in terms of their associated contribution to the overall performance of the trained network. Working along these lines, we will empirically prove that the absolute value of the eigenvalues is an excellent marker of the node's significance in carrying out the assigned discrimination task. This observation can be effectively exploited, downstream of training, to filter the nodes in terms of their relative importance and prune the unessential units so as to yield a more compact model, with almost identical classification abilities. The effectiveness of the proposed method has been tested for different feed-forward architectures, with just a single or multiple hidden layers, by invoking several activation functions, and against distinct datasets for image recognition, with various levels of inherent complexity. Building on these findings, we will also propose a two stages training protocol to generate minimal networks (in terms of allowed computing neurons) which outperform those obtained by hacking off dispensable units from a large, fully trained, apparatus. This strategy can be seen as an effective way to discover sub-networks (a.k.a. "winning tickets" [18]) with recorded performance comparable to those displayed by their unaltered homologues, after a proper round of training [18]. More specifically, after a first round of training which solely acts on the eigenvalues, one can identify the most relevant nodes, as follows the magnitude of the associated eigenvalues. Since the first training stage is just targeted to eigenvalues, the eigenvectors obtained after pruning are still bearing reflexes of the random initialization and thus represent a sort of "winning ticket"[18]. In this respect, according to the above reasoning, the proposed two stages strategy can be seen as a novel and efficient way to discover optimal sub-networks.

2.2 Conventional Pruning Techniques

Generally speaking, it is possible to ideally group various approaches for network compression into five different categories: Weights Sharing, Network Pruning, Knowledge Distillation, Matrix Decomposition and Quantization [10], [19].

Weights Sharing defines one of the simplest strategies to reduce the number of parameters, while allowing for a robust feature detection. The key idea is to have a shared set of model parameters between layers, a choice which reflects back in an effective model compression. An immediate example of this methodology are the convolutional neural networks [20]. A refined approach is proposed in Bat et al. [21] where a virtual infinitely deep neural network is considered. Further, in Zhang et al. [12] an ℓ_1 group regularizer is exploited to induce sparsity and, simultaneously, identify the subset of weights which can share the same features.

Network Pruning is arguably one of the most common technique to compress Neural Network: in a nutshell it aims at removing a set of weights according to a certain criterion (magnitude, importance, etc). Chang et al. [22] proposed an iterative pruning algorithm that exploits a continuously differentiable version of the $\ell_{\frac{1}{2}}$ norm, as a penalty term. Molchanov et al. [23] focused on pruning convolutional filters, so as to achieve better inference performances (with a modest impact on the recorded accuracy) in a transfer leaning scenario. Starting from a network fine-tuned on the target task, they proposed an iterative algorithm made up of three main parts: (i) assessing the importance of each convolutional filter on the final performance via a Taylor expansion, (ii) removing the less informative filters and (iii) re-training the remaining filters, on the target task. Inspired by the pioneering work in [18], Pau de Jorge et al. [24] proved that pruning at initialization leads to a significant performance degradation, after a certain pruning threshold. In order to overcome this limitation they proposed two different methods that enable an initially trimmed weight to be reconsidered during the subsequent training stages.

Knowledge Distillation is yet another technique, firstly proposed by Hinton et al. [11]. In its simplest version Knowledge Distillation is implemented by combining two objective functions. The first accounts for the discrepancy between the predicted and true labels. The second is the cross-entropy between the output produced by the examined network and that obtained by running a (generally more powerful) trained model. In [25] Polino et al. proposed two approaches to mix distillation and quantization (see below): the first method uses the distillation during the training of the so called student network under a fixed quantization scheme while the second exploits a network (termed the teacher network) to directly optimize the quantization. Mirzadeh et al. [26] analyzed the regime in which knowledge distillation can be properly leveraged. They discovered that the representation power gap of the two networks (teacher and student) should be bounded for the method to yield beneficial effects. To resolve this problem, they inserted an intermediate network (the assistant), which sits in between the teacher and the student, when their associated gap is too large.

Matrix Decomposition is a technique that remove redundancies in the parameters by the means of a tensor/matrix decomposition. Masana et al. [27] proposed a matrix decomposition method for transfer learning scenario. They showed that decomposing a matrix taking into account the activation outperforms the approaches that solely rely on the weights. In [28], Novikov et al. proposed to replace the dense layer with its Tensor-Train representation [8]. Yu et al. [29] introduced a unified framework, integrating the low-rank and sparse decomposition of weight matrices with the feature map reconstructions.

Quantization, as also mentioned above, aims at lowering the number of bits used to represent any given parameter of the network. Stock et al. [30] defined an algorithm that quantize the model by minimizing the reconstruction error for inputs sampled from the training set distribution. The same authors also claimed that their proposed method is particularly suited for compressing residual network architectures and that the compressed model proves very efficient when run on CPU. In Banner et al. [31] a practical 4-bit post-training quantization approach was introduced and tested. Moreover, a method to reduce network complexity based on node-pruning was presented by He et al. in [32]. Once the network has been trained, nodes are classified by means of a node importance function and then removed or retained depending on their score. The authors proposed three different node ranking functions: entropy, output-weights norm (onorm) and input-weights norm (inorm). In particular, the input-weights norm function is defined as the sum of the absolute values of the incoming connections weights. As we will see this latter defines the benchmark model that we shall employ to challenge the performance of the trimming strategy here proposed. Finally, it is worth mentioning the Conditional Computation methods [33]–[35]: the aim is to dynamically skip part of the network according to the provided input so as to reduce the computational burden.

Summing up, pruning techniques exist which primarily pursue the goal of enforcing a sparsification by cutting links from the trained neural network and have been reviewed above. In contrast with them, the idea of our method is to a posteriori identify the nodes of the trained network which prove unessential for a proper functioning of the device and cut them out from ensemble made of active units. This yields a more compact neural network, in terms of composing neurons, with unaltered classification performance. The method relies on the spectral learning [2], [36] and exploits the fact that eigenvalues are credible parameters to gauge the importance of a given node among those composing the destination layer. In short, our aim is to make the network more compact by removing nodes classified as unimportant, according to a suitable spectral rating.

2.3 Results

In order to assess the effectiveness of the eigenvalues as a marker of the node's importance (and hence as a potential target for a cogent pruning procedure) we will consider a fully connected feed-forward architecture. Applications of the explored methods will be reported for $\ell = 3$ and $\ell > 3$ configurations. The nodes that compose the hidden layers are the target of the implemented pruning strategies. As we shall prove, it is possible to get rid of the vast majority of nodes without reflecting in a sensible decrease in the test accuracy, if the filter, either in its pre- or post-training versions, relies on the eigenvalues ranking. Moreover, it is also important to stress that, in general terms, the pruning of unessential nodes improves the computational efficiency of the network. As a matter of fact, reducing the number of output nodes leads a compression in terms of both memory and inference time which is directly proportional to the number of removed elements. As an example, by pruning a fraction α (< 1) of the total nodes, we obtain a new layer with $\alpha \cdot N$ less neurons and a memory reduction of $\alpha \cdot N$ times the number of input features.

For our test, we used three different datasets of images, already presented in the previous chapter. The first is the celebrated MNIST database of handwritten digits [13], the second is Fashion-MNIST (F-MNIST), a dataset of Zalando's article images, the third is CIFAR-10 a collection of images from 10 different classes. In the main text we report our findings for Fashion-MNIST. Analogous investigations carried out for MNIST and CIFAR10 will be reported as supplementary information in appendix (B). Further, different activation functions have been employed to evaluate the performance of the methods. In this chapter, we will show the results obtained for the ELU. The conclusion obtained when operating with the ReLU and tanh are discussed in the appendix (B). In the following we will report into two separate sub-sections the results pertaining to either the single or multiple hidden layers settings.

2.3.1 Single hidden layer ($\ell = 3$)

In Figure 2.1 the performance of the inspected methods are compared for the minimal case study of a three layers network. The intermediate layer, the sole hidden layer in this configuration, is set to $N_2 = 500$ neurons. The accuracy of the different methods are compared, upon cutting at different percentile, following the strategies discussed in the Methods and compared with the benchmark model (the orange profile). In the benchmark model, the neural network is trained in direct space, by adjusting the weights of each individual inter-nodes connection. Then, the absolute value of the incoming connectivity is computed and used as an importance rank of the nodes' influence on the test accuracy (analogous to the way in which we use the eigenvalues). Such a model has been presented and discussed by He et al. in [32]. Following this assessment, nodes are progressively removed from the trained network, depending on the imposed percentile, and the ability of the trimmed network to perform the sought classification (with no further training) tested. We choose this particular type of trimming as a benchmark to our spectral pruning technique for the following reasons. First, it also amount to removing nodes from the collection, and not just sparsify the weight of the associated transfer matrices. Then, both approaches build on the concept of nodes ranking, as obtained from a suitable metric, which is respectively bound to direct vs. spectral domains. The abovementioned procedure is repeated 5 times and the mean value of the accuracy plotted in the orange curve of Figure 2.1. The shaded region stands for the semi dispersion of the measurements. A significant drop of the network performance is found when removing a fraction of nodes larger than 60 % from the second layer.

The blue curve Figure 2.1 refers instead to the post-processing spectral pruning based on the eigenvalues and identified, as method (ii), in the Methods Section. More precisely, the three layers network is trained by simultaneously acting on the eigenvectors and the eigenvalues of the associated transfer operators, as illustrated above. The accuracy displayed by the network trained according to this procedure is virtually identical to that reported when the learning is carried out in direct space, as one can clearly appreciate by eye inspection of Figure 2.1. Removing the nodes based on the magnitude their associated eigenvalues, allows one to keep stable (practically unchanged) classification performance for an intermediate layer that is compressed of about 70% of its original size. In this case the spectral pruning is operated as a post-processing filter, meaning that the neural network is only trained once, before the nodes' removal takes eventually place.

At variance, the green curve in Figure 2.1 is obtained following method (i) from the Methods Section, which can be conceptualized as a pre-training manipulation. Based on this strategy, we first train the network on the set of tunable eigenvalues, than reduce its size by performing a compression that reflects the ranking of the optimized eigenvalues and then train again the obtained network by acting uniquely on the ensemble of residual eigenvectors. The results reported in Figure 2.1 indicate that, following this procedure, it is indeed possible to attain astoundingly compact networks with unaltered classification abilities. Moreover, the total number of parameters that need to be tuned following this latter procedure is considerably smaller than that on which the other methods rely. This is due to the fact that only the random directions (the eigenvectors) that prove relevant for discrimination purposes (as signaled by the magnitude of their associated eigenvalues) undergoes the second step of the optimization. This method can also be seen as a similar kind of [18]. As a matter of fact, the initial training of the eigenvalues uncovers a sub-network that, once trained, obtains performances comparable to the original model. More specifically, the uncovered network can be seen as a *winning ticket* [18]. That is, a sub-network with an initialization particularly suitable for carrying out a successful training.

Next, we shall generalize the analysis to the a multi-layer setting ($\ell > 3$), reaching analogous conclusions.

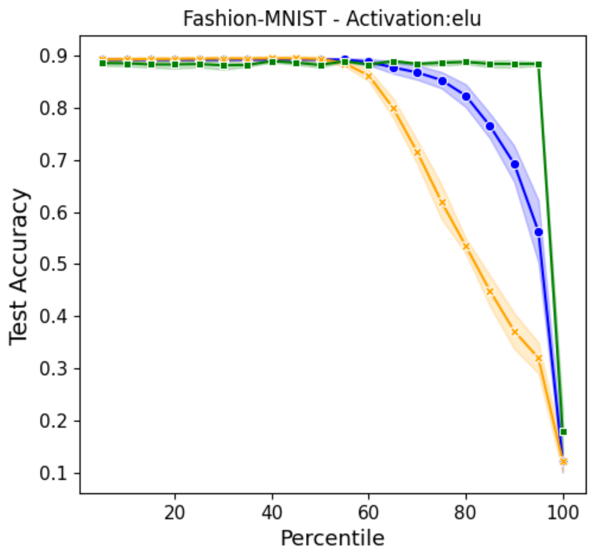


Figure 2.1: Accuracy on the Fashion-MNIST database with respect to the percentage of trimmed nodes (from the hidden layer), in a three layers feedforward architecture. Here, $N_2 = 500$, while $N_1 = 784$ and $N_3 = 10$, as reflecting the structural characteristics of the data. In orange the results obtained by pruning the network trained in direct space, based on the absolute value of the incoming connectivity (see main text). In blue, the results obtained when filtering the nodes after a full spectral training (post-training). The curve in green reports the accuracy of the trimmed networks generated upon application of the pre-training filter. Symbols stand for the averaged accuracy computed over 5 independent realizations. The shadowed region is traced after the associated semi-dispersion.

Multiple hidden layers ($\ell > 3$)

Quite remarkably, the results achieved in the simplified context of a single hidden layer network also apply within the framework of a multi-layers setting.
To prove this statement we set to consider a $\ell = 5$ feedforward neural network with ELU activation. Here, $N_1 = 784$ and $N_5 = 10$ as reflecting the specificity of the employed dataset. The performed tests follows closely those reported above, with the notable difference that now the ranking of the eigenvalues is operated on the pool of $N_2 + N_3 + N_4$ neurons that compose the hidden bulk of the trained network. In other words, the selection of the neuron to be removed is operated after a global assessment, i.e. scanning across the full set of nodes, without any specific reference to an a priori chosen layer.

In Figure 2.2 the results of the analysis are reported, assuming $N_2 = N_3 = N_4 = 500$. The conclusions are perfectly in line with those reported above for the one layer setting, except for the fact that now the improvement of the spectral pruning over the benchmark reference are even superior. The orange curve drops at percentile 20, while the blue begins its descent at about 60 %. The green curve, relative to the sequential two steps training, stays stably horizontal up to about 90 %.

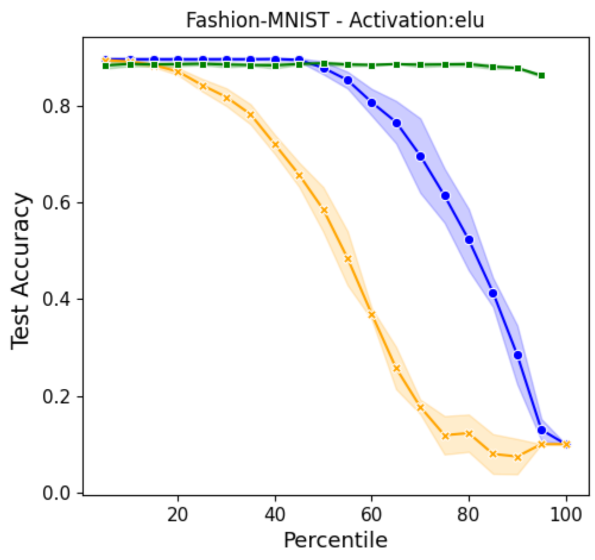


Figure 2.2: Accuracy on the Fashion-MNIST database with respect to the percentage of pruned nodes (from the hidden layers), in a five layers feedforward architecture. Here, $N_2 = N_3 = N_4 = 500$, while $N_1 = 784$ and $N_5 = 10$, as reflecting the structural characteristics of the data. Symbols and colors are chosen as in Figure 2.1.

2.4 Methods

We detail here the spectral procedure to make a trained network smaller, while preserving its ability to perform classification.

To introduce the main idea of the proposed method, we make reference to formula (1.2) and assume the setting where $\lambda^{(k)}_{m(j)} = 0$. The information travelling from layer k to layer $k+1$ gets hence processed as follows: first, the activity on the departure node j is modulated by a multiplicative scaling factor $\Phi^{(k)}_{l(i),m(j)}$, specifically linked to the selected (i, j) pair. Then, all incoming (and rescaled) activities reaching the destination node i are summed together and further weighted via the scalar quantity $\lambda^{(k)}_{l(i)}$. This latter eigenvalue, downstream of the training, can be hence conceived as a distinguishing feature of node i of layer $k+1$. Assume for the moment that $\Phi^{(k)}_{l(i),m(j)}$ are drawn from a given distribution and stay put during optimization. Then, every individual neuron bound to layer $k+1$ is statistically equivalent (in terms of incoming weights) to all other nodes, belonging to the very same layer. The eigenvalues $\lambda^{(k)}_{l(i)}$ gauge therefore the relative importance of the nodes, within a given stack, and as reflecting the (randomly generated) web of local inter-layer connections (though statistically comparable). Large values of $|\lambda^{(k)}_{l(i)}|$ suggest that node i on layer $k+1$ plays a central role in the economy of the neural network functioning. This is opposed to the setting when $|\lambda^{(k)}_{l(i)}|$ is found to be small. Stated differently, the subset of trained eigenvalues provide a viable tool to rank the nodes according to their degree of importance. As such, they can be used as reference labels to make decision on the nodes that should be retained in a compressed analogue of the trained neural network, with unaltered classification performance. As empirically shown in the Results section with reference to a variegated set of applications, the sorting of the nodes based on the optimized eigenvalues turns out effective also when the eigenvectors get simultaneously trained, thus breaking, at least in principle, statistical invariance across nodes.

As we will clarify, the latter setting translates in a post-training spectral pruning strategy, whereas the former materializes in a rather efficient pre-training procedure. The non linear activation function as employed in the training scheme leaves a non trivial imprint, which has to be critically assessed.

More specifically, in carrying out the numerical experiments here reported we considered two distinct settings, as listed below:

- (i) As a first step, we will begin by considering a deep neural network made of N neurons organized in ℓ layers. The network will be initially trained by solely leveraging on the set of tunable eigenvalues. Then, we will proceed by progressively removing the neurons depending on their associated eigenvalues (as in the spirit discussed above). The trimmed network, composed by a total of $M < N$ units, still distributed in ℓ distinct layers, can be again trained acting now on the eigenvectors, while keeping the eigenvalues frozen to the earlier determined values. This combination of steps, which we categorize as pre-training, yields a rather compact neural network (M can be very small) which performs equally well than its fully trained analogue made of N computing nodes.

- (ii) We begin by constructing a deep neural network made of N neurons organized in ℓ layers. This latter undergoes a full spectral training, which optimizes simultaneously eigenvectors and the eigenvalues. The trained network can be compressed, by pruning the nodes which are associated to eigenvalues (see above) with magnitude smaller that a given threshold. This is indeed a post-training pruning strategy, as it acts *ex post* on a fully trained device.

To evaluate the performance of the proposed spectral pruning strategies (schematically represented in the flowchart of Figure 2.3), we also introduced a reference benchmark model. This latter can be conceptualized as an immediate overturning of the methods in direct space. Simply stated, we train the neural network in the space of the nodes, by using standard approaches to the learning. Then, we classify the nodes in terms of their relevance using a proper metric to which shall make reference below, and consequently trim the nodes identified as less important. When adopting the spectral viewpoint, one can rely on the eigenvalues to rank the nodes importance. As remarked above, in fact, the eigenvalues at the receiver nodes set a local scale for the incoming activity, the larger the eigenvalue (in terms of magnitude) the more important the role played by the processing unit. As a surrogate of the eigenvalues, when anchoring the train in direct space, we can consider the quantity $\sum_{j=1}^{N_k} |w_{ij}|$, for each neuron i belonging to layer $k+1$, see also [32]. The absolute value prevents mutual cancellations of sensible contributions bearing opposite signs, which could incidentally hide the actual importance of the examined node.

In all explored cases, the pruning is realized by imposing a threshold on the reference indicator (be it the magnitude of the eigenvalues or the cumulated flux of incoming –and made positive– weights). Pointedly, the respective indicator is extracted for every node in the arrival layer. Then a percentile q is chosen and the threshold fixed to the q-th percentile. Nodes displaying an indicator below the chosen threshold are removed and the accuracy of the obtained (trimmed) neural network assessed on the test-set. The codes employed, as well as a notebook to reproduce our results, can be found in the public repository of this project [1].

2.5 Conclusions

The eigenvalues of the transfer operator that connects adjacent stacks in a multi-layered architecture provide an effective measure of the nodes importance in handling the information processing. By exploiting this fact we have introduced and successfully tested two distinct procedures to yield compact networks –in terms of number of computing neurons– which perform equally well than their untrimmed original homologous. One procedure (referred as (ii) in the description)

[1] `https://github.com/Buffoni/spectral_learning`

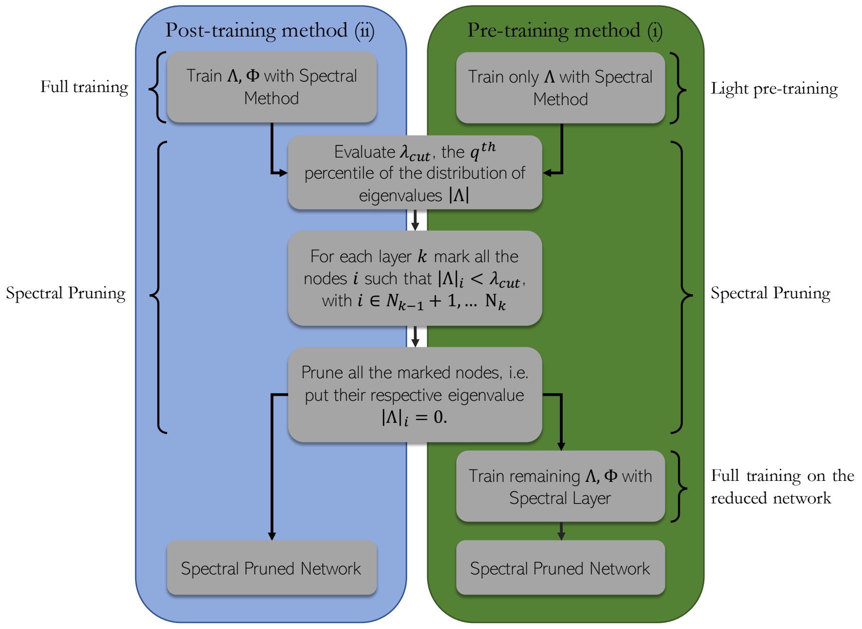

Post-training method (ii)
Pre-training method (i)
Full training
Train Λ, Φ with Spectral Method
Train only Λ with Spectral Method
Light pre-training
Evaluate λ_{cut}, the q^{th} percentile of the distribution of eigenvalues $|\Lambda|$
Spectral Pruning
For each layer k mark all the nodes i such that $|\Lambda|_i < \lambda_{cut}$, with $i \in N_{k-1}+1, \dots N_k$
Spectral Pruning
Prune all the marked nodes, i.e. put their respective eigenvalue $|\Lambda|_i = 0$.
Train remaining Λ, Φ with Spectral Layer
Full training on the reduced network
Spectral Pruned Network
Spectral Pruned Network

Figure 2.3: Flowchart of the pre- and post- speactral training pruning strategies as presented in section 2.4.

is acknowledged as a post processing method, in that it acts on a multi-layered network downstream of training. The other (referred as (i)) is based on a sequence of two nested operations. First the eigenvalues are solely trained. After the spectral pruning took place, a second step in the optimization path seeks to adjust the entries of the eigenvectors that populate a trimmed space of reduced dimensionality. The total number of trained parameters is small as compared to that involved when the pruning acts as a post processing filter. Despite that, the two steps pre-processing protocol yields compact devices which outperform those obtained with a single post-processing removal of the unessential nodes.

As a benchmark model, and for a neural network trained in direct space, we decided to rank the nodes importance based on the absolute value of the incoming connectivity. This latter appeared as the obvious choice, when aiming at gauging the local information flow in the space of the nodes, see also [32]. In principle, one could consider to diagonalizing the transfer operators as obtained after a standard approach to the training and make use of the computed eigenvalues to a posteriori sort the nodes relevance. This is however not possible as the transfer operator that links a generic layer k to its adjacent counterpart $k+1$, as follows the training performed in direct space, is populated only below the diagonal, with all diagonal entries identically equal zero. All associated eigenvalues are hence are zero and they provide no information on the relative importance of the nodes of layer $k+1$, at variance with what happens when the learning is carried out in the reciprocal domain.

Summing up, by reformulating the training of neural networks in spectral space, we identified a set of sensible scalars, the eigenvalues of suitable operators, that unequivocally correlate with the influence of the nodes within the collection.

This observation translates in straightforward procedures to generate efficient networks that exploit a reduced number of computing units. Tests performed on different settings corroborate this conclusions. As an interesting extension, we show in the appendix (B)n that a suitable regularization of the eigenvalues yields a general improvement of the proposed method.

With this chapter we conclude the general discussion aimed at introducing the background and application of the spectral parametrization of feedforward neural networks.We have shown how through such new training scheme (which acts on global parameters, eigenvalues and eigenvectors) fundamental contribution can be identified that hide inside the global structure of the network. These substructures (the nodes identified by the largest eigenvalues) emerge spontaneously during the training phase. Understanding what the fundamental structures are could be a the target for future investigations to shed light on the relevant explainability issue. However, as we will discover in the next two chapters, the spectral formulation can also be valuable in developing machine learning models that, albeit with a modest level of approximation, emulate biological decision-making processes. In the next chapter, we will begin by exploring the possibility of incorporating dynamics within the process of decision making that underlies the actual functioning of an artificial neural networks. Indeed, biological neural processes follow highly complex temporal dynamics and introducing temporal evolution in ANNs defines a fundamental step forward for the development of bio-inspired models. In the following two chapters we will delve into this direction. We will start by introducing a model called Recurrent Spectral Network (RSN) consisting of a discrete map that, thanks to constraints imposed in the spectral regime, naturally evolves towards a stable subspace where classification is carried out.

Chapter 3

Recurrent Spectral Networks - RSN

In this chapter a novel strategy to automated classification is introduced which exploits a fully trained dynamical system to steer items belonging to different categories toward 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.

3.1 Introduction

In the standard feedforward scheme the target of the optimization are the weights of the links that connect pair of nodes belonging to adjacent stacks of the multilayered arrangement. In the previous chapters an alternative training scheme has been proposed and explored which anchors the learning to reciprocal domain: the eigenvalues and the eigenvectors of the transfer operators get adjusted by the optimization.

Delving into the principles of the spectral methodology, we here propose a radically novel approach to computational deep 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 categorised. 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 [1]. 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, 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 [37]–[39]. 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,

[1]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.

with an independent dataset to be processed. This is at present arduous with standard approaches to deep learning, as the second learning stage causes the so-called catastrophic forgetting taking over any form of digital consciousness inherited from the first [40]–[42]. Few attempts have been so far reported which aim at overcoming this limitation [43]–[45].

In the next Section we intoduce 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 toward 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 [13]. 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.

3.2 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 [46]–[48]. 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:

$$\vec{x}^{(k+1)} = \vec{f}_k\left(\mathbf{A}\vec{x}^{(k)}\right) \tag{3.1}$$

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 $\lim_{k\to\infty} \vec{f}_k \to \vec{\mathbb{1}} \equiv (1, 1, ..., 1)^T$, in such a way that, for large enough k, the system approximately follows a linear update rule. This is achieved by setting:

$$\vec{f}_k(\cdot) = \vec{\mathbb{1}} + \frac{\vec{g}(\cdot)}{k^{\gamma}} \tag{3.2}$$

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 $\mathbf{A} = \mathbf{\Phi}\mathbf{\Lambda}\left(\mathbf{\Phi}\right)^{-1}$, by invoking spectral decomposition. Here, $\mathbf{\Lambda}$ is the diagonal matrix of the eigenvalues $(\lambda_1, \lambda_2, ..., \lambda_N)$. Working in the spectral domain enables us to enforce n-dimensional attracting subspace. To this end, we impose $\lambda_1 = \lambda_2 = ... = \lambda_n = 1$, and assume $|\lambda_i| < 1$ for $i > n$. These latter $N - n$ quantities are among the target of the optimization scheme. Moreover, we assume $\vec{\phi}_1$, $\vec{\phi}_2$,..., $\vec{\phi}_n$, 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 $k >> 1$, 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 toward 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 i-th element $(\vec{c}_{\bar{k}})_i$ 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 zeroes 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 [49]) 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 equation (3.2), 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.

3.3 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, centred at the origin, define the first class (displayed in yellow, in Figure 3.1). 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 Figure 3.1). 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 generated to belong to the square domain of width L - to its reference portion, as introduced above.

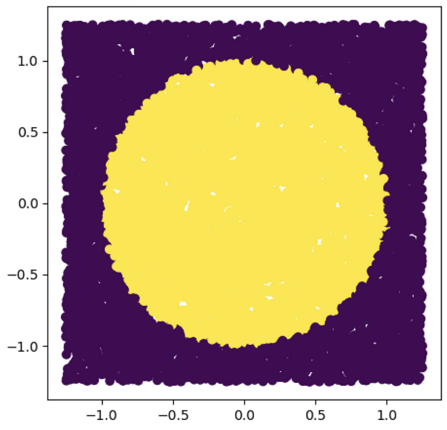


Figure 3.1: The dataset used as a validation test for the RSN scheme. Points populate two different regions, of equal relevance, separated by a sharp non-linear boundary, which we identify as the unitary circle.

For 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 optimisation. 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 Figure 3.2, 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.

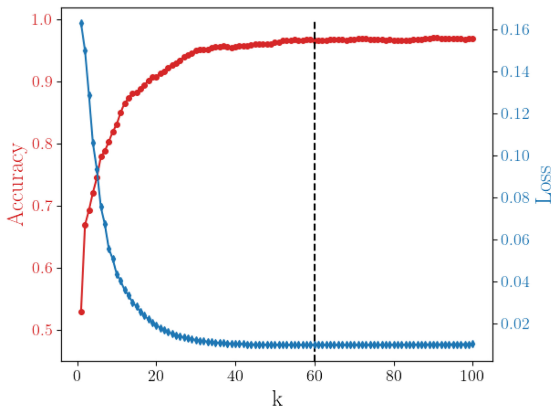


Figure 3.2: Accuracy (in blue) and loss (in red) against the iteration k, for a trained RSN with $\bar{k} = 60$ (vertical dashed line). Data refer to just one realization of the training procedure.

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 favours the eigen-direction $\vec{\phi}_2$. The coefficient c_2 should therefore prevail over the others. This scenario is confirmed by inspection of Figure 3.3, where c_1 and c_2 are plotted against the iteration number for data points falling respectively 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.

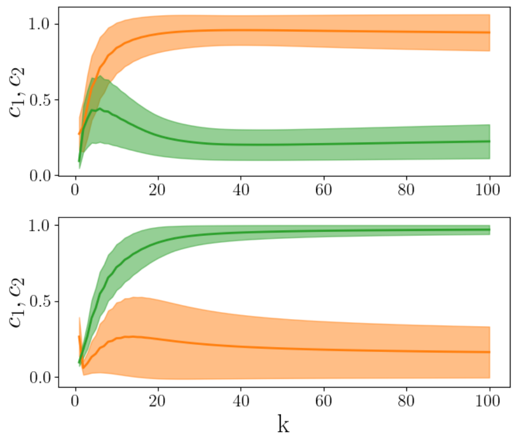


Figure 3.3: The evolution of the coefficients c_1 (orange) and c_2 (green) is plotted for points of the test set positioned respectively inside (top panel) and outside (lower panel) the unitary circle. The shadowed region points to the standard deviation of the collected signal when averaging over the population of supplied input, organized in groups which reflect their domain of pertinence.

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 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 under exam, we consider $\langle c_i \rangle$, the average of the i-th 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 Figure 3.4 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 Figure 3.4 we plot $\langle c_6 \rangle$, as an illustrative example as well as $\overline{\langle c \rangle} = \left(\sum_{i=3}^{10} \langle c_i \rangle \right) / 7$). 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 sepratrix. 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 disangaged, 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 Figure 3.5 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.

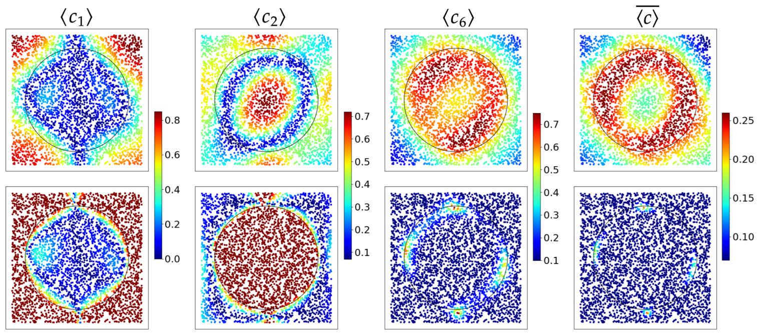


Figure 3.4: The quantities $\langle c_i \rangle$ for $i = 1, 2, 6$ and $\overline{\langle c \rangle}$ are plotted, for different (x, y), i.e moving on the plane of the initial condition. Top panels refer to $k_F = 5, k_I = 1$. Lower panel to $k_F = 50, k_I = 40$. The separatrix between the two considered 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 $\langle c_2 \rangle$.

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

3.4 Applying RSN to the MNIST dataset

As a further step in the analysis, we apply the RSN to the celebrated MNIST dataset of handwritten digits [13], which has been previously used in the preceding

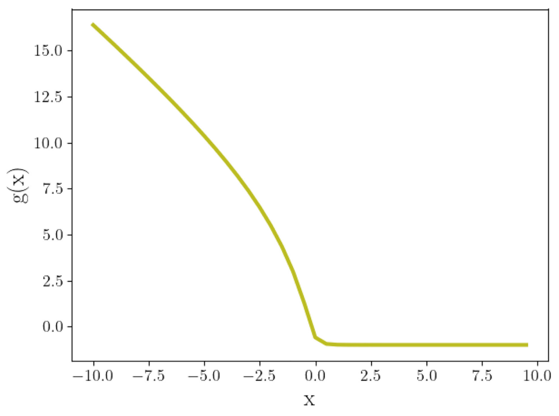


Figure 3.5: The non-linear function $g(x)$ as obtained from the regression neural model that is associated to each computing neuron of the RSN.

chapters. Each image of the dataset 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 Figure 3.6, 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 Figure 3.6 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 deep learning. Figure 3.7 illustrates the progressive

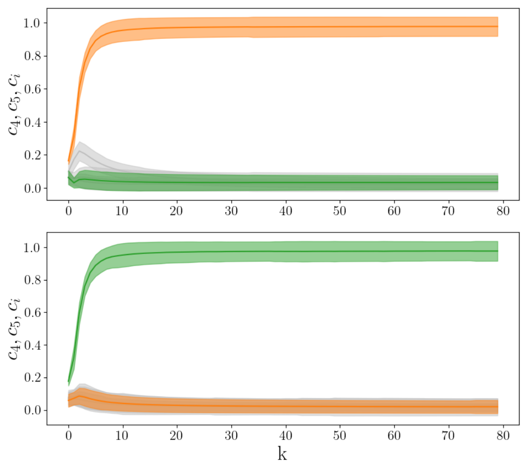


Figure 3.6: Top panel: the full set of handwritten four available in the test set is provided as an input to the trained RSN and the response monitored in terms of the obtained c_i, with $i = 1, ..., 10$. As expected, c_4 (orange) emerges and converges to unit, for $k > 10$ ($\bar{k} = 10$ being the 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.

convergence of the scheme, for two distinct exemplaries of input images. The RSN converges asymptotically 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 stylised 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,

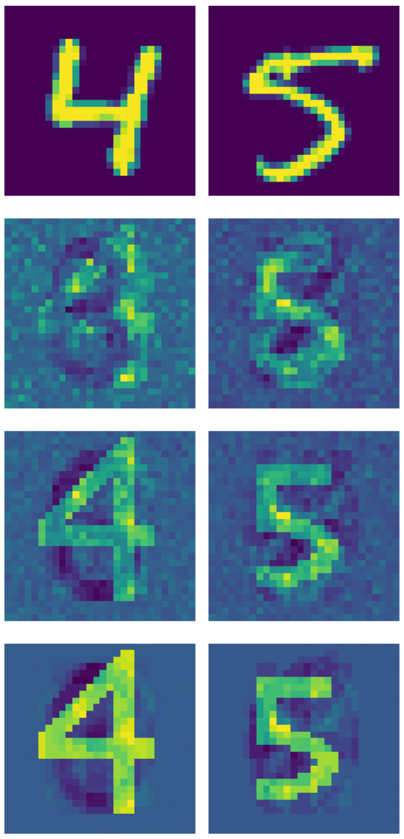


Figure 3.7: In each row we plot the activity on each node of the RSN, at different iterations and for two input numbers that belong to two distinct categories, respectively a four (left) and a five (right), see top panels. After a few iterations the RSN converges asymptotically to the eigenvectors $\vec{\phi}_4$ (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 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 [50]–[52]. 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 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 Figure 3.8, 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.

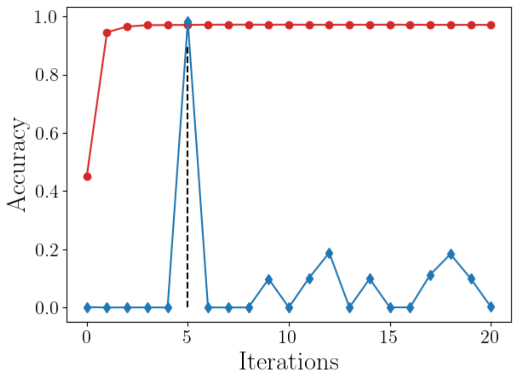


Figure 3.8: Evolution of the accuracy as computed on the test set of the MNIST dataset. Red symbols stand for a RSN model, trained at $\bar{k} = 5$ (black dashed line); blue symbols refer to a RNN, with $\bar{k} + 1$ consecutive layers, i.e. with $\bar{k}$ nested applications of the same $N \times N$ 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 $k > \bar{k}$.

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 Figure 3.9 - 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

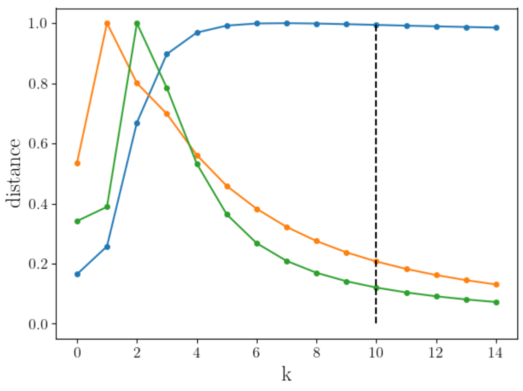


Figure 3.9: Euclidean distance (normalized to its maximum value) between the three sets of coefficients as described in the main body of the chapter and respectively referred to fours and fives, against the iteration index k. The orange curve refers to (10) coefficients, associated to modes with small 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 $\bar{k}$.

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.

3.5 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 [40]–[42].

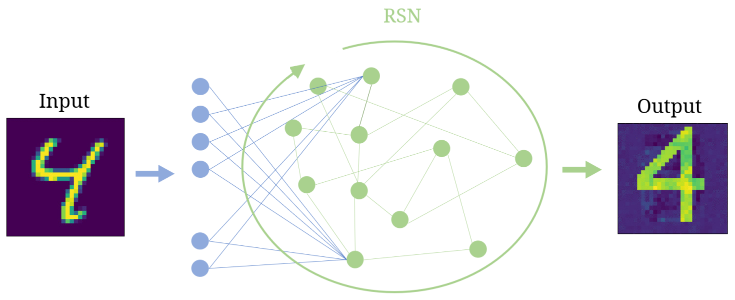


Figure 3.10: A schematic layout of the architecture employed to handle sequential learning. The information stemming from the image presented as an input are passed to the RSN, and therein iteratively elaborated until convergence to the deputed stationary solution (here exemplified as a stylized version of the input number).

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 Figure 3.10. 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 $\mathbf{\Phi}$:

$$\mathbf{\Phi} = \left(\begin{array}{c|c} \mathbf{\Phi}_{11} & \epsilon\mathbf{\Phi}_{12} \\ \hline \epsilon\mathbf{\Phi}_{21} & \mathbf{\Phi}_{22} \end{array}\right) \tag{3.3}$$

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 $\big(1, 1, 1, 1, 1, \lambda_6, ..., \lambda_{N/2}\big)$ and $\big(1, 1, 1, 1, 1, \lambda_{N/2+6}, ..., \lambda_N\big)$. 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:

- 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 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 Figure 3.11 (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 Figure 3.11 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 [40]–[42]. 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.

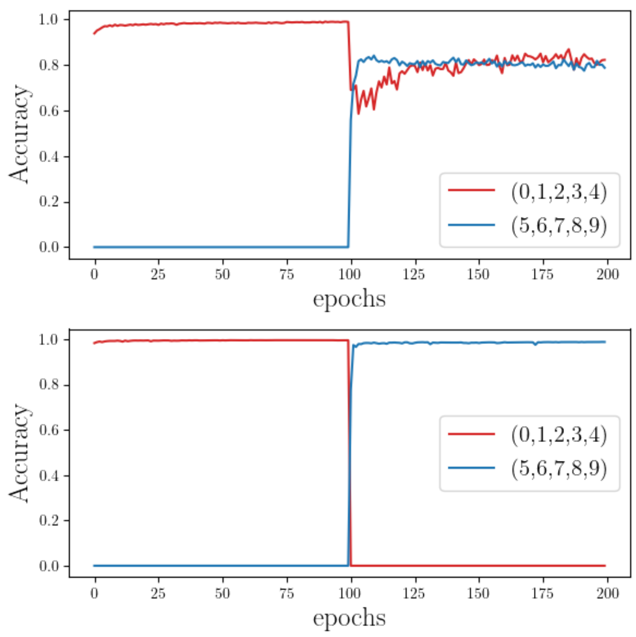


Figure 3.11: Top panel: accuracy against the epochs number for the RSN. The first 100 epochs refer to the RSN confronted with the task of classifying the images of the dataset made of digits from zero to four. Then, the second range of epochs, refers to the RSN while learning to classify numbers from 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 $\mathbf{\Phi}_{12}$ and $\mathbf{\Phi}_{21}$ 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 $N = 1000$.

3.6 Conclusions

In this chapter 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 deep 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 disfavour 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.

Chapter 4

Complex Recurrent Spectral Networks- $\mathbb{C}$-RSN

4.1 Introduction

In the previous chapter, we introduced a new strategy for machine learning in which the evaluation process is associated with the evolution of a dynamical system. The formulation of such strategy is possible thanks to the spectral parametrization of the adjacency matrix of a fully connected network. The constraints imposed in the parametrization cause the dynamics to forcibly evolve towards a vector subspace defined by certain eigenvectors locked in the spectral decomposition. During the learning process, the parameters are selected in such a way that, by using a given set of data as initial condition, the system collapses into one among the possible final states, specifically associated to the class the supplied data belongs to. By making use of this strategy some important properties have been observed. Firstly, the evaluation process appears to be independent of the number of steps (i.e. the system's integration time) taken during the training phase. In other words, the accuracy (or equivalently the measured loss) converges steadily to an asymptotic value which is indefinitely maintained across successive iterations, also beyond the limited horizon of the training. This property is not observed, for example, in standard Recurrent Neural Networks where the number of iterations is a fundamental parameter to be provided for a correct assessment of the data belonging to the test to be analyzed. Furthermore, as was discussed in the previous chapter, the problem of sequential learning can be successfully addressed through RSN . However, the RSN model predicts a static final state. This is at variance with what it happens for real biological systems, where dynamics plays a crucial role. Indeed, neurons involved in the examined process display a continuous activity over time. To try to overcome this critical issue, in this chapter we will briefly introduce a variant of the RSN model called Complex Recurrent Spectral Network ($\mathbb{C}$-RSN). The main new ingredients are:

- The non linearity is here localized on a subset of the nodes, call non-linear part.

- The fixed eigenvalues are complex value of the type $e^{2\pi i/T_i}$.
- The memory is confined in the linear part.
- The input is read in the non-linear part.

By introducing such new ingredients in the the model, as we shall see in detail in the next section, the system evolves towards a final state that oscillates in time. The final state will be a linear combination of the eigenvectors associated with the complex and fixed eigenvalues. The period of final state depends on the values of the constants T_i involved in the definition of the eigenvalues kept fixed. Moreover, the obtained shape of the emergent wavefront reflects the specific combination of eigenvectors which remains active in the asymptotic state. In other words, the complex eigenvalues constitute a basis of frequencies (or periods) on which the state can be written at very large times. Using these frequencies (assuming they are sufficiently large in number), it is possible to write a functions $f_c(t)$ by means of the activity of the neurons in a sub-portion of the network. Thus, it is possible to define a classification problem by asking the network to generate a specific function of time when the initial activity on the network corresponds to one of the data to be classified. As will be elaborated later, we have here considered the case where the function $f_c(t)$ is 'written' in a single neuron. Therefore, unlike in the RSN model, the information acquired during the evaluation process is stored in a minimal portion of the network, leaving the remaining part of the network free to perform other evaluation processes.

4.2 Neural network as a discrete map

The mathematical foundation of the ℂ-RSN is here presented. Let be $\tilde{f}$ the non-linear function involved in the process, W the weighted adjacency matrix of the fully connected neural network of size N and $\vec{x}$ the $\mathbb{R}^N$ vector describing the neurons activity.
The non linear function $\tilde{f}$ is defined as follows:

$$\tilde{f}(x_i) = \begin{cases} \tanh(x_i) & \text{if} \quad i \leq L \\ x_i & \text{if} \quad i > L \end{cases} \tag{4.1}$$

Thus, in the following we will refer to the sets of neurons $\mathcal{NL} = \{1, 2, \dots, L-1, L\}$ and $\mathcal{L} = \{L, L+1, \dots, N-1, N\}$ as the *non-liner* part and the *linear* part of the network.

The square matrix W is described by means of the spectral decomposition:

$$W = \Phi \Lambda \Phi^{-1} \tag{4.2}$$

where Λ is the diagonal matrix of the eigenvalues λ_i and the Φ is the basis of the decomposition. We introduce some constrains on the eigenvalues and organize them as follows:

$$\lambda_i = \begin{cases} e^{2\pi i/T_i} & \text{if} \quad i < M \\ \lambda_i \quad | \quad |\lambda_i| < 1 & \text{if} \quad i > M \end{cases} \tag{4.3}$$

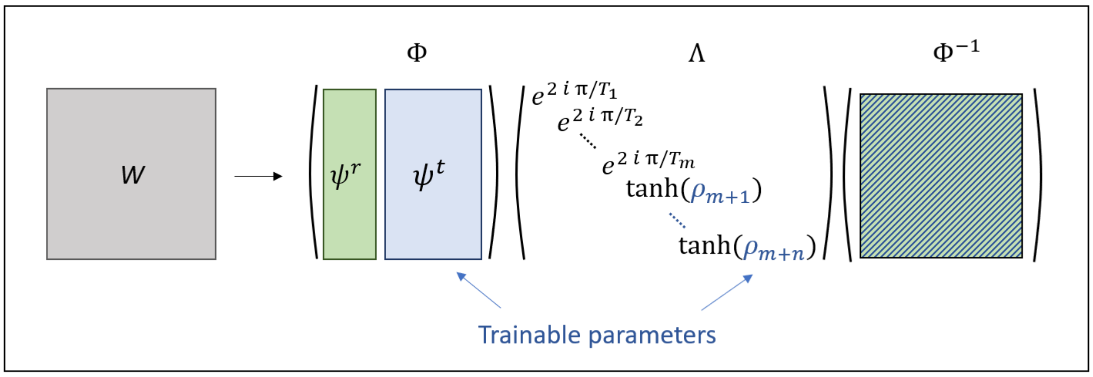


Figure 4.1: Spectral decomposition used to describe the linear transformation. The trainable parameters are highlighted in blue. The third matrix is the inverse matrix of the basis Φ and depend to both trainable elements and fixed elements of the matrix Φ

Namely, the first M eigenvalues are fixed to be equal to M complex values with modulus equal to 1 and a phase that depends on the parameter T_i, called *period*. The second set of eigenvalues are bounded to have a magnitude lower than 1.
The columns of the matrix Φ are the eigenvectors ψ_k of the decomposition. The first M eigenvectors are fixed and localized in the linear part of the network. In particular, for $k < M$, we set:

$$\psi_k(j) = \begin{cases} 1 & \text{if} \quad j \in N, N-k \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

That is, in each fixed eigenvector only two neurons are involved: the last one and one that depends on the specific eigenvector considered. For example, the last three fixed eigenvectors are:

$$\psi_{M-2} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad \psi_{M-1} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \quad \psi_{M} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}. \tag{4.5}$$

Finally, the Complex Recurrent Spectral Network ($\mathbb{C}$-RSN) is defined as the discrete map:

$$\vec{x}_{n+1} = \tilde{f}(\Phi\Lambda\Phi^{-1}\vec{x}_n) = \mathcal{M}(\vec{x}_n). \tag{4.6}$$

As for the RSN model, the trainable parameters of the ℂ-RSN architecture are the $N - M$ bounded eigenvalues and the relative eigenvectors. Since the constrain on the eigenvalues magnitude must be satisfied during every single step of the training procedure, the trainable eigenvalues λ_i are hereafter written as:

$$\lambda_i = \tanh(\rho_i) \qquad i = M+1, \dots, N, \quad \rho_i \in \mathbb{R}. \tag{4.7}$$

In the next section some properties of the forward evolution of the ℂ-RSN will be introduced.

4.3 Forward evolution of the map $\mathcal{M}$

Let call $\vec{x}_0$ the initial condition for the map $\mathcal{M}$ and $\vec{x}_t$ the vector obtained after t iterations of the discrete map (4.6). Assume the dynamics to be linear for the sake of argument. Under this condition, due to the constraints on the eigenvalues, the asymptotic dynamics collapses within the subspace defined by the first M eigenvectors. However, for some values of the trainable parameters, the effect of non-linearity can prevent the system to eventually converge towards the deputed asymptotic manifold, i.e. the subspace spanned by the eigenvectors associated to fixed (and complex) eigenvalues. It can be however shown that these latter unsuited behaviours occur on just few occasions over a large sample of statistically equivalent realizations of the training. Furthermore, as will be clearer later on, these solutions that we might call divergent with am abuse of language are unfavorable for the loss minimization problem and they are not explored during the training phase. In other words, although divergent solutions are theoretically possible, the typical range of parameter values and the learning process prevent such solutions from emerging. For this reason, from now on we will consider the dynamics to evolve iteration by iteration towards the final subspace spanned by the fixed eigenvectors.

Suppose the dynamics after t^* iterations be confined in the subspace of the first M eigenvectors, so that:

$$\vec{x}_{t^*} = \sum_{k=1}^{M} \alpha_k \psi_k. \tag{4.8}$$

Since the form of the eigenvectors described by (4.4) and the non-linearity (4.1), the dynamics on the subspace is linear and the activity is forced to remain confined in the subspace. In particular after a new application of the map $\mathcal{M}$, the activity become:

$$\vec{x}_{t^*+1} = \sum_{k=1}^{M} \alpha_k e^{2\pi i/T_k} \psi_k. \tag{4.9}$$

and after t more iterations:

$$\vec{x}_{t^*+t} = \sum_{k=1}^{M} \alpha_k (e^{2\pi i/T_k})^t \psi_k = \sum_{k=1}^{M} \alpha_k e^{2\pi i t/T_k} \psi_k. \tag{4.10}$$

The coefficients α_k are in general complex values due to the presence of the complex eigenvalues in the decomposition. Let us recall that the training parameters are hidden into the definition of the coefficients α_k. In fact, depending on the specific selection of the underlying trainable parameter (the element of the free eigenvectors and the associated eigenvalues) the system will generate different evolution patterns. This will leave an indirect mark in the coefficients of the recorded activity as expressed in the reference basis of the explored manifold. To keep track of this inherent dependence we make explicit in α_k the dependence of a vector of trainable quantities that we shall label $\vec{\beta}$. Namely:

$$\alpha_k(\vec{\beta}) = r(k, \vec{x}_0, \vec{\beta}) + \mathrm{i}c(k, \vec{x}_0, \vec{\beta}), \tag{4.11}$$

where it has been made explicit the fact that the coefficients depend also on the initial conditions $\vec{x}_0$. Thus, the equation (4.10) rewrites as:

$$\begin{aligned}\vec{x}_{t^*+t} &= \sum_{k=1}^{M} \alpha_k(\vec{\beta}) e^{2\pi\mathrm{i}t/T_k}\psi_k = \sum_{k=1}^{M}(r(k, \vec{x}_0, \vec{\beta}) + \mathrm{i}c(k, \vec{x}_0), \vec{\beta}) e^{2\pi\mathrm{i}t/T_k}\psi_k = \\ &= \sum_{k=1}^{M} \big(r(k, \vec{x}_0, \vec{\beta}) + \mathrm{i}c(k, \vec{x}_0, \vec{\beta})\big)\bigg(\cos(\frac{2\pi t}{T_k}) + \mathrm{i}\sin(\frac{2\pi t}{T_k})\bigg)\psi_k.\end{aligned} \tag{4.12}$$

By remembering the chosen form of the eivengectors and in particular that $\psi_k(N) = 1 \; \forall \; k \in \{1, .., M\}$, the N-*th* component of $\vec{x}_{t^*+t}$ is:

$$\begin{aligned}\vec{x}_{t^*+t}(N) &= \sum_{k=1}^{M} \big(r(k, \vec{x}_0, \vec{\beta}) + \mathrm{i}c(k, \vec{x}_0, \vec{\beta})\big)\bigg(\cos(\frac{2\pi t}{T_k}) + \mathrm{i}\sin(\frac{2\pi t}{T_k})\bigg)1 = \\ &= \sum_{k=1}^{M} \bigg(r(k, \vec{x}_0, \vec{\beta})\cos(\frac{2\pi t}{T_k}) - c(k, \vec{x}_0, \vec{\beta})\sin(\frac{2\pi t}{T_k})\bigg) + \\ &\qquad + \mathrm{i}\bigg(c(k, x0, \vec{\beta}) cos(\frac{2\pi t}{T_k}) + r(k, \vec{x}_0), \vec{\beta}\sin(\frac{2\pi t}{T_k})\bigg).\end{aligned} \tag{4.13}$$

Focusing on the real part:

$$\mathcal{R}(t, \vec{x}_0, \vec{\beta}) = \mathrm{Re}\left(\vec{x}_{t^*+t}(N)\right) = \sum_{k=1}^{M} \bigg(r(k, \vec{x}_0, , \vec{\beta})\cos(\frac{2\pi t}{T_k}) - c(k, \vec{x}_0, \vec{\beta})\sin(\frac{2\pi t}{T_k})\bigg) \tag{4.14}$$

where $\vec{\beta}$ stands for the trainable parameters of the model. This latter equation will be used in the definition of the loss as explained in the following.

4.4 Classification task

Let be D a data set made by couples $(\vec{x}_0, \hat{y})_d$ with $d = 1, ..., |\mathcal{D}|$. The vector $\vec{x}_0$ is the input vector and $\hat{y} \in \{1, .., C\}$ is the associated label. Different labels refer to different classes of the problem. Let now introduce C discrete functions of time $f_c(t)$ one for each class of the problem. A classification task is formulated by asking the network to minimize the following loss function:

$$\begin{aligned} \mathcal{L} &= \sum_{d \in \mathcal{D}} \mathcal{L}(\vec{x}_0^{\,d}, \vec{\beta}) \\ \text{where} \quad \mathcal{L}(\vec{x}_0^{\,d}, \vec{\beta}) &= \sum_{t=0}^{T} |\mathcal{R}(t, \vec{x}_0^{\,d}, \vec{\beta}) - f_{\hat{y}^d}(t)|^2. \end{aligned} \tag{4.15}$$

The minimization is operated by acting on the trainable spectral parameters β introduced in the section (4.2). In other words, we ask the network to learn how reproduce the sought target function of time on one neuron (the last one) during a time window of length T when an input is used as initial condition of the dynamics. In the following, we will show the results obtained by working with the already presented MNIST dataset [13].

4.5 Results

To each image is associated a label corresponding to the class of the image, i.e. the digit shown in the image. We trained a ℂ-RSN network of size $N = 1000$. Te non-linearity (4.1) acts only on the first $L = 800$ neurons, while the number of fixed eigenvectors and eigenvalues was set to 5. The model was trained to reproduce the corresponding target discrete function of time in a 20 steps long time window, during which the loss is computed. Before that window, the network is left to evolve for 10 time steps starting from an initial condition which is shaped by the selected image. More specifically, any item from the MNIST data base is properly normalized so to have a signal to lay in the range $[0, 1]$ pixelwise. The obtained information is poured on a selected subset of nodes belonging to the portion of the network which decorated with the inclusion of the non linear processing units. During the first ten steps the network carries out the classification by separating the dynamics resulting from initial conditions belonging to different classes in the corresponding final states.

In figure 4.2 the discrete time function $f(t)$ produced as an output in the last neuron is reported in blue. The red line represents the target discrete time function for the input class. The two curves are very close: the model has successfully learned to reproduce the target function. Note how the two curves are close even after the window in which the loss is calculated (indicated in yellow in the figure). Furthermore, the two curves begin to get closer even before this time window. Both of these observations are in agreement with what was found in the RSN model presented in [4]. In particular, the fact that the curves remain close for successive time steps is a direct consequence of the stability of the final subspace.

It is possible to calculate the accuracy of the model by evaluating for each input which of the possible target functions is closest (according to the distance

L2) to the function observed in the last neuron. For the MNIST dataset, we found an accuracy on the test set of 0.9784.

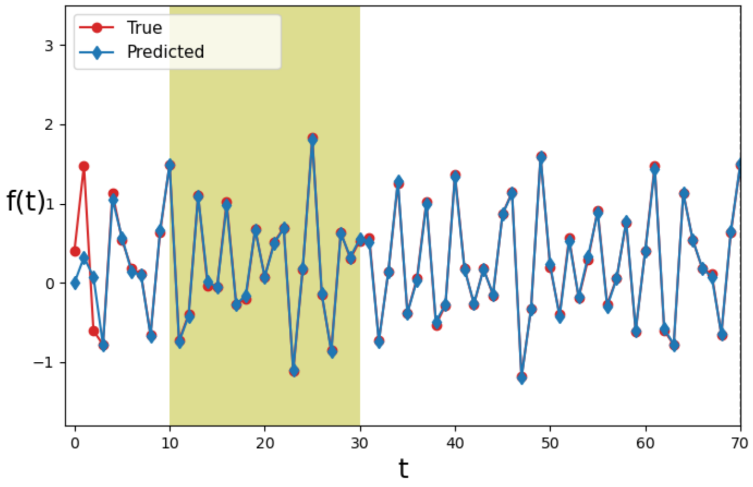


Figure 4.2: Activity recorded in the last neuron (blue line) compared with the target temporal function (red line). In yellow the time window during which the loss is calculated. The example shown refers to an input belonging to the class "1".

4.5.1 The role of the basis

Looking at equation (4.14) it is clear that the function observed in the last neuron is a linear combination of sinusoidal functions characterized by the periods T_i. Recall that each of the fixed eigenvectors possesses only two unit entries (the other being set to zero). They all share one element identically equal to unit which sits on position N. The other non trivial entry is located in different positions depending on the considered eigenvector. On this latter node, the recorded signal will be purely sinusoidal - by definition - with a characteristic period which reflects the chosen value of T_i in the definition of the associated eigenvalue. The amplitude of the sinusoidal wave are an indirect measure of the relative importance of the selected mode in the emerging time resolved pattern as displayed on node N. In other words, the parameters T_i introduced in (4.3) select some elements from the Fourier basis that define the basis the network can use to reproduce the signal in the last neuron. In figure (4.3) the network is visualized in two different phases of the dynamics: the initial condition and a successive time that belong to the finite time windows where loss evaluation is carried out. The activity starts in the non-linear part of the network (panel A) and flows to the linear part after some iterations. After enough iterations the activity is completely confined in the linear part (panel B). The real part of the signal in the last neuron reproduces well the target time function while the activities in the neurons involved in the fixed eigenvectors are characterized by sinusoidal activity as we expected.

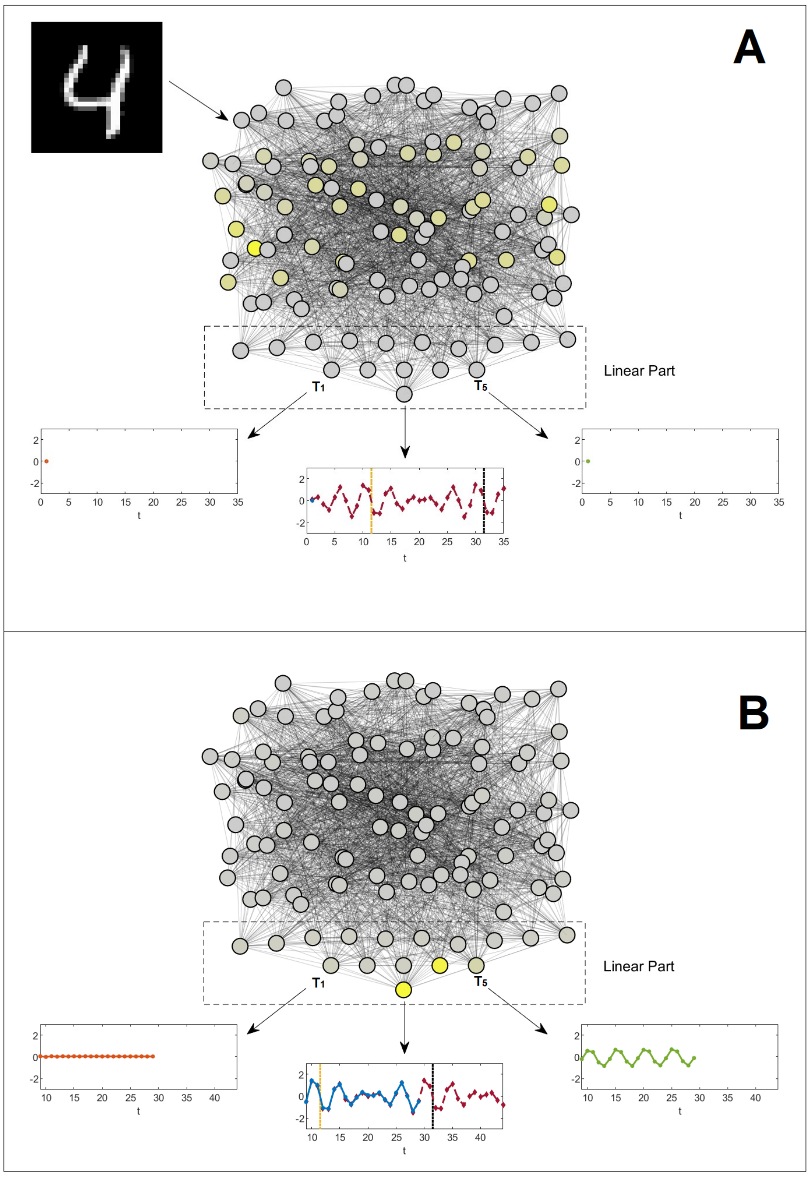


Figure 4.3: Visual representation of the ℂ-RSN network in two different moments of the dynamics. To simplify the representation, not all the nodes are represented: each of the other nodes represents a group of nodes, except for the last node and the five nodes involved in the fixed eigenvectors. The activity on the last neuron is represented by the blue line in the bottom sub-panel, while the target function is shown in red in the same sub-panel. The non trivial entries of the vectors that shape the attracting manifold are arranged as a linear horizontal layer (just above the neuron where the final signal is displayed) in the scheme depicted above. The two sub-panels on the right and on the left show the activity of two of them, characterized by the periods $T_1 = \infty$ and $T_5 = 5$ respectively. Panel A refers to the initial condition: the input data enters the network as activity of the neurons in the non-lnear part. During this phase the neurons belonging to the linear part are not activated. In panel B a snapshot of the network state during the loss window is shown. In that phase the activity is entirely confined to the linear part. The activity on the last neuron is in accordance with the target function. The activities on the two selected neurons are sinusoidal with amplitude corresponding to the coefficients of the target function when decomposed in the Fourier basis.

4.5.2 Multiple evaluations

$\mathbb{C}$-RSN networks are able to manage multiple input data sequentially, without the processing of the new data influencing and disturbing the dynamics relative to the first data. This is possible provided that the second input data is read after a sufficiently long time to ensure that the dynamics of the first data is already collapsed into the stable subspace. Indeed, as we demonstrate in the following, under that hypothesis the two dynamics evolve separately.

Let introduce a new formalism and split the activity vector $\vec{x}(t)$ in two components: one involving the elements belonging to the linear part of the network, and one for the elements of the non-linear part:

$$\vec{x}(t) = \vec{x}^{\,nl}(t) + \vec{x}^{\,l}(t) \tag{4.16}$$

where:

$$\vec{x}^{\,nl}(t) = \begin{pmatrix} x_1 \\ \vdots \\ x_L \\ 0 \\ \vdots \\ 0 \end{pmatrix} \qquad\qquad \vec{x}^{\,l}(t) = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ x_{L+1} \\ \vdots \\ x_N \end{pmatrix}. \tag{4.17}$$

Under this new formalism, the action of the function $\tilde{f}$ is:

$$\tilde{f}(\vec{x}(t)) = \tilde{f}(\vec{x}^{\,l}(t) + \vec{x}^{\,nl}(t)) = \vec{x}^{\,l}(t) + \tanh(\vec{x}^{\,nl}). \tag{4.18}$$

Let $\vec{w}(t)$ be the activity vector after t iterations of the map $\mathcal{M}$ starting from the initial condition $\vec{w}_0$, where $t >> \bar{t}$ and $\bar{t}$ is the last time step during which the loss is computed. As we explained in section (4.3), $\vec{w}(t)$ is confined in the subspace defined by the fixed eigenvectors:

$$\vec{w}(t) = \sum_{k=1}^{M} \gamma_k \psi_k = \vec{w}^{\,l}(t), \tag{4.19}$$

where we have made explicit the fact that the final state includes only neurons of the linear part. Moreover, eq. (4.9) tells us that also $\vec{w}(t+1)$ is confined in the same subspace and in particular:

$$\vec{w}(t+1) = \Phi\Lambda\Phi^{-1}\vec{w}(t) = \vec{w}^{\,l}(t+1) \tag{4.20}$$

We now introduce a new activity vector $\vec{y}(t)$ which is added to the vector $\vec{w}(t)$. The activities of the network neurons are now described by the sum vector $\vec{s}(t) = \vec{w}(t) + \vec{y}(t)$. The action of the map $\mathcal{M}$ on the sum vector is:

$$
\begin{aligned}
\mathcal{M}(\vec{w}(t)+\vec{y}(t)) &= \tilde{f}(\Phi\Lambda\Phi^{-1}(\vec{w}(t)+\vec{y}(t))) = \\
&= \tilde{f}(\Phi\Lambda\Phi^{-1}\vec{w}(t)+\Phi\Lambda\Phi^{-1}\vec{y}(t)) = \\
&= \tilde{f}(\vec{w}\,^{l}(t+1)+\Phi\Lambda\Phi^{-1}\vec{y}(t)) = \\
&= \vec{w}\,^{l}(t+1)+\tilde{f}(\Phi\Lambda\Phi^{-1}\vec{y}(t)) = \\
&= \vec{w}(t+1)+\mathcal{M}\vec{y}(t) = \vec{w}(t+1)+\vec{y}(t+1).
\end{aligned}
\tag{4.21}
$$

So, the dynamics of the vectors $\vec{w}(t)$ and $\vec{y}(t)$ evolve separately. This is a direct consequence of the fact that the evaluation of the input data leads to a state of the network that evolves while remaining confined in a subspace involving only neurons belonging to the linear part of the network.

This property can be used to evaluate and classify different inputs sequentially by working with a trained ℂ-RSN on a specific dataset. In fact, once the evaluation process of a first input has converged, a second input can be read by the ℂ-RSN model which will be classified by the network independently of the first input. The global final state of the network can be read in the real part of the activity on the last neuron as a function of time. The resulting time function will be the linear superposition of the two functions associated with the corresponding classes of the inputs, shifted by a phase that depends on the time distance elapsed between the reading of the two inputs. In formulae, calling t_1 and t_2 the entry times of the two inputs, the real part of the activity in the last neuron is:

$$
\mathcal{R}(t) = f_{c1}(t) + f_{c2}(t+\gamma) \qquad \text{for t } >> t_1, t_2, \tag{4.22}
$$

where f_{c1} and f_{c2} are the target functions for the inputs classes and γ is the time shift between the two entries. A sequential evaluation example involving the MNIST data set is represented in the figure (4.4). Four different frames of the network evolution are shown. In the panel A ($t = 0$) a first input is put in the non-linear part of the network. The dynamics evolves following the rule explained in section (4.3) and after some time steps ($t = 39$) it collapses into the linear part showing in the last neuron a temporal evolution that mimic the target function (panel B). At $t = 100$ a new input enters the network (panel C). The global dynamics collapses again into the linear part of the network while the activity on the last neurons converges to the linear combination of the two target functions as described by the eq. (4.22). So, by using a ℂ-RSN previously trained to classify separately the elements of a dataset, it is possible to evaluate different inputs sequentially. The final information obtained allows to identify both the two classes of the inputs seen and the time shift separating the entries. This information is completely stored in the simple function of time obtained in the last neuron. It is important to note that this property is automatically obtained from the properties of the ℂ-RSN model and does not require further adjustments during the training phase.

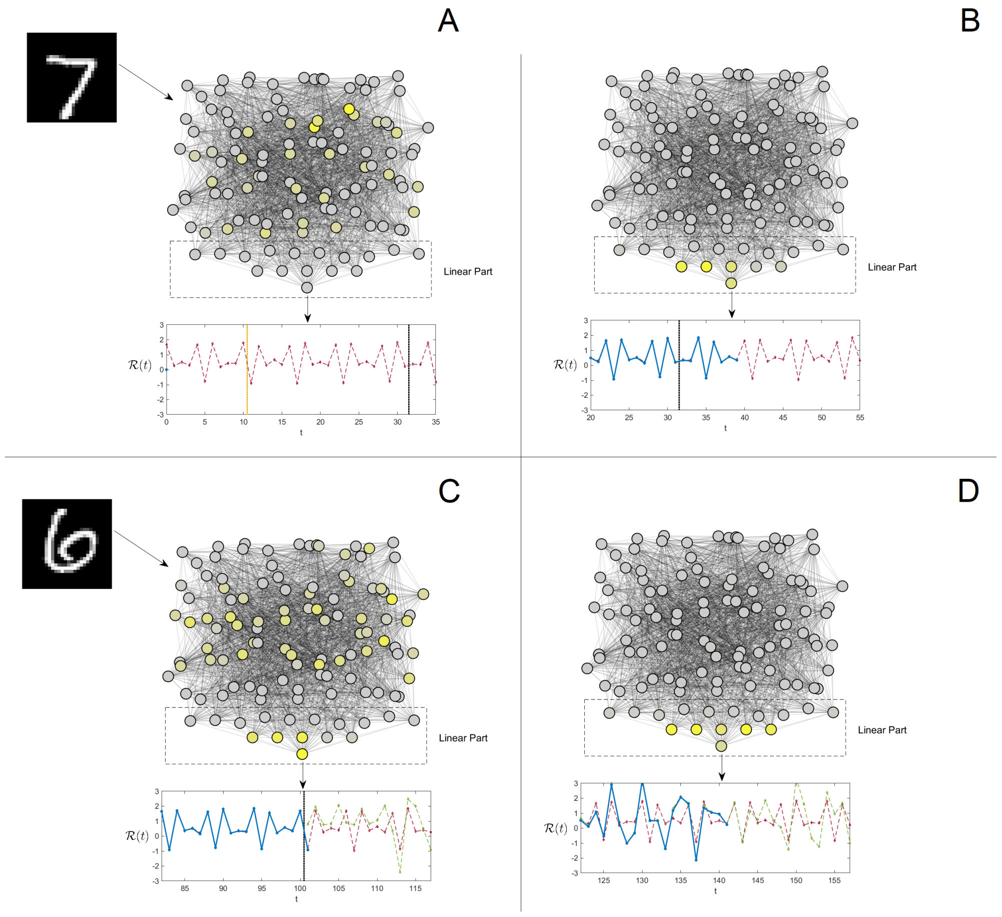


Figure 4.4: Four different steps of the evolution of a ℂ-RSN trained on the MNIST dataset. Panel A shows the step of entry of the first data. Panel B shows a subsequent step in which the dynamics and evaluation process of the data reached convergence. The sub-panel shows how the real part of the activity recorded in the last neuron (blue line) is in perfect agreement with the target function (red line). Panel C shows the moment at which a second input is inserted into the network without the residual activity related to the first data point being removed. Finally, panel D shows how the global dynamics (the residual one and the one due to the new input) collapses again towards the linear part of the network. In particular, the wave front observed in correspondence of the last node (or entry of the eigenvectors) results from a linear combination of the two selected target function. Moreover, and surprisingly enough, it also keep memory of the time delay between the successive insertion of the two examined images.

4.6 Conclusion

Artificial neural network models have historically been inspired by biological observations of how the human brain works. However, these tools show huge differences with current neuroscience models and experimental observations. First of all, brain is a dynamical entity, and different processes are ultimately shaped by a time resolved evolution of different neuronal populations. In the last two chapters we have introduced two new artificial neural network models that mimic some of the behaviors seen in nature. In the previous chapter the RSN model was presented in detail. This latter model,constitutes a first step in the direction of creating bio-inspired models but it presents some critical issues. First of all, it is necessary to define an initial time after which the nonlinearity begins to vanish. A second criticality is due to the fact that the final state appears to be a stable and charatcreized by a constant activity level. At variance in nature we observe continuous dynamics that never converges to a constant state. To overcome these critical issues in this chapter we have introduced an extension of the RSN model as discussed in chapter 3 called Complex Recurrent Network ($\mathbb{C}$-RSN). First of all, in this new version the non-linearity is confined in space (i.e., in a subset of nodes), rather than vanishing over time. That is, while the RSN model has a time dependence converging towards a linear model, the $\mathbb{C}$-RSN does not exhibit this dependence and the model remains unchanged over time. Similarly to wha introduced in chapter 3 and in [4] for the RSN model, in the $\mathbb{C}$-RSN the adjacency matrix that defines the interactions between neurons is described by means of a spectral decomposition. The set of trainable parameters of the model consists of some of the eigenvectors and their related eigenvalues, the latter constrained to have a modulus less than one. The other eigenvectors and eigenvalues are fixed and are not modified during the training phase. In particular, the subset of fixed complex eigenvalues display unitary modulus. Each of the fixed eigenvalues is associated with a particular frequency. The structure of the fixed eigenvectors a the constraints on the eigenvalues mean that once the dynamics reaches the subspace described by these eigenvectors, it remains confined. Furthermore, these eigenvectors share the last neuron of the network which can be used to write a signal as a weighted sum of sinusoidal functions with frequencies equal to those involved in writing the fixed eigenvalues. Thus, a classification problem can be defined by asking the network to learn to reproduce on the last neuron a different temporal activity for different classes of input. The model was then tested with the MNIST dataset showing excellent accuracy. The ability to classify remains unchanged over time as the system reaches a stable steady state. Furthermore, once the model has been trained on a dataset it can be used to classify multiple inputs sequentially. The final activity read on the last neuron contains information on the classes of the different inputs read and the temporal distance elapsed between two successive inputs.

With this chapter we conclude the part of the thesis relating to bio-inspired models. In the next chapters we will see how the neural networks that we have studied for the development of the theoretical results presented so far can be applied to different research areas. In particular, in this manuscript we will present two different applications: one in the epidemiology and the other concerning the analysis of the internal morphology of volcanoes.

Chapter 5

Mobility-based prediction of SARS-CoV-2 spreading

From March 2020, for years the rapid spreading of SARS-CoV-2 and its dramatic consequences were forcing policymakers to take strict measures in order to keep the population safe. At the same time, societal and economical interactions were to be safeguarded. A wide spectrum of containment measures have been hence devised and implemented, in different countries and at different stages of the pandemic evolution. Mobility towards workplace or retails, public transit usage and permanence in residential areas constitute reliable tools to indirectly photograph the actual grade of the imposed containment protocols. In this chapter, taking Italy as an example, we will develop and test a deep learning model which can forecast various spreading scenarios based on different mobility indices, at a regional level. We will show that containment measures contribute to "flatten the curve" and quantify the minimum time frame necessary for the imposed restrictions to result in a perceptible impact, depending on their associated grade.

5.1 Introduction

Machine Learning (ML) [53], [54] has been extensively employed in the context of time series modeling and forecasting [55]. Groundbreaking applications in natural language processing [56], financial forecasting [57], speech recognition [58] have earned this particular subfield of ML lots of investments and attention. Notably, the use of Deep Neural Networks [46], [47], with respect to traditional approach to time series analysis, enabled the algorithm itself to learn from the data the relevant variables and their associated correlations. Following the rapid spreading of SARS-CoV-2, numerous attempts have been made for predicting the time evolution of epidemics across different spatial scales [59]–[61]. To this end ML techniques have been also employed [62]–[64]. Although very accurate and useful, these models often lack the ability to incorporate the effects of containment measures as implemented by local governments and solely rely on selected epidemiological variables (e.g. number of tests performed, number of deaths) to predict the spreading of the virus. The putative impact of different containment strategies as devised by local governments is hence customarily modeled by resorting to standard epidemiological tools [65], [66],a choice which potentially limits the predictive ability of the trained ML devices. Starting from these premises, we suggest that mobility indices provide solid, almost real-time, indicators of the implemented containment strategies. When included in the training, they are processed as key information for future forecasting of ML algorithm. A self-consistent argument allows in turn to estimate the time it takes for the imposed mobility restrictions to materialize in an effective drop of the curve of infected individuals.

In the following, we will describe the adopted machine learning approach which is tailored to predicting the SARS-CoV-2 epidemic evolution in the twenty regions of Italy [1]. The model is trained by using the time series of selected epidemic quantities (number of infections, number of death, etc..) and includes information on the population mobility. We will show that, by looking at epidemic and mobility trends during the n_p past days, the model is able to return sensible information on the values of a target epidemiological parameter in the next n_f days. Working in the proposed framework, we are also able to estimate the time needed for the imposed restriction to yield consequences that can be appreciated at the scale of the whole community in terms of reduction of hospitalized individuals. To this end, we consider different grades of imposed restrictions on individual mobility ranging from a complete, nationwide lockdown to milder, regional-level restrictions to virtually no restrictions at all.

5.2 Methods

5.2.1 Architecture

We worked with Recurrent Neural Networks (RNN) [67], a class of deep learning architectures widely used in time series machine learning modeling. Such architecture is designed to be sensitive to the ordering of the elements in the input sequence [67]. This is achieved by introducing an inner state vector that is updated by the network itself, during each successive iteration. This latter vector allows the network to "keep memory" of the past input values. RNNs suffer of the so-called vanishing gradient problem: the gradients in later steps of the sequence fade away quickly in the backpropagation process, without reaching earlier input signals and thus making it hard for the RNN to apprehend and correctly incorporate long-range dependencies [67]. To oppose this problem, gating-based architectures, such as the Long short-term memory (LSTM), have been proposed [68]. Trainable vectors, called gates, are accommodated for in the architecture and control the inner state update, at each iteration. This technical solution makes it possible for the network to "forget" or "store" the novel bits of information that are processed at each time step, along the sequence of collected

[1]Valle d'Aosta, Piemonte, Lombardia, Trentino - Alto Adige, Veneto, Friuli Venezia Giulia, Liguria, Emilia Romagna, Toscana, Marche, Umbria Lazio,Abruzzo, Molise, Campania, Apulia, Basilicata, Calabria, Sicilia e Sardegna

events. In this way, early information deemed crucial for handling the forecasting task can be stored in the bulk while, recent inputs, identified as unessential, are safetly removed from the memory kernel. This is precisely the reason why we have decided to employ a LSTM-like architecture for the problem at hand. In the following we shall operate with a deep architecture composed by two LSTM hidden layers of 300 and 20 nodes, respectively. Moreover, an additional dense layer is introduced to produce the sought output. Further, use is made of Adam [69] optimizer with a learning rate of 0.0005. The batch size is set to 100 and the number of epochs during the learning procedure is assumed equal to 100. We hand picked these hyper-parameters without any ad-hoc optimization.

5.2.2 Data set

The dataset consists in discrete daily series of length T of selected epidemic and mobility parameters for each of the 20 regions in Italy. More specifically, we focus on the following quantities: (i) number of patients in *intensive care* (ii) number of *hospitalized* patients (iii) number of patients in *home isolation* (iv) number of *deaths*. Data from the COVID-19 Community Mobility Reports of Google [70] are employed to track the change in time of the degree of mobility, as associated to different regions of Italy. We calculate in particular the evolution as percentile change from baseline values[2] of the reported mobility indexes in the following areas:

1. retail and recreation
2. grocery and pharmacy
3. parks
4. transit stations
5. workplaces
6. residential

In Fig. 5.1 the evolution of the reference mobility indicators are displayed for the case of Lombardy. The impact of the imposed restrictions on the mobility indexes can be clearly appreciated by visual inspection of the depicted global trends. It is hence surmised that the aforementioned mobility indicators provide a faithful barometer to gauge the actual impact of the imposed containment measures. As such, they could be accounted for when training the LSTM to forecasting the future evolution of the epidemics.

Combining all these together, for each day of the time series, we access 10 distinct parameters, either referred to the evolution of the epidemics or the mobility trends. Input entries are normalized so as to operate, during training

[2]The base value is defined as the average value in the five weeks between 3th January and 6th February 2020 for the considered week day, as explained in [70]

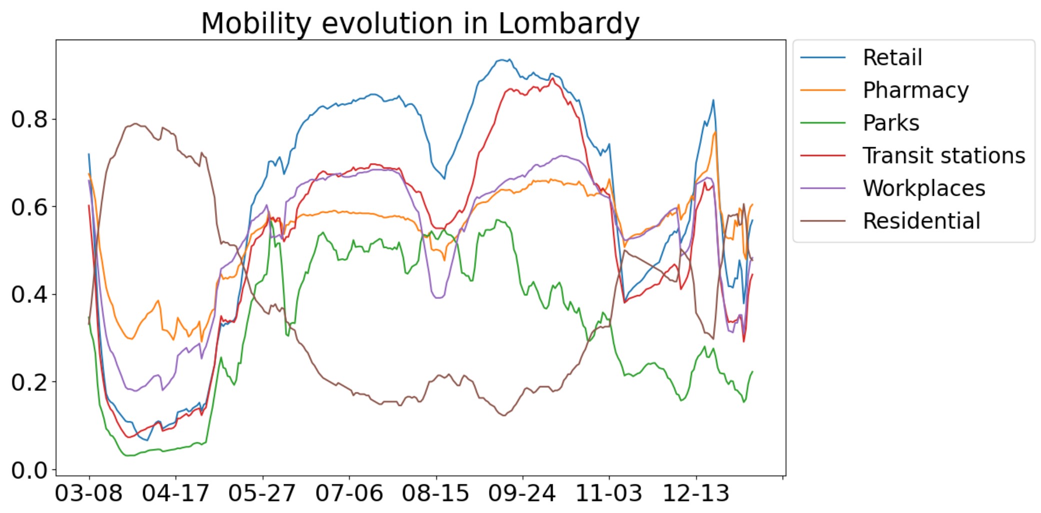


Figure 5.1: Time evolution of mobility parameters in Lombardy. A seven-days moving average was performed to filter out the weekly fluctuations and highlight the global trend.

stage, with quantities that span a definite range (details are provided in the Figures' captions). The above information are used in the supervised learning problem and organized as follows. The input vector collects information referred to the past n_p days. Epidemics and mobility data sum up to a total of 10 scalar parameters per day of acquisition. The output target vector has length n_f, the time horizon of the prediction. More specifically, each entry of the output vector returns a prediction for the number of patients in intensive care (IC) units, at the day of forecast, up to n_f days in the future. A schematic representation of data structure and processing handling is provided in Fig. 5.2. The mobile window is made to slide along the scrutinised time series, day after day. For each position of the window, the information stemming from the n_p preceding days (including the current day of observation) are acquired and confronted with the desired output, the number of occupied IC units in the future n_f days. During the training phase, this information is used to adjust the weights of the LSTM. When properly trained, this device is used for forecasting purposes by letting the sliding window to explore a portion of the times series not supplied during the learning phase. The computing apparatus is fed with the needed input information referred to the past n_p days (including the day of elaboration) to anticipate the future (the following n_f days) in terms of expected COVID-19 patients necessitating IC units.

Summing up, the training data set is made of $20(T - n_p - n_f)$ examples (that is, couples *input-target*) where the factor 20 stems from the number of considered regions. In the analysis reported below $n_p = 21$ (meaning that we process data from the last 21 days of observation) and $n_f = 7$ (hence, for each position of the sliding window, we look forward in time with a horizon of prediction that covers one week in the future).

The data set is divided into two subsets: *training and test sets*. The first set

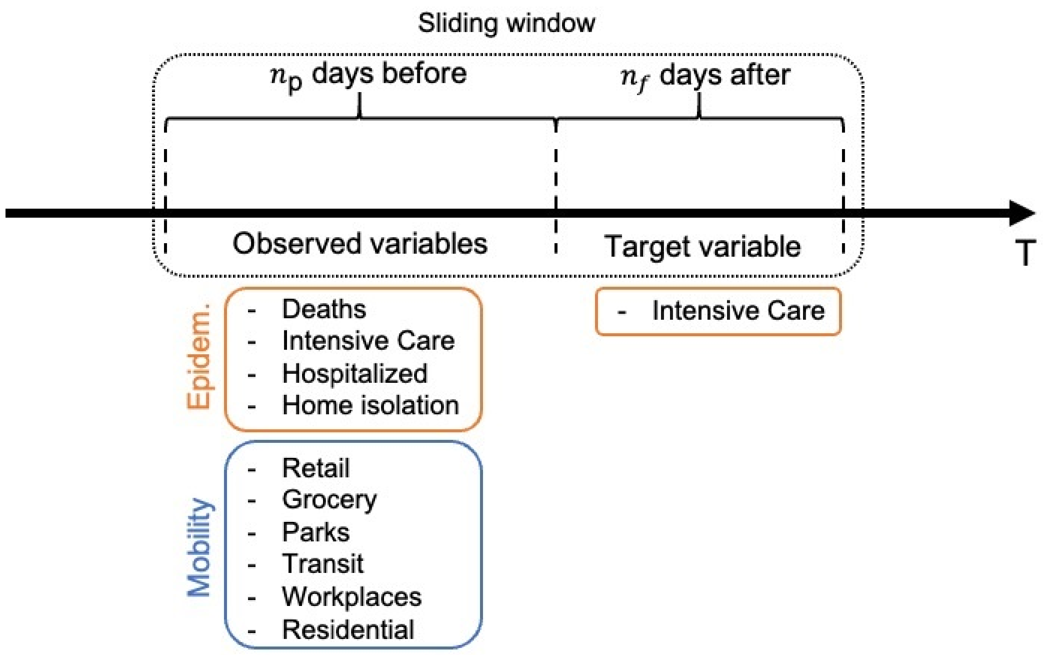


Figure 5.2: Each data consist of an ensemble of input parameters (concerning both epidemiological and mobility quantities and associated to the n_p days that precede the day where the analysis is carried out.) and an output set that contains our forecast. This is the number of occupied Intensive Care (IC) units in the following n_f days (from the last day of observation).

is used to train the model, whereas the second one is employed to test the ability of the trained device to cope with data that were not supplied during the learning stages. To probe the robustness of the method we have devised two different procedures to split the data in training and test set, respectively. These are listed in the following:.

1. *The training set consists in a limited segment of the available times series.* In this case, the training procedure is carried out by solely employing data up a prescribed date. The future evolution of the system, beyond the last day of the processed observation, is used to test the model. This makes it possible to test the performance of the LSTM model against data that refer to a time window not processed during learning.

2. *The training set is a sub-set of the available regions.* In this second case, the learning process is carried out over the entire length of the time series associated to a subset of the 20 regions. The accuracy of the prediction is tested against data referred to regions that were not used during the learning phases.

In the next subsection we will discuss the obtained results and validate the model as a viable tool to anticipate SARS-CoV-2 spreading across the Country.

5.3 Results

We used the architecture and data set as described in the previous section to define a learning problem that allows one to predict the evolution of the number of intensive care units (IC) occupied by COVID-19 patients in different regions of Italy.
We begin by adopting the first of the two aforementioned frameworks. This implies dealing with the full set of available time series, up to a given time, for the training phase. The trained network is then employed to forecast the evolution of the epidemics. Results are reported in Fig. 5.3 for a subsets of regions, namely Piedmont, Umbria and Veneto. The evolution of occupied IC units (orange trace) is nicely predicted by the model (coloured dots).

For each day, the number of truly occupied IC units is compared to the corresponding value, as predicted by the LSTM with different time horizons. More specifically, yellow dots refer to predictions which exploit information made accessible up to the preceding day. On the opposite limit, black dots are forecast that process information older than one week (7 days). Intermediate color grades refer to predictions which interpolate between these two extremes. The data reported (Fig. 5.3) are obtained by training the LSTM with data up to November 16th, where the dashed line is positioned. From here on, predictions are obtained by sliding the computing window (as depicted in Fig. 5.2) forward in time. The information relative to the n_p input days are processed and used to anticipate the expected load of IC units in the next n_f days. The forecasted evolution agrees pretty well with the observed curve of occupied IC units. Remarkably, the position of the peak is nicely captured by the computed time series which is hence capable to anticipating the evolution of the examined system.

In Fig. 5.4 the results obtained when dealing with the alternative setting as listed above, are depicted. As mentioned, we now train the model by focusing on a subset of the available regions and use this knowledge to predict the evolution of IC units occupied by COVID-19 patients in regions that were not supplied as part of the training set. Color are assigned following the same code introduced above: yellow dots refer to prediction that looks to just one day in the future. Black dots stand for the opposite extreme: the LSTM anticipates the evolution one week ahead in time. The agreement between predicted and observed times series is again remarkable.

In Fig. 5.5 the Root-Mean-Square Error (RMSE) associated to the predictions is plotted. Each bar represents the error made when trying to predict the target values $d \in [1, n_f]$ days in the future, where the parameter n_f defines the forecast horizon of the model. The RMSE is computed over the test set. Panel A of Fig. 5.5 is referred to $n_f = 7$, whereas panel B is obtained for $n_f = 14$. As expected, the accuracy goes down when d becomes larger. Although the model with larger n_f allows us to make early predictions, the accuracy of the predictions get worse when confronted with actual data: a lower accuracy is found not only for distant predictions but also for closer ones. The choice $n_f = 7$ is a compromise between the need to cope with reliable predictions, on the one side, and the request of

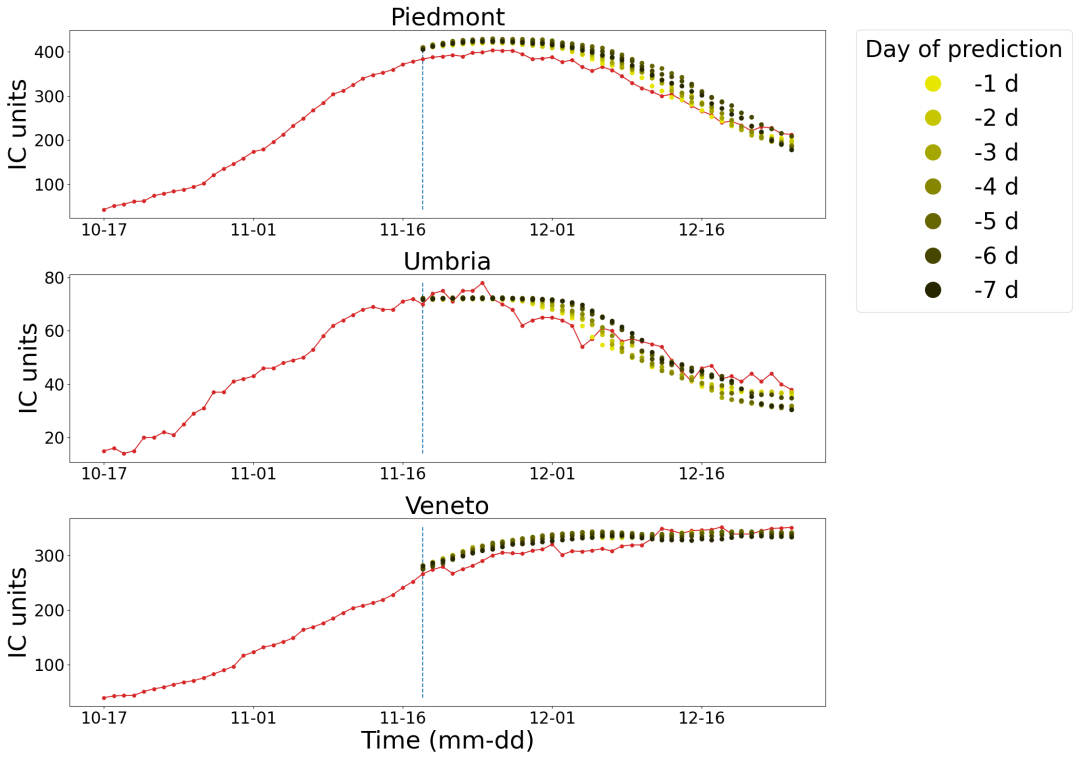


Figure 5.3: Predicted evolution as compared to the experimentally recorded time series: the plotted curves refer to the number of occupied Intensive Care (IC) units by SARS-CoV-2 patients in Piemonte, Umbria and Veneto. Red lines stand for the observed (hence, real) evolution. Coloured dots represent the forecast of the LSTM model. Yellow dots are predictions that look at one day in the future. Black symbols rely instead on processing one week old data. Different color gradings, ranging from yellow to black, interpolate between these two limiting scenarios. In this case n_p is set to 21. Data are rescaled by using, for each variable of the set, its corresponding maximum value, as displayed in the training interval. This latter value is also used to normalize data from the test set, so that only information from the training set are effectively employed.

imposing a plausible temporal horizon, i.e. useful for forecast, on the other.

In the following, we will shortly elaborate on the role of the mobility and highlight its reflexes on the evolution of the epidemics.

5.3.1 Change the mobility

Different containment measures have been imposed to contrast the spreading of the COVID-19 epidemic. Such measures, like social distancing and lockdown, result in a clear impact on the mobility. In Fig. 5.6 the evolution of five normalized mobility parameters are plotted for Piedmont and Tuscany. Data are processed by operating a 7 days moving average to obtain smoother profiles and remove weekly fluctuations. The averaged mobility indexes display global trends which bear the

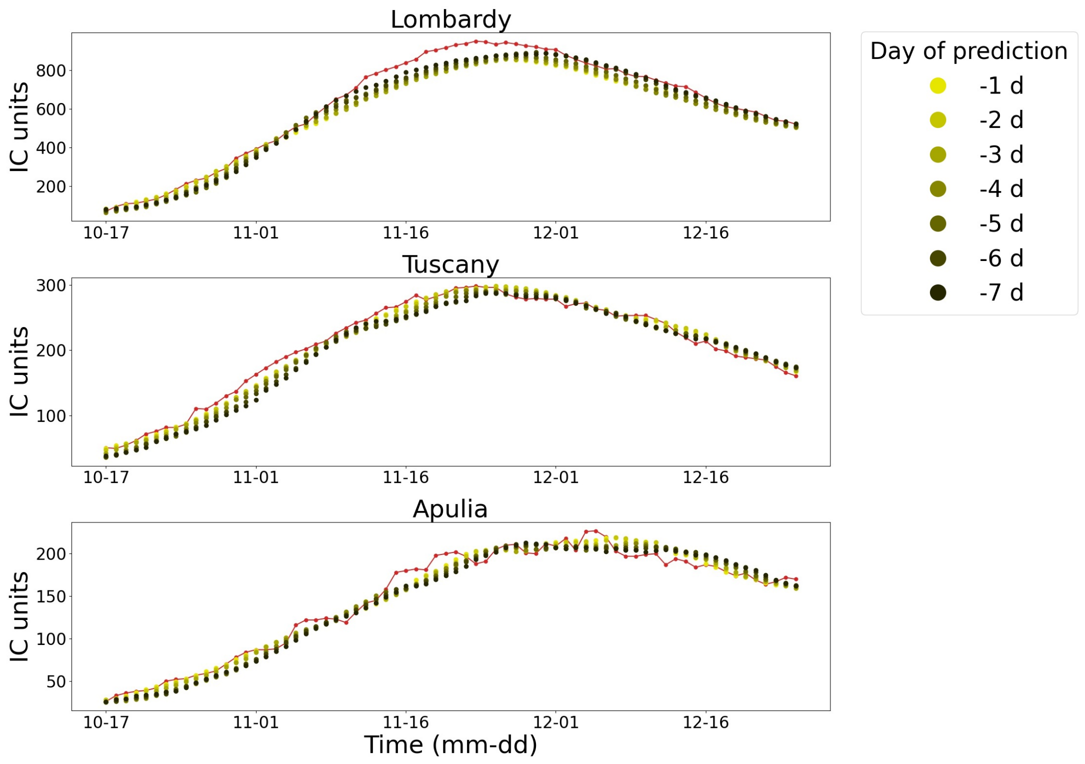


Figure 5.4: The number of occupied IC is plotted against time, expressed in days. Red traces reflect the observed, hence true, numbers. Colored dots stand for the LSTM forecasst. Here, the analysis is carried out for three regions that were not part of the training set. The adopted color code is specified in the caption of Fig. 5.3. The analysis is carried out by using $n_p = 21$. Here, epidemiological data are normalized by an arbitrary constant that we assume to extensively scale with the size of the examined region.

imprint of the containment measures as imposed by national and local authorities. To support this claim, for each region, five time intervals associated to different containment measures have been identified. At the beginning of the time series, a depression of all the mobility scores is detected (except for the parameter that quantifies mobility in residential areas – purple lines – which in general, and as expected, shows opposite trends as compared to those stemming from other parameters). This has to be put in relation with the strict lockdown taken by the Italian government in the spring of 2020. Subsequently, the curves associated to the parameters of not-residential areas grow up until they reach a new plateau. The plateau follows a no-restrictions (or few-restrictions) period, during the summer, when containment measures had been relaxed. Other characteristic periods can be indeed identified, specifically at the end of November and at the beginning of December. This last segment of the recorded time series is indirectly influenced by the introduced color code labelling of the regions, as reflecting the degree of local severity of the epidemics. Each region is in fact associated

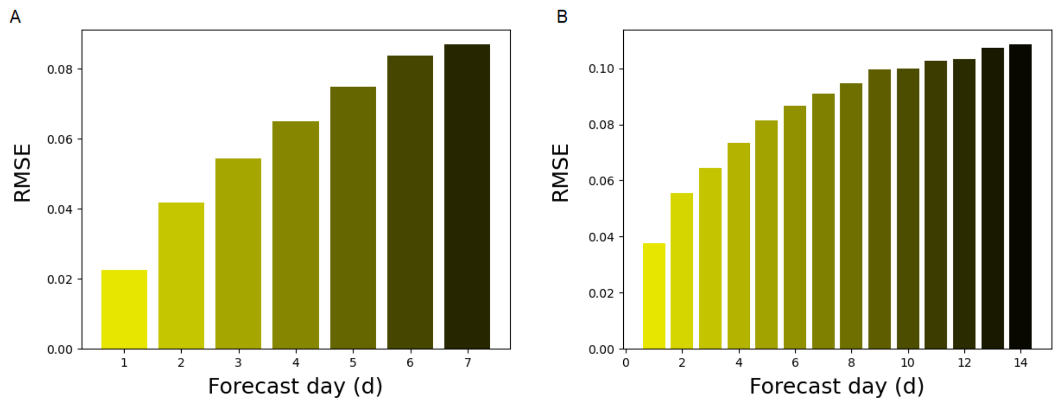


Figure 5.5: Root-Mean-Square Error (RMSE) computed on the test set: the error compares the real evolution of IC occupation number and the expected evolution on the basis of the LSTM prediction at day $d \in [1, n_f]$, from the last processed observation. Panel (A) and panel (B) referred to LSTM models with $n_f = 7$ and $n_f = 14$, respectively.

to a color (respectively, yellow, orange and red in ascending order of severity) and a different level of restrictions are adopted depending on the region color. The correlation between the actual severity of the imposed restrictions and the displayed mobility trends can be clearly appreciated by visual inspection of Fig. 5.6. To help visualization few (colored) vertical stripes are depicted which refer to different conditions of the mobility, as outlined above. The first bar, colored in grey, is traced in correspondence of the strict lockdown back in the spring 2020. By averaging over the selected time interval (the width of the greyish bar) we obtain an average estimate of the mobility parameters, as associated to the lockdown phase. Similarly, the other depicted bars identify other characteristic instances of the epidemics evolution: the green stripe is meant to select mobility score referred to the summer 2020. The red/orange/yellow bars identify the status of the region, as follow the novel strategy to label the severity of the disease at the local scale. Also in this case, by averaging over the width of the corresponding intervals, one obtains a set of values for the mobility indexes which indirectly reflect the imposed containment action (from draconian lockdown to no restrictions, via the intermediate settings as associated to different labelling colors).

This information can be used in the attempt to predict the role of an enforced modulation of the mobility, as follow the different scenarios recalled above. More specifically, at any given day, one can change the mobility entries as supplied to the trained LSTM (by fishing from the aforementioned alternative classes, identified via the corresponding averaged entries). The aim is examine the ensuing effect which materializes at the level of the forecasted evolution of the occupied IC units, the target of the LSTM. In Fig. 5.7 the result of the analysis is displayed for two reference regions, although the reached conclusion holds in general. A punctual modulation of the mobility (i.e. a change in the mobility that is confined

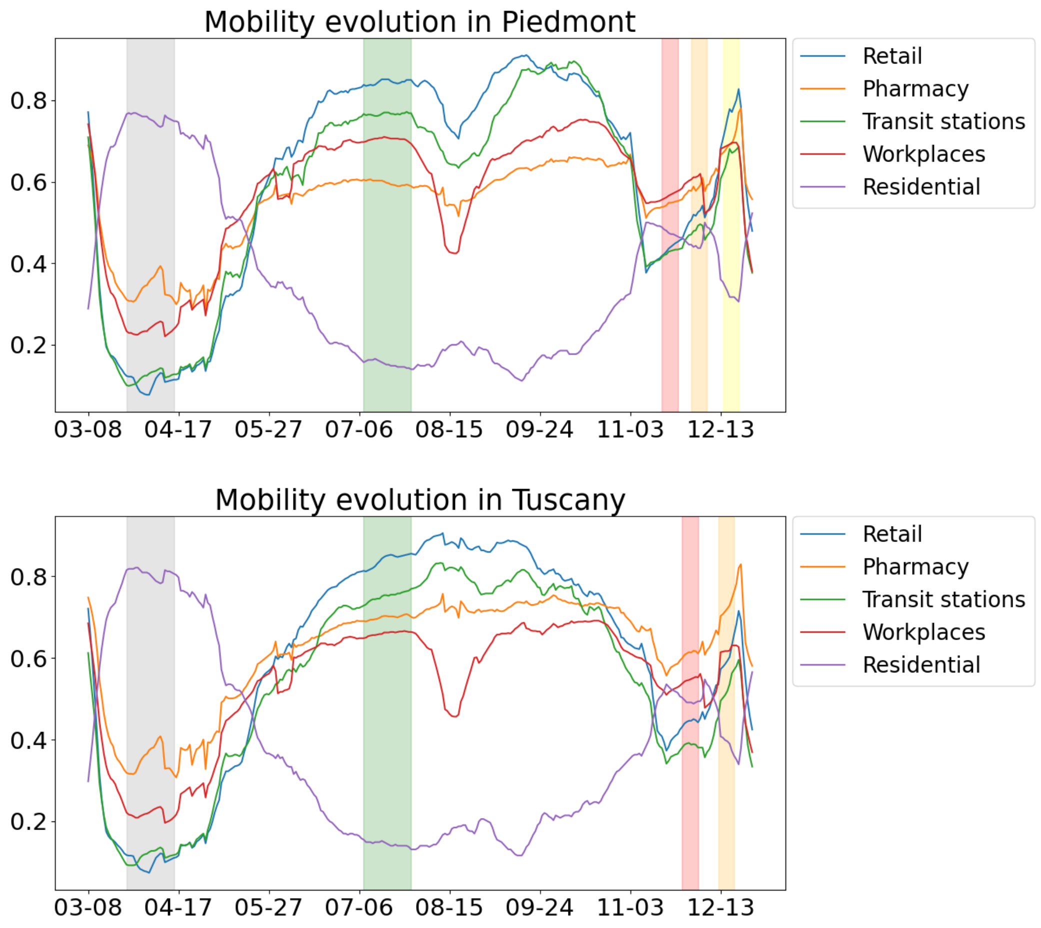


Figure 5.6: Mobility evolution in Piedmont and Tuscany, with reference to five distinct categories, as outlined in the annexed legend. Data are from [70] and have been normalized to yield quantities that range in the interval $[0, 1]$. A moving average of 7 days is operated so as to remove spurious weekly fluctuations. The four vertical bars define the time intervals over which the reference mobility values have been estimated.

to just one day) produced sensible changes in the predicted hospitalization, the response being more marked the stricter the reduction of the mobility being imposed. Remarkably, and according to the LSTM, the effect of a local change in the mobility becomes visible 8-10 days in the future, a plausible outcome of the analysis which calls for a timely planning of the containment protocols. On the basis of the above, it is hence surmised that machine learning schemes of the type here analyzed could help devising optimal strategies for an intelligent combination of openings and closures, at the local scale. Furthermore, notice that the lag time quantified above provides an a posteriori justification for choosing $n_f = 7$ as a forecast horizon of the LSTM machinery.

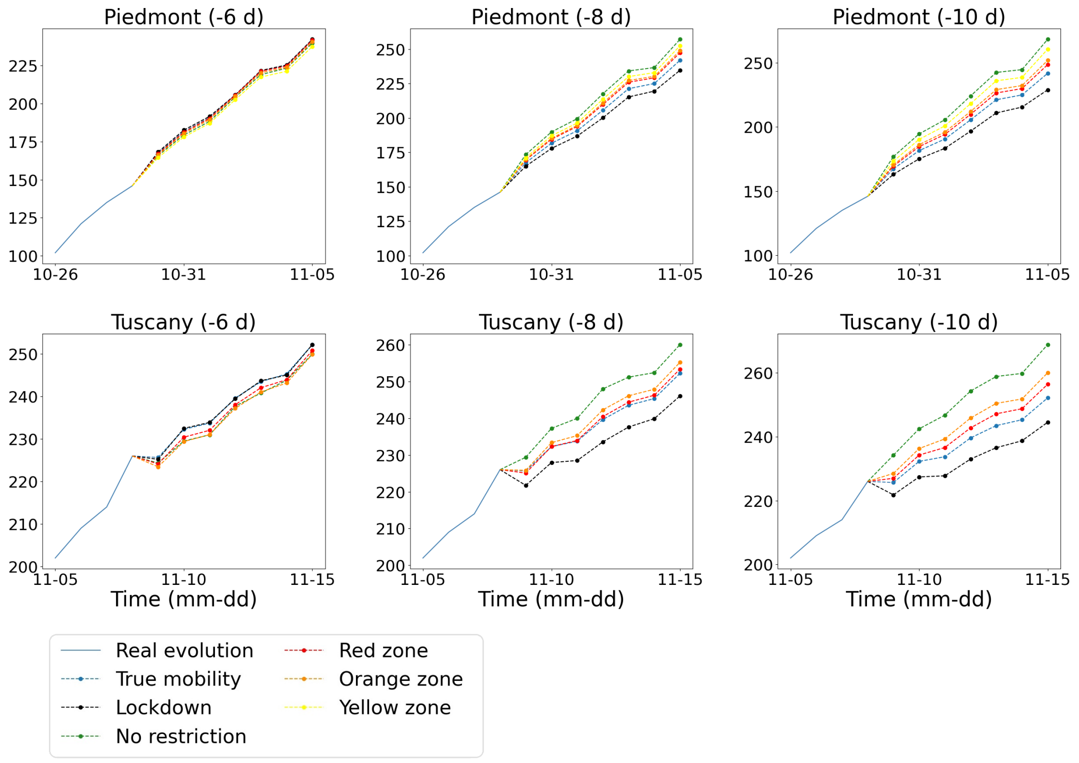


Figure 5.7: Predictions of IC units occupied by COVID-19 patients in Piedmont and Tuscany made under the hypotheses of 6 different past mobility scenarios: true mobility (blue dots), lockdown (black dots), no restrictions (green dots), red zone (red dots), orange zone (orange dots) and yellow zone (yellow dots). In each column, from left to right, the change in the mobility scores is operated at a different day, as measured from the time the first prediction is made (see annexed legend).

5.4 Conclusions

To summarize our findings, using a simple LSTM model trained on both epidemiological and mobility data we were able to correctly forecast the spreading of SARS-CoV-2 across different regions and at different times. Our model proved robust to alternative train/test splits in the spatial (hold out a region) and temporal (hold out a temporal interval) domains. The choice of employing available information on human mobility constitutes the novelty of the proposed approach. The obtained forecasts are indeed shown to sensibly depend on the imposed mobility scores. When artificially reducing the degree of imposed mobility, yields a consistent flattening of the curve of expected occupied intensive care units. Importantly, the effect of an abatement of the mobility materializes in a consequent contraction of the occupied IC. The contraction becomes visible after 8-10 days, from the time the mobility change became effective. Interestingly, punctual mobility stops (one day) seem to generate a noticeable effect on the predicted IC occupation curve. Elaborating further along these lines could help devising viable strategies to oppose the spreading of the epidemics, with a minimal impact on both social and economical activities.

In the next chapter we will show how neural networks can be successfully applied in the field of geothermobarometry. In particular, a deep architecture of feed forward neural networks was trained to obtain an estimate of the temperature and pressure at which a sample of volcanic rock was created starting from the data obtained from geochemical analyses. Furthermore, an open source web app will be presented, called GAIA, which allows to easily use the trained models to obtain estimates of the quantities mentioned above.

Chapter 6

GAIA, a novel Deep Learning-based tool for volcano plumbing systems

The anatomy of the plumbing system of active volcanoes is fundamental to understand how magma is stored and channeled to the surface. Reliable geothermobarometric estimates are, therefore, critical to assess the depths and temperatures of the complex system of magmatic reservoirs that form a volcano apparatus. Here, we developed a novel Machine Learning approach (named GAIA, Geo Artificial Intelligence thermobArometry) based upon Feedforward Neural Networks to estimate P-T conditions of magma (clinopyroxene) storage and migration within the crust. Our Feedforward Neural Network method applied to clinopyroxene compositions yields better uncertainties (Root-Mean-Square Error and R2 score) than previous Machine Learning methods and set the basis for a novel generation of reliable geothermobarometers, which extends beyond the paradigm associated to crystal-liquid equilibrium. Also, the bootstrap procedure, inherent to the Feedforward Neural Network architecture, permits to perform a rigorous assessment of the P-T uncertainty associated to each clinopyroxene composition, as opposed to the Root-Mean-Square Error representing the P-T uncertainty of whole set of clinopyroxene compositions. As a test, we applied GAIA to assess P-T conditions of five Italian volcanoes (Somma-Vesuvio, Campi Flegrei, Etna, Stromboli, Volcano), which are among the most dangerous volcanic centres in Europe. The results on the depths of the plumbing systems are in excellent agreement with those obtained with independent geophysical and geodetic surveys, and provide further evidence to current models of volcano plumbing systems consisting of physically-separated reservoirs interconnected by a network of conduits channelling magma en route to the surface. The results on the magma (clinopyroxene crystallization) temperatures are also in agreement with other estimates, albeit obtained considering - mainly but not only - thermodynamically-based clinopyroxene-liquid geothermometers. GAIA can set robust estimates of magma storage, segregation, and ascent conditions within the plumbing system of active volcanoes, helping to unravel P-T variations, if any, during their eruptive history and providing robust clues to volcanic hazard assessment.

6.1 Introduction

Understanding the dynamics of active volcanic systems is paramount to volcanic hazard assessment and can set the basis to forecast volcanic eruptions. This includes the geochemical and rheological characteristics of primary and evolved magmas, crustal storage conditions (P-T), ascent dynamics and interaction with crustal rocks, timescales of pre-eruptive magmatic processes, along with geophysical surveys [71]–[79]. In the last decades, our understanding of how magma is stored, migrates and feeds volcanoes has changed from a "relatively simple" melt-dominated magma chamber to more complex networks of lenses of melts, crystal mushes, and exsolved volatiles that extend throughout the crust up to the uppermost mantle [75]. This network plays an integral role in the eruption dynamics triggered by influx of fresh, hot, and deeper magmas into shallower reservoirs through prolonged magma segregation, stoping, and crystallization. In this context, thermobarometry is critical to decipher the anatomy of the plumbing system of active volcanoes, providing clues on the depth of magmatic reservoirs and their time-related migration. This information, which is among the main parameters controlling magma properties and eruptive styles, is fundamental for volcanic hazard assessment. Among minerals employed to routinely assess magma storage conditions, clinopyroxene is definitely the best candidate, as it is widespread in mafic to intermediate and even evolved magmas [80]–[82]. Here, we present GAIA (Geo Artificial Intelligence thermobArometry), a new geothermobarometer based on clinopyroxene-only compositions able to assess P-T conditions of magma storage and ascent within the crust, which relies on Machine Learning technology [83]–[85]. More specifically, a Feedforward Neural Network (FNN) has been trained to predict the anatomy of the plumbing systems in active volcanoes. GAIA provides more robust results than thermodynamically-based clinopyroxene-liquid geothermobarometers [80]–[82], [86] and other machine learning-based geothermobarometers developed in the last years [85], [87]–[89].

6.2 GAIA, a novel Machine Learning-based strategy to P-T estimates in volcanic systems

To overcome the abovementioned pitfalls that are inherently enfolded in the customarily employed geothermobarometers in volcanic systems, we used dedicated Machine Learning (ML) strategies for supervised assessment [83], [84]. In particular, FNN bear an unprecedented - still largely unexplored – potential to yield a novel generation of reliable geothermobarometer sensors, that extend beyond the equilibrium paradigm.

6.3 Dataset

We have worked on the same LEPR [90] dataset as Putirka [80] supplemented with recent compilations of experimental petrology data. The dataset consists of clinopyroxene-liquid pairs, covering a range of some 40 years and retrieving 6768 experimental data. To this dataset we have applied a few filters to focus the Feedforward Neural Network (FNN) method on volcanic systems. The intensive parameter P was filtered between 1 bar and 10 kbar (6118 data), while T was limited to <1500°C (6689 data). Melt composition was restricted to SiO2 >35 wt.% (6190 data) and MgO <20wt.% (6606 data) to have a wide spectrum of melts from mafic to felsic compositions. Clinopyroxenes were filtered on the basis of multiple parameters to check chemical analysis quality and include mainly quadrilater compositions and a few Ca-Na pyroxenes. The details on the applied filters are shown in [6]. In the end, the combination of all applied filters resulted in 5594 clinopyroxene-melt pairs and 5599 clinopyroxenes, which were used in the FNN method.

6.3.1 Calculation of clinopyroxene components

Chemical analysis of experimental mineral samples is commonly conducted by using an electronic microprobe, which enables non-destructive analysis of small sample portions. Through these analyses, the concentrations of the elements within the examined sample are determined. By leveraging on the known structural properties and composition of clinopyroxenes, reconstructing the concentrations of the various clinopyroxenes present within the sample becomes feasible. We then calculated clinopyroxene components following a novel procedure, slightly different from that commonly used [80]. We wish to point out that this procedure is not meant to reproduce the actual components occurring in clinopyroxene, simply it is a working procedure to be used in our FNN method. For the purpose of this thesis, we have chosen not to report onto the technical details of this procedure. However, these details are explicitly documented in [6]. The computed components were then used as data input to the FNN method.

6.4 Dataset elaboration and bootstrap procedure

The available dataset was initially divided into two distinct portions: the data points belonging to the first group were kept aside for subsequent inspection (the test set), while the elements defining the second ensemble were employed for training the neural network (the global training set). Two different fractions of the dataset were kept aside as test set for temperature and pressure regression, because pressure, contrary to temperature, is heterogeneously distributed across the explored range (1 bar $<$ P $<$ 10 kbar). For this reason, and to maximize the chance that sparsely populated pressure ranges were adequately represented, we have chosen to include in the test set only 5% of the total dataset. In contrast, we have decided to use a larger test set for temperature (20% of the total dataset) because it is evenly distributed across the considered temperature interval (<1500°C). To this end, data were first normalized to belong to the interval [0,1]. This was achieved by dividing each individual instance by the corresponding maximum, that is the largest value displayed by the homologous quantities defining the training set. A bootstrap procedure was then implemented following the prescriptions recalled hereafter. The global training set was split into two parts, the actual training set and the so-called validation set. The training set contains a number of items equal to 80% of the total. The instances that populate the set are sampled with replacement. The elements that have not been selected (at least 20% of the total) define the validation set. This operation is repeated M times and each configuration employed to train a selected model. After training has been completed, we are hence left with M distinct feedforward models that can be challenged against data. The accuracy of the predictions can be assessed versus the validation set (for each of the trained model) and, more importantly, against the test set, that has been protected from further scrutiny at the beginning of the procedure. Dealing with M distinct trained feedforward networks allows for a careful assessment of the prediction of the uncertainty (1σ) associated to each analyzed instance.

6.5 GAIA architecture and training details

Data were processed via a deep neural network with a feedforward architecture [84], [91]–[93]. Two separate networks were used for temperature and pressure regressions. The architecture was chosen after careful evaluation of the performance of different candidate models against validation set. In particular, a network consisting of three hidden layers of 1000 nodes each was used for temperature regression, while a network made of three hidden layers of 100 nodes each was employed to forecast the associated pressure. In both applications, a batch normalization (a dedicated normalization of the outputs in each layer to enhance training stability and convergence) is performed and a dropout layer (a regularization technique that randomly deactivates neurons during training, so as to prevent overfitting and promoting generalization) with rate=0.1 is inserted, after the first hidden layer. The last layer is linear with no activation function. Hence, the model has not been forced to yield positive defined output values, meaning that negative predictions for the estimated pressure are not excluded (the range of pertinence including zero at one side of the interval). The fact that negative outputs are only rarely obtained (see results) constitutes an a posteriori check on the overall reliability of the proposed methodology. Forcing the network to generate positive outputs represents an a priori bias that we have deliberately chosen to omit, relying on the inherent ability of the trained network to self-consistently cope with the relevant interval of definition for the examined variables. Each training iteration consisted in 500 epochs with a batch size fixed to 50. The Adam algorithm [94] has been used as optimizer, with a learning rate of 5·10-5. Interestingly, a larger network is required for the temperature to be adequately predicted. We believe that this is somehow related to the distribution of the input temperatures which cover - almost uniformly - a bound, though extended domain, across the real axis. Conversely, pressures are adequately predicted by means of a significantly smaller network. This is probably because, within the analyzed dataset, pressures tend to cluster around a few representative values. Results are stable for network larger than those employed in our analysis, at fixed architecture. The code[1], available on GitHub, was written in Python 3.7.9, the architecture and the fitting procedure were defined by means of the tensorflow library (2.1.0) [83].

6.6 Results of training and comparison with other machine learning methods

In recent years, a few works have begun to employ machine learning-based methods to estimate pressure and temperature of magmatic reservoirs. The first work that aimed at applying model free regression techniques to predict P-T conditions from clinopyroxene-melt and clinopyroxene-only, dates back to 2020 [87]. In this work, Petrelli and co-workers employed a wide range of methods, including gradient boosting, extremely randomised trees, random forest, k-nearest neighbours and decision trees to establish a causal relation between the supplied input to the sought output. Successive refinements of the above procedures have been reported in Li and Zhang [85] and involved either linear (principal component regression, partial least square and elastic net regression) and non-linear models (multivariate adaptive regression, k-nearest neighbours, support vector machine with radial kernel, random forest and extremely randomised trees). It is also worth mentioning the work by Higgins et al. [88] and Jorgenson et al. [89], who applied random forest methods. To the best of our knowledge, our work reports the first attempt to apply Feedforward Neural Networks (FNN) to geothermobarometry. Remarkably, and as we shall prove in the following paragraphs, the performance of the trained FNN overcome those reported in the literature, in terms of ability to accurately predict temperature and pressure of magmatic reservoirs from clinopyroxene-only data.

The results of the FNN model are subdivided in sequential steps. Temperature predictions of the validation set using clinopyroxene-only and clinopyroxene-liquid pairs plotted against the experimental temperature are reported in the two panels of Fig. 6.1. The accuracy (indicated by the R2 score) and the RMSE at 1σ level is excellent in both cases, albeit the performance of the clinopyroxene-liquid geothermometer is better than the clinopyroxene-only geothermometer. Pressure predictions of the validation set are reported in the two panels of Fig. 6.2 plotted against experimental pressure. The accuracy and the RMSE is excellent also in these two cases and, as with temperature, the clinopyroxene-liquid geobarometer performs better than the clinopyroxene-only geobarometer. Then, we compared the results of the validation set of the clinopyroxene-only and clinopyroxene-liquid pairs geothermobarometers in the two panels of Fig. 6.3. The excellent agreement between the two geothermobarometers in terms of accuracy and RMSE, gave us confidence to apply the FNN method to estimate P and T using clinopyroxene data only, even though the uncertainties on P-T estimates are worse than those

[1]github.com/GAIA-geothermobarometry/GAIA

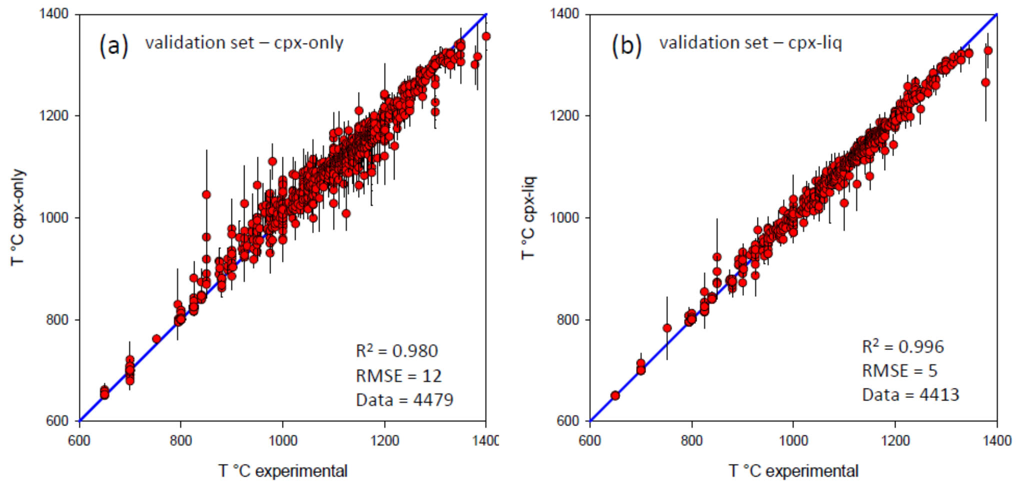


Figure 6.1: Predicted temperature of the validation dataset vs. experimental temperature using the Feedforward Neural Network method. (a) Clinopyroxene-only geothermometer, and (b) linopyroxene-liquid geothermometer. Each data point represents the average prediction and associated uncertainty (1σ) computed after M independent replicas trained with the bootstrap procedure (see text). Regression statistics [R2 score, Root-Mean-Square Error (RMSE), and number of data (n)] is reported in the two insets.

of clinopyroxene-liquid pairs (Figs. 6.1 and 6.2). We reckon, however, that the slightly worse P-T uncertainties are counterbalanced by the fact that considering only clinopyroxene does not require any equilibrium test on the coexisting melt, which, as we have shown above, is rather questionable. The next step has been to assess the performance of the FNN method using the test set, that is the clinopyroxene data kept aside from the training procedure during the global training set (see dataset elaboration). The P-T prediction are reported in the two panel of Fig. 6.4 in terms of the difference between experimental and estimated P-T. The two different sizes of the test sets (P = 280 cpx, T = 1120 cpx) are due to the fact that they represent 5% (P) and 20% (T) of the total clinopyroxene dataset, respectively (see dataset elaboration). The regression statistics yields an accuracy (R2 score) of 0.90 and 0.88 for temperature and pressure, respectively, whereas the RMSE attests to ± 28°C and ± 0.90 kbar (Fig. 6.4). Analogous results have been obtained for different (random) selections of the test set. Overall, the performance of GAIA is much better than the best uncertainties reported in previous machine learning-based approaches on clinopyroxene-only data (RMSE = ± 57°C, and ± 2.3 kbar, Higgins et al.,[88]). To help comparison we have also assessed the performance of our Deep Learning method on the subset of data provided by Higgins et al. [88]. The obtained uncertainty is equal to ±0.9 kbar, identical to that obtained for the whole dataset. It is also worth emphasising that we have here carried out a rigorous assessment of the uncertainty associated with predictions by improving on other reported approaches [85], [87]–[89], and accounting for the peculiarities of the employed dataset. In particular, the bootstrap procedure allows us to associate an error to a given prediction. This is

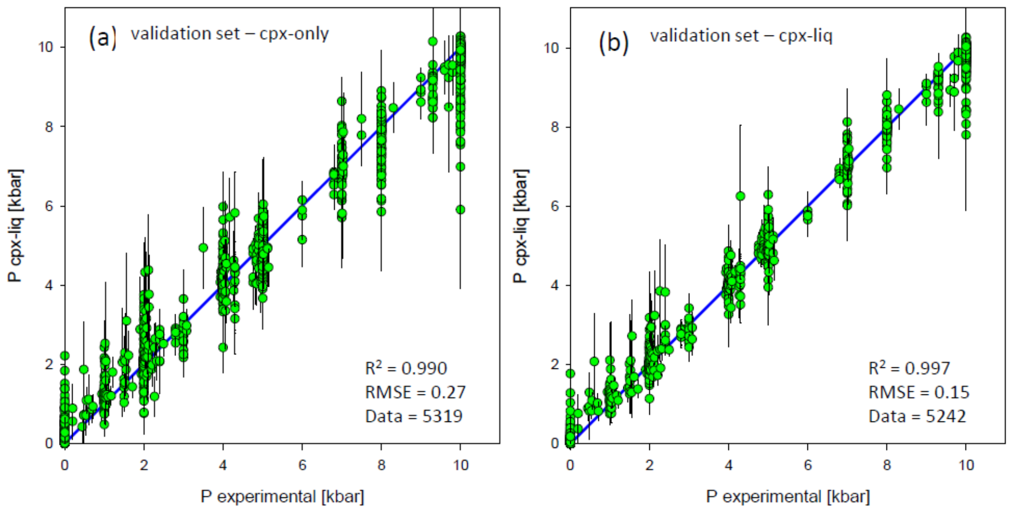


Figure 6.2: Predicted pressure of the validation dataset vs. experimental pressure using the Feedforward Neural Network method. (a) Clinopyroxene-only geobarometer, and (b) clinopyroxene-liquid geobarometer. Each data point represents the average prediction and associated uncertainty (1σ) computed after M independent replicas trained with the bootstrap procedure (see text). Regression statistics [R2 score, Root-Mean-Square Error (RMSE), and number of data (n)] is reported in the two insets.

achieved by looking at the statistical distribution of the responses obtained when operating the collection of trained models on the very same data supplied as an input. The bootstrap error can be computed for any given input as opposed to the RMSE which can be solely computed for datapoints belonging to training, validation, and test sets (Figs. 6.2, 6.3 and 6.4). Moreover, to account for the inherent degree of heterogeneity in the distribution of recorded pressures, we performed an analysis of the associated uncertainties on validation set. We aimed, in particular, to a quantitative assessment of the reliability of a given prediction, depending on the position it occupies in the space of the output variables (Fig. 6.5b). The analysis demonstrates that the data points sampled from the most represented experimental pressure values (1 bar, 2 kbar, 4 kbar, 8 kbar, and 10 kbar, Fig. 6.5a) are correctly identified by the trained FNN model, whereas the data points laying in between (e.g., from 2 to 4 kbar) are also traced back to their correct domain of pertinence although yielding higher uncertainties (see Fig. 6.5 for details). The uncertainty in P prediction is better visualized in the 9 panels of Fig. 6.6. P estimates falling within the 5 classes most represented by the experimental data points (1 bar, 2 kbar, 4 kbar, 8 kbar, and 10 kbar, Fig. 6.6) are associated with low uncertainty ($<<$ 1 kbar), whereas P estimates falling within the intermediate ranges yield, as expected, larger, although limited, uncertainty. Overall, the statistical treatment we have performed on prediction uncertainties with our FNN model (Figs. 6.5 and 6.6) constitutes an a posteriori proof of the ability of the network to cope with the supplied dataset and eventually materialize in reliable predictions for both pressure and temperature (Figs. 6.1, 6.2 and 6.4).

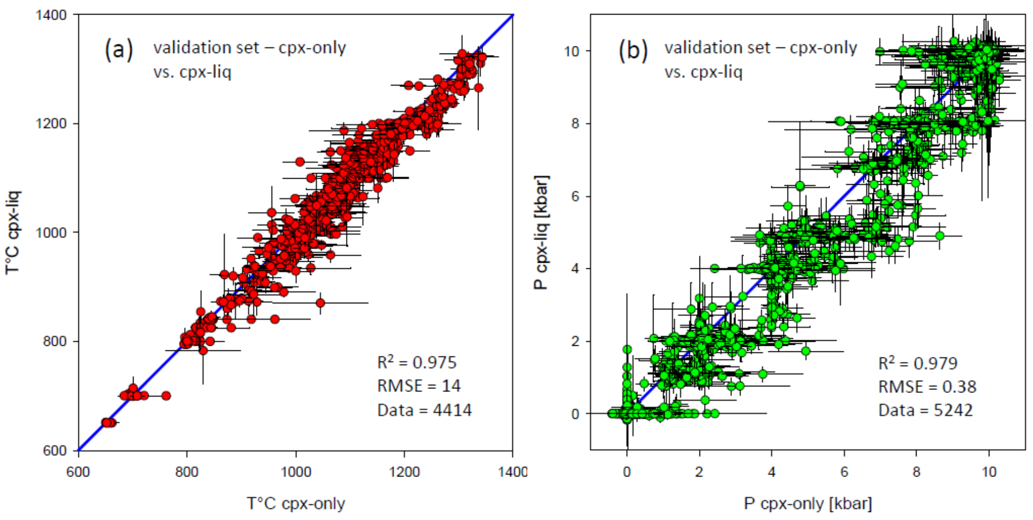


Figure 6.3: Comparison between clinopyroxene-only and clinopyroxene-liquid geothermometer (a) and geobarometer (b) of the validation dataset using the Feedforward Neural Network method. Each data point represents the average prediction and associated uncertainty (1σ) computed after M independent replicas trained with the bootstrap procedure (see text). Regression statistics [R^2 score, Root-Mean-Square Error (RMSE), and number of data (n)] is reported in the two insets.

The uncertainty in pressure estimates will be assumed to be ± 1 kbar (Fig. 6.4, Fig. 6.6) in all pressure ranges

6.7 Application to the Italian volcanoes

Italy, along with Greece, has the most spectacular active volcanic centres in Europe. Etna and Vesuvius, in particular, are located within densely populated areas and the International Association of Volcanology and Chemistry of the Earth's Interior (IAVCEI) has included them into the list of volcanoes to be kept under closer surveillance (http://www.sveurop.org/gb/articles/articles/decade.htm). Here, as an example, we want to apply and compare GAIA to assess the P-T conditions of magma storage and ascent within the crust to five active Italian volcanoes, namely: Etna and Stromboli (currently active), Campi Flegrei and Vulcano-Vulcanello (in volcanic unrest), and Somma-Vesuvio (quiescent). Their clinopyroxene compositions were downloaded from the GEOROC Data Repository (https://georoc.eu/georoc), and the results obtained with our FNN model have been compared, whenever possible, to those coming from geophysical and geodetic investigations. Exhaustive reviews on the volcanological history of these volcanoes have been reported elsewhere [71], [72], [95]–[103]. Before discussing the results of the FNN model (Figs. 6.7, and 6.8), we wish to make the following preliminary considerations on the clinopyroxene dataset downloaded from the GEOROC Data Repository:

(i) The clinopyroxene dataset has not been subdivided according to the different eruptive periods of each volcano, implying that we are dealing with their overall plumbing system. We acknowledge that during the history of each volcano, different magmatic reservoirs may have contributed to a given eruption, but the main objective of the present study is to critically compare the outcome of our Neural Network model with geophysical surveys, deferring the analysis of the plumbing system of each single volcano (e.g., correlation with VEI of single eruptions) to other companion works;

(ii) Also, the clinopyroxene dataset is not necessarily and homogeneously representative of all magmatic reservoirs of the plumbing system of each volcano, because it depends on the crystal cargo sampled by a given eruption;

(iii) The estimated depth of the different magmatic reservoirs is always reported in kbar from the summit of the volcanic cone down to the Moho using the crustal stratigraphy - and corresponding rock density - of each volcanic centre;

(iv) The uncertainty (1σ) on the predicted pressures and temperature of crystallization of the clinopyroxene-only geothermobarometer is generally <1 kbar (Figs. 6.4b, and 6.6) and < 30°C (Fig. 6.4a), respectively;

(v) The frequency of negative pressure estimates is generally $<2\%$ for all volcanoes, also considering that pressure estimates from 0 to -1 kbar are within error with positive values, indicating a very shallow, just below the surface, clinopyroxene crystallization;

(vi) Temperature estimates refer to that of magma during clinopyroxene crystallization.

6.7.1 Etna, 3357 m a.s.l.

The magmas of Mt. Etna are hawaiites and mugearites and erupt mainly as lava flows and lava fountains [72]. The plumbing system structure of Mt. Etna has been imaged from the summit crater (3357 m a.s.l.) down to ca. 6 kbar (ca. 20 km b.s.l.) using volcanic tremor, geodetic and geophysical data along with thermodynamic equilibria [104]–[107]. The main magmatic reservoirs (M), physically separated and variably connected to each other, have been located at (i) $\leq$ 0.3 kbar (M2) (>1.9 km a.s.l), (ii) 1.2-1.6 kbar (M1b) (ca. 1.5-3.1 km b.s.l.) (; (iii) 2.3-2.9 kbar (M1a) (ca. 6-8 km b.s.l.); (iv) 3.8-4.0 kbar (M0) (ca. 11-13 km b.s.l.); (v) 5.5-6.5 kbar (M00) (ca. 18-22 km b.s.l.). The inferred depths of these magmatic reservoirs are totally consistent with the results obtained with GAIA (Fig. 6.7a). Neglecting the 10% clinopyroxenes yielding $-1<P<0$ kbar, which, taking into account the uncertainty, indicate crystallization just below the surface, most clinopyroxenes (>70%) yield pressures corresponding to the shallower reservoirs (M2, M1b, M1a) from 0 to 3 kbar with a peak (ca. 30%) between 0 and 1 kbar. The Kernel density estimate (KDE) reveals other two small peaks at 3.5-4 kbar (ca. 5%) and 5.5-6.5 kbar (ca. 2%) corresponding to the depths of the M0 and M00 reservoirs. It is worth noting that the pressure distribution indicates a continuum of clinopyroxene crystallization, supporting a mechanism of crystallization during magma rising and transfer along a network of interconnected reservoirs [72]. The temperature of Etnean basalts ranges from 1080°C to 1136°C into the M2, M1b, and M1a reservoirs and from 1150°C to 1210°C in the deepest reservoirs [72], [108]. The KDE temperature distribution of GAIA has a quasi-gaussian shape with a peak at 1075±30°C, and ca. 85% of data between 1025 and 1125°C (Fig. 6.8a). These results are in agreement with temperature estimates of Etnean basalts of the M2, M1b, and M1a reservoirs as P estimates (>70% clinopyroxenes) indicate a provenance from the shallowest reservoirs (Fig. 6.7a). The limited data frequency between 1125°C and 1150°C (<7%) suggests that clinopyroxene did not start to crystallize in the deepest reservoirs and the liquidus phase was olivine [72].

6.7.2 Stromboli, 924 m a.s.l.

The Stromboli stratovolcano has a current steady-state activity where continuous input of high-K calcalkaline basalts has maintained persistent volcanism for centuries, with mild eruptions of crystal-rich black scoriae (i.e., Strombolian activity). The Strombolian activity is periodically interrupted by more energetic short-lived eruptions and basaltic lava flows (e.g., Francalanci et al., 1999, 2005, 2013; Bragagni et al., 2014; Petrone et al., 2018). The anatomy of the plumbing system of Stromboli has been reconstructed from the summit crater (924 m a.s.l.) down to ca. 5.5 kbar (ca. 20 km b.s.l.) using artificial and natural seismic data sources, volcano-tectonic events, volcanic tremor, geodetic data, volatile (CO2, H2O, S and Cl) content in melt inclusions along with thermodynamic equilibria (Allard et al., 1994; Vaggelli et al., 2003; Mattia et al., 2008; Métrich et al., 2010; Patanè et al., 2017; Gambino et al., 2018; Ubide et al., 2019). According to these studies, the architecture of the plumbing system consists of a succession of magmatic reservoirs at different depths, where magma accumulates and differentiates, interconnected by channels that form preferential pathways of magma en route to the surface. These interconnected magmatic reservoirs are located from 3.9-4.4 kbar (14-16 km b.s.l.) to 1.8-2.6 kbar (6-9 km b.s.l.), 0.5-1.3 kbar (1-4 km b.s.l.), and at ca. 0.1 kbar (500 m a.s.l.), just beneath the craters. Pressure estimates of GAIA mimic the distribution of these magmatic reservoirs (Fig 6b). Most clinopyroxenes (ca 80%) crystallized in the last 2.5 kbar of the plumbing system although the KDE indicates a minor peak (some 4-5%) at 4 kbar as well, i.e., the onset of clinopyroxene crystallization in the deepest reservoir beneath Stromboli. It is to note that the largest peak at 1-1.5 kbar (>25%) provides evidence for a clinopyroxene cargo originating in the magmatic reservoir located at the root of the volcanic edifice. As observed for Etna (and other volcanoes studied in this work, see below) the continuum pressure distribution of clinopyroxene supports a process of crystallization both in the magmatic reservoirs and along the network of conduits that channel magma en route to the surface. The most recent temperature estimates on clinopyroxenes of Stromboli basalts have been thermodynamically determined and have a variation from 1082±7°C (Ubide et al., 2019) to 1180-1090°C and 1140-1060°C (Scarlato et al., 2021). The KDE temperature distribution obtained with GAIA has a gaussian shape with a peak at 1100±30°C, and >80% of data between 1075°C and 1125°C (Fig. 6.8b), in remarkable agreement with previous estimates.

6.7.3 Vulcano and Vulcanello, 501 m a.s.l.

Vulcano and Vulcanello consist of high-K calcalkaline and shoshonitic magmas exhibiting a wide sprectrum of compositions with a progressive increase in silica and potassium contents with time from basalts and shoshonites to trachytes and rhyolites. The volcanic activity mainly consists of fallout deposits, lava flows, pyroclastic flows, and hydrothermal eruptions (Peccerillo et al., 2006; De Astis et al., 2013; Costa et al., 2020). The internal structure of the plumbing system of Vulcano has been assessed by integrated geophysical surveys, fluid-inclusion studies, and thermodynamic equilibria (Clocchiatti et al., 1994, Zanon et al., 2003, Peccerillo et al., 2006; De Astis et al., 2013, Costa et al., 2020). The magmatic reservoirs consist of two major storage zones at 4.8-6 kbar (17–21.5 km bsl) and 2.2-3.7 kbar (7.8–13.5 km b.s.l.), and a minor shallower storage zone, beneath La Fossa Cone, at 0.4-1.6 kbar (1.3–5.5 km b.s.l). The deepest magmatic reservoir is located at the transition between the upper mantle and a granulitic lower crust, and a network of conduits connects these three magma accumulation reservoirs. The pressure distribution yielded by GAIA is able to locate these reservoirs (Fig. 6.7c). The KDE delineates 5 major peaks at 5.5, 4, 3, 2, and 1 kbar. The clinopyroxenes indicating a pressure of 5.5 kbar (ca. 3%) originate from the deepest reservoir at the crust-mantle transition, whereas those resulting in a pressure of 1 kbar (ca. 15%) originate from the shallowest magmatic reservoir beneath La Fossa Cone. The intermediate reservoir at 2.2-3.7 kbar is indicated, considering an error of ±1 kbar, by most of the other clinopyroxenes. Temperature estimates of Vulcano and Vulcanello magmas have a wider spectrum owing to the compositional range from basalts to rhyolites. Latites and trachytes yield 1080±10°C and 1050±10°C, respectively (Clocchiatti et al., 1994). A lower temperature of 1010°C is reported for latites by Vetere et al. (2015) along with 950°C for rhyolite. Other studies reported 1004±14°C for a K-rich trachyte, 1007±9°C for other trachytes, 1027±5°C for latites, 955±8°C for rhyolites (Costa et al., 2020). The KDE temperature distribution obtained with GAIA is consistent with these estimates and reveals three major peaks at ca. 1000±30°C, 1050±30°C, and 1125±30°C (Fig. 6.8c). The highest temperature (1100-1150°C, ca. 22% of data) potentially reflects the temperature of basaltic magmas s.l., which are not included in the abovementioned estimates. The intermediate temperature (1025-1075°C, ca. 33% of data) agrees with the temperature reported for latites and trachytes (Clocchiatti et al., 1994), whereas the lowest temperature (975-1025°C, ca. 26% of data) is consistent with that reported for other trachytes and latites (Costa et al., 2020). The temperature related to rhyolites (955±8°C) has been determined using the plagioclase-liquid thermometer (Putirka, 2008), hence cannot be established by our clinopyroxene-only geothermometer.

6.7.4 Somma-Vesuvio, 1281 m a.s.l.

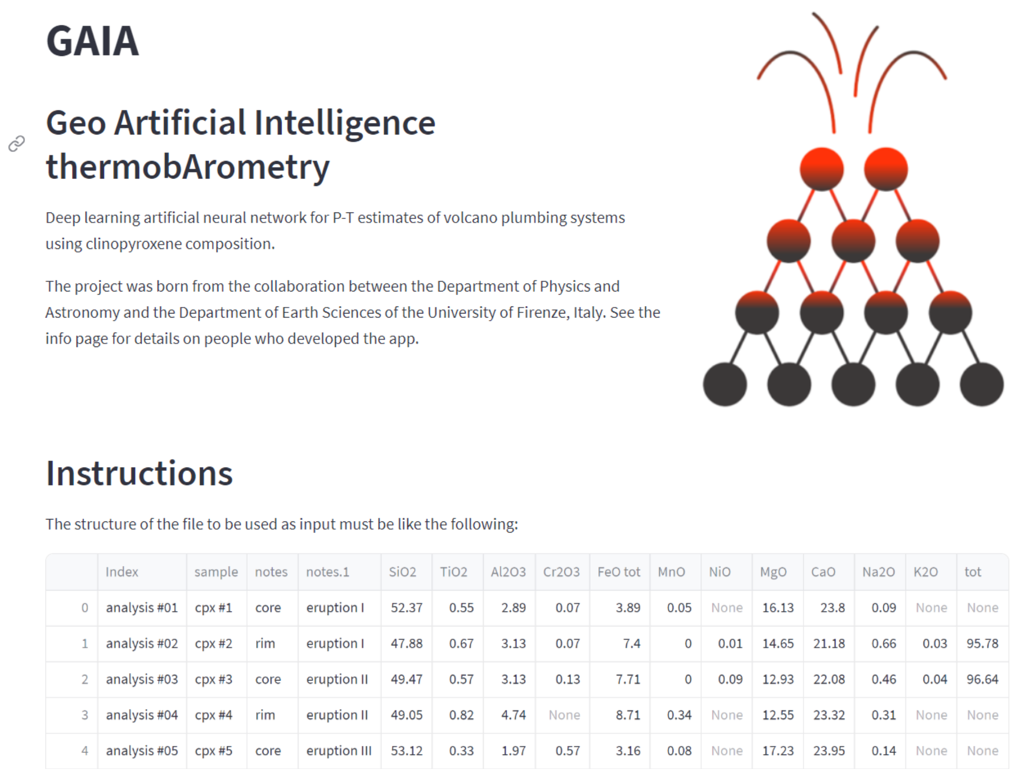
The Somma-Vesuvio stratovolcano consists of a wide spectrum of compositions from leucite-bearing trachybasalts to leucite latites and trachytes, along with leucite basanites, leucite tephrites and phonolites. The volcanic activity is characterized by explosive Plinian and Subplinian eruptions in addition to effusive lava flows (Cioni et al., 1998; Scaillet et al., 2008). As for other volcanoes of this study, the anatomy of the plumbing system of Mt. Vesuvio consists of multi-depth magmatic reservoirs detected on the basis of geophysical and geodetic surveys, magnetotelluric data, and electrical conductivity measurements (Iuliano et al., 2002; Nunziata et al., 2006; De Natale et al., 2006; Pommier et al., 2010; Fedi et al., 2018). The magmatic reservoirs are randomly distributed and interconnected to each other from the mantle-crust transition at ca. 7.5 kbar (25 km bsl) up to 2.5 kbar (8 km bsl), and then between 1.9 and 0.8 kbar (6 and 2km bsl). The KDE pressure distribution obtained with GAIA (Fig. 6.7d) indicates 4 peaks at 1.5, 3.5, 6, and 9 kbar. The major peak at 1.5 kbar (ca. 34% of data) is consistent with an origin of clinopyroxenes from the shallowest magmatic reservoirs, whereas the other two peaks at 3.5 and 6 kbar (ca. 16% and 4% of data) delineate a reservoir located in the mid-crust and another one close to the mantle-crust transition, likely representing magma ponding and crystallizing at the base of the crust. It is to note that the very minor amount of clinopyroxenes yielding a pressure of 9 kbar (ca. 1% of data) is suggestive of magma crystallization during rising from the mantle (Fedi et al., 2018). The other pressure distributions of clinopyroxene can indicate either other reservoirs randomly distributed in the 7.5-2.5 kbar range or the network of channels interconnecting the different magmatic reservoirs. Temperature estimates of Somma-Vesuvio have been related to three stages of evolution of the plumbing system: an initial stage with mafic magma at ca. 1100°C, a young stage with evolved mafic magma at ca.1050°C progressively grading to felsic magma at 850-900°C, and a mature stage with evolved mafic magma at ca. 1050°C and a stratified upper portion of felsic magma at 800-950°C (Cioni et al., 1998). The GAIA KDE temperature distribution (Fig. 6.8d) yields a major peak at 1075±30°C and two minor peaks at 950°C and 900°C. The high-T peak from 1025°C to 1125°C (ca. 75% of data) is consistent with temperature estimates of the mafic magma (trachybasalt, basanite) feeding the magmatic reservoirs of Somma-Vesuvio, and the evolved mafic magma (tephrite), whereas the other minor peaks from 850°C to 1000°C (ca. 12% of data) correspond to the temperatures of felsic magma (latite, phonolite).

6.7.5 Campi Flegrei (caldera assumed at sea level)

The Campi Flegrei volcanic field consists predominantly of pyroclastic rocks and subordinate lavas flows and domes, with a compositional range from minor shoshonites to more abundant trachytes and trachyphonolites (e.g., Orsi et al., 2022). The anatomy of the plumbing system of Campi Flegrei volcanic field is similar to that of the nearby Somma-Vesuvio on the basis of geophysical and geodetic surveys, and volatile (H2O, CO2, H2S, Ar, N2, H2, He, CH4, and CO) content of fumarolic gases (e.g., De Siena et al., 2017, Buono et al., 2022, Calò et al., 2018; Costanzo and Nunziata, 2017; De Natale et al., 2006; Fedi et al., 2018; Zollo et al., 2008; Fedi et al., 2018). According to these studies the deepest reservoirs are located from ca. 7.1-2.1 kbar (ca. 8 to 25 km b.s.l.) and consist of a continuum of several melt pockets and crystal mushes variably interconnected to each other which, through decompression-induced, recharge shallower reservoirs at 0.5-1.2 kbar (2.-4.5 km b.s.l). The former possibly represents permanent long-lived reservoirs where magma is stored and differentiates, whereas the latter could represent small and ephemeral reservoirs, further differentiating and mixing (Pappalardo et al., 2012; Fedi et al., 2018). The GAIA KDE pressure distribution (Fig. 6.7e) yields two major peaks at 3-4 kbar (ca. 30% of data) and 0-1.5 kbar (ca. 27% of data). The former, along with the tail towards higher depths (ca. 15% of data), provide evidence for the origin of clinopyroxenes from the melt pockets and crystal mushes of the deepest reservoirs, whereas the latter indicates an origin from the shallower reservoirs. Similar to Somma-Vesuvio, clinopyroxenes of Campi Flegrei have a small peak (ca. 2% of data) at 5.5-7 kbar suggestive of magma ponding and crystallizing at the base of the crust. The P values between -1 and 0 kbar are within error of positive values, indicating a subsurface crystallization. Based upon thermodynamic equilibria, Forni et al. (2018) and Pelullo et al. (2022) reported a temperature range of Campi Flegrei magmas from ca. 1120°C to 870°C during the evolution from shoshonite to trachyte and trachyphonolite, with most magmas falling in the 1000°C-900°C range (Fig. 6.5 in Forni et al., 2018). The KDE temperature distribution obtained with GAIA (Fig. 6.8e) yields a major peak at 1100±30°C (ca. 70% of data) with a tail towards lower temperatures of 900°C (<20% of data). Broadly speaking, the two temperature estimates approaches (GAIA and thermodynamic equilibria geothermometers) are consistent to each other, although, to a more careful attention, we note that most magmas have 1000°C-900°C using the thermodynamic equilibria geothermometer, in contrast to the peak at 1100°C yielded by GAIA. We think that this 100-200°C difference could be ascribed to the fact that most phenocrysts (olivine, clinopyroxene, feldspar) of Campi Flegrei magmas are not in equilibrium with their host liquid (Pelullo et al., 2022). This may severely undermine the results of the thermodynamic approach, and it is the main reason why we decided to tackle P-T estimates using a machine learning strategy.

6.8 A web app to facilitate research

To make our model easy to be used by researchers, we have developed a web free and open-source app. The app, which can be reached at this link: https://gaia-geothermobarometry-gaia-home-6ol8kg.streamlit.app/, was developed in Python with the help of the open-source app framework Streamlit. To use the app, one can simply upload a file in “.xlsx" format which contains the results obtained from the analysis of the samples whose temperature and pressure one wants to recover. you can download a blank file with the correct formatting, as well as view an example of input. Once the file has been uploaded, it is possible to start the calculation process. The app will first of all calculate the components of the clinopyroxenes which will be fed to the neural networks. Finally, an output file is produced in which estimates for pressure and temperature are reported for each of the data entered.


GAIA

Geo Artificial Intelligence thermobArometry

Deep learning artificial neural network for P-T estimates of volcano plumbing systems using clinopyroxene composition.

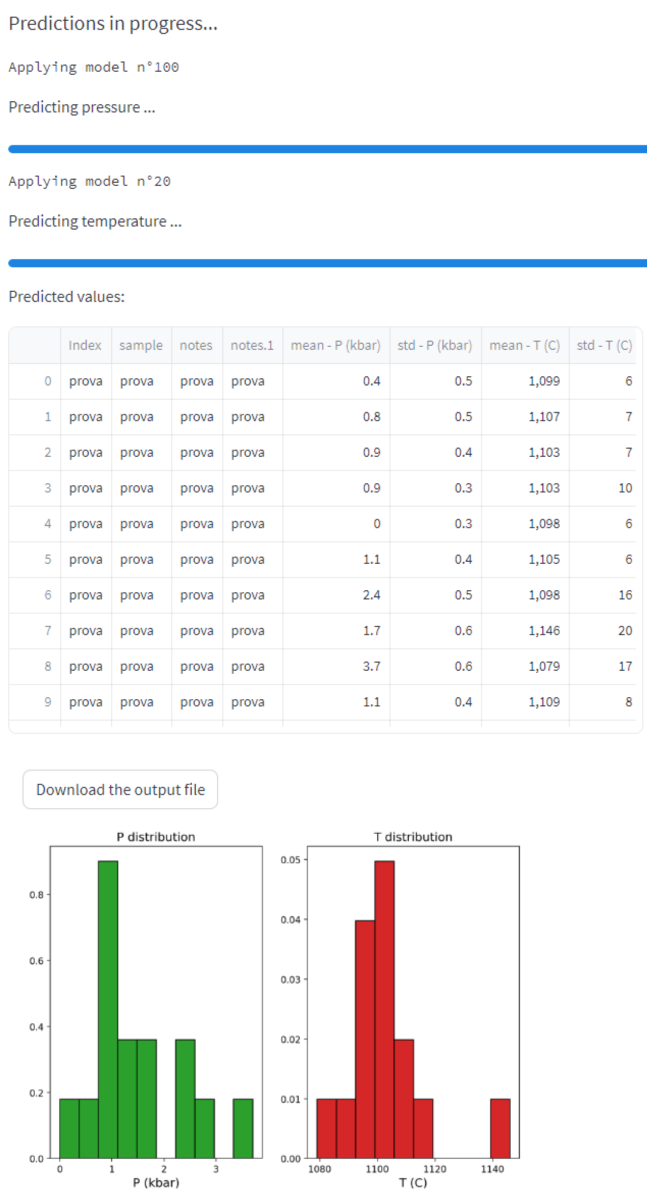
The project was born from the collaboration between the Department of Physics and Astronomy and the Department of Earth Sciences of the University of Firenze, Italy. See the info page for details on people who developed the app.

Instructions

The structure of the file to be used as input must be like the following:

	Index	sample	notes	notes.1	SiO2	TiO2	Al2O3	Cr2O3	FeO tot	MnO	NiO	MgO	CaO	Na2O	K2O	tot
0	analysis #01	cpx #1	core	eruption I	52.37	0.55	2.89	0.07	3.89	0.05	None	16.13	23.8	0.09	None	None
1	analysis #02	cpx #2	rim	eruption I	47.88	0.67	3.13	0.07	7.4	0	0.01	14.65	21.18	0.66	0.03	95.78
2	analysis #03	cpx #3	core	eruption II	49.47	0.57	3.13	0.13	7.71	0	0.09	12.93	22.08	0.46	0.04	96.64
3	analysis #04	cpx #4	rim	eruption II	49.05	0.82	4.74	None	8.71	0.34	None	12.55	23.32	0.31	None	None
4	analysis #05	cpx #5	core	eruption III	53.12	0.33	1.97	0.57	3.16	0.08	None	17.23	23.95	0.14	None	None

Figure 6.9: Home page of the web app GAIA.



Predictions in progress...

```
Applying model n°100
```

Predicting pressure ...

```
Applying model n°20
```

Predicting temperature ...

Predicted values:

Index	sample	notes	notes.1	mean - P (kbar)	std - P (kbar)	mean - T (C)	std - T (C)
0	prova	prova	prova	0.4	0.5	1,099	6
1	prova	prova	prova	0.8	0.5	1,107	7
2	prova	prova	prova	0.9	0.4	1,103	7
3	prova	prova	prova	0.9	0.3	1,103	10
4	prova	prova	prova	0	0.3	1,098	6
5	prova	prova	prova	1.1	0.4	1,105	6
6	prova	prova	prova	2.4	0.5	1,098	16
7	prova	prova	prova	1.7	0.6	1,146	20
8	prova	prova	prova	3.7	0.6	1,079	17
9	prova	prova	prova	1.1	0.4	1,109	8

Figure 6.10: The input data are processed and a file with the predictions is made. The file can be easily downloaded and two histograms, for the pressure and the temperature respectively, are shown.

6.9 Conclusions

We have developed GAIA, a new Deep Learning Feedforward Neural Network approach, to assess intensive parameters (P-T) of volcano plumbing systems requiring no assessment of equilibrium with coexisting melt (Figs. 6.1 6.2 6.4). The application of GAIA to five Italian volcanoes yielded strict and comparable depth storage conditions obtained with independent geophysical and gravimetric methods (Fig. 6.7). The pressure estimated results reinforce current models of volcano plumbing systems (e.g., Cashman et al., 2017) consisting of physically-separated reservoirs interconnected by a network of conduits channelling magma en route to the surface. Overall, we are confident that GAIA can provide a robust tool to unravel magma storage, segregation, and ascent conditions (P-T) within the plumbing system of active volcanoes. These results, coupled with the timescales of pre-eruptive processes (radiogenic isotopes, diffusion geochronometry) and volatile emissions will set further basis for one of the key research in modern volcanology, that is the prevention and mitigation of volcanic hazard.

In a subsequent development of this work, the spectral method presented in the first two chapters of this thesis was successfully used to address the same regression problem shown here. The use of the spectral method has multiple advantages. First of all, through the pruning strategy (see chapter 2), It is possible to obtain more compact networks that can be managed more easily and implemented more efficiently in the web app. Moreover, by assigning an eigenvalue to each input node in the network, you can derive an importance ranking for the input features based on the post-training eigenvalue values. This approach allows us to estimate the significance of each input element for pressure and temperature estimation.

While this remains an ongoing development, the results obtained so far suggest that employing the spectral method to tackle this problem (and regression problems in general) could be an effective strategy.

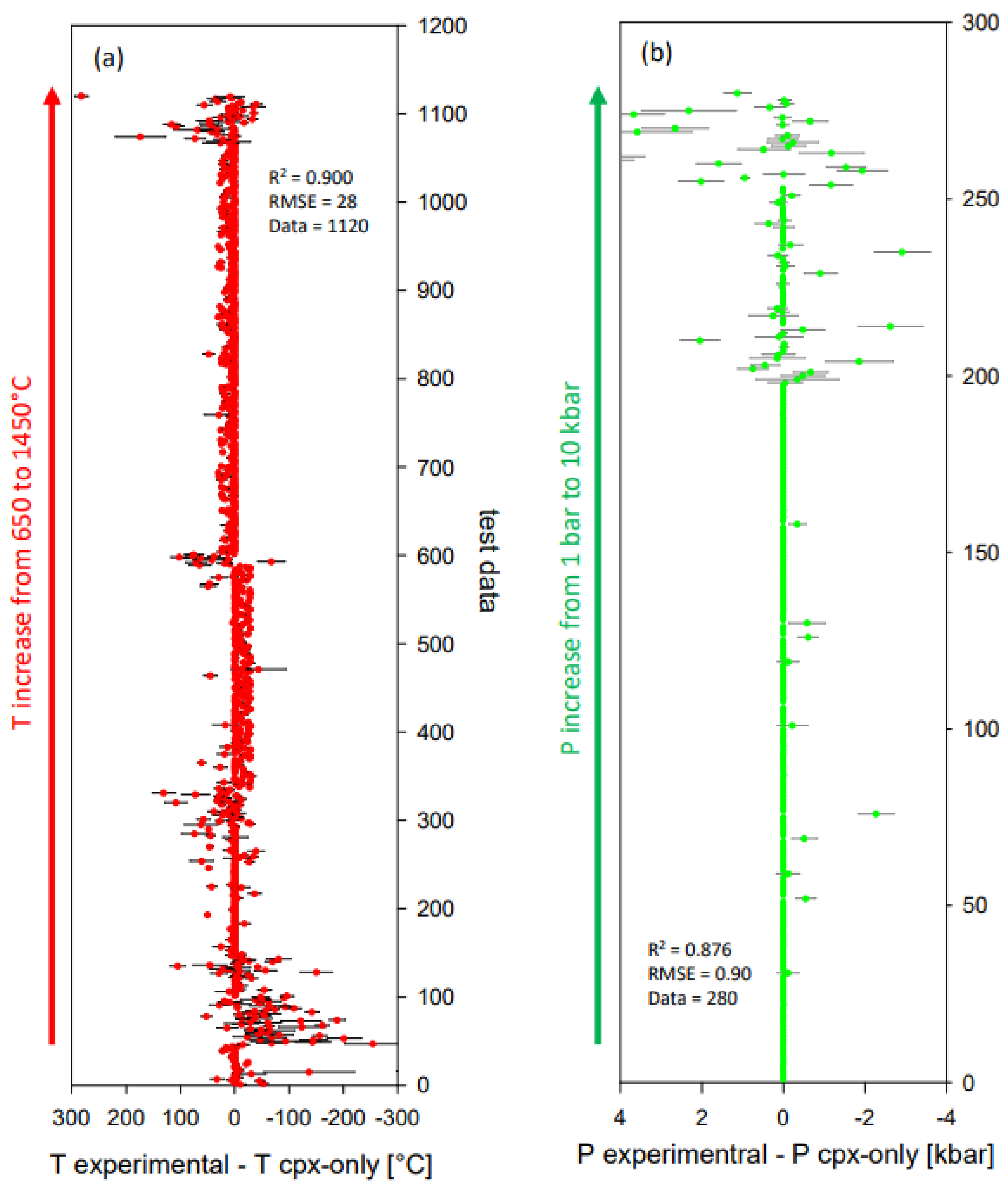


Figure 6.4: Difference between experimental intensive parameters and predicted intensive parameters of the test set using the Feedforward Neural Network method applied to clinopyroxene-only. (a) temperature, test set 20% of the total available dataset, (b) pressure, test set 5% of the total available dataset (see text). Each data point represents the average difference and associated uncertainty (1σ) computed after M independent replicas trained with the bootstrap procedure (see text). Regression statistics [R2 score, Root-Mean-Square Error (RMSE), and number of data (n)] is reported in the two insets. For the sake of clarity, the data points have been reported as sequential data from the lowest T and P values (bottom of the y-axis) to the highest T and P values (top of the y-axis) to have a better visualization of the difference between experimental and predicted T and P. This is particularly important in the case of pressure because of its localized distribution (1 bar, 2 kbar, 4 kbar, 8 kbar, 10 kbar) across the explored range (Fig. 6.5).

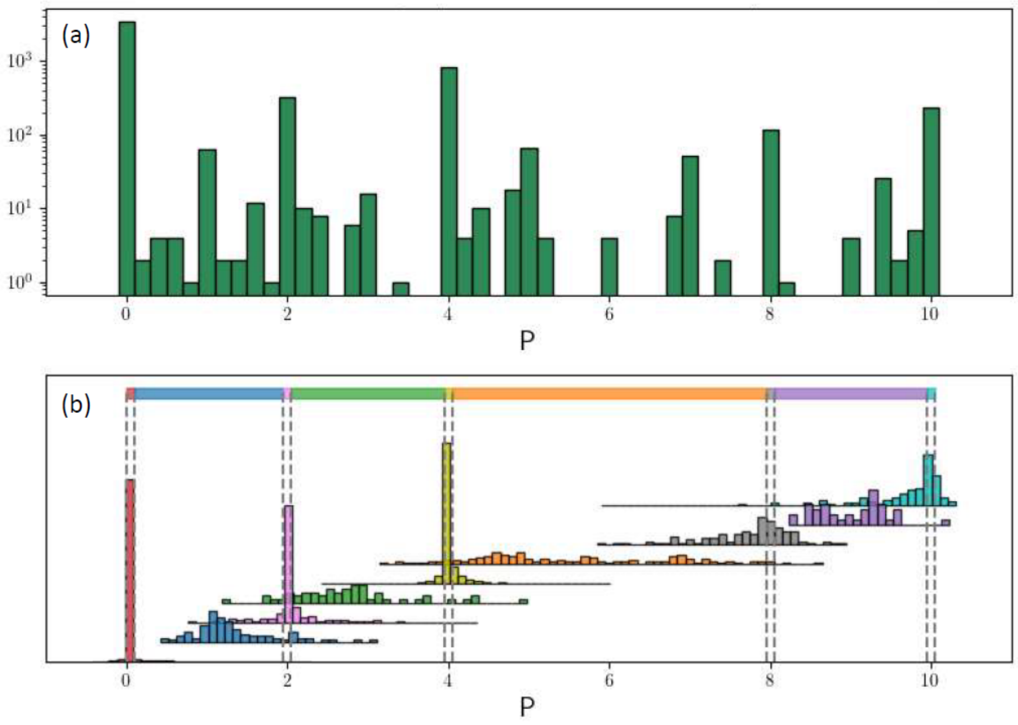


Figure 6.5: Top panel (a): distribution of the experimental pressure values used as target for the training procedure. The distribution displays a few localized peaks corresponding to the pressure values most represented within the employed dataset. Bottom panel (b): distribution of the pressure values as predicted by the Feedforward Neural Network method using the clinopyroxene-only geobarometer. Each color corresponds to a different class, as identified from direct inspection of the examined dataset (top panel). Data points sampled from the peaks (five classes representing the most represented pressure values in the data set) are correctly identified by the trained network, with a small degree of uncertainty. Data points laying in between peaks (four classes) are also traced back to their correct domain of pertinence, although with a higher degree of uncertainty. This constitutes an a posteriori proof of the ability of the Feedforward Neural Network to cope with the supplied dataset and eventually materialize in reliable predictions.

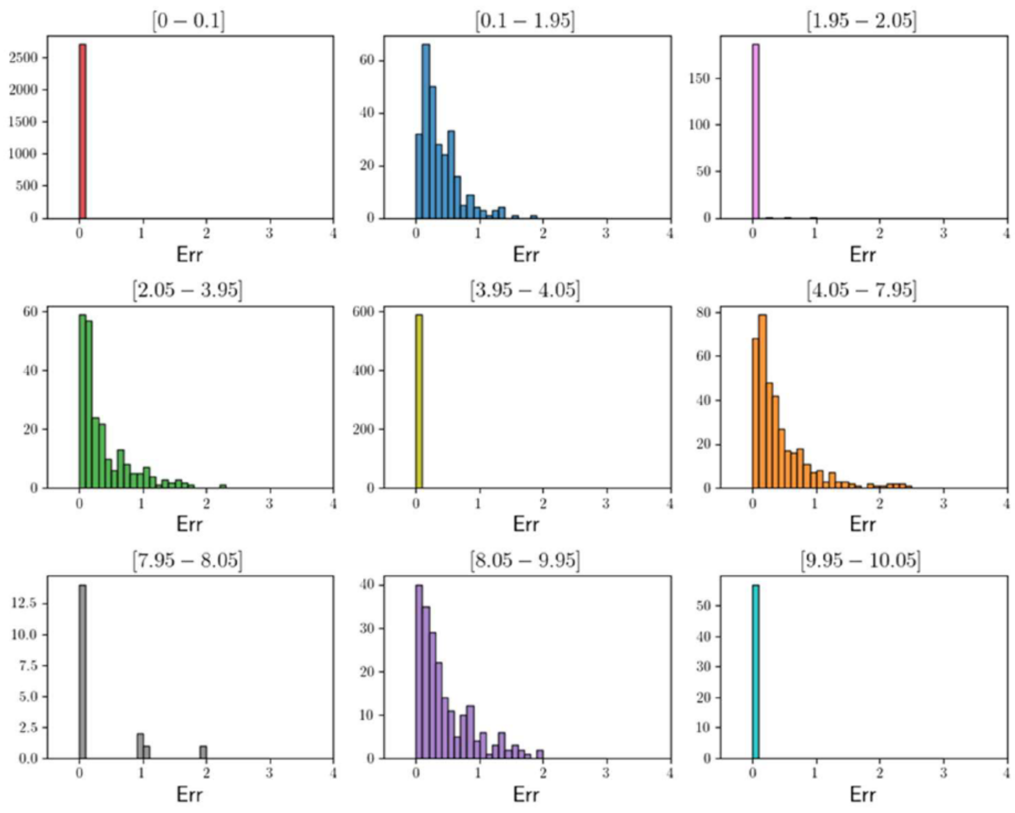


Figure 6.6: Absolute error distribution (i.e., difference between predicted and experimental pressure) of the clinopyroxene-only geobarometer for the nine different classes, as introduced in Fig. 6.5 Each data point has been associated to a class based on the values of the model prediction. Predictions that fall within the five peaks of the global distribution are associated with uncertainty << 1 kbar, whereas predictions that fall within the intermediate regions display as expected a larger, although limited, uncertainty. Overall, the average pressure uncertainty can be safely assumed to be ±1kbar.

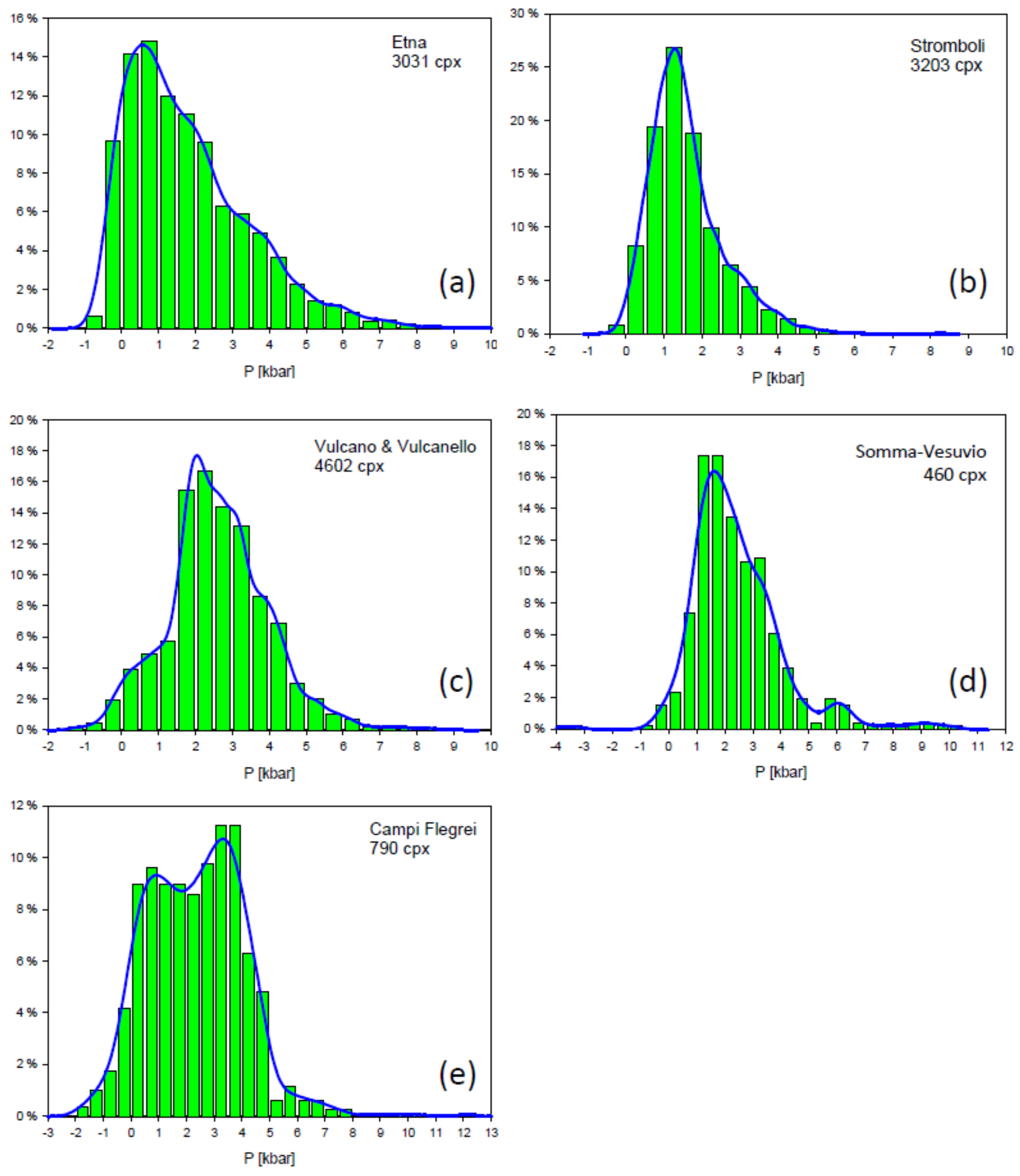


Figure 6.7: Histograms of pressure predictions from the Feedforward Neural Network method applied to the plumbing system of the five Italian volcanoes: (a) Etna, (b) Stromboli, (c) Vulcano and Vulcanello, (d) Somma-Vesuvio, (e) Campi Flegrei. The frequency of pressure prediction is every 0.5 kbar. The Kernel Density Estimates (KDEs) highlight the major peaks of pressure distribution within the plumbing system of each volcano (see text).

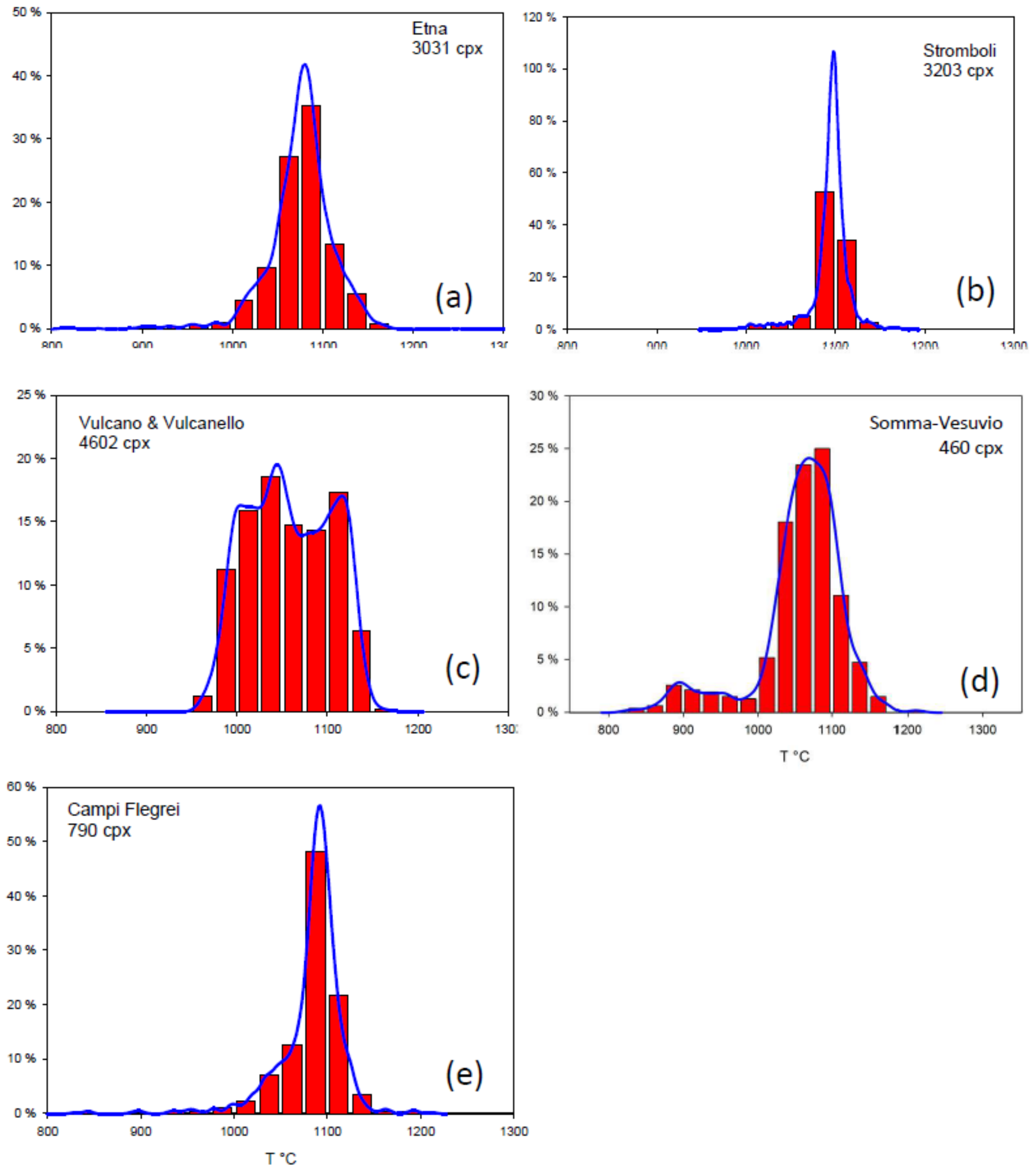


Figure 6.8: Histograms of temperature predictions from the Feedforward Neural Network method applied to the plumbing system of the five Italian volcanoes: (a) Etna, (b) Stromboli, (c) Vulcano and Vulcanello, (d) Somma-Vesuvio, (e) Campi Flegrei. The frequency of temperature prediction is every 25 °C. The Kernel Density Estimates (KDEs) highlight the major peaks of temperature distribution within the plumbing system of each volcano (see text).

Chapter 7

Conclusion

Over the course of the last three years, I have explored a few ideas for improving artificial neural network models beyond current state of the art. In particular, the spectral formulation introduced in the firts chapter has proven to be remarkably valuable. We explored how the spectral formulation can be extended by introducing additional decompositions to achieve performances akin to those obtained with standard architectures, while still employing a total number of parameters which scales linearly with the number of nodes. Moreover, through the presented extension, we have also shown how to define an effective strategy for training sparse networks.

As described in the second chapter, the introduction of the global parameters on which the spectral formulation is based provides us with strategies for identifying the fundamental nodes of a network, i.e. where the information learned during the training phase it is actually stored. The ability to identify a fundamental subnetwork dedicated to task-solving enables the removal of a significant amount of residual noise within the network. Consequently, this could enhance the interpretability of the remaining network components.

In the third and fourth chapters, we introduced a novel artificial neural network model inspired to biological systems that we termed RSN. Data processing and subsequent evaluation (classification) are performed by a discrete-time dynamic system described by a map. Each class is linked to a final state of the dynamics and this marks a distinction as compared to conventional approaches.

Finally, in the last two chapters of the thesis, we demonstrated successful applications of various neural network architectures in two distinct research areas: epidemiology and thermobarometry. Notably, the second application led to the creation of a web app (GAIA) accessible to researchers interested in thermobarometry. This app can provides valuable insights into the internal morphology of volcanoes based on geochemical analysis of rock samples.

The results found during my doctoral experience constitute encouraging evidence of the contribution that the physics community can bring to the study of the foundations of artificial intelligence. Too often, in fact, we rely on these innovative tools based only on empirical results. The scientific approach must play a fundamental role in understanding machine learning models: one of the central challenges of these years. Furthermore, as shown in this thesis, the convergence of ideas from various fields can allow the necessary change of perspective. In particular, the use of the typical concepts of the theory of networks and dynamical systems in the context of artificial neural networks has allowed the formulation of new models. These models, on one hand, offer greater control over the roles of architectural components (spectral pruning), and on the other hand, emulate to some extent the behavior of a biological neuronal system (RSN, $\mathbb{C}$-RSN).

These two latter directions may be explored further in the future and some steps forward have already been made. In particular, in [109] we show how the spectral formulation can allow us to obtain an estimate of the complexity of the target function. This is possible by applying the spectral method in the teacher-student paradigm where a randomly initialized neural network acts as a target function for another neural network. By placing a regularization on the eigenvalues of the network written in the spectral formulation, it is possible to identify a sub-structure of the network that shares the same topological properties of the target network. At the same time, we started to explore developments of the RSN and $\mathbb{C}$-RSN models by working with continuous dynamic systems, rather than discrete maps as in the case of the models described in this thesis. In these models under development it will therefore be possible to study typical properties of dynamical systems such as, for example, the characterization of the basins of attraction. Furthermore, the continuous dynamic systems to be trained will become progressively more similar to the models used in neuroscience, thus reducing the gap between computational neuroscience and artificial intelligence.

Appendix A

Improving the spectral learning

A.1 Analytical characterisation of inter-nodes weights in direct space

In the following, we will derive Eq. (1.2) as reported in the chapter 1 of the thesis. Recall that $\mathbf{A}^{(k)}$ is a $N \times N$ matrix. From $\mathbf{A}^{(k)}$ extract a square block of size $(N_k + N_{k+1}) \times (N_k + N_{k+1})$. This is formed by the set of elements $\mathbf{A}^{(k)}_{i',j'}$ with $i' = \sum_{s=1}^{k-1} N_s + i$ and $j' = \sum_{s=1}^{k-1} N_s + j$, with $i = 1, ..., N_k + N_{k+1}$, $j = 1, ..., N_k + N_{k+1}$. For the sake of simplicity, we use $\mathbf{A}^{(k)}$ to identify the obtained matrix. We proceed in analogy for $\mathbf{\Lambda}^{(k)}$ and $\mathbf{\Phi}^{(k)}$. Then:

$$
\begin{aligned}
A^{(k)}_{ij} &= \left[\mathbf{\Phi}^{(k)}\mathbf{\Lambda}^{(k)}\left(2I - \mathbf{\Phi}^{(k)}\right)\right]_{ij} \\
&= \left[2\mathbf{\Phi}^{(k)}\mathbf{\Lambda}^{(k)}\right]_{ij} - \left[\mathbf{\Phi}^{(k)}\mathbf{\Lambda}^{(k)}\mathbf{\Phi}^{(k)}\right]_{ij} \\
&= \alpha^{(k)}_{ij} - \beta^{(k)}_{ij}
\end{aligned}
\tag{A.1}
$$

From hereon, we shall omit the apex (k). Let $\lambda_1 \dots \lambda_{N_k+N_{k+1}}$ identify the eigenvalues of the transfer operator $\mathbf{A}$, namely the diagonal entries of Λ. In formulae, $\Lambda_{ij} = \sum_{j=1}^{N_k+N_{k+1}} \delta_{ij}\lambda_j$.
The quantities α_{ij} and β_{ij} can be cast in the form:

$$
\begin{aligned}
\alpha_{ij} &= 2 \sum_{k=1}^{N_k+N_{k+1}} \Phi_{ik}\lambda_k\delta_{kj} = 2\Phi_{ij}\lambda_j \\
\beta_{ij} &= \sum_{k,m=1}^{N_k+N_{k+1}} \Phi_{ik}\lambda_k\delta_{km}\Phi_{mj} = \sum_{m \in \mathcal{I}\cup\mathcal{J}} \delta_{im}\lambda_m\Phi_{mj}
\end{aligned}
$$

where $j \in \mathcal{J} = (1, ..., N_k)$ runs on the nodes at the departure layer (k), whereas $i \in \mathcal{I} = (N_k + 1, ..., N_k + N_{k+1})$. Hence, $\mathcal{I} \cup \mathcal{J} = [1, ..., N_k + N_{k+1}]$. The above expression for β_{ij} can be further processed to yield

$$
\beta_{ij} = \sum_{m \in J} \Phi_{im}\lambda_m\Phi_{mj} + \sum_{m \in I} \Phi_{im}\lambda_m\Phi_{mj} = \Phi_{ij}\lambda_j + \lambda_i\Phi_{ij}
$$

Finally we can express the difference in (A.1) as

$$\alpha_{ij} - \beta_{ij} = 2\Phi_{ij}\lambda_j - \Phi_{ij}\lambda_j - \lambda_i\Phi_{ij} = (\lambda_j - \lambda_i)\phi_{ij} \tag{A.2}$$

From the above expression, one readily obtains the sought equation, upon shifting the index i to have it spanning the interval $[1, ..., N_{k+1}]$. Recall in fact that, by definition, $\mathbf{w}$ (the matrix of the weights, see chapter (1) is a $N_k \times N_{k+1}$ matrix.

A.2 Testing the *S-SVD* and *S-QR* methods on F-MNIST and CIFAR-10 database

In the following we report on the accuracy of the *S-SVD* and *S-QR* methods when applied to the case of F-MNIST and CIFAR-10. The analysis refers to a three layers setting. The results displayed in Figs. A.1 and A.2 are in line with those discussed in the chapter (1) of this thesis.

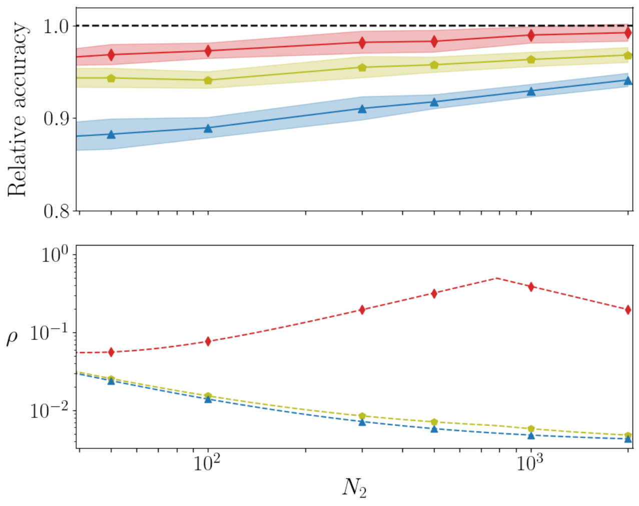


Figure A.1: **The case of F-MNIST.** Upper panel: the accuracy of the different learning strategies, normalised to the accuracy obtained for an identical deep neural network trained in direct space, as a function of the size of the intermediate layer, N_2. Triangles stand for the relative accuracy obtained when employing the spectral method (*Spectral*). Pentagons refer to the setting which extends the training to the eigenvectors' blocks via a SVD decomposition. Specifically, matrices $\mathbf{U}_k$ and $\mathbf{V}_k$ are randomly generated (with a uniform distribution of the entries) and stay unchanged during optimisation. The singular values are instead adjusted together with the eigenvalues which originate from the spectral method (*S-SVD*). Diamonds are instead obtained when the eigenvalues and the elements of matrix $\mathbf{R}$ (in a QR decomposition of the eigenvectors' blocks) are simultaneously adjusted *S-QR*). Errors are computed after 10 independent realisations of the respective procedures. Lower panel: the ratio of the number of tuned parameters (*Spectral*, *S-SVD*, *S-QR* methods) is plotted against N_2. As a reference, the best accuracy obtained over the explored range for the deep network trained with conventional means is 90%.

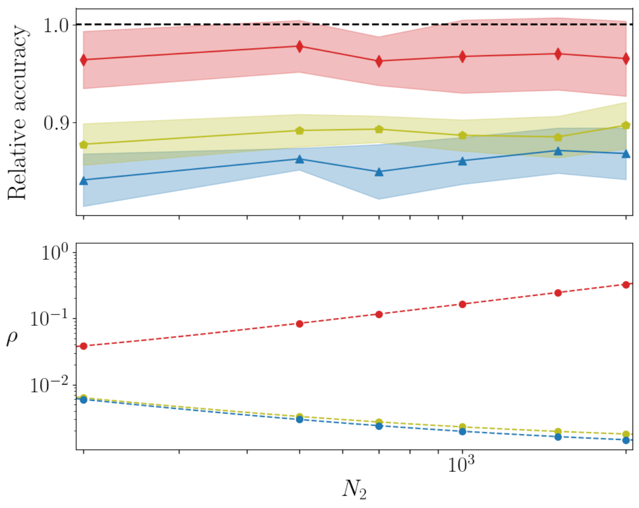


Figure A.2: **The case of CIFAR-10.** Same as in Fig. A.1. The best accuracy obtained over the explored range for the deep network trained with conventional means is 52%.

A.3 Reducing the number of trainable parameters in the S-QR method

Introduce $p \in [0, 1]$. When $p = 0$, the diagonal elements of $\mathbf{R}$ in the S-QR method are solely trained. The off-diagonal elements are instead frozen to random values. In the opposite limit, when $p = 1$ all elements of matrix $\mathbf{R}$ are assumed to be trained. Intermediate values of p interpolate between the aforementioned limiting conditions. More specifically, the entries that undergo optimisation, are randomly chosen from the pool of available the available ones, as reflecting the selected fraction. In Fig. A.3 the relative accuracy for MNIST is plotted against p. Here, the network is made of $\ell = 3$ layers with $N_2 = 500$. A limited fraction of parameters is sufficient to approach the accuracy displayed by the network trained with conventional means. In Figs. A.4 and A.5 the results relative to F-MNIST and CIFAR-10 are respectively reported.

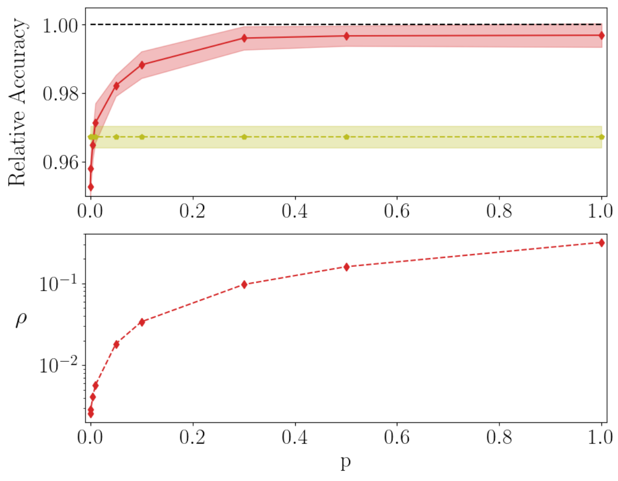


Figure A.3: **The case of MNIST.** The (relative) classification accuracy is plotted (red, diamond and solid line) against p, the probability to train the entries that populate the non null triangular part of R. The corresponding value of the relative accuracy as computed via the *S-SVD* is also reported (green, pentagons and solid lines). Here, $\ell = 3$, with $N_2 = 500$.

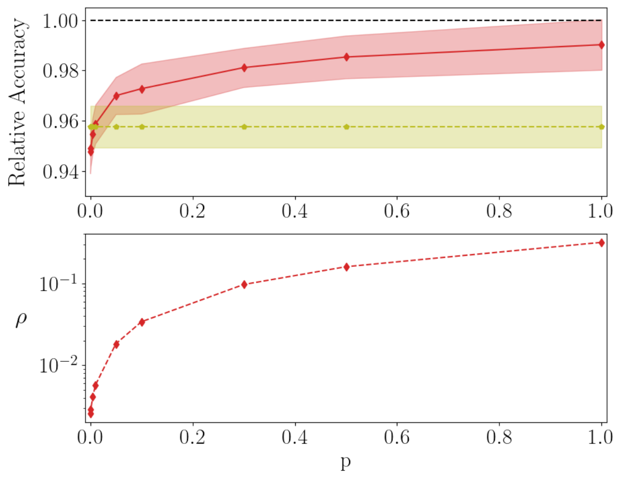


Figure A.4: **The case of F-MNIST.** As in the caption of Fig. A.3. Here, $N_2 = 500$.The averages are carried out over 10 independent realisations.

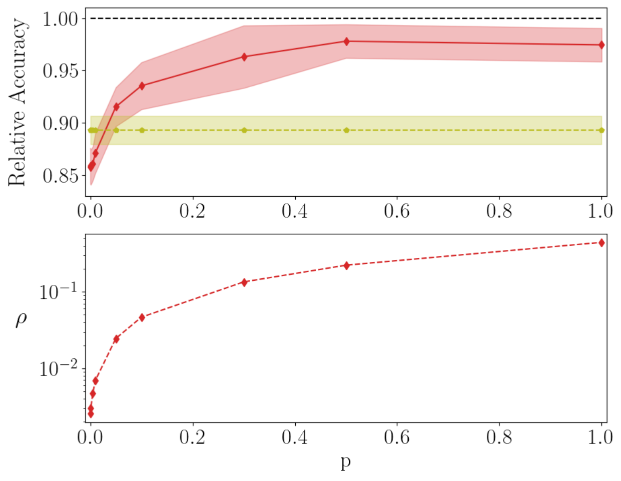


Figure A.5: **The case of CIFAR-10.** As in the caption of Fig. A.3. Here, $N_2 = 700$. The averages are carried out over 5 independent realisations.

A.4 Testing the performance of the introduced methods on a multi-layered architecture.

In this section we will test the setting of a multi-layered architecture by generalising beyond the case study $\ell = 3$ that we employed in chapter (1). More specifically, we have trained according to different modalities a four-layer ($\ell = 4$) deep neural network, by modulating $N_2 = N_3$ over a finite window. As usual, the size of the incoming and outgoing layers are set by the specificity of the examined datasets. The results reported in Fig. A.6 refer to F-MNIST and confirm that the S-QR strategy yields performance that are comparable to those reached with conventional learning approaches, but relying on a much smaller set of trainable parameters. In Fig. A.7 the effect of the imposed sparsity on the classification accuracy is displayed for both conventional and S-QR method. Similar conclusions can be reached for MNIST and CIFAR-10.

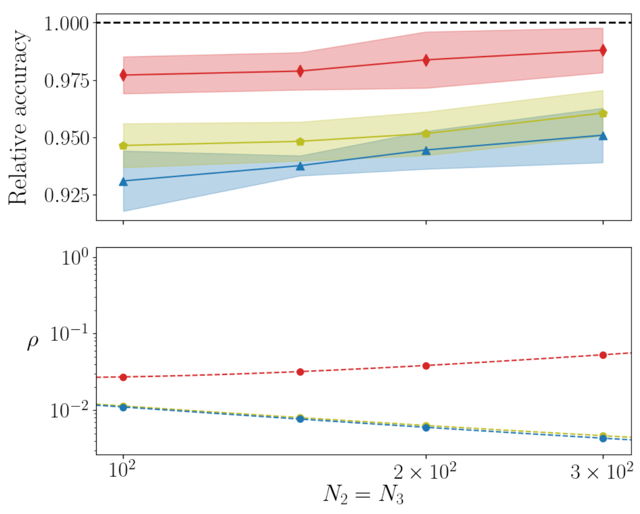


Figure A.6: **The case of a multi-layered architecture: the relative accuracy.** The relative accuracy as obtained by training a four layer network with $N_2 = N_3$ via different strategies. The symbols are as specified in Fig. A.1. The analysis refers to F-MNIST.

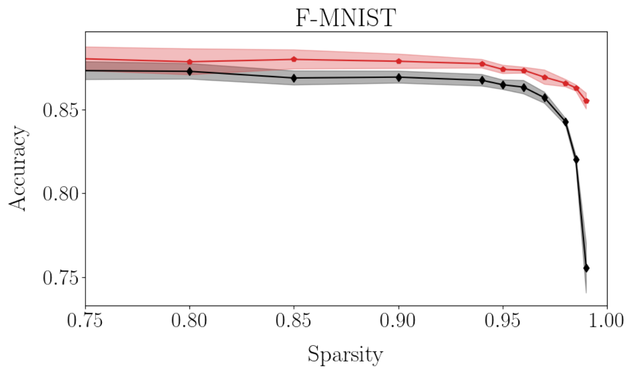


Figure A.7: **The case of a multi-layered architecture: training a sparse network.** The accuracy of the trained network against the degree of imposed sparsity. Black diamonds refer to the usual training in direct space, while red pentagons refer to the S-QR method.The analysis is carried out for F-MNIST. Here, $N_2 = N_3 = 500$.

Appendix B

Eigenvalues driven pruning

B.1 MNIST and Fashion-MNIST: single hidden layer with different activation functions.

We shall here report (see Figures B.1a, B.1b, B.1c, B.2a and B.2b) on the performance of the proposed trimming strategies, as applied to MNIST and Fashion-MNIST, for a single hidden layer architecture and beyond the setting reported in the chapter (2). In particular, we will assume (i) ELU, tanh and ReLU for MNIST (ii) tanh and ReLU activation function for Fashion-MNIST (the ELU activation was employed in the results shown in chapter (2)). Here, $N_2 = 500$, while $N_1 = 784$ and $N_3 = 10$.

B.2 MNIST and Fashion-MNIST: multiple hidden layers with different activation functions.

We will here generalize the analysis carried out in the preceding section to the case of a multilayered ($\ell > 3$) architecture (see Figures B.3a, B.3b, B.3c, B.4a and B.4b). In line with the choice operated in chapter (2, we will assume a five layered deep neural network with $N_2 = N_3 = N_4 = 500$, and $N_1 = 784$ and $N_5 = 10$.

B.3 Testing the trimming strategies on CIFAR10 dataset.

To assess the flexibility of the schemes outlined in Section III-B we here consider the CIFAR10 dataset and assume a modified MobileNetV2 [110] adding two dense layer at the end of the network. During training we freeze all the layers, except for the two appended dense layers. These latter are trained in the spectral domain. Working in this setting, the pruning is performed on the first dense layer by using strategies both (i) and (ii), as introduced in chapter (2. Here again the results are compared to those obtained when using the absolute value of the incoming connectivity as an alternative trimming criterion (see Figures B.5a, B.5b and B.5c). As a further step in the analysis, we also introduce and test a ℓ_1-norm regularization acting on the eigenvalues, so as to induce a sparse solution [111]. All experiments are performed by using a MobileNetV2 based architecture. The first dense layer is made of 512 nodes with an ELU activation function (others activation functions yield analogous results). The following regularization loss functions are considered depending on whether the training takes place in the reciprocal (spectral layer) or direct space:

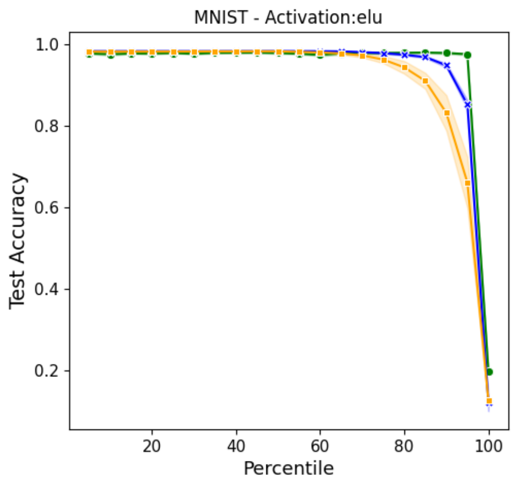
- Spectral regularization

$$L_r^{\text{spec}} = \gamma * \sum_{i=1}^{N_{\ell-1}} |\lambda_i^{(\ell-1)}|$$

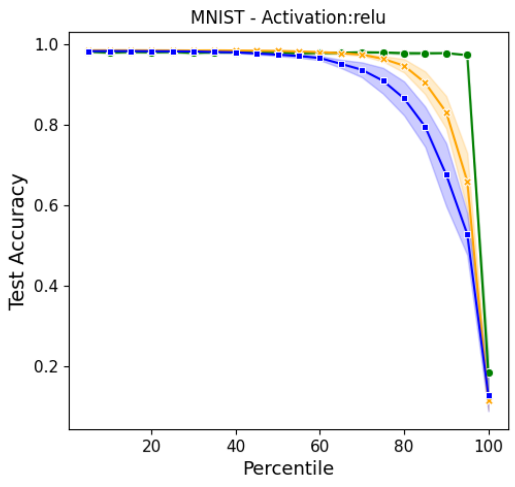
- Connectivity regularization

$$L_r^{\text{conn}} = \gamma * \sum_{i,j} |w_{ij}^{(\ell-1)}|$$

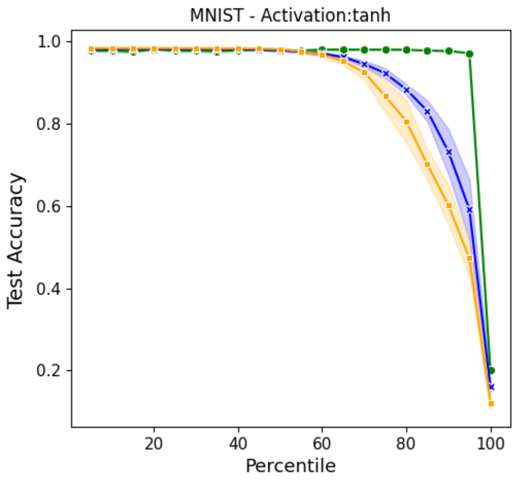
where γ stands for a suitable regularizer weight.
Clearly L_r^{conn} is equivalent to a regularization which acts on the incoming absolute connectivity. In fact, $|\sum_i |x_i|| = \sum_i |x_i|$.
The ℓ_1 regularization impacts significantly on the classification accuracy, as it can be clearly appreciated by direct inspection of Figure B.6.
Choosing the correct regularizer weight (γ), the performance of the network are stable across various range of pruning thresholds, even at the highest percentile.



(a)

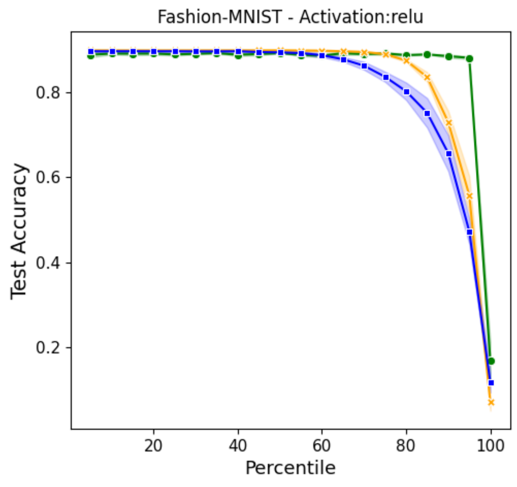


(b)

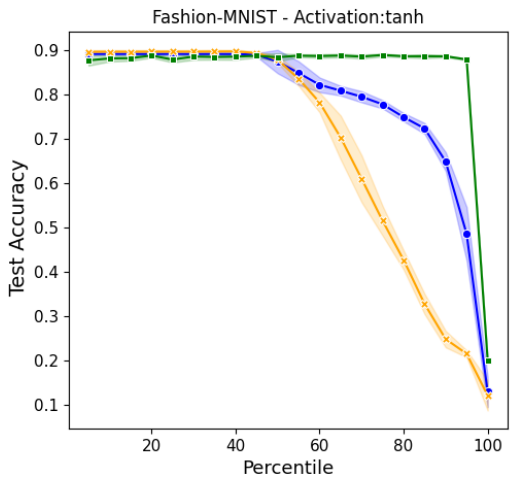


(c)

Figure B.1: Accuracy on the MNIST database with respect to the percentage of trimmed nodes (selected from the 500 neurons that compose the sole hidden layer), in a three layers feedforward architecture. The results reported in each panel refer to a different selection of the nonlinear activation functions, respectively ELU (a), ReLU (b) and tanh (c). In orange, the results obtained by using the trimming procedure based on the absolute value of the incoming connectivity. In blue, the results obtained when filtering the nodes after a full spectral training (post-training). The curve in green displays the accuracy of the trimmed networks generated upon application of the pre-training filter. In this case, the examined network is initially trained on the set of eigenvalues, while keeping the eigenvectors frozen. After having removed unessential nodes, based on their associated eigenvalues, the network undergoes another training phase that is solely targeted to adjusting the entries of the residual eigenvectors. The shadowed region represents the semi-dispersion over 5 independent realizations. When using the Relu function, trimming on the absolute value of the incoming connectivity yields slightly better results than what found when using the post-training spectral filter. The two stages spectral trimming proves always more effective.



(a)



(b)

Figure B.2: Accuracy on the Fashion-MNIST database with respect to the percentage of trimmed nodes (selected from the 500 neurons that compose the sole hidden layer), in a three layers feedforward architecture. The results reported in each panel refer to a different selection of the nonlinear activation functions, respectively ReLU (b) and tanh (c). Symbols and conclusions are in line with those reported for the case of MNIST.

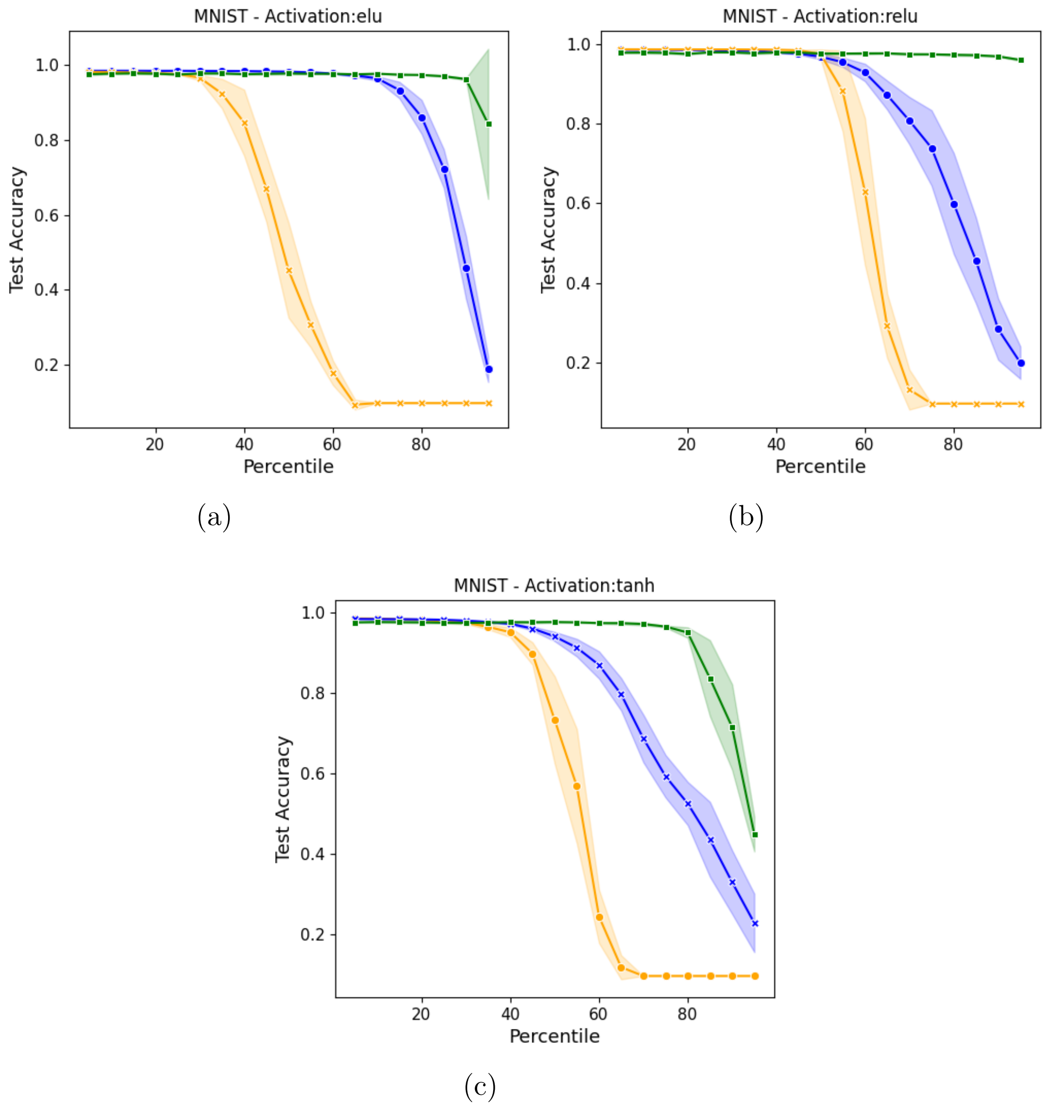
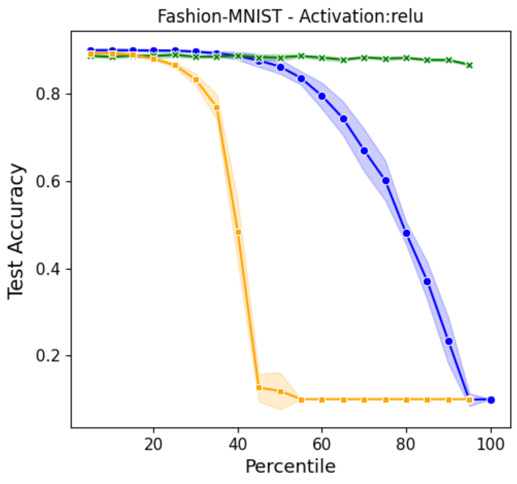
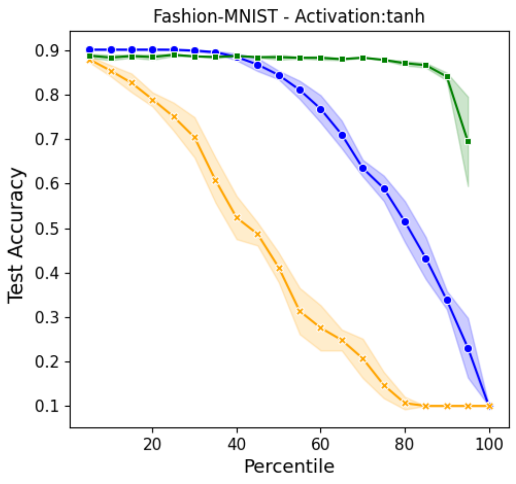


Figure B.3: Accuracy on the MNIST database with respect to the percentage of trimmed nodes (from the set of $N_2 + N_3 + N_4$ neurons). The results in each panel refer to different choices of the non linear function, ELU (a), ReLU (b) and tanh (c). Symbols are chosen as for the case of the single hidden layer setting. It should be remarked that the spectral trimming strategies proves definitely more effective than the benchmark model anchored to direct space, also when the Relu function is employed, in the case of multiple hidden layers.

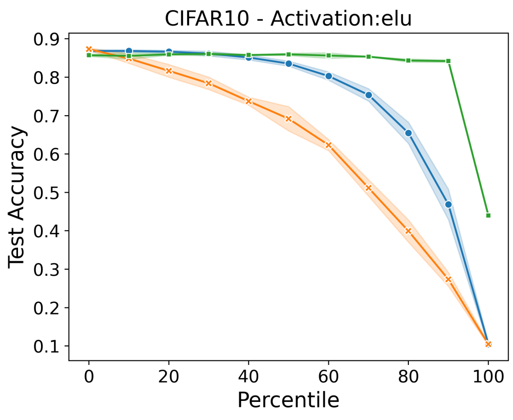


(a)

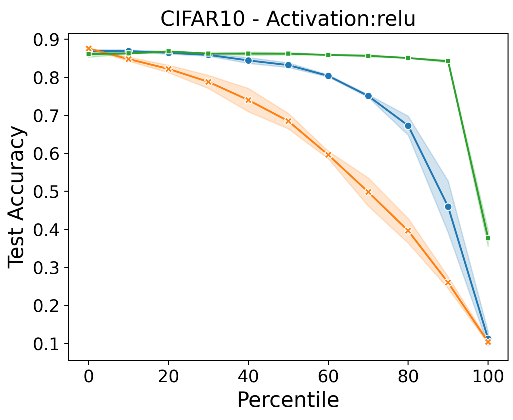


(b)

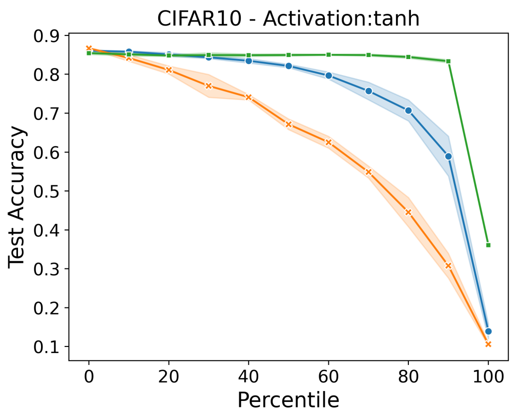
Figure B.4: Accuracy on the Fashion-MNIST database with respect to the percentage of trimmed nodes (from the set of $N_2 + N_3 + N_4$ neurons). The results in each panel refer to different choices of the non linear activation function, ReLU (a) and tanh (b). For the symbols, see the caption of the Figures above. Also in this case the spectral filters prove always superior.



(a)

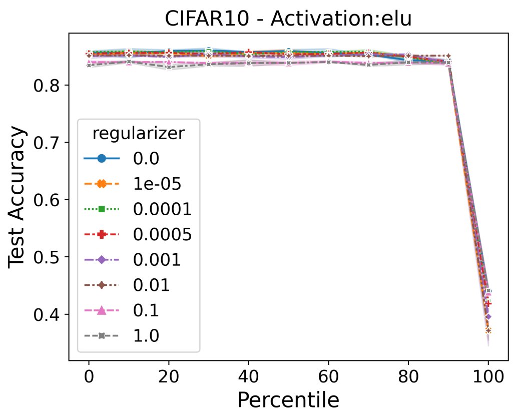


(b)

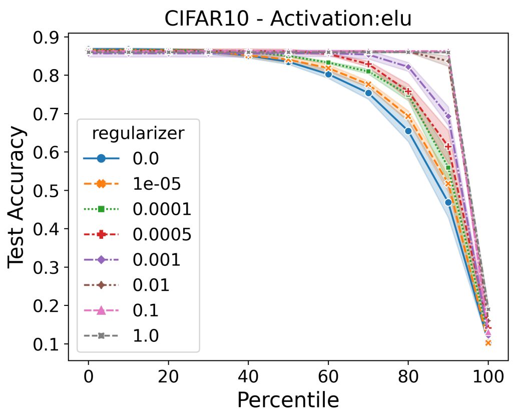


(c)

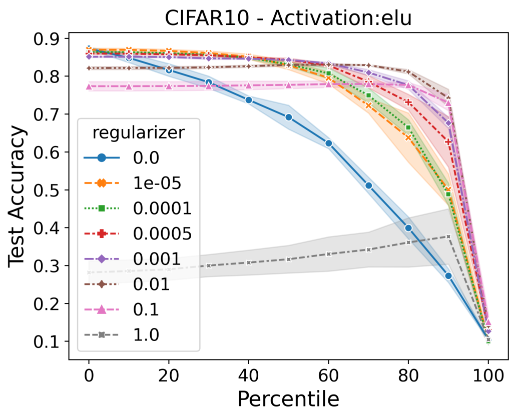
Figure B.5: Accuracy on the CIFAR10 database with respect to the percentage of trimmed nodes (from the $\ell - 1$ layer). The results in each panel refer to different non linear functions, respectively ELU (a), ReLU (b) and tanh (c). Symbols are chosen in analogy with the above (the result drawn in green are based on two different runs).



(a)



(b)



(c)

Figure B.6: Computed accuracy on the CIFAR10 dataset against the percentage of trimmed nodes (from the first of the two dense layers appended to the MobileNet-like architecture). The panels displays the performance of the network as according to each trimming procedure, and using weights (W) for the ℓ_1 regularizer. In panel (a) and (b) pre-training (based on two runs) and post-spectral filter, respectively; in panel (c) the reduction schem based on the absolute connectivity.

Bibliography

[1] L. Giambagli, L. Buffoni, T. Carletti, W. Nocentini, and D. Fanelli, "Machine learning in spectral domain," *Nature communications*, vol. 12, no. 1, p. 1330, 2021.

[2] L. Chicchi, L. Giambagli, L. Buffoni, T. Carletti, M. Ciavarella, and D. Fanelli, "Training of sparse and dense deep neural networks: Fewer parameters, same performance," *Physical Review E*, vol. 104, no. 5, p. 054 312, 2021.

[3] L. Buffoni, E. Civitelli, L. Giambagli, L. Chicchi, and D. Fanelli, "Spectral pruning of fully connected layers," *Scientific Reports*, vol. 12, no. 1, Jul. 2022. DOI: `10.1038/s41598-022-14805-7`. [Online]. Available: `https://doi.org/10.1038/s41598-022-14805-7`.

[4] L. Chicchi, D. Fanelli, L. Giambagli, L. Buffoni, and T. Carletti, "Recurrent spectral network (rsn): Shaping a discrete map to reach automated classification," *Chaos, Solitons & Fractals*, vol. 168, p. 113 128, 2023.

[5] L. Chicchi, L. Giambagli, L. Buffoni, and D. Fanelli, "Mobility-based prediction of sars-cov-2 spreading," *arXiv preprint arXiv:2102.08253*, 2021.

[6] L. Chicchi, L. Bindi, D. Fanelli, and S. Tommasini, "Frontiers of thermobarometry: Gaia, a novel deep learning-based tool for volcano plumbing systems," *Earth and Planetary Science Letters*, vol. 620, p. 118 352, 2023.

[7] D. J. Surmeier and R. Foehring, "A mechanism for homeostatic plasticity," *Nature neuroscience*, vol. 7, no. 7, pp. 691–692, 2004.

[8] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.

[9] M. Dusenberry, G. Jerfel, Y. Wen, *et al.*, "Efficient and scalable bayesian neural nets with rank-1 factors," in *International conference on machine learning*, PMLR, 2020, pp. 2782–2792.

[10] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," *arXiv preprint arXiv:1710.09282*, 2017.

[11] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[12] D. Zhang, H. Wang, M. Figueiredo, and L. Balzano, "Learning to share: Simultaneous parameter tying and sparsification in deep learning," in *International Conference on Learning Representations*, 2018.

[13] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[14] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[15] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," Citeseer, 2009.

[16] C. C. Aggarwal, *Neural Networks and Deep Learning.* Springer, 2018, pp. 15–16.

[17] M. Gabrié, A. Manoel, C. Luneau, *et al.*, "Entropy and mutual information in models of deep neural networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2019, no. 12, p. 124 014, Dec. 2019. DOI: 10.1088/1742-5468/ab3430. [Online]. Available: https://doi.org/10.1088%2F1742-5468%2Fab3430.

[18] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018.

[19] J. O. Neill, "An overview of neural network compression," *arXiv preprint arXiv:2006.03669*, 2020.

[20] Y. LeCun, B. Boser, J. S. Denker, *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[21] S. Bai, J. Z. Kolter, and V. Koltun, "Deep equilibrium models," *arXiv preprint arXiv:1909.01377*, 2019.

[22] J. Chang and J. Sha, "Prune deep neural networks with the modified $L\_\{1/2\}$ penalty," *IEEE Access*, vol. 7, pp. 2273–2280, 2018.

[23] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

[24] P. de Jorge, A. Sanyal, H. S. Behl, P. H. Torr, G. Rogez, and P. K. Dokania, "Progressive skeletonization: Trimming more fat from a network at initialization," *arXiv preprint arXiv:2006.09081*, 2020.

[25] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," *arXiv preprint arXiv:1802.05668*, 2018.

[26] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh, "Improved knowledge distillation via teacher assistant," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 5191–5198.

[27] M. Masana, J. van de Weijer, L. Herranz, A. D. Bagdanov, and J. M. Alvarez, "Domain-adaptive deep network compression," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 4289–4297.

[28] A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov, "Tensorizing neural networks," *arXiv preprint arXiv:1509.06569*, 2015.

[29] X. Yu, T. Liu, X. Wang, and D. Tao, "On compressing deep models by low rank and sparse decomposition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7370–7379.

[30] P. Stock, A. Joulin, R. Gribonval, B. Graham, and H. Jégou, "And the bit goes down: Revisiting the quantization of neural networks," *arXiv preprint arXiv:1907.05686*, 2019.

[31] R. Banner, Y. Nahshan, E. Hoffer, and D. Soudry, "Post-training 4-bit quantization of convolution networks for rapid-deployment," *arXiv preprint arXiv:1810.05723*, 2018.

[32] T. He, Y. Fan, Y. Qian, T. Tan, and K. Yu, "Reshaping deep neural network for fast decoding by node-pruning," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 245–249. DOI: `10.1109/ICASSP.2014.6853595`.

[33] X. Wang, F. Yu, L. Dunlap, *et al.*, "Deep mixture of experts via shallow embedding," in *Uncertainty in Artificial Intelligence*, PMLR, 2020, pp. 552–562.

[34] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, "Skipnet: Learning dynamic routing in convolutional networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Sep. 2018.

[35] E. Bengio, P.-L. Bacon, J. Pineau, and D. Precup, "Conditional computation in neural networks for faster models," *arXiv preprint arXiv:1511.06297*, 2015.

[36] L. Giambagli, L. Buffoni, T. Carletti, W. Nocentini, and D. Fanelli, "Machine learning in spectral domain," *Nature communications*, vol. 12, no. 1, pp. 1–9, 2021.

[37] D. J. Gauthier, E. Bollt, A. Griffith, and W. A. Barbosa, "Next generation reservoir computing," *Nature communications*, vol. 12, no. 1, p. 5564, 2021.

[38] G. Tanaka, T. Yamane, J. B. Héroux, *et al.*, "Recent advances in physical reservoir computing: A review," *Neural Networks*, vol. 115, pp. 100–123, 2019.

[39] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[40] M. McCloskey and N. J. Cohen, “Catastrophic interference in connectionist networks: The sequential learning problem,” in *Psychology of learning and motivation*, vol. 24, Elsevier, 1989, pp. 109–165.

[41] S. Lewandowsky and S.-C. Li, “Catastrophic interference in neural networks: Causes, solutions, and data,” in *Interference and inhibition in cognition*, Elsevier, 1995, pp. 329–361.

[42] R. Kemker, M. McClure, A. Abitino, T. Hayes, and C. Kanan, “Measuring catastrophic forgetting in neural networks,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[43] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, *et al.*, “Overcoming catastrophic forgetting in neural networks,” *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[44] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, “An empirical investigation of catastrophic forgetting in gradient-based neural networks,” *arXiv preprint arXiv:1312.6211*, 2013.

[45] X. Li, Y. Zhou, T. Wu, R. Socher, and C. Xiong, “Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting,” in *International Conference on Machine Learning*, PMLR, 2019, pp. 3925–3934.

[46] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[47] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016, `http://www.deeplearningbook.org`.

[48] L. Deng, D. Yu, *et al.*, “Deep learning: Methods and applications,” *Foundations and trends® in signal processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[49] F. Chollet *et al.*, “Keras: The python deep learning library,” *Astrophysics source code library*, ascl–1806, 2018.

[50] A. Sherstinsky, “Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network,” *Physica D: Nonlinear Phenomena*, vol. 404, p. 132 306, 2020.

[51] Y. Goldberg, *Neural network methods for natural language processing.* Springer Nature, 2022.

[52] L. R. Medsker and L. Jain, “Recurrent neural networks,” *Design and Applications*, vol. 5, no. 64-67, p. 2, 2001.

[53] C. M. Bishop, *Pattern Recognition and Machine Learning*, English, 1st ed. 2006. Corr. 2nd printing 2011 edition. New York: Springer, Apr. 2011, ISBN: 978-0-387-31073-2.

[54] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction.* Springer Science & Business Media, 2009.

[55] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: A review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[56] K. Chowdhary, "Natural language processing," in *Fundamentals of Artificial Intelligence*, Springer, 2020, pp. 603–649.

[57] P. Tino, C. Schittenkopf, and G. Dorffner, "Financial volatility trading using recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 12, no. 4, pp. 865–874, 2001.

[58] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: An overview," in *2013 IEEE international conference on acoustics, speech and signal processing*, IEEE, 2013, pp. 8599–8603.

[59] D. Fanelli and F. Piazza, "Analysis and forecast of covid-19 spreading in china, italy and france," *Chaos, Solitons & Fractals*, vol. 134, p. 109 761, 2020.

[60] T. Carletti, D. Fanelli, and F. Piazza, "Covid-19: The unreasonable effectiveness of simple models," *Chaos, Solitons & Fractals: X*, vol. 5, p. 100 034, 2020.

[61] A. Vespignani, H. Tian, C. Dye, *et al.*, "Modelling covid-19," *Nature Reviews Physics*, vol. 2, no. 6, pp. 279–281, 2020.

[62] S. Shastri, K. Singh, S. Kumar, P. Kour, and V. Mansotra, "Time series forecasting of covid-19 using deep learning models: India-usa comparative case study," *Chaos, Solitons & Fractals*, vol. 140, p. 110 227, 2020.

[63] J. Farooq and M. A. Bazaz, "A deep learning algorithm for modeling and forecasting of covid-19 in five worst affected states of india," *Alexandria Engineering Journal*, vol. 60, no. 1, pp. 587–596, 2020.

[64] J. Mathew, R. K. Behera, *et al.*, "A deep learning framework for covid outbreak prediction," *arXiv preprint arXiv:2010.00382*, 2020.

[65] M. U. Kraemer, C.-H. Yang, B. Gutierrez, *et al.*, "The effect of human mobility and control measures on the covid-19 epidemic in china," *Science*, vol. 368, no. 6490, pp. 493–497, 2020.

[66] C. Ilin, S. E. Annan-Phan, X. H. Tai, S. Mehra, S. M. Hsiang, and J. E. Blumenstock, "Public mobility data enables covid-19 forecasting and management at local and global scales," National Bureau of Economic Research, Tech. Rep., 2020.

[67] Y. Goldberg, "Neural network methods for natural language processing," *Synthesis Lectures on Human Language Technologies*, vol. 10, no. 1, pp. 1–309, 2017.

[68] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[69] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[70] Google, *Mobility reports*, 2020. [Online]. Available: `https://www.google.com/covid19/mobility/?hl=en`.

[71] B. Scaillet, M. Pichavant, and R. Cioni, "Upward migration of vesuvius magma chamber over the past 20,000 years," *Nature*, vol. 455, pp. 216–219, 2008. DOI: `10.1038/nature07232`.

[72] M. Kahl, S. Chakraborty, M. Pompilio, F. Costa, and O. Bachmann, "Constraints on the nature and evolution of the magma plumbing system of mt. etna volcano (1991-2008) from a combined thermodynamic and kinetic modelling of the compositional record of minerals," *Journal of Petrology*, vol. 56, pp. 2025–2068, 2015. DOI: `10.1093/petrology/egv063`.

[73] G. Saccorotti, M. Iguchi, and A. Aiuppa, "In situ volcano monitoring: Present and future," in *Volcanic Hazards, Risks and Disasters*, Elsevier Inc., 2015, pp. 169–202. DOI: `10.1016/B978-0-12-396453-3.00007-1`.

[74] C. Petrone, G. Bugatti, E. Braschi, and S. Tommasini, "Pre-eruptive magmatic processes re-timed using a non-isothermal approach to magma chamber dynamics," *Nature Communications*, vol. 7, pp. 1–11, 2016. DOI: `10.1038/ncomms12946`.

[75] K. Cashman, R. Sparks, and J. Blundy, "Vertically extensive and unstable magmatic systems: A unified view of igneous processes," *Science*, vol. 355, pp. 1–9, 2017. DOI: `10.1126/science.aag3055`.

[76] C.-H. Lin, Y.-C. Lai, M.-H. Shih, H.-C. Pu, and S.-J. Lee, "Seismic detection of a magma reservoir beneath turtle island of taiwan by s-wave shadows and reflections," *Scientific Reports*, vol. 8, pp. 1–12, 2018. DOI: `10.1038/s41598-018-34596-0`.

[77] C. Magee, C. T. E. Stevenson, S. K. Ebmeier, *et al.*, "Magma plumbing systems: A geophysical perspective," *Journal of Petrology*, vol. 59, pp. 1217–1251, 2018. DOI: `10.1093/petrology/egy064`.

[78] A. Mohamed, M. Al Deep, K. Abdelrahman, and A. Abdelrady, "Geometry of the magma chamber and curie point depth beneath hawaii island: Inferences from magnetic and gravity data," *Frontiers in Earth Science*, vol. 10, pp. 1–17, 2022. DOI: `10.3389/feart.2022.847984`.

[79] D. Rasmussen, T. Plank, D. Roman, and M. Zimmer, "Magmatic water content controls the pre-eruptive depth of arc magmas," *Science*, vol. 375, pp. 1169–1172, 2022. DOI: `10.1126/science.abm5174`.

[80] K. Putirka, "Thermometers and barometers for volcanic systems," *Reviews in Mineralogy and Geochemistry*, vol. 69, pp. 61–120, 2008. DOI: `10.2138/rmg.2008.69.3`.

[81] M. Masotta, S. Mollo, C. Freda, M. Gaeta, and G. Moore, "Clinopyroxene-liquid thermometers and barometers specific to alkaline differentiated magmas," *Contributions to Mineralogy and Petrology*, vol. 166, pp. 1545–1561, 2013. DOI: `10.1007/s00410-013-0927-9`.

[82] D. A. Neave and K. D. Putirka, "A new clinopyroxene–liquid barometer, and implications for magma storage pressures under icelandic rift zones," *American Mineralogist*, vol. 102, pp. 777–794, 2017. DOI: `10.2138/am-2017-5968`.

[83] M. Abadi, A. Agarwal, P. Barham, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[84] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, pp. 362–386, 2020. DOI: `10.1002/rob.21918`.

[85] X. Li and C. Zhang, "Machine learning thermobarometry for biotite-bearing magmas," *Journal of Geophysical Research: Solid Earth*, vol. 127, e2022JB024137, 2022. DOI: `10.1029/2022JB024137`.

[86] D. Neave et al, "Clinopyroxene–liquid equilibria and geothermobarometry in natural and experimental tholeiites: The 2014-2015 holuhraun eruption, iceland," *Journal of Petrology*, vol. 60, pp. 1653–1680, 2019. DOI: `10.1093/petrology/egz042`.

[87] M. Petrelli, L. Caricchi, and D. Perugini, "Machine learning thermobarometry: Application to clinopyroxene-bearing magmas," *Journal of Geophysical Research: Solid Earth*, vol. 125, e2020JB020130, 2020. DOI: `10.1029/2020JB020130`.

[88] O. Higgins, T. Sheldrake, and L. Caricchi, "Machine learning thermobarometry and chemometry using amphibole and clinopyroxene: A window into the roots of an arc volcano (mount liamuiga, saint kitts)," *Contributions to Mineralogy and Petrology*, vol. 177, pp. 1–22, 2022. DOI: `10.1007/s00410-021-01874-6`.

[89] C. Jorgenson, O. Higgins, M. Petrelli, F. Bégué, and L. Caricchi, "A machine learning-based approach to clinopyroxene thermobarometry: Model optimization and distribution for use in earth sciences," *Journal of Geophysical Research: Solid Earth*, vol. 127, e2021JB022904, 2022. DOI: `10.1029/2021JB022904`.

[90] M. M. Hirschmann, M. S. Ghiorso, F. A. Davis, *et al.*, "Library of experimental phase relations (lepr): A database and web portal for experimental magmatic phase equilibria data," *Geochemistry, Geophysics, Geosystems*, vol. 9, 2008. DOI: `10.1029/2007GC001894`.

[91] N. Sebe, I. Cohen, A. Garg, and T. S. Huang, *Machine learning in computer vision.* Springer Science & Business Media, 2005, vol. 29.

[92] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6645–6649. DOI: 10.1109/ICASSP.2013.6638947.

[93] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[94] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations*, 2017. DOI: 10.48550/arXiv.1412.6980.

[95] L. Francalanci, S. Tommasini, S. Conticelli, and G. R. Davies, "Sr isotope evidence for short magma residence time for the 20th century activity at stromboli volcano, italy," *Earth and Planetary Science Letters*, vol. 167, pp. 61–69, 1999. DOI: 10.1016/S0012-821X(99)00013-8.

[96] T. Ubide, J. Caulfield, C. Brandt, *et al.*, "Deep magma storage revealed by multi-method elemental mapping of clinopyroxene megacrysts at stromboli volcano," *Frontiers in Earth Science*, vol. 7, pp. 1–23, 2019. DOI: 10.3389/feart.2019.00239.

[97] G. Orsi, M. D'Antonio, and L. Civetta, *Campi Flegrei: a Restless Caldera in a Densely Populated Area.* Springer Berlin Heidelberg, 2022. DOI: 10.1007/978-3-642-85905-7.

[98] L. Francalanci, F. Lucchi, J. Keller, G. De Astis, and C. A. Tranne, "Eruptive, volcano-tectonic and magmatic history of the stromboli volcano (north-eastern aeolian archipelago)," *Memoirs*, vol. 37, pp. 397–471, 2013. DOI: 10.1144/M37.13.

[99] L. Francalanci, G. R. Davies, W. I. M. Lustenhouwer, S. Tommasini, P. R. D. Mason, and S. Conticelli, "Intra-grain sr isotope evidence for crystal recycling and multiple magma reservoirs in the recent activity of stromboli volcano, southern italy," *Journal of Petrology*, vol. 46, pp. 1997–2021, 2005. DOI: 10.1093/petrology/egi045.

[100] A. Peccerillo, M. Frezzotti, G. De Astis, and G. Ventura, "Modeling the magma plumbing system of vulcano (aeolian islands, italy) by integrated fluid-inclusion geobarometry, petrology, and geophysics," *Geology*, vol. 34, pp. 17–20, 2006. DOI: 10.1130/g22117.1.

[101] A. Bragagni, R. Avanzinelli, H. Freymuth, and L. Francalanci, "Recycling of crystal mush-derived melts and short magma residence times revealed by u-series disequilibria at stromboli volcano," *Earth and Planetary Science Letters*, vol. 404, pp. 206–219, 2014. DOI: 10.1016/j.epsl.2014.07.028.

[102] G. De Astis, F. Lucchi, P. Dellino, *et al.*, "Geology, volcanic history and petrology of vulcano (central aeolian archipelago)," *Memoirs*, vol. 37, pp. 281–349, 2013. DOI: 10.1144/M37.11.

[103] C. Petrone, E. Braschi, L. Francalanci, M. Casalini, and S. Tommasini, “Rapid mixing and short storage timescale in the magma dynamics of a steady-state volcano,” *Earth and Planetary Science Letters*, vol. 492, pp. 206–221, 2018. DOI: `10.1016/j.epsl.2018.03.055`.

[104] A. Bonaccorso, A. Cannata, R. Corsaro, *et al.*, “Multidisciplinary investigation on a lava fountain preceding a flank eruption: The 10 may 2008 etna case,” *Geochemistry, Geophysics, Geosystems*, vol. 12, 2011. DOI: `10.1029/2010GC003480`.

[105] M. Palano, M. Viccaro, F. Zuccarello, and S. Gresta, “Magma transport and storage at mt. etna (italy): A review of geodetic and petrological data for the 2002–03, 2004 and 2006 eruptions,” *Journal of Volcanology and Geothermal Research*, vol. 347, pp. 149–164, 2017. DOI: `10.1016/j.jvolgeores.2017.09.009`.

[106] A. Borzi, M. Giuffrida, F. Zuccarello, M. Palano, and M. Viccaro, “The christmas 2018 eruption at mount etna: Enlightening how the volcano factory works through a multiparametric inspection,” *Geochemistry, Geophysics, Geosystems*, vol. 21, pp. 1–15, 2020. DOI: `10.1029/2020GC009226`.

[107] M. Giuffrida, D. Scandura, G. Costa, *et al.*, “Tracking the summit activity of mt. etna volcano between july 2019 and january 2020 by integrating petrological and geophysical data,” *Journal of Volcanology and Geothermal Research*, vol. 418, p. 107 350, 2021. DOI: `10.1016/j.jvolgeores.2021.107350`.

[108] M. Giuffrida and M. Viccaro, “Three years (2011–2013) of eruptive activity at mt. etna: Working modes and timescales of the modern volcano plumbing system from micro-analytical studies of crystals,” *Earth-Science Reviews*, vol. 171, pp. 289–322, 2017. DOI: `10.1016/j.earscirev.2017.06.003`.

[109] L. Giambagli, L. Buffoni, L. Chicchi, and D. Fanelli, “How a student becomes a teacher: Learning and forgetting through spectral methods,” *NeurIPS Conference*, Dec. 2023.

[110] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2018.

[111] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski, “Optimization with sparsity-inducing penalties,” *Foundations and Trends® in Machine Learning*, vol. 4, no. 1, pp. 1–106, 2012, ISSN: 1935-8237. DOI: `10.1561/2200000015`. [Online]. Available: `http://dx.doi.org/10.1561/2200000015`.