UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

DSIP Research Unit - FBK

Master Degree in Physics of Data

Final Dissertation

Complex-Valued Deep Learning for Condition Monitoring

Thesis supervisor

Dr. Marco Cristoforetti

Thesis co-supervisor

Prof. Samir Suweis

Candidate

Mattia Pujatti

Academic Year 2021/2022

Contents

Abstract

At present, the vast majority of deep learning architectures are based on real-valued operations and representations. However, recent works and fundamental theoretical analyses suggest that complex numbers can have richer expressiveness: many deterministic wave signals, such as seismic, electrical, or vibrational, contain information in their phase, which risks being lost when studied using a real-valued model. However, despite their attractive properties and potential, only recently complex-valued algorithms have started to be introduced in the deep neural networks frameworks.
In this work, we move forward in this direction developing and implementing a coherent and working structure to train complex-valued models, remaining rigorous from a mathematical perspective but, at the same time, seeking for stability and accuracy of the training process.
As a first application of this solution, we show the results obtained applying complex-valued deep neural networks for condition monitoring in industrial applications. Different Deep Network architectures have been trained on vibrational signals extracted from sensors attached to gearmotors to detect failures. Finally, we compare the performances obtained with real and complex-valued neural networks.

Chapter 1

Introduction

1.1 Intro

Since the early years of deep learning development, researchers have tried to provide a coherent and working extent of those techniques and methodologies also in the complex domain. The first motivation was obviously that it appeared as the most "natural" extent of ordinary machine learning (literally, as Neuronal Synchrony [4] guarantees also a biological interpretation), but also thanks to more recent works and fundamental theoretical analyses suggesting that complex numbers could have richer expressiveness: many deterministic wave signals, such as seismic, electrical, or vibrational, in fact, contained information in their phase, which risked being lost when studied using a real-valued model. In the last thirty years, however, despite the large amount of papers published in this sense, proposing the most disparate theories, the research world still lack a definite complex-valued deep learning framework. The motivations of this very slow growth are various, from the theoretical difficulties of re-interpreting the properties of the complex plance in this sense, to the big shortcomings in the support of complex numbers by hardware accelerators. And all of this still holds even tho complex data arise in various practical contexts, such as medicine (e.g. Magnetic Resonance Images), communication areas (audio and radar signals) and industrial applications (vibration signals). Furthermore, thanks to a powerful instrument like the Fourier transform, we can study time series in the momentum space, which is inherently complex-valued.
In this thesis, we want to recover the most promising extents of deep learning techniques and operators to the complex domain, re-organize them and build a complex-valued deep learning framework that is both efficient and rigorous from a mathematical point of view. After a first part in which we discuss how to define structures like neurons and layers in the complex domain, considering also an alternative version of the backpropagation algorithm, we go through a practical implementation of the problem: in order to overcome the limited support of complex operators by existing deep learning frameworks (like Tensorflow and Pytorch), we decided to write our own Python library. The code is actually still quite raw, and supports only the most fundamental operations in deep learning, but is written on top of JAX, a Python library recently developed by Google DeepMind, that provides such an high level of optimization that we can partially fulfill the lack of hardware acceleration.
After the technical part, we proceed with the practical implementation and training of complex-valued models on simple datasets, focusing also on the comparison, in term of performances, with equivalent real-valued architectures. The conventional approach foresees, in presence of complex inputs, to drop the phase information in data, keeping only their magnitude, or to split them into real and imaginary parts, considered now as two independent channels: we will see that ignoring inherent relationships in complex data can drastically affect the training behavior in our models, making also an interesting study on the circularity property, peculiar of complex random variables. We will also discuss about the apparent "higher robustness" of complex-valued models to overfitting.
In the end, we will treat the problem of complex-valued deep learning in industrial applications considering the task of condition monitoring: we will show that complex models can effectively work with real world problems, guaranteeing also very nice performances.

1.2 Previous work

Several attempts have been performed during the years, in order to develop the complex framework we are looking for. The very fist trial of developing a complex backpropagation algorithm even dates back to 1991, by Nitta [5]. This extension was anything but straightforward: moving into the complex domain, in fact, you encounter a notion of differentiability that is far stronger than its counterpart in the real space. And the presence of non-holomorphic functions (non-differentiable, in the complex sense) was a huge problem. He was the first, then, to introduce the idea of exploiting Wirtinger calculus to limit the large operational complexity required to compute backpropagation of non-holomorphic functions. Several other works ([3, 6, 7]) have succeeded this one, almost all based on Wirtinger calculus, each one adding a slight improvement to the algorithm.
From now on, many papers were published in order to contribute building this new deep learning area based on complex numbers: from proposing complex multi-layer perceptrons [8], to whole new deep learning models [1], to simple new activation functions [9, 10], suitable to work in this new domain. In particular, a certain attention should be deserved by complex-valued convolutional neural networks [11], that, according to Guberman [12], are the only models able to detect meaningful phase structures in the data.
Even if there is no definitive proof that complex models are better than real, more and more clues are being collected in the most various areas: complex-valued neural networks have been successfully applied to seismic data [13], non-linear signal processing [8], MRI classification [10] and even differential privacy [14] (we did on vibration signals). Furthermore, in a very recent work by Barrachina et [15], it seemed that the circularity property, peculiar of complex-random variable, could draw a clear line among complex vs real networks. We achieved something similar in our thesis.

1.3 Thesis Organization

After providing a brief explanation of the problem in the previous introduction, the thesis is organized into two bug parts: part I is dedicated to the theoretical analysis and implementation of a complex-valued deep learning frameworks, while in part II we proceed with a real-world application of the methodology developed.
In particular, in chapter 2 we go through a brief theoretical introduction of complex analysis, paying particular attention to the notions of complex differentiability and circularity; we will also introduce the theoretical advantages that complex-valued deep learning should provide over the real counterpart.
In chapter 3, instead, we will continue with a more precise formulation of the framework we are trying to build, first with a working complex backpropagation algorithm and then with some possible extents, in the complex domain, of the most common machine learning layers and activations. In chapter 4 we added a practical implementation of the operations defined, together with the comparison with an equivalent real-valued procedure. There is also an interesting section examining the impact of the circularity quotient when training complex models.
In chapter 5, we will finally discuss and implement the framework in a real world situation, that is the problem of condition monitoring in industrial applications. In particular, we will work on a dataset of vibration signals, provided by Bonfiglioli, an important gearmotors producer, comparing the effective performances achieved with both real and complex-valued models.

—

Part I

Complex-Valued Deep Learning

Chapter 2

Complex Analysis

Introduction

At present, the vast majority of deep learning architectures are based on real-valued operations and representations. However people knew that there was a numerical domain even larger, and more general, than the real one: the complex space $\mathbb{C}$. We cannot forget also about the fact that Quantum Mechanics told us that the "true nature" of reality is inherently complex (since quantum states and operators lives in a complex Hilber space).
All of this is implicitly suggesting that actual deep learning techniques can be further extended to an even more general, and hopefully more efficient, representation. This possibility was already under the eyes of researchers from the dawn of deep learning, since the first publications covering this hypotheses are almost thirty years old.
In this chapter we are going to recall the main concepts, from complex analysis, that will be helpful during the practical extent of real-valued deep learning techniques in the complex world. We will address the main issues encountered in this development, mainly due to characteristics of the complex domain without an effective real counterpart (e.g. Liouville's and Identity theorems), but we will also focus on the property known as *circularity*, that in principle could be the key to prove the effective advantages brought by complex-valued models. In the end, we will examine the core of modern deep learning approach in presence of a complex dataset, focused on the existing mapping between numbers in $\mathbb{C}$ and points in $\mathbb{R}^2$, proving that in many cases it is inadequate, and should be replaced with a new and proper methodology living in the complex domain.

2.1 Complex Numbers

A natural way to build complex-valued neural networks is to extend real-valued models to handle complex-valued neurons. Obviously, this extension requires also the parameters (weights, biases) and the activations to be complex-valued. In contrast, the loss function should be real-valued, in order to allow for an empirical risk minimization during the training process. Despite all the operations that need to be redefined for this adaptation, the most challenging part of the problem consist into constructing a coherent and stable algorithm to train such networks.
The benefits of retaining a complex representation remains an open question; however, several works and implementations, have recently proved that complex networks could leverage mathematical properties of $\mathbb{C}$ to learn more efficient transform functions than with real-valued networks.
But before getting into the motivation and background for complex-valued deep learning, we should first recall a few concepts of *complex analysis* [1].

Definition 2.1.1. A **complex number** $z \in \mathbb{C}$ takes the form

$$z = x + iy$$

[1] A nice book to recall all the concepts explained is [16].

where $x \in \mathbb{R}$ and $y \in \mathbb{R}$ are its real and imaginary components, respectively, and $i = \sqrt{-1}$.
The same complex number may be written in exponential, or polar, form in terms of its magnitude $m \in \mathbb{R}^+$, and its phase $\theta \in \mathbb{R}$, as

$$z = me^{i\theta}$$

Magnitude and phase can then be extracted as follow:

$$m = \|z\| = (z\bar{z})^{1/2} \qquad \theta = -i \log \frac{z}{\|z\|}$$

Each complex number in $\mathbb{C}$ can be viewed as a point (x, y) in the Euclidean space $\mathbb{R}^2$; the two, however, are not isomorphic, especially because the first is a field, while the second a vector space, and also because they have distinct notions of differentiability.

Observation 2.1.1. Given two complex numbers in polar form, $z = re^{i\theta}$ and $w = se^{i\varphi}$, then

$$zw = rse^{i(\theta+\varphi)}$$

So, multiplication by a complex number corresponds to an *homothety* (rotation composed with a dilation) in $\mathbb{R}^2$.

Definition 2.1.2. The **complex conjugate** of $z = x + iy$ is defined by $\bar{z} = x - iy$ (or, equivalently, $\bar{z} = me^{-i\theta}$), and it is obtained by a *reflection* across the real axis in the plane.

The functions defined in the complex plane can be decomposed as well:

$$f : D \subseteq \mathbb{C} \to \mathbb{C} \qquad f(z) = u(x, y) + iv(x, y)$$

with $u, v : \mathbb{R}^2 \to \mathbb{R}$ begin real-valued functions. Also the notion of continuity is the same that holds for functions in $\mathbb{R}$.
Now, since the notion of *convergence* is the same for $\mathbb{C}$ and $\mathbb{R}^2$ (because absolute values in the first and euclidean distances in the second coincide), we can recover as well the triangle inequality,

$$\|z + w\| \leq \|z\| + \|w\| \qquad \forall z, w \in \mathbb{C}$$

that will hold also in the complex plane. Because of the latter relation, it is immediate that if $f : D \to \mathbb{C}$ is continuous, then the real-valued function defined by $z \mapsto \|f(z)\|$ is continuous too. We say that f attains a **maximum** at the point $z_0 \in D$ if

$$\|f(z)\| \leq \|f(z_0)\| \qquad \forall z \in D$$

with the inequality reversed for the definition of a **minimum**.
This definition will be relevant when reformulating the learning process as an optimization problem, since $\mathbb{C}$ is not an ordered field with respect to the canonical addition and multiplication. In order to compare two complex numbers, in fact, one needs first to establish a *total ordering* on the set, like the lexicographical order (which prioritizes real parts), or the polar order (which prioritizes magnitudes).
The next notions are central in complex analysis, effectively without any real counterpart.

Definition 2.1.3. Let D be and open set in $\mathbb{C}$, and f a complex-valued function on D. The function f is **holomorphic** at the point $z_0 \in D$ if the limit

$$\lim_{h \to 0} \frac{f(z_0 + h) - f(z_0)}{h} = f'(z_0)$$

that corresponds to the *derivative* of f, exists and is finite.
Equivalently, f is holomorphic on a domain $D \subseteq \mathbb{C}$ if it is for every $z_0 \in D$.

It should be emphasized that in the above limit, $h \in \mathbb{C}$ is a complex number that may approach 0 from any direction. The concept of differentiability is, in fact, much stronger in $\mathbb{C}$ than with functions of real variables: a holomorphic function will actually be infinitely many times complex differentiable, i.e., the existence of the first derivative will guarantee the existence of derivatives of any order. In the real case, instead, derivability does not necessary implies the continuity of the derivative. Furthermore, the slope of f needs to be identical for every trajectory in the complex plane through z_0 [2].
Even more is true: every holomorphic function is also *analytic*, in the sense that it has a power series expansion near every point. Again, the same does not hold in $\mathbb{R}$.

2.2 Complex Differentiability

We have already given a brief overview on the concept of complex differentiation 2.1.3. The fact is that, unfortunately, even seemingly simple functions are not holomorphic. The square modulus itself, $f(z) = \|z\|^2 = z\bar{z}$, for example.
But, under this perspective, how can we provide an efficient and coherent way to *optimize* any loss function associated to a machine learning problem? How can we exploit gradient descent if most of the times losses and neurons' activations are non-holomorphic?
A possible alternative exists: just as subgradients enable us to optimize functions that are not differentiable across their domain (e.g. ReLU), we can leverage **Wirtinger calculus** (or $\mathbb{CR}$**-calculus**) to optimize functions that are not holomorphic but still differentiable with respect to their real and imaginary components. The core idea behind this approach is basically a "change of basis" for the complex number $z = x + iy$, from the traditional representation as a vector in $\mathbb{R}^2$, i.e. $r = (x, y)^T$, to the so called **conjugate coordinates** [6]:

$$c \equiv (z, \bar{z})^T \in \mathbb{C} \times \mathbb{C}, \qquad z = x + iy \quad \text{and} \quad \bar{z} = x - iy \tag{2.1}$$

Wirtinger operators permit the construction of a differential calculus for complex-valued functions, that is entirely analogous to its ordinary real-valued counterpart. Furthermore, it allow us to avoid the expansion of the input into its real and imaginary components and the successive derivation (that would be just a simple real derivative). Those operators we were talking about are the $\mathbb{R}$**-derivative**, $\partial f/\partial z$ (computed treating z as a real variable and holding instances of $\bar{z}$ constant) and the **conjugate** $\mathbb{R}$**-derivative**, $\partial f/\partial \bar{z}$ (same but this time treating z like a constant). More formally, we can construct the following operators:

$$\frac{\partial}{\partial z} \equiv \partial_z = \frac{1}{2}\left(\partial_x - i\partial_y\right) \qquad \frac{\partial}{\partial \bar{z}} \equiv \partial_{\bar{z}} = \frac{1}{2}\left(\partial_x + i\partial_y\right) \tag{2.2}$$

In backpropagation we need to compute the derivatives of the final loss function with respect to the parameters of the network across the various layers. The task can be achieved applying the *chain rule* multiplying the upstream derivatives times the local derivatives for a specific layer function. For this reason it may be useful to write down also the chain rule for the $\mathbb{CR}$-derivatives, even if, in the end, it is just and extension of the real case (in which we consider z and $\bar{z}$ as independent random variables):

$$\frac{\partial(f \circ g)}{\partial z} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial z} + \frac{\partial f}{\partial \bar{g}}\overline{\left(\frac{\partial g}{\partial \bar{z}}\right)} \qquad \frac{\partial(f \circ g)}{\partial \bar{z}} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial \bar{z}} + \frac{\partial f}{\partial \bar{g}}\overline{\left(\frac{\partial g}{\partial z}\right)} \tag{2.3}$$

where g is a complex-valued function of z.
Rearranging the previous functions one can derive also the following identities:

$$\frac{\partial \bar{f}}{\partial \bar{z}} = \overline{\left(\frac{\partial f}{\partial z}\right)} \qquad \frac{\partial \bar{f}}{\partial z} = \overline{\left(\frac{\partial f}{\partial \bar{z}}\right)} \tag{2.4}$$

Furthermore, in the case of a real-valued output, with $f(z) : \mathbb{C} \to \mathbb{R}$, there is an additional pair of identities that hold:

$$\frac{\partial f}{\partial z} = \overline{\left(\frac{\partial f}{\partial \bar{z}}\right)} \qquad \frac{\partial f}{\partial \bar{z}} = \overline{\left(\frac{\partial f}{\partial z}\right)} \tag{2.5}$$

We will see, in the next chapter, when implementing a complex backpropagation algorithm, that these last identities will greatly simplify the computations, since the final loss function of any network must be real-valued.

2.3 Other theorems

There is a couple of further theorems that is worth to recall, that we will need to take into account in order to define a learning algorithm for complex neural networks.

Theorem 2.3.1. ***(Liouville's theorem)*** *If f is entire (holomorphic on all $\mathbb{C}$) and bounded (i.e. $\exists N \in \mathbb{R}$ such that $\|f(z)\| \leq N \ \forall z \in \mathbb{C}$), then f is constant.*

Equivalently, non-constant holomorphic functions on $\mathbb{C}$ are unbounded.
This theorem is particularly relevant when trying to define a set of activation functions suitable for the complex layers of our network. Because of this, in fact, we need to choose functions that are unbounded, otherwise they would return always a constant output, that is not desirable behavior. Moreover, a direct consequence of the Liouville's theorem is the following corollary:

Corollary 2.3.1.1. *If f is smaller or equal to a scalar times its input (i.e. $\exists M \in \mathbb{R}$, $M > 0$, such that $\|f(z)\| \leq M\|z\|$), then f is linear.*

This is another undesirable behavior, since non-linearity is an important requirement in the setup of a backpropagation algorithm.

Theorem 2.3.2. ***(Identity theorem)*** *Let $D \in \mathbb{C}$ be a domain, and f, g holomorphic functions on D. If the set $E = \{z \in D : f(z) = g(z)\}$ contains a non-isolated (i.e. accumulation) point, then $f(z) = g(z)$ for all $z \in D$.*

We reported the identity theorem because, according to [5], there are clues that a complex-valued network is able to learn any complex function just training over part of its domain.

2.4 Circularity

An important characteristic of a complex random variable is the so-called circularity property, or lack of it. Circular random variables have, in fact, vanishing pseudo-variance, index that its real and imaginary parts are statistically uncorrelated. Under this perspective, a recent work [15] have shown that, the circularity property of a dataset can significantly impact on the different performances obtained using a complex-valued model with respect to its real counterpart. At least in principle, in fact, complex-valued networks seems to benefit more of dataset presenting inherent correlations.
Let us denote the vector $\mathbf{u} \triangleq [X, Y]^T$ as the real vector built by stacking the real and imaginary parts of a complex random variable $Z = X + iY$. The probability density function of Z can be identified with the pdf of $\mathbf{u}$. The *variance* of Z is defined by:

$$\sigma_Z^2 \triangleq \mathbb{E}\left[|Z - \mathbb{E}[Z]|^2\right] = \mathbb{E}\left[|Z|^2\right] - |\mathbb{E}[Z]|^2 = \sigma_X^2 + \sigma_Y^2$$

where σ_X^2 and σ_Y^2 are respectively the variance of X and Y. However, this parameters does not bring any information about the *covariance* of Z,

$$\sigma_{XY} \triangleq \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$$

for which we need to rely on another statistical quantity defined for complex random variables, i.e. the *pseudo-variance*:

$$\tau_Z \triangleq \mathbb{E}\left[(Z - \mathbb{E}[Z])^2\right] = \sigma_X^2 - \sigma_Y^2 + 2i\sigma_{XY}$$

Unlike the variance of Z, which is always real and positive, the pseudo-variance is in general complex. We define the **circular quotient** ρ_Z as:

$$\rho_Z = \frac{\tau_Z}{\sigma_Z^2}$$

Additionally, we can define also a *correlation coefficient* among real and imaginary parts of Z:

$$\rho = \frac{\sigma_{XY}}{\sigma_X \sigma_Y}$$

In some papers, the circular quotient is defined as a covariance measure between Z and $\bar{Z}$, so among a random variable and its complex conjugate, rather than considering real and imaginary parts. We believe that those formulations in the end are equivalent from a practical point of view and so they can be used interchangeably.

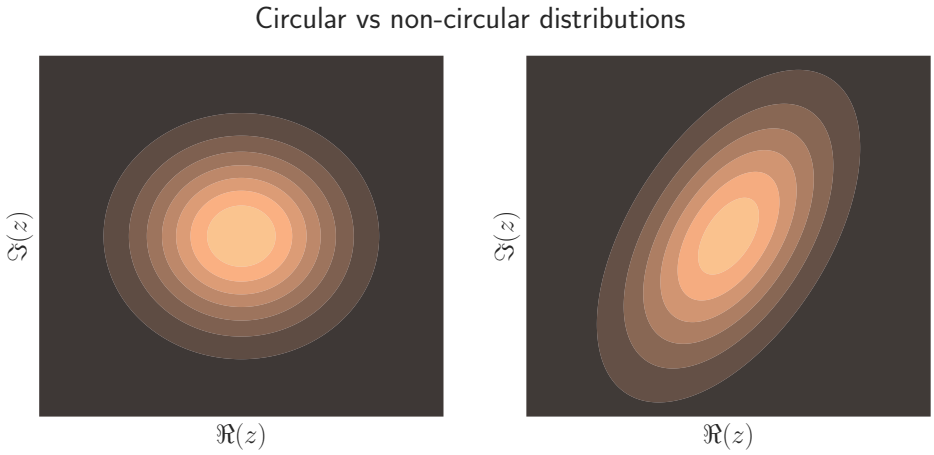


Figure 2.1: Example of a perfectly circular distribution ($\rho_z = 0$) and a non-circular one (right).

From another perspective, a complex random vector $\mathbf{Z}$ is called circularly symmetric if, for every $\varphi \in [-\pi, \pi)$, the distribution of $e^{i\varphi}\mathbf{Z}$ is the same of $\mathbf{Z}$. The vector's PDF then satisfies $f(\mathbf{Z}) = cg(|\mathbf{Z}|^2)$ for some non-negative function g and normalizing constant c. Hence, the regions of constant contours, for this distribution, are circles in the complex plane (as can be observed from figure 2.1).
Another interesting fact is that ρ_Z possess an intuitive geometrical interpretation since the modulus and phase of its principal square-root are equal to the eccentricity and angle of orientation of the ellipse defined by the covariance matrix of the real and imaginary parts of Z [17].

2.5 Why not to prefer real-valued network with two channels?

As we explained in the previous section, complex numbers can also be represented as points in $\mathbb{R}^2$, considering the "isomorphism" $\mathbb{C} \ni x + iy \leftrightarrow (x, y)^T \in \mathbb{R}^2$. Consequently, one of the main approach that are followed when a complex input is fed to a real-valued model is exactly to split such input into two independent channels, one for the real components and one for the imaginary. That's similar to what happens for the three channels of an RGB image, that are sent, one at a time independently, trough a convolutional (or fully-connected) layer.
Unfortunately, even tho this approach is efficient and coherent with the way complex numbers are managed in modern calculators (real and imaginary parts stored as separated floating point numbers), it turns out to be non-proper, even "dangerous", from an analytic point of view, since the real-valued inner product implemented in linear layers dismisses the mathematical correlation between the real and imaginary parts.

Complex Multiplication

Complex multiplication, in facts, has a specific protocol that combine the real and imaginary components of the input to the corresponding ones of the output (figure 2.2, top left). In the case of a complex input that has been split into its components, trying to reproduce the multiplication using the naive real-valued implementation of the "inner product" breaks this protocol. The reason is simply that real and imaginary parts of the output depend on the constituents of both factors.
Nevertheless, it is possible to connect the (real-valued) network's components in order to "mimic" exactly the behavior of complex multiplication: you just need to exploit *shared weights*, as in the example in figure 2.2 (on the right). However, this do not come without any cost, since $6x$ floating point operations are required (4 multiplications + 2 sums), with respect to a standard product, making this approach quite inefficient.
For a better understanding of the differences between complex and real multiplication it may be worth to give a look at figure 2.3, in which are represented the various degrees of freedom of those operations. Complex multiplication naturally has two degrees of freedom, *scaling* and *rotation* (check 2.1.1), com-

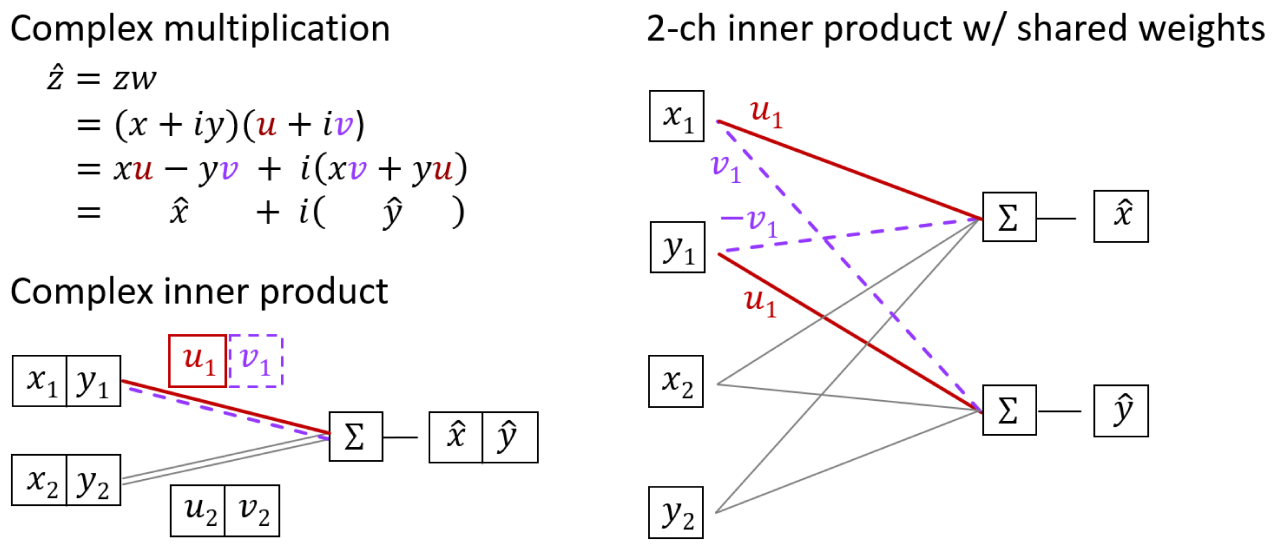


Figure 2.2: Visual representation of the multiplication and the corresponding implementation in a complex-valued network (bottom left). In order to replicate the same behavior on a real-valued network, the weights' components must be replicated, or shared, in the 2-channels layer (on the right). (source: [18])

pared to the four that characterize a product among vectors in $\mathbb{R}^2$. Let's consider the black letter "R" in the image, as the input z_{in} of a fully-connected layer in our network. This letter may change in different way depending on the multiplication protocol implemented:

- with single-channel real weights (2.3, left), the only possible effects is a re-scaling of the input;
- in the complex plane (2.3, center), as repeated several times, the multiplication depends on two parameters (magnitude and phase of the complex-valued weights) and represents and homothety in the plane;
- if the multiplication is instead implemented with a real-valued layer with two inputs and two outputs (2.3, right), there are four weights to be learned; that's will be for sure more expensive, but can guarantees two additional transformation like *reflect* and *shear* (bad or good thing depending on the dataset used).

In many applications the relative change in phase between neighboring pixels may be of importance and the absolute phase value irrelevant. Two-channel real networks that treat real and imaginary parts independently can be problematic if a certain application needs to be invariant with respect to this arbitrary global complex scaling across an image.

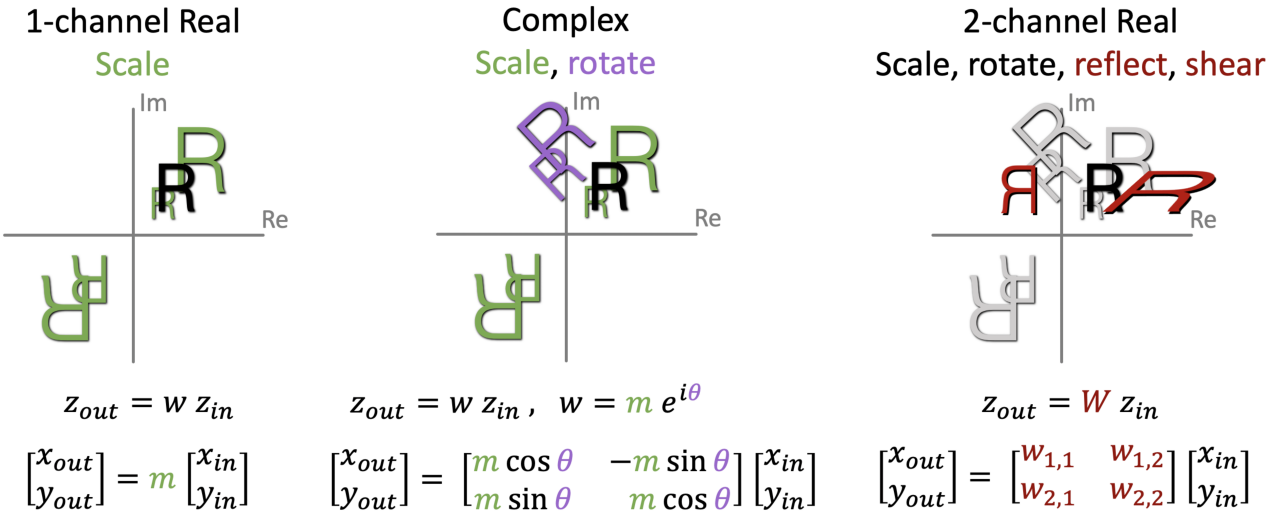


Figure 2.3: Visual representation of the complex multiplication (center) compared to its real-valued implementations (1-ch left, 2-ch right). (source: [18])

Non-Linear Activations

Not only linear operations would be affected by this complex decomposition, but also non-linear ones. In generic neural networks the main non-linear contribute is due to activation functions. But while complex multiplication can be in some way reproduced also by real-valued layers, for those non-linear activations the situation is not so straightforward. Working with a 2-channels input implies that the components will pass as independent variables trough the activation, which again dismisses eventual natural correlations between real and imaginary parts. Furthermore, when working, for example, with the canonical `ReLU` activation, one can fell into undesirable scenarios like a number with a large magnitude not allowed to pass (because of a negative component) or even unexpected rotations (because either real or imaginary parts got strongly altered). We cannot even renounce to them, since they are basically the core of a backpropagation algorithm. The only possibility, then, is to define a whole set of new activation functions, more suitable for working in $\mathbb{C}$.
Later, we will see how the choice of the activation was one of the most discussed obstacles in developing a definitive complex-valued deep learning framework.

Chapter 3

Extent

From Wirtinger calculus backpropagation to specific software implementation challenges, in this chapter are described the fundamental details of complex-valued neural network components and how they are related to existing real-valued network implementations. We will show how existing layers and functionalities can be extended to work also with a complex-valued input and which of them needs to be completely redefined.
We address the problem of re-adapting the training process building a complex backpropagation algorithm on top of many prior works, that allows for an optimization when the loss function is real-valued, thanks to Wirtinger calculus.
Furthermore, we will discuss in details the problem of building complex-valued activation functions, that was one of the main obstacles in the development of deep learning in this direction.
In the end, we will provide a brief presentation of the high level library, built on top of `JAX`, that we have realized in order simplify the setup and train of those kind of networks. Nowadays, in fact, the internet is full of deep learning libraries implementing basically every kind of known model, with different optimizations, parallelizations, etc. However, for some reason, many of them still does not provide support to complex data types: a huge obstacle in the growth of complex-valued deep learning.

3.1 Problems in the extent

Considering just their fundamental structure, complex-valued neural networks work exactly like their real counterpart, and are again constituted by neurons connected among each other (figure 3.1): the only difference is that now those neurons are complex-valued.

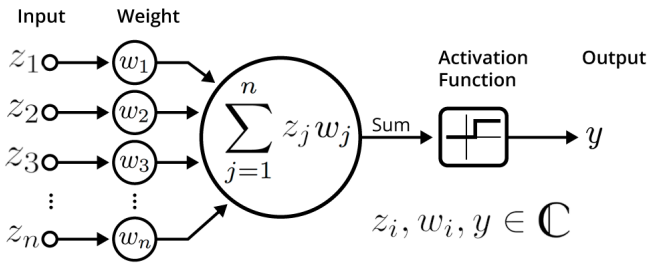


Figure 3.1: Fundamental unit (neuron) of a complex-valued neural network.

Each neuron receives a weighted input signal $\mathbf{z}$, that this time is complex valued (as the weights $\mathbf{w_i}$); this signal in summed up and added to a bias $\mathbf{b}$, and then passed through an activation function $f : \mathbb{C} \to \mathbb{C}$, that most of the times is non-linear. If we denote with the subscript ℓ the forward pass of

a neuron in the ℓ-th layer, then the output can be expressed with the following formula:

$$\mathbf{y}_\ell = f_\ell \left(\sum_{i=1}^{N} \mathbf{w_i} z_i + \mathbf{b}_\ell \right)$$

where N are the neurons in layer ℓ, M the neurons in layer $(\ell - 1)$, $\mathbf{z_{\ell-1}} \in \mathbb{C}^M$ was the output of the previous layer, $\mathbf{w}_\ell \in \mathbb{C}^{N \times M}$ and $\mathbf{b}_\ell \in \mathbb{C}^N$ are the learnable parameters of this level, f_ℓ the activation function and $\mathbf{y}_\ell \in \mathbb{C}^N$ the effective output.
However, when considering a possible extension from $\mathbb{R}$ to $\mathbb{C}$, we need to take into account a few inconveniences, since we look for a coherent and rigorous framework.

Max operator is undefined

As explained also in the introductory mathematical section, $\mathbb{C}$ is not an ordered field, in the sense that we cannot define a comparison relation among complex numbers that makes everybody agree. In principle you can define one, like the lexycographical ordering, that compares first the real part and only after the imaginary, or relying on establishing this relation among the magnitudes of those numbers. The latter is actually the preferred approach. This brief overview is important, since many non-linear functions in deep learning, like `ReLU` and `Max-Pooling` necessitate of a *maximum* operation in order to fulfill their purpose of increasing numerical stability and dimensionality reduction, respectively.

Unstable Activations

As we will see in a dedicated section, the problem of defining stable and coherent activation functions is one of the main issues that limited the development of complex-valued deep learning during the years. Complex functions, in fact, necessitate of further limitations to be suitable as activations: because of the Liouville's theorem 2.3.1, for example, they can't be limited, and neither grow too slow, otherwise their derivative would always vanish during the backpropagation. So, simply re-adapting existing activations to support complex-valued inputs, maybe redefining ambiguous operations like `max`, is not enough, especially because you need to care about the eventual loss of complex correlations if the activation is applied independently on the real and imaginary components.

Lost Probabilistic Interpretation

One nice property of real-valued neural network classifiers is the probabilistic interpretation that we can associate to its final layer, mainly due to the normalization in the range $[0, 1]$ provided by sigmoid/softmax activation functions. But now, the final output of the network will be a set of complex numbers, that we cannot interpret anymore as a probability distribution over a set of probabilistic outcomes. This nice property can be partially recovered if we add a *magnitude* layer just before the last activation: in this way we drop all the phase information but we move back to a real-valued problem. Anyway, it always depends on the final objective of the model.

Optimization of a Complex-Valued Output

Another problem related to having a complex-valued output, beside its interpretation, is the loss associated. If you have a complex final loss, how can you effectively minimize it? In the first chapter we have defined the minimum of a function defined in $\mathbb{C}$ as the point $z_0 \in \mathbb{C}$ in which its modulus is minimized. But this definition, provided by the author of that complex analysis book, is referred to a total ordering in which we refer first to the magnitudes, and so also this is just a convention. Notice that, at the end, minimizing the modulus of a complex loss is equivalent to defining a real-valued loss, i.e. $\mathscr{L} : \mathbb{C} \to \mathbb{R}$, and minimizing it. And that's exactly what we are going to do when setup a backpropagation.

After all this steps, it is clear that we cannot simply reuse deep learning architectures designed for real values without first understanding the complex mathematics in neural networks forward and backward.

3.2 Complex Backpropagation

As anticipated in the introductory section, the interest of researchers in this complex-valued deep learning area started arising many years ago; in fact, the first trial to setup a complex backpropagation algorithm even dates back to 1991 [5]. According to the author, not only its algorithm turns out to have an higher convergence speed (and the same generalization performances) with respect to its real counterpart, but also that is able to learn an entire class of transformations (e.g. rotations, similarities, parallel displacements, etc.) that the real method cannot. However, having read the work of Nitta [5], I feel it is better to remark a couple of things, mainly because many years have passed from its publication. First of all, the author used a suboptimal setup:

- the network followed one of the "conventional" approaches that we are proving to be unefficient, i.e. treating real and imaginary parts of the data as independent random variables;
- he computed the derivatives $\partial f/\partial x$ and $\partial f/\partial y$, instead of relying on Wirtinger calculus 2.2; even if this is a working alternative to ours, we will see that it is suboptimal;
- he relied on "bad" activation functions, since, as told by he himself, many times the algorithm failed to converge.

I decided to report his work because it was still one of the first and working attempts to develop a complex backpropagation algorithm, but also because of the purely theoretical analysis realized on the transformations that a complex network can learn. Nitta, managed to teach its networks several transformations in $\mathbb{R}^2$, like rotations, reductions and parallel displacements, that the corresponding real-valued model didn't make. He understood first that this was possible thanks to the higher degrees of freedom offered by complex multiplication (discussed in section 2.5). But what I believe it is even more interesting, is the relation that Nitta have found among complex-valued networks and the **Identity theorem** 2.3.2:
"We believe that Complex-BP networks satisfy the Identity Theorem, that is, Complex-BP networks can approximate complex functions just by training them only over a part of the domain of the complex functions."
This means that exploiting holomorphic functions when building a complex-valued network can sometimes impact on its generalization capabilities (since its shape will be rigidly determined by its characteristics on a small local region of its domain) [11]. Unfortunately, no additional work have been realized on this statement during the years, but I think it is an aspect deserving further attention.
In section 2.2 we have discussed about complex differentiability, and we also said that holomorphicity is not a property assured for most functions, and even simple ones, like the square modulus, can be not differentiable in the complex sense. In our architectures we have mainly two sources of *nonholomorphicity*: the loss and the activations. For reasons that will be clearer later on, boundedness and analiticity cannot be achieved simultaneously in the complex domain, and the first feature is often preferred [7].
An elegant approach that can save computational labor is the usage of Wirtinger calculus to setup optimization problems, solvable via gradient descent, for functions that are not holomorphic but at least differentiable with respect to their real and imaginary components.
Also for neural network functions we have a similar problem: requiring them to have complex differentiable properties can be quite limiting, and we should again rely on $\mathbb{CR}$-calculus. To complicate matters, the chain rule requires now to compute two terms rather than one (both the $\mathbb{R}$-derivative and the conjugate $\mathbb{R}$-derivative), causing more than $4x$ calculations during the backward pass.
"Fortunately", we can significantly reduce both memory and computation time during complex backpropagation by assuming that our final network loss function is *real-valued*. This is a strong but valid assumption since, as already discussed in section 2.1, minimizing a complex-valued function is an **ambiguous** operation. Also, minimizing the modulus of a complex loss, as suggested by the magnitude ordering introduced, in the end is exactly like optimizing a real-valued function.

But why does it turn out to be more efficient?

3.2.1 Steepest Complex Gradient Descent

In the previous sections we have widely discussed about the fact that complex-valued functions cannot be minimized, and so we ended up with the conclusion that we must use a real-valued loss in order to formulate the training process of a complex model as an optimization problem, in continuation to what happens inside real models.
Let's then consider a function $f(z) : \mathbb{C} \to \mathbb{R}$: if we decide to proceed with a gradient descent algorithm, which direction are we supposed to take since there are two different derivatives that one can compute ($\partial/\partial z$ and $\partial/\partial \bar{z}$)?
It can be proved ([3, 7] and appendix A) that the direction of the *steepest gradient descent* is the **complex cogradient**, $\nabla_{\bar{\mathbf{z}}} f$. So, given a function f that depends on a complex random variable $z \in \mathbb{C}$, the update rule that minimizes it is:

$$\mathbf{z} \leftarrow \mathbf{z} - \alpha \nabla_{\bar{\mathbf{z}}} f \tag{3.1}$$

where $\alpha \in \mathbb{R}$ is the learning rate.

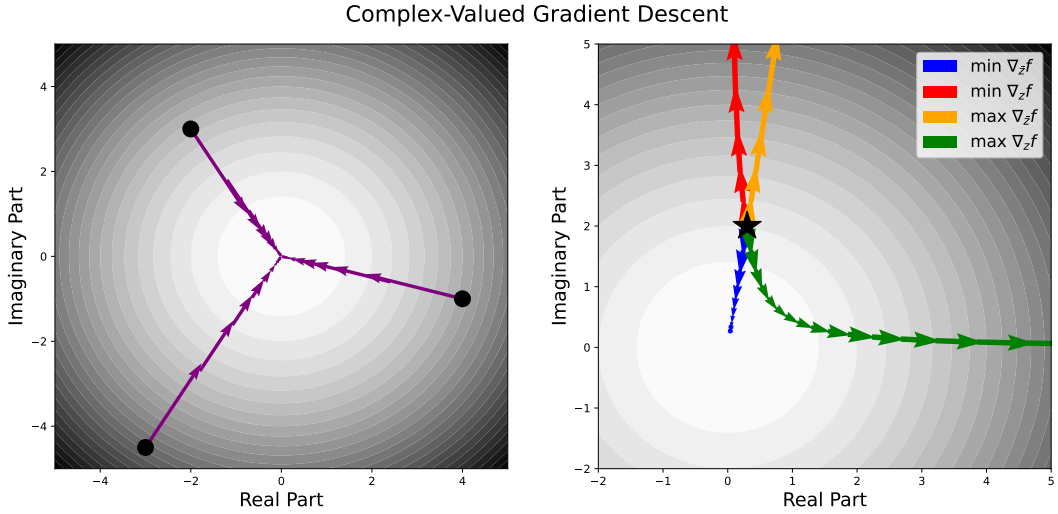


Figure 3.2: Complex Gradient Descent.

In order to provide also a visual representation, in figure 3.2 we have considered a simple, non holomorphic, real-valued function like $f(z) = z\bar{z} = \|z\|^2$, that has a unique global minimum at $z = 0 + 0j$. We have then applied the gradient descent and ascent rules in both directions of the gradient, $\nabla_{\mathbf{z}} f$, and the cogradient $\nabla_{\bar{\mathbf{z}}} f$, in order to verify what said above. In the plot we clearly see that the only direction that approaches the true minimum (starting from a random point in the dominium of f) is exactly the one determined by the complex cogradient, while the complex gradient moves in a completely wrong direction. Also considering the ascent rules we observe that the steepest direction maximizing f is again the one determined by the cogradient.

3.2.2 Backpropagation with a Real-valued Loss

With the real-valued loss assumption (proposed in 3.2) and the update rule 3.1, complex backpropagation turns out to be quite efficient and not so computationally expensive as we thought in section 3.2. The situation can be summarized with table 3.1 and figure 3.3.

Now the algorithm needs to pass only one of the two $\mathbb{CR}$ derivatives back to the earlier layers, even tho, because of the chain rule 2.3, it must compute both of them, at least for the final layer. In figure 3.3 it is effectively illustrated how input and derivatives flow through the layers during the forward and backward pass, respectively.

Standard Real Calculus	**Complex Calculus**	**Complex Calculus, assuming real-valued loss**
Input to layer $\ell+1$:		
$\frac{\partial f_L}{\partial x_\ell}$	$\frac{\partial f_L}{\partial z_\ell}$ and $\frac{\partial f_L}{\partial \bar{z}_\ell}$	$\frac{\partial f_L}{\partial \bar{z}_\ell}$
Output from layer ℓ:		
$\frac{\partial f_L}{\partial x_{\ell-1}} = \frac{\partial f_L}{\partial x_\ell}\frac{\partial f_\ell}{\partial x_{\ell-1}}$	$\frac{\partial f_L}{\partial z_{\ell-1}} = \frac{\partial f_L}{\partial z_\ell}\frac{\partial f_\ell}{\partial z_{\ell-1}} + \frac{\partial f_L}{\partial \bar{z}_\ell}\overline{\left(\frac{\partial f_\ell}{\partial \bar{z}_{\ell-1}}\right)}$	
	$\frac{\partial f_L}{\partial \bar{z}_{\ell-1}} = \frac{\partial f_L}{\partial z_\ell}\frac{\partial f_\ell}{\partial \bar{z}_{\ell-1}} + \frac{\partial f_L}{\partial \bar{z}_\ell}\overline{\left(\frac{\partial f_\ell}{\partial z_{\ell-1}}\right)}$	$\frac{\partial f_L}{\partial \bar{z}_{\ell-1}} = \overline{\left(\frac{\partial f_L}{\partial \bar{z}_\ell}\right)}\frac{\partial f_\ell}{\partial \bar{z}_{\ell-1}} + \frac{\partial f_L}{\partial \bar{z}_\ell}\overline{\frac{\partial f_\ell}{\partial z_{\ell-1}}}$

Table 3.1: Comparison of backpropagation calculus. (source: [18])

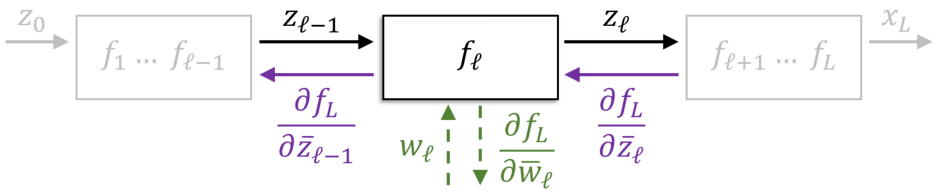


Figure 3.3: Forward and backward pass through a complex layer. (source [18])

Given the steepest complex gradient descent rule 3.1, we can then write down the equivalent expression for our network function depending on a set of parameters $\mathbf{w}$:

$$\mathbf{w}_n \leftarrow \mathbf{w}_n - \alpha \frac{\partial f}{\partial \bar{\mathbf{w}}_n}$$

3.3 Re-definition of the main neural network layers

We have started writing down this chapter with the purpose of building a working and coherent framework for complex-valued machine learning. And we also In section 3.1 we have analyzed the main obstacles that we encounter during this extension, while in 3.2 we

Fully-Connected Layers

For the fundamental layers of a neural network the extension is quite trivial. Consider a layer with K complex-valued units, a weight vector $\mathbf{w} \in \mathbb{C}^K$ and a bias term $\beta \in \mathbb{C}$; then, the response to a certain input vector $\mathbf{z} \in \mathbb{C}^N$ (given also the activation function f) is:

$$\mathbf{y} = f(\mathbf{z}, \{\mathbf{w}, \beta\}) = f\left(\sum_k z_k w_k + \beta\right) = f\left(\mathbf{z}^T\mathbf{w} + \beta\right)$$

So it is just like the real case, with the only difference that now the multiplication is in $\mathbb{C}$ and not in $\mathbb{R}$, with all the consequences already discussed in 2.5.

What we should care about is, instead, the initial values of the networks' parameters. Proper initialization can, in principle, help reducing the risk of vanishing or exploding gradients. Conventionally, researchers follow the approaches proposed by Glorot [19] or by He [20] (the latter designed specifically for `ReLU` activations), with the final objective of ensuring that the variance of input, output and their gradients are the same. According to these two methods, weights should be initialized with a normal distribution (or truncated-normal) with zero mean and standard deviation that depends on the number of units in that specific layer. Thanks to Trabelsi [1] (derivation in appendix A.1) we could provide two equivalent procedures, but in the complex domain.
To put in place them, we need first to consider the weights in polar form, i.e. $\|\mathbf{w}\|e^{i\theta}$, and then:

- random sampling the magnitude according to a **Rayleigh distribution** with parameter $\sigma = 1/\sqrt{n_{in} + n_{out}}$ (`Complex Xavier` initialization) or $\sigma = 1/\sqrt{n_{in}}$ (`Complex He` initialization);
- random sampling the phases according to a **uniform distribution** in $[-\pi, \pi$‘].

The biases β, instead, can all be initialized simply to 0, or uniformly in a small interval $[-\varepsilon, \varepsilon]$.

Convolutional Layers

In order to perform the equivalent of a traditional real-valued 2D convolution in the complex domain, we convolve a complex filter matrix $\mathbf{W} = \mathbf{A} + i\mathbf{B}$ by a complex vector $\mathbf{h} = \mathbf{x} + i\mathbf{y}$, where $\mathbf{A}, \mathbf{B}$ are real matrices and $\mathbf{x}, \mathbf{y}$ are real vectors, since we are simulating complex arithmetic using real-valued entries. As the convolution operator is distributive, we have:

$$\mathbf{W} * \mathbf{h} = (\mathbf{A} * \mathbf{x} - \mathbf{B} * \mathbf{y}) + i\,(\mathbf{B} * \mathbf{x} + \mathbf{A} * \mathbf{y}) \tag{3.2}$$

This is a very nice behavior, since the a complex convolution can be decomposed into two real-valued independent operations. And this means that we can exploit already existing algorithms (like this one, from Gauss [1]) to perform efficiently this (already expensive) computation.
Notice also that complex convolutional layers' weights basically, can learn to rotate the phase of desirable data toward the positive real axis; in fact, if we rewrite 3.2 in a matrix form:

$$\begin{bmatrix} \Re(\mathbf{W} * \mathbf{h}) \\ \Im(\mathbf{W} * \mathbf{h}) \end{bmatrix} = \begin{bmatrix} \mathbf{A} & -\mathbf{B} \\ \mathbf{B} & \mathbf{A} \end{bmatrix} * \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}$$

In figure 3.4 we can observe a visual representation of the complex convolution proposed by [1].

In principle, however, even tho complex convolution should have the same computational cost as its real-valued counterpart, in practice it is four times more expensive, just because complex multiplication in the worst case requires four products and two additions.

Pooling Layers

The pooling operation involves sliding a n-dimensional filter over each channel of the feature map and summarizing the features lying within the region covered by the filter. Pooling layers are essentially dimensionality reduction levels inside the network, with the purpose of making the model more robust to positional variations in the input.
There are mainly two kinds of pooling operations that we want to extend to the complex domain:

- `Average Pooling`: replaces the elements in the filtered region of the feature map with their mean. The average of a set of complex number is a well defined operation and so no further work is needed in this case.
- `Max Pooling`: replaces the elements in the filtered region of the feature map with their maximum. In this case, instead, we have that the maximum operation is ambiguous in $\mathbb{C}$, and so we must establish an ordering to re-define this layer. Our choice that was to setup the max pooling operation to return the complex number with the highest magnitude, inside the region covered by the filter. Namely:

$$\text{MaxPool}(\{z_k\}) = z_n \qquad \text{where } n = \underset{k}{argmax}\ \|z_k\|$$

[1] As explained in this brief Wikipedia section, we can reduce the cost of complex multiplication, in some cases.

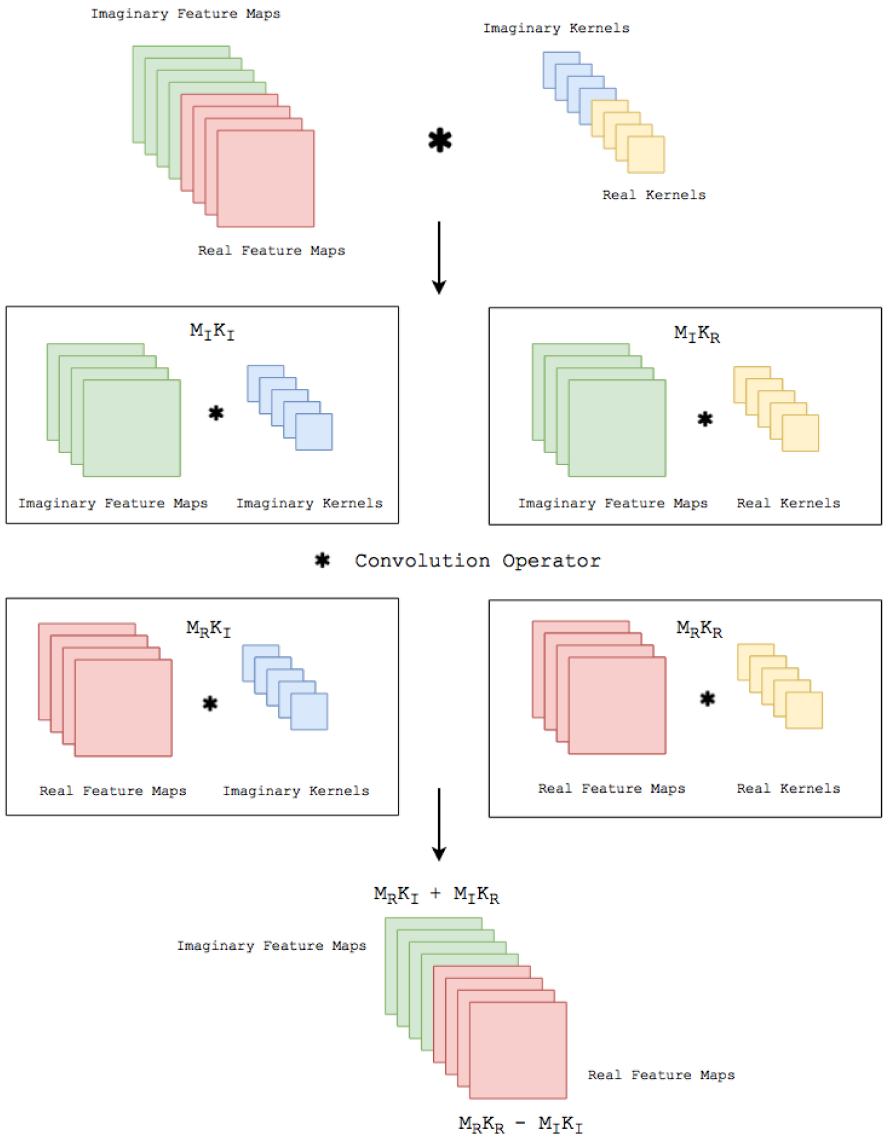


Figure 3.4: Implementation details of the Complex Convolution (by [1]).

This choice is not unique, since, as repeated several times, there are more than one total ordering for $\mathbb{C}$.

Normalization Layers

Normalization layers are usually inserted inside the architecture of a neural network with the purpose of improving the stability of the network optimization, especially in cases of an high depth, and for a better control of the gradients. In this case the extension not so straightforward, since you need to re-define several statistical objects.
In standard `Batch-Normalization` we train two internal parameters, a scale and an offset, such that each batch of data has zero mean and standard deviation one. This approach, however, is valid only for real data, since it does not guarantee equal variance for both the real and imaginary components, with a resulting distribution turning out to be non-circular. Following the method proposed by [1], the idea is to normalize data to obtain a standard complex normal distribution (basically the one in figure **??**), achieved by multiplying the zero-centered data $(\mathbf{z} - \mathbb{E}[\mathbf{z}])$ by the inverse square root of

their $2x2$ covariance matrix $\mathbf{V}$ [2]:

$$\tilde{\mathbf{z}} = (\mathbf{V})^{-\frac{1}{2}} (\mathbf{z} - \mathbb{E}[\mathbf{z}]) \qquad \text{where} \quad \mathbf{V} = \begin{pmatrix} \text{Cov}(\text{Re}\{\mathbf{z}\}, \text{Re}\{\mathbf{z}\}) & \text{Cov}(\text{Re}\{\mathbf{z}\}, \text{Im}\{\mathbf{z}\}) \\ \text{Cov}(\text{Im}\{\mathbf{z}\}, \text{Re}\{\mathbf{z}\}) & \text{Cov}(\text{Im}\{\mathbf{z}\}, \text{Im}\{\mathbf{z}\}) \end{pmatrix} \tag{3.3}$$

From a practical point of view, the implementation is the same of the real case. You have again two trainable parameters: $\beta \in \mathbb{C}$, i.e. the complex mean, and $\gamma \in \mathbb{C}^2$, the complex-valued positive-defined covariance matrix. The `complex batchnorm` operation is then defined as

$$BN(\mathbf{z}) = \gamma \mathbf{z} + \beta$$

We need, however, to be careful when relying on batchnormalization. This procedure allows, in fact, to avoid co-adaptation between real and imaginary parts of data, effectively reducing the risk of overfitting. But there is a cost to this, since you are basically decorrelating your complex data, partially losing the advantage over two-channels networks [21].
For this reason, we sometimes prefer to rely on other kind of normalization layers that, instead, allows to preserve complex data correlations.
In simple `Complex Normalization` we scales a complex scalar input $\mathbf{z}$ such that its magnitude is set to one, while the phase remains unchanged. In practice we project $\mathbf{z}$ onto the unit circle. The forward pass is then

$$\hat{\mathbf{z}} = e^{i\angle \mathbf{z}} = \frac{\mathbf{z}}{\|\mathbf{z}\|} = \frac{\mathbf{z}}{(\mathbf{z}\bar{\mathbf{z}})^{1/2}}$$

Other Layers

There are many layers that do not need any further re-definition to work also in the complex domain: `Dropout`, Pad or Attention layer, for example. There are also many other structures that should be re-derived (e.g Recurrent layers, LSTM, etc.), but that were out of our scope and so we haven't examined. This should be interpreted just as a starting point in the development of an higher level complex-valued deep learning framework.

3.4 Complex-Valued Activation Functions

One of the main issues encountered in the last 30 years in the developing of a complex-valued deep learning framework was exactly the definition of reliable activation functions. The extension from the real-valued domain turned out to be quite challenging: because of the Liouville's theorem 2.3.1, in fact, stating that the only complex-valued functions that are bounded and analytic everywhere are constants, one have necessarily to choose between boundedness and analyticity, in the design of those activations. Furthermore, before the introduction of ReLU, almost all the activation functions known in the real case were bounded. And for the same ReLU the extent was not trivial, since operations like *max* are not defined in the complex domain, Additionally, with complex-valued outputs, we have lost the probabilistic interpretations that functions like `sigmoid` and `softmax` used to provide.
We have to say, however, that most of the candidate functions that have been proposed, have been developed in a split fashion, i.e. by considering the real and imaginary parts of the activation separately. But, as discussed also in the previous chapter, this approach should be abandoned, since you risk losing the complex correlations stored in those variables.
In this section, we will explore a few complex-valued activations proposed during the years: first with the ones that are direct extensions of their real counterparts, and then with more "abstract" candidates, that have more reasons to live and work in the complex domain.

[2] The existence of the inverse matrix is guaranteed by the positive (semi-) definiteness of $\mathbf{V}$. Eventually, you can enforce this condition by adding a small quantity $+\varepsilon\mathbb{I}$ to the matrix (Tikhonov regularization).

Extent from the real case

The first class of viable approaches consists into barely extending real-valued activations in the complex domain, like the `sigmoid` and the `hyperbolic tangent`:

$$\sigma_{\mathbb{C}}(z) = \frac{1}{1+e^{-z}} \qquad \tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$$

These functions are fully-complex, and also analytic and bounded almost everywhere, at the cost of introducing a set of singular points: both function, in fact, seems to diverge periodically in the imaginary axis of the complex plane 3.5. Limiting and scaling carefully the input values seems to help avoiding then those singularities and so partially containing the instability. However, I believe that there are simpler and more efficient alternatives.

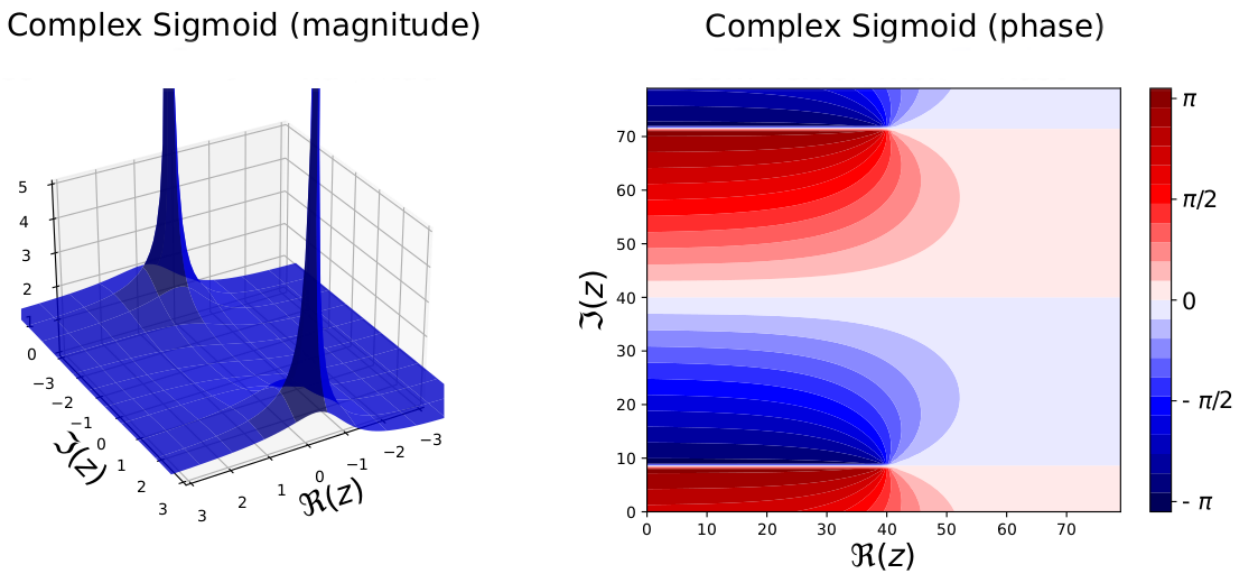


Figure 3.5: Sigmoid activation function extended in the complex plane. The magnitude (left) goes to $\pm\infty$ at $i\pi(2N-1)$. These singularities may also be seen in the output phase (right).

Separable Activations

As already explained, the main tendency in the development of complex-valued activation functions was basically getting back the "old" designs for real-valued models and using them independently on the real and imaginary components of the input.
This can be done easily with both the sigmoid and the hyperbolic tangent, mapping real and imaginary parts among input and output as they were independent channels:

$$f(\mathbf{z}) = g\left(\mathrm{Re}(\mathbf{z})\right) + ig\left(\mathrm{Im}(\mathbf{z})\right), \qquad \text{where} \quad g(x) = \frac{1}{1+e^{-x}} \quad \text{or} \quad g(x) = \tanh(x)$$

Notice that this approach maps the phase of the signal into $[0, 2\pi]$, since the function g returns always a positive value.
There are also interesting variations to the separable sigmoid, properly designed to work using a complex-valued network on real-valued data. But, for this reason, they are functions with values in $\mathbb{R}$ and not in $\mathbb{C}$, and so we won't go through them in this work.
After the advent of the ReLU activation functions, two designs where developed in this fashion, the $\mathbb{C}$`ReLU` and the z`ReLU`:

$$\mathbb{C}ReLU(z) = ReLU(\mathrm{Re}(z)) + iReLU(\mathrm{Im}(z)) \qquad zReLU = \begin{cases} z & \text{if } \theta_z \in [0, \pi/2] \\ 0 & \text{otherwise} \end{cases}$$

These functions also share the nice property of being holomorphic in some regions of the complex plane: $\mathbb{C}$`ReLU` in the first and third quadrants, while z`ReLU` everywhere but the set of points $\{\text{Re}(z) > 0,\ \text{Im}(z) = 0\} \cup \{\text{Re}(z) = 0,\ \text{Im}(z) > 0\}$.

Phase-preserving Activations

Phase-preserving complex-valued activations are those functions that usually act only on the magnitude of input data, preserving the pre-activated phase during the forward pass. Those are usually non-holomorphic, but at least bounded in the magnitude. They are all based on the intuition that, altering the phase could severely impact the complex representation.
The first proposal is the so called `siglog` activation function: It is called in this way because it is equivalent to applying the sigmoid to the log of the input magnitude and restoring the phase:

$$siglog(z) = g\left(\log \|z\|\right) e^{-i\angle z} = \frac{z}{1 + \|z\|}, \qquad \text{where} \quad g(x) = \frac{1}{1 + e^{-x}}$$

Unlike the sigmoid and its separable version, the siglog projects the magnitude of the input from the interval $[0, \infty)$ to $[0, 1)$. The authors of this proposal suggested also the addition of a couple of parameters to adjust the *scale*, r, and the *steepness*, c, of the function:

$$siglog(z; r, c) = \frac{z}{c + \frac{1}{r}\|z\|}$$

The main problem with *siglog* is that the function has a nonzero gradient in the neighborhood of the origin of the complex plane, which can lead to gradient descent optimization algorithms to continuously stepping past the origin rather then approaching the point and staying here.
For this reason, an alternative version have been proposed, this time with a better gradient behavior (approaching zero as the input approaches zero), that goes under the name of `iGaussian`.

$$iGauss(z; \sigma^2) = g(z; \sigma^2)n(z) \qquad \text{where} \quad g(z; \sigma^2) = 1 - e^{-\frac{z\bar{z}}{2\sigma^2}}, \quad n(z) = \frac{z}{\|z\|}$$

This activation is basically an inverted gaussian (and this is the reason for which it is more smooth around the origin) and so depends only on one parameter, i.e. its standard deviation σ.
The last activation that we want to consider for this class is another variation of the rectified linear unit, this time called `modReLU`:

$$modReLU(z) = ReLU(\|z\| + b)e^{i\theta_z} = \begin{cases} (\|z\| + b)\,\frac{z}{\|z\|} & \text{if } \|z\| + b \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

where $z \in \mathbb{C}$, θ_z is the phase of z, and $b \in \mathbb{R}$ is a learnable parameter. The idea of this activation is to create a "dead zone" of radius b around the origin, where the neuron will be inactive, while it will be active outside. We decided to consider this function, even if apparently was designed for Recurrent neural networks.

Complex Cardioid

Even if the `iGaussian` has nice gradients properties around the origin in the complex plane, the same cannot be said when in input are given large values: in that situation there is high risk of vanishing gradients, the same that have historically hindered the performances of sigmoid-like activations. Because of this, recently a new complex activation function have been proposed: the `Complex Cardioid` [18]. The cardioid acts as an extension of the ReLU function to the complex domain, rescaling from one to zero all the values with non-zero imaginary component, based on how much the same input is rotated in phase with respect to the real axis. Also, the cardioid is sensitive to the input phase, rather than the modulus: the output magnitude is, in fact, attenuated based on the input phase, while the output phase remains equal to the original one.
The analytical expression for this activation function is:

$$cardioid(z) = \frac{1}{2}\left(1 + \cos(\angle z)\right) z$$

Another very nice property is that when the inputs are restricted to real values, this function becomes simply the ReLU activation 3.6. We will see, in our applications, that the cardioid effectively allows

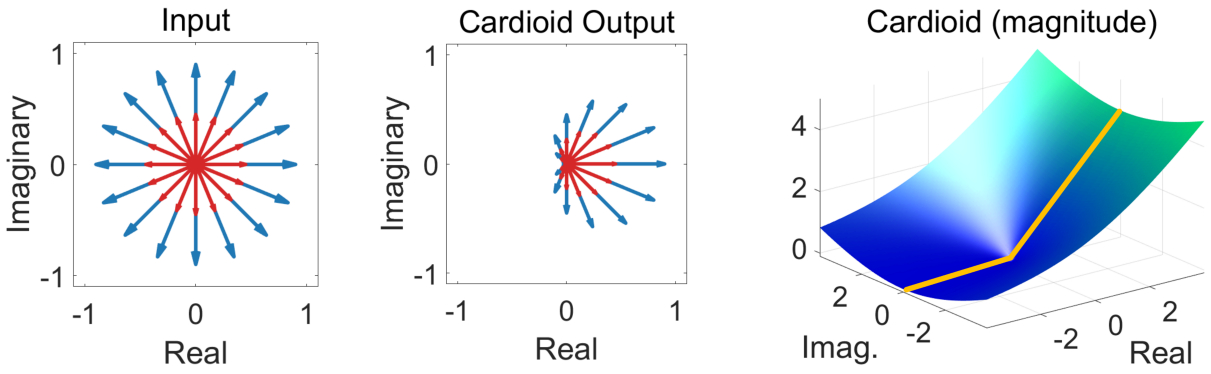


Figure 3.6: Complex cardioid activation function. The magnitude transformation of the cardioid shows that it is reduced to the ReLU on the real axis (right, orange line).

complex networks to converge in a fast and also stable way.

Activations Recap

In table 3.2 we left a brief recap of all the activations function in $\mathbb{C}$ discussed up to now. We provided also the relative plots in appendix B.

Activation	Analytic Form	Reference
Sigmoid	$\sigma(z)$	//
Hyperbolic Tangent	$\tanh(z)$	//
Split-Sigmoid	$\sigma(\mathrm{Re}(z)) + i\sigma(\mathrm{Im}(z))$	[5]
Split-Tanh	$\tanh(\mathrm{Re}(z)) + i\tanh(\mathrm{Im}(z))$	[5]
$\mathbb{C}$ReLU	$ReLU(\mathrm{Re}(z)) + iReLU(\mathrm{Im}(z))$	[1]
zReLU	z if $z \in [0, \pi/2]$, 0 otherwise	[12]
Siglog	$\sigma(\log \|\|z\|\|)e^{-i\theta_z}$	[22]
iGauss	$\left(1 - e^{-\frac{z\bar{z}}{2\sigma^2}}\right)\frac{z}{\|\|z\|\|}$	[18]
modReLU	$ReLU(\|\|z\|\| + b)e^{i\theta_z}$	[23]
Cardioid	$\frac{1}{2}(1 + \cos(\angle z))\, z$	[18]

Table 3.2: Recap of the most popular complex-valued activation functions.

3.5 JAX Implementation

From a practical perspective, in order to setup and realized all the studies and analysis we are going to present, I had to realize a dedicated `Python` library.
In the previous sections we have analyzed all the theoretical obstacles that researchers had to overcome in order to develop a working complex-valued deep learning framework. But, in reality, there are many drawbacks also at the implementation level: first of all the most popular hardware acceleration architectures, `CUDA` and `CuDNN` doesn't own a native support for complex-valued data types. And this is by itself a huge limitation, since we cannot train our networks efficiently, and we will have to rely on simple models with few parameters.
Regarding existing deep learning libraries, like `Keras`, `TensorFlow` and `Pytorch`, we have to say that they provide a lot of interesting high-level architectures to setup deep learning algorithms, and also they "officially" support complex inputs. Even better, they support complex derivatives. But when you try to implement an effective complex-valued network, you understand how much

are they far from an effective comprehensive support (just some known issues: 1, 2, 3): regardless of the many errors you can get from structures that should work, from an accurate analysis of the source code of the main layers, you notice that many operations are ambiguous or at least not compatible with the reasoning we adopted to develop the layers extents in this chapter. We could have probably found a way to redefine or override those structures, but we felt that modifying a so large and complex library, without getting undesired side effects, would have been too much work. For this reason we had to rely on a more niche library, called `JAX`, and recently developed by `Google DeepMind`. JAX is **Autograd** and **XLA** (a domain-specific compiler for linear algebra designed for TensorFlow models), brought together for high-performance numerical computing and machine learning research. It provides composable transformations of Python and NumPy programs: differentiate, vectorize, parallelize, Just-In-Time compile to GPU/TPU, and more. The main advantages that we found using JAX were:

- it supports and optimize complex differentiation, for holomorphic and non functions;
- it is extremely optimized, with XLA + JIT that partially compensate the lack of native hardware acceleration;
- many complex operations/layers are already supported and well defined.

Figure 3.7: JAX logo.

More specifically, for building our complex-valued neural networks we will rely on `dm-haiku`, another library built on top of JAX with the purpose of covering the same role that `Sonnet` (widely used ad DeepMind) has for Tensorflow, and to simplify the approach of users that are familiar with object oriented programming.
Since these are quite recent libraries, and also a definitive approach to complex-valued deep learning does not exists yet, I made a careful and complete analysis of the source code of Haiku and of the most important JAX functions. Thanks to this I effectively noticed that much work is still needed: even if in a small quantity respect to Tensorflow/Pytorch, also in this case many operations turns out to be ambiguous or bad-defined for complex-valued data types (e.g. the `square` or the `max` operators). Many of them are also completely undefined (e.g. initialization or, more in general, random complex distributions). We still decided to proceed with JAX mainly because of its flexibility, and many new functions/operations can be redefined without caring of implicit undesired side effects. This is possible also thanks to the design of the training loop, that is quite "explicit" and customizable.
From a practical point of view, I had basically to realize small a '`complex_nn`' library on top of Haiku, containing the definitions (and re-definitions) of all the necessary components of a complex-valued neural network:

- `layers`: the adaptations derived before for linear, convolutional, pooling and normalization operations;
- `activations`: all functions listed in table 3.2;
- `initializers`: weights initializers following uniform or truncated random normal distributions, together with the modified approaches described above by Xavier and He;
- `metrics`: categorical accuracy and categorical cross-entropy, useful for the successive classifiers designs;
- `optimizers`: a *complex-Adam* algorithm;
- a `classifier wrapper` with the purpose of collecting all the necessary functions to setup a training loop with JAX and Haiku;
- for completeness, also some utility functions, mainly to realize plots or wrapping more complex structures.

This code has been tested over several datasets. There are in fact a few `Jupyter Notebooks` providing

a detailed explanation of analysis we are going to see, together with some basic setup of some learning procedures.
All the implementation and complete analysis is actually available at my gitlab page [3].

[3] https://gitlab.fbk.eu/mpujatti/complex-valued-deep-learning-for-condition-monitoring

Chapter 4

Complex-Valued vs Real Neural Networks

4.1 Implementation of a Complex-valued Classification Problem

Another big obstacle, beyond the lack of hardware acceleration procedures, in the development of a complete and working complex-valued deep learning framework was the absolute absence of available datasets. Surfing the internet, the number of inherently complex datasets available for machine learning purposes, is very very small. That's reasonable, from a certain point of view, since real world measures are only real-valued. A possibility, also coherent with the final application areas of complex-valued models, could have been looking for datasets of signals: electric, seismic or even biological signals can easily be studied also in the momentum space, moving from the time series to the corresponding spectrograms via the Fourier transform.
However, in this section, we wanted to make just a superficial analysis, leaving real-world challenging tasks to future works. For this reason, we needed a sufficiently small and "simple" dataset to train small/medium-sized models in a reasonable amount of time, several times each. We wanted something more similar to a qualitative, rather than quantitative, analysis, giving more attention to the stability of the loss function, during the training phase, or to configurations that are more likely to overfit, rather than to the final accuracy reached. In our help it comes a very recent work by Ziller et. al. [14]: regarding this article, we are not really interested to this branch of deep learning but more on the dataset they have used. For the area of supervised deep learning, `MNIST` is probably one of the most used benchmark datasets for real-valued models. In $\mathbb{C}$ there is not such availability, and so the authors came up with this proposal: a complex-valued adaptation of `MNIST`, called `PhaseMNIST`. In brief, for each example in the original set with label $L_R \in \{0, \dots, 9\}$ select another image with label L_I such that $L_R + L_I = 9$, resulting in an input image rearrangement (0,9), (1,8), ..., (9,0). You build then new complex data stacking this pair of images, and keeping as label the one of the real part (imagine the lack of datasets, if those people had to come up with such an idea). In figure 4.1 we can observe an example sampled from `PhaseMNIST`. Obviously, we could have directly used the original `MNIST` even with a complex-valued model, but in that case the imaginary part of our weights would have not learned anything, vanishing the benefits that we instead would like to prove.

Real vs Complex-Valued Networks for PhaseMNIST images classification

In this section, we are going to verify if the complex-valued deep learning framework we have developed in the last chapters, effectively manage to learn something from a dataset. Furthermore we will make a comparison with an equivalent real-valued formulation.
In order to properly compare a real and a complex-valued model we need to take into account that the parameters of the latter have two components, real and imaginary parts, and so twice the learning power of the first. For more equity in the comparison we will setup the real model as follow: we first split the complex input into its real and imaginary parts that will be sent through two independent

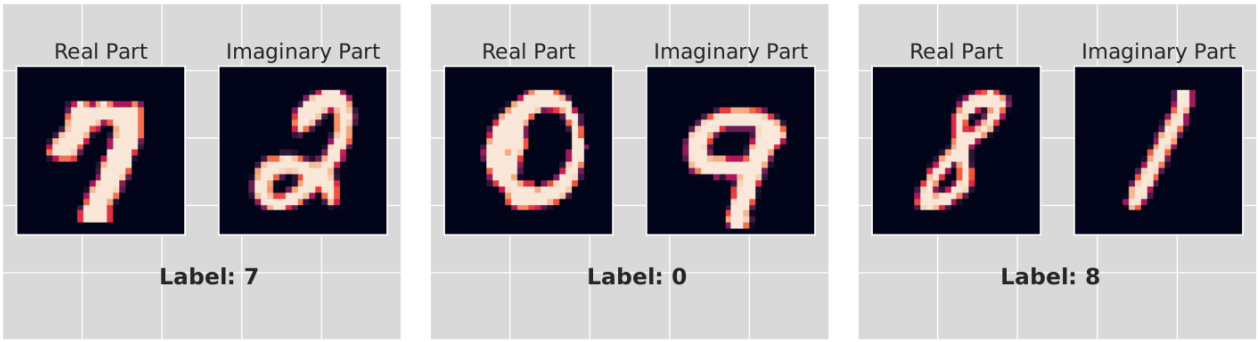


Figure 4.1: Sample from the `PhaseMNIST` dataset.

branches, with the same architecture of the complex model (but this time with real-valued weights), and then we concatenate the results into a unique output layer with number of units equal to the classes of the classification problem. In this way we are basically treating the components of the complex dataset as two independent channels, as the actual state-of-the-art approaches foresees in case of complex inputs. The model used are simple multi-layer perceptrons, with a couple of regularization layers, and are plotted in figure 4.2: notice that the number of trainable parameters of the complex architecture is half of the real one. Working on `PhaseMNIST` we notice one similarity shared with the

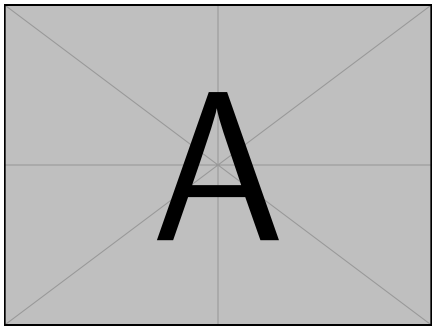


Figure 4.2: Real and complex-valued models used to classify `PhaseMNIST` samples.

original `MNIST`: this dataset is too simple. The large number of samples (70000), together with their relatively small size (2x28x28 samples) allows to reach very high performances (train/test accuracy $> 97\%$) even after a few epochs.
So, in order to stress the effective power of our networks, pushing them to the limit, we had two possible directions to take: reducing the number of training samples and reducing the number of trainable parameters in the architectures. Adopting those precautions, we effectively noticed some interesting behaviors. From a practical point of view, we changed the structure of the linear layers of the models above, from $64 \rightarrow 32 \rightarrow 10$ to $5 \rightarrow 10$ (so a drastical worsening, with the trainable parameters reduced from (103146, 205898) to (7930, 15820)), and we changed also the size of the datasets, with only 200 training samples and 10000 test samples. The results are summarized in figure 4.3. Apparently, the complex model seems to perform better than its real counterpart, whose loss seems to be more likely to diverge. We recognized this behavior also in other trials using this kind of "bad conditions", but we didn't manage to find any kind of regularity or general relation among this performance difference and the size of the training set. In reality, we are not really surprised about this behavior, since many previous works, treating this comparison, have found signs that complex-valued models are less likely to overfit. However, a definitive statement, in this situation, is not possible, especially because of the limits indirectly imposed by this dataset.

Weights Initialization

During the previous chapter, during the formulation of a possible extent of deep learning into the complex domain, at a certain point, in section 3.3 we described a couple of methods for initializing the

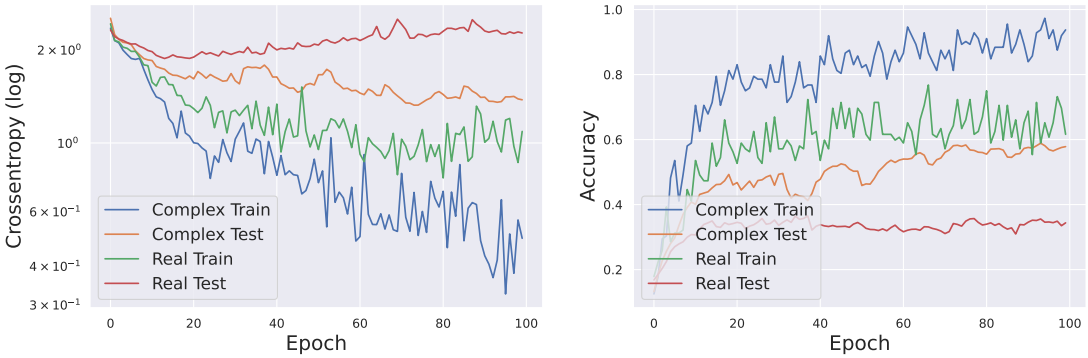


Figure 4.3: Performance comparison of real and complex-valued model over 200 samples from the `PhaseMNIST` dataset.

parameters in these new complex layers. For canonical neural networks, it has already been widely proven that a nice initialization for the parameters can drastically improve the performances and make the training faster. So, we are going to exploit the recently discussed `PhaseMNIST` dataset to verify if the same conclusions hold in the real world.
So, we setup a training loop with the same complex-valued architecture described before, whose layers have been initialized with:

- a complex uniform distribution in $[-1-i, +1+i]$;
- two independent random normal distributions (for real and imaginary parts) with parameters $(\mu = 0, \sigma = 1)$ truncated in $(-2, 2)$;
- two independent truncated normal distributions but with Glorot variation ($\sigma = 1/\sqrt{n_{input}}$);
- complex Xavier initialization (described in 3.3);
- complex He initialization (described in 3.3).

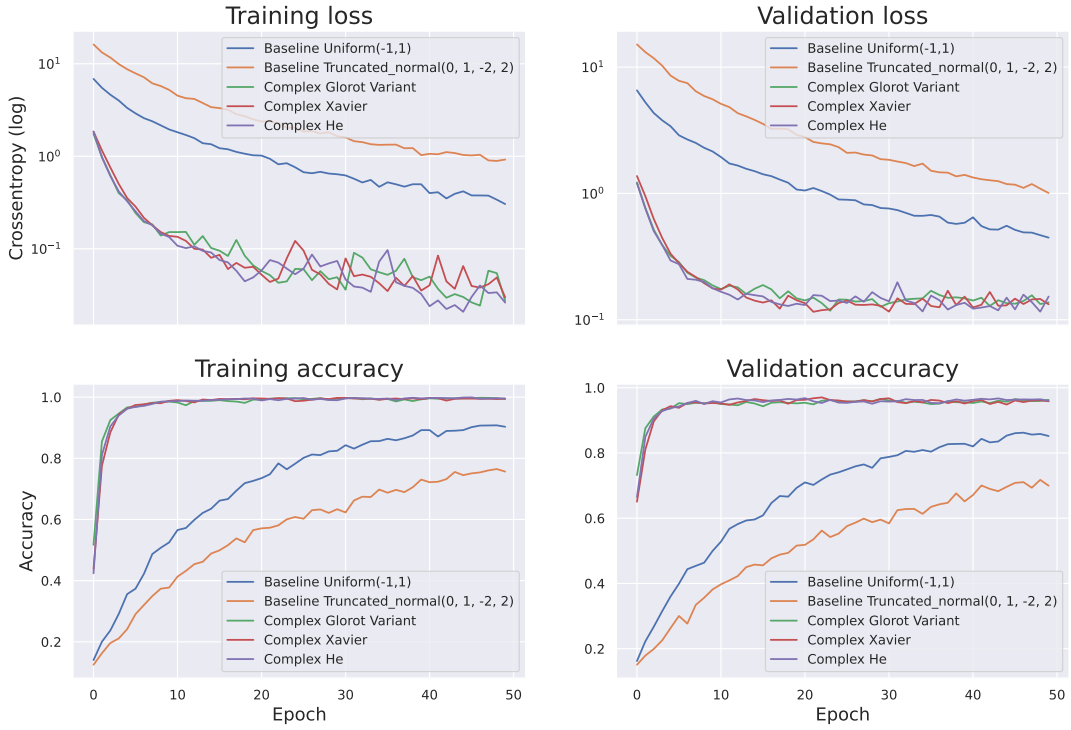


Figure 4.4: Performance comparison of different weights initializations over the `PhaseMNIST` dataset.

As we can see from the results in figure 4.4, the baseline random implementations do not work very well and lead to a quite slow training, while the proposal of Glorot and He, together with their complex-valued variants following the Rayleigh distribution, instead, guarantee very nice (and fast) convergence properties (at least in this particular case). It is better to recall, however, that we are still working with a "homemade" dataset, whose real and imaginary parts have been composed ad-hoc from real-valued data. This could be the reason for which an initialization like the one proposed by Glorot and adapted to the complex case, being performed independently among the real and imaginary parts of the weights, still manage to achieve results as good as the ones relatable to the methods exploiting the Rayleigh distribution.

Testing Activation Functions

We didn't lose the possibility of testing also all the complex-valued activation functions, proposed during the years and discussed in section 3.4. To do this we simply re-run the training loop over a subset of `PhaseMNIST`, with the same models as before, just with a different activation. What we noticed is that, as already said, this dataset is quite simple to learn, and so it becomes difficult to distinguish among similar training configurations, as all will reach high accuracies. For almost all the activations, in fact, the network managed to converge. Only in two cases, with the `complex sigmoid` and with the `complex tanh` the model returned suboptimal performances, effectively proving that "forced re-adaptation" of existing functions to the complex domain, are not sufficient to guarantee nice training property in complex-valued deep learning.

4.2 Impact of the Circularity

In chapter 2.4 we dedicated a whole section to the discussion about the circularity property of a complex random variable: we provided rigorous definitions of the circularity and the correlation coefficients, together with a brief explanation of their geometrical interpretation. We also cited a recent work by Barracchina et al. [15], suggesting the impact that datasets with such properties could have in machine learning applications. The authors, in fact, set up a classification task with two different distributions at a given circularity value, and then they use these to train real and complex-valued models. However they focused mostly on the training stability rather then on the difference in term of performances, that we, instead, believe could bring to very interesting results. For this reason, we decided to set up a similar problem in order to confirm first what they obtained, and then to come up with a more quantitative analysis of the accuracy that complex models can effectively achieve.

Data Generation

The idea behind this analysis is to understand how much complex-valued models can effectively benefit of data with good circularity properties. For this reason we will generate two distinct complex-valued datasets, one with high correlations among real and imaginary components, and one with poor correlations.
An easy way to generate data with pre-determined circularity is to rely on the *complex normal distribution*, that characterizes complex random variables whose real and imaginary parts are jointly normal. So, we can construct the samples we need simply exploiting the `multivariate random normal` distribution, supported by `numpy`: we just need to generate two-dimensional values that will constitute the real and imaginary parts of our data. Recalling the definition of the circularity coefficient

$$\rho_z = \frac{\sigma_x^2 - \sigma_y^2}{\sigma_x^2 + \sigma_y^2} + i\frac{2\sigma_{xy}}{\sigma_x^2 + \sigma_y^2}$$

we can then tune the covariance matrix Γ of the generating distribution to regulate the two main sources of non-circularity:

$$\Gamma = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix} \qquad (I) \quad \sigma_x \neq \sigma_y \qquad (II) \quad \sigma_{xy} \neq 0$$

As in the paper discussed above, we want to setup a classification problem among data following two different distributions:

- with label "0", a perfectly circular complex normal ($\rho_z = 0$), with zero-centered data and the identity as covariance matrix;
- with label "1", still a complex normal with mean zero, but with an arbitrary covariance matrix (variable ρ_z).

In practice, we will construct a dataset made of 5000 samples for each distribution, with each of them being constituted by 128 complex values. In figure 4.5 we represent an example for each class of data. Successively the dataset will be partitioned in a train and a test set with proportions $80\% - 20\%$.

It is important to note that the distinction between the classes is entirely contained in the relation-

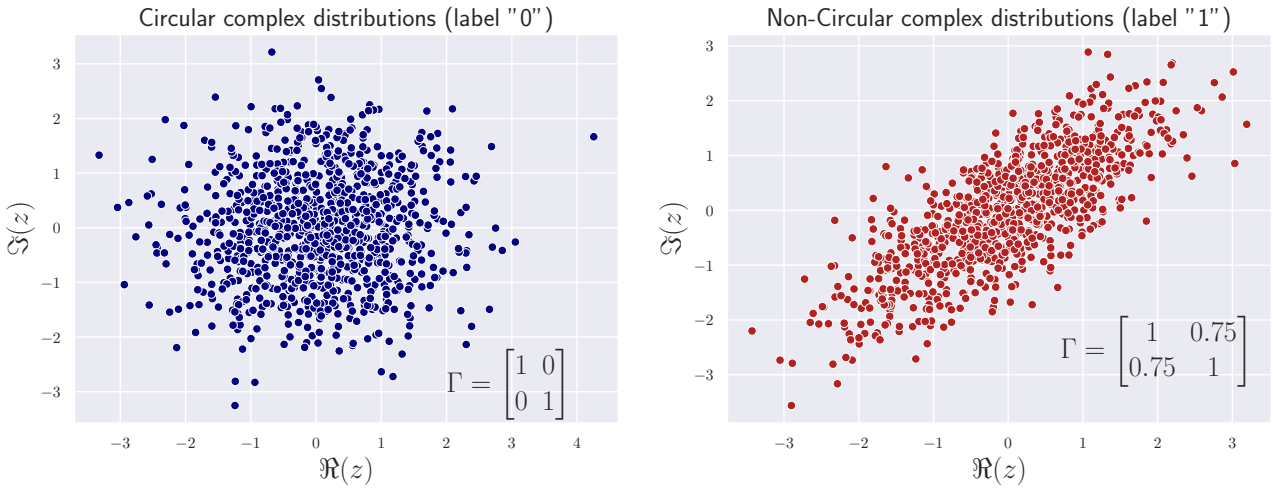


Figure 4.5: Samples coming from a perfectly circular distribution (left) and from a non-circular one (right).

ship between the real and imaginary parts. This means that removing, for example, the imaginary part of the dataset will result in both classes being statistically identical, and therefore, making the classification impossible.

Classification Task

To complete our setup, we just need to define a couple of neural networks to train: a complex-valued model, and its equivalent real-valued counterpart (treating the real and imaginary parts of the input as two independent channels) with double parameters respect to the first one. Datasets are quite simple by themselves, and so we used very elementary classifiers: we used a complex network with a couple of hidden layers (64 and 16 units respectively), and a small dropout (20%). The same structure has been duplicated over two independent channels to obtain the real-valued network.

We made our tests for different label 1 distributions (changing the second covariance matrix while leaving the first equal to the identity), seeing that both models are able to learn easily from the dataset reaching also high accuracies ($> 95\%$). But we also noticed that there are some configurations for which one of them outperform the other, sometimes even by a lot. In order to better investigate this situation, we proceed examining the two sources of non-circularity separately. Just to remark, the samples labeled with "0" have:

$$\sigma_x = \sigma_y = 1 \qquad \sigma_{xy} = 0 \qquad \implies \rho_z = \rho = 0$$

Let's examine the first source of non-circularity in a complex distribution, i.e. data with a non-zero covariance factor ($\sigma_{xy} = 0$). Considering the distribution "1", let's fix, for simplicity, $\sigma_x = \sigma_y = \sigma = 1$ and leave the covariance σ_{xy} free to space the interval $(-1, 1)$ (to keep the covariance matrix positive defined). In this case we will have: $\rho_z = i\sigma_{xy}/\sigma^2$, $\rho = \sigma_{xy}/\sigma^2$ and so $|\rho_z| = |\rho|$.

We generated several datasets with the setup just described (fixed variances but different σ_{xy}) and trained both models over all of them for 60 epochs. As final "performance score" we kept the average

classification computed over the last 10 epochs for the test set. To improve the reliability of the statistics found, we repeated the entire procedure for different random seeds (taken into account in the errorbars of the plots) and represented the results in figure 4.6 (left). The behavior of the points is quite clear and regular: for this kind of non-circularity, the complex model outperforms the real one, also by a significative percentage, as the modulus of the circularity coefficient ρ_z approaches 1 (and so does the correlation ρ).
Then we examined also the second source of non-circularity, i.e. different variances among real and imaginary parts of the variables ($\sigma_x \neq \sigma_y$). This time we started fixing $\sigma_{xy} = 0$ and also, for simplicity, $\sigma_x = 1$ (because, in the end, it is just the shape of the distribution that matters). We leaved, instead, the second variance σ_y free to move in an interval $(0, 3]$. This time we have $\rho_z = (\sigma_x^2 - \sigma_y^2)/(\sigma_x^2 + \sigma_y^2)$ and $\rho = 0$.
We repeated again the same procedure as for the first source and obtained the results in figure 4.6 (right). The behavior of the accuracies is again quite clear and regular, but this time we see the real model performing better than its complex-valued counterpart; however, this advantage is relatively small, and seems to completely vanish as $|\rho_z|$ approaches 1. A possible justification to this behavior stands on the fact that the real and imaginary part of the data are statistically not correlated, since $\rho = 0$. We believe, in fact, that the two independent channels of the real model are probably more adapt to learn such internal structure with respect to the complex-valued architecture, whose parameters still preserve some local correlations, even after the training loop.
In the previous analysis we found some really interesting regular behaviors, with situations in which

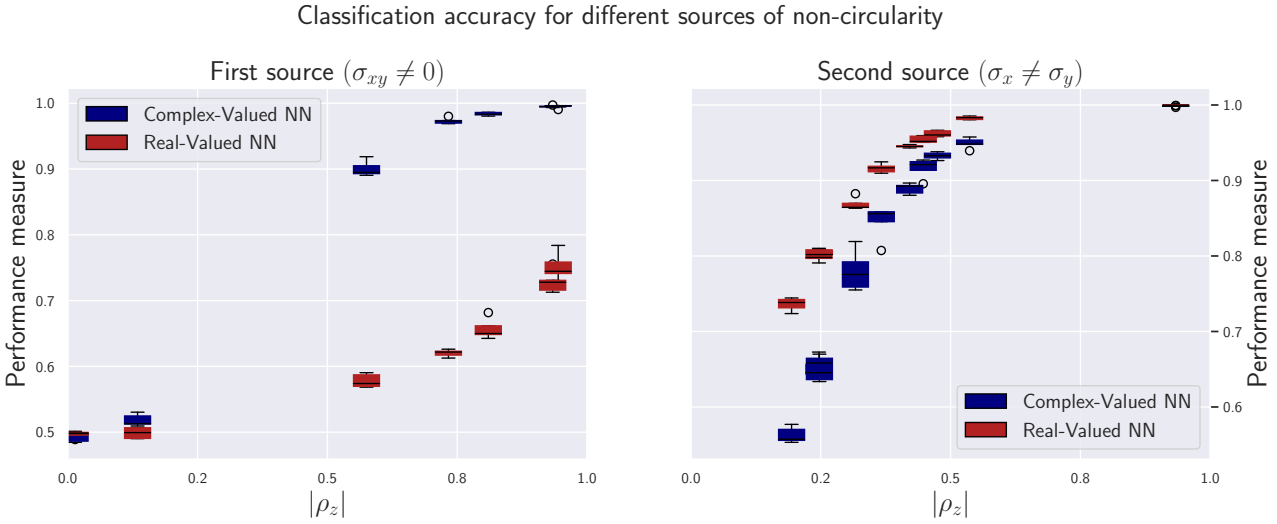


Figure 4.6: Classification accuracy for real and complex-valued networks as function of different source of non-circularity.

complex models are dominating and others in which are the real ones performing better. But, is there any general law, or rule, that we can find, in order to understand which characteristics should our data have, in order to guarantee nice performances for complex networks?
Let's proceed in the same way as before, fixing a distribution "0" that is perfectly circular, again with the identity as covariance matrix, and let's try to distinguish its samples from the ones coming from a second distribution "1", in which we add some non-circularity from both sources. We decide again to fix $\sigma_x = 1$ for the second distribution, while the two remaining degrees of freedom σ_y and σ_{xy} are let free. Let's vary those parameter in order to space all the possible values that ρ_z can assume, i.e. all the points in the unitary complex circle, and train again the models.
Because of the finite precision and the low amount of points in our samples (128), we cannot construct an accurate and uniform grid spacing all the possible values, and so we decided to adopt a stochastic method: sampling random combinations of (σ_y, σ_{xy}), computing the relative performance measure for the architectures, and reconstructing the accuracy distribution with the functionality `tri.LinearTriInterpolator`, provided by `matplotlib`. The final results of this analysis are presented in figure 4.7

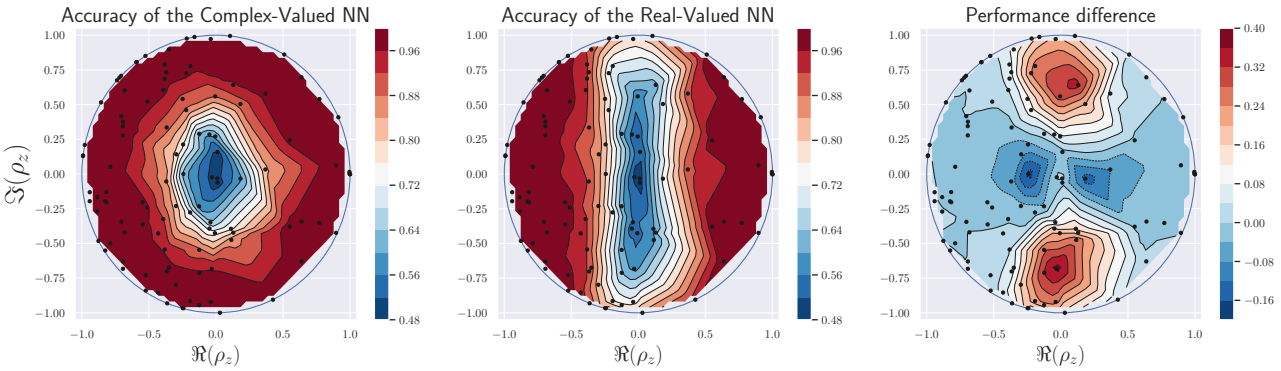


Figure 4.7: Comparison of the accuracy achieved by a complex-valued model (left) and its real equivalent counterpart (center) in distinguishing a perfectly circular distribution from a second one as function of its circularity quotient. The third plot (right), instead, represents simply the differences among the first two.

As we can see, there are effectively some regularities in the behavior of those models. For the complex-valued network, we can see that its discrimination performances raises uniformly with the modulus of the circularity quotient, achieving the maximum accuracy, and training stability, on the edge of unitary circle. For their real-valued counterpart, instead, the situation is a bit different: this models, in fact, seems to not handle very well the "first source" of non-circularity ($\sigma_{xy} \neq 0$), probably because it coincide with a non-zero correlation coefficient among real and imaginary parts of data. Looking at the right figure, the one with the effective comparison among the models, we can clearly see that there are still some configurations in which real networks seems to perform better (-16%), but the theoretical advantage ($+40\%$) guaranteed by the complex model in the two red regions is too high to not be taken into account.

Part II

Condition Monitoring

Chapter 5

Healthy-Faulty Signal Classification

In the second part of this thesis, we will apply the complex-valued deep learning framework, developed in part 1, to a real world problem of condition monitoring in industrial applications.
Condition Monitoring is defined as *the process of monitoring a parameter of condition in machinery (vibration, temperature, etc.), in order to identify a significant change which is indicative of a developing fault.*
In our technological and industrial society, the use and mastery of condition monitoring turned out to be of extreme importance, since it allows to be schedule maintenance, or other actions to be taken in order to prevent consequential damage and avoid its, often expensive, consequences. Furthermore, it can helps in increasing the lifespan of many engines, preventing faults that could develop in major failures.
Condition Monitoring techniques are normally used on rotating equipment, auxiliary systems and other dynamic machinery (compressors, pumps, electric motors, etc.) while for static devices, usually, periodic inspections are sufficient.
In our analysis, we will propose a modification (or, better, an extension) to the existing approach to the strategy that is usually followed for rotating machines, i.e. **vibration analysis**.

What engineers do, is usually taking measurements on machine bearing casings with *accelerometers* to measure the casing vibrations, sometimes provided also of electric transducers able to directly observe the rotating shafts and detect their radial (and axial) displacements. Data collected are then set of vibrational signals that can be analyzed and studied to detect clues of something bad that is happening. Vibration levels can then be compared with some historical baseline values (a "ground truth") derived after long periods of experiments, and in some cases compared with established standards such as load changes, to keep high the attention. Vibration limits can also be defined based on the machine design or components, knowing their fault frequencies of bearings.
Interpreting the vibration signal obtained is an elaborate procedure that requires specialized training and experience. With the technology progresses and increasingly complicated machines, simple comparisons and vibration limits are not anymore sufficient to guarantee the performances and accuracies needed by the companies. Luckily, also the techniques available have widely improved: thanks especially to the recent advances of machine and deep learning, signal analysis and classification is strongly simplified. Several state-of-the-art approaches have been developed, able to provide the vast majority of data analysis automatically, retrieving information instead of raw data.
In general one commonly employed technique is to examine the individual frequencies present in the signal. These frequencies, in fact, can correspond to certain mechanical components or specific malfunctions (e.g. unbalance or misalignment). So, by analyzing these frequencies and their harmonics, a specialist can in principle be able to identify the location or the type of a problem, sometimes even the cause. And this is possible weeks, even months, before the effective failure or damage of the apparatus, giving ample time to the technicians for replacing or repairing the machine.
But the interesting part in this approach is that we have a mathematical instrument that allow us

to move from the time to the frequency domain, with all the consequent advantages: the **Fourier Transform**. We will see, in the next section, that this operator not only permits the analysis of the frequency spectrum of the signal, but also represents the connection among the real-valued measurements of physical quantities and the complex-valued domain in which we can effectively put to test the framework we have developed.
It is however better to specify that frequency analysis tends to be most useful on machines that employ rolling element bearings and whose main failure modes tend to be the degradation of those bearings, which typically exhibit an increase in characteristic frequencies associated with its geometries and constructions.
Tons of different "manual" analysis are possible for those kind of signals in the frequency domain, a study of its phase spectrum for example, but in this thesis we are not really interested in them since we will rely on modern deep learning approaches, providing also efficient alternatives to the actual state-of-the-art techniques.

5.1 State-of-the-Art

In chapter **??**, we were complaining about the fact that literature does not provide almost any kind of complex-valued dataset to effectively test the models we have developed. There, we were forced to "manually construct" our data, with simple distributions of points satisfying certain properties **??**, or clever modifications of existing real collections **??**. Relying on them could have been fine for that preliminary analysis, mainly focused on the convergence and the stability of the training process, rather then on the final performances. However, we believe that, in order to prove the efficacy of our method, we should stress our complex-valued architectures on more concrete and reliable datasets.
But, is there the possibility of measuring in some way physical quantities that are inherently complex-valued? From this point of view, the answer is yes, and we just need to think about magnetic momenta. The PHD thesis we have based our work on **??**, is constructed exactly on these kind of data: Magnetic Resonance Images contains, in fact, much information in their phase, that, with a complex neural network the author managed to outperform many existing approaches that used to discard such information.
But we should not limit our horizons when we have a powerful instrument like the Fourier transform $\mathcal{F}$, that allow us to associate a complex-valued frequency domain representation to a function of time. In this new representation, the magnitude represents the amount of a certain frequency present in the original signal, while the argument is its phase offset with respect to the basic sinusoid. All of this means that, given any dataset composed of waves in the time domain, the same can be studied in the frequency domain exploiting the new power of complex-valued models, keeping all the physical information without need of expedients like before. Several papers have been published, suggesting complex-valued models to study electric, seismic **??** and vibrational signals in the complex domain. So, in principle, for any signal registered in the time domain, we can recover, and maybe exploit, the phase information thanks exactly to the Fourier transform.
Studying the problem in the frequency domain is a quite common approach to signal processing, since many operations in the time domain have a complex counterpart that sometimes turns out to be easier to perform (eg. differentiation and convolution). Furthermore, also from a purely theoretical point of view, several concepts are easier to derive and understand with complex notations.

5.1.1 Baseline Machine Learning

But how can we effectively distinguish among vibration signals? According to the traditional machine learning approach we should rely on some hand-crafted features extractable from the wave in the time domain; those features can then face a dimensionality reduction step (PCA, t-SNE, etc.) and then sent through baseline classifiers like Support Vector Machines or Multi-Layer Perceptrons. This was the fundamental strategy for years: in term of memory and time complexity it is very efficient, because each sample is reduced to a handful of meaningful values, but, in general, it lacks of generalization capabilities, since the high-level features to be used are chosen every time by an expert, depending

on the situation, and also from the point of view of the accuracy reached, more modern technologies employing deep learning are already able to outperform it.

5.1.2 Study in the frequency domain

However, even if condition monitoring is not one of the main tasks over which machine learning researchers focused mostly in the last years, several techniques and improvements could be recovered and re-adapted from the popular area of audio applications. In the end, in fact, sounds are vibrations of the air, and so their analysis can present interesting similarities with our problem.
It is exactly from the widely developed theory of deep learning applied to audio processing, that we inherit techniques suitable to study a wave signal in the frequency domain. The computational instrument that allow us to do so is the **Short-Time Fourier Transform** (**STFT**). Discrete STFT is a method of partitioning continuous signals (in the time-dependent representation) over a long period into shorter segments (usually overlapped) at short time intervals, and applying a Fourier transform to each of those segments. An additional step can be taken, at this point, deriving the power spectrum of the signal: taking the square modulus of the STFT we can construct a 2D sample, i.e. an image, representing the amplitude of the signal for a particular frequency at a particular time.
In figure 5.1 we took a random signal from the dataset we are going to study, and we transformed it in the Fourier representation: the leftmost picture represents the original wave, the central one is the result after the application of the Fast-Fourier transform (so the signal in the frequency domain), while the rightmost is its power spectrum. After the computation of the power spectrum for each

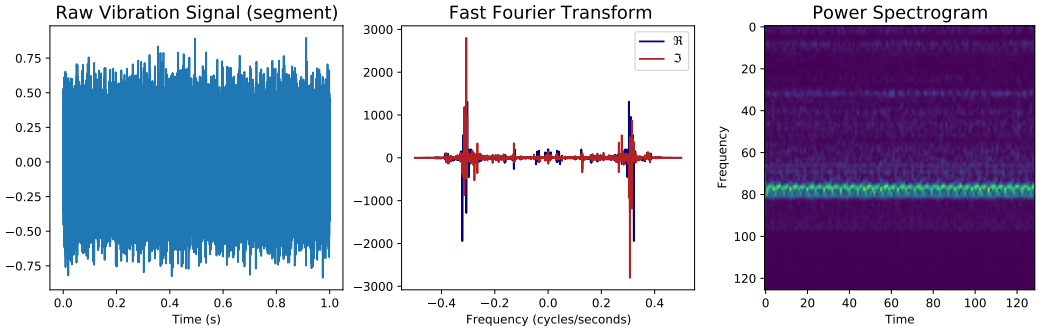


Figure 5.1: Representation of a vibration signal in the time domain (left), in the frequency domain via FFT (center) and with its power spectrum (right).

vibration signal at this point, there are two directions that the researcher can decide to follow:

- keeping the two-dimensional input and building a machine learning classifier able to work directly with the spectrograms (usually a Convolutional Neural Network);
- taking a further step and extracting from the spectrogram the so called *Mel Frequency Cepstral Coefficients* (MFCC): these coefficients, widely used in speech recognition tasks, permit an efficient compression of the "important information" carried by the signal, and can be simply given in input to an ordinary classifier.

Even if they were not properly designed for vibration analysis applied to condition monitoring, the usage of MFCCs can bring very good results together with an high efficiency in term of space and time complexity, because of the information compressed.
In figure 5.2 are represented schematically the two main state-of-the-art approaches (high-level features extraction vs power spectrograms) for healthy-faulty vibration signal classification, that we just described.
In the successive analysis, we will propose an alternative to the approaches listed above, that is more suitable to complex-valued deep learning: the coefficients of a Fourier transform are, in fact, inherently complex, and so, instead of losing all the phase information taking the power spectrum (that is constituted only by their amplitudes), we stop one step before keeping the original complex-valued variables. In the end, deriving the complex spectrum is equivalent to computing a discrete Fourier transform over short overlapping windows of the signal.

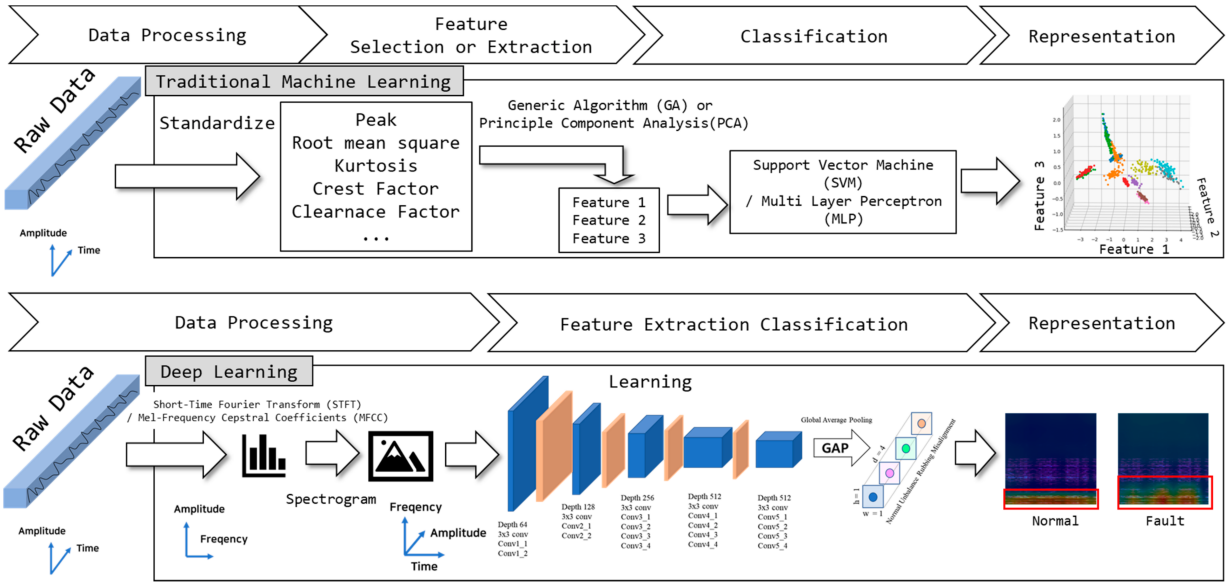


Figure 5.2: State-of-the-art approaches for vibration data classification applied to predictive maintenance (source: [24]).

5.2 Bonfiglioli

The main motivation of this thesis work was the development and testing of an unconventional deep learning approach, based on complex-valued neural networks, on a series of datasets provided to FBK by Bonfiglioli 5.3, a worldwide designer, manufacturer and distributor of a complete range of gearmotors, drive systems, planetary gearboxes and inverters.

The company provided a large number of datasets collected from as many experiments, with the purpose of simulating artificially the most frequent faults, that usually occur over the bearings of different kind of engines. Our objective will be then to find an efficient approach for failure detection and classification.

Figure 5.3: Bonfiglioli logo.

5.2.1 Simulation Environment

Data have been collected at the Bonfiglioli headquarters of Rovereto (TN), with the experimental setup schematized in figure 5.4. The laboratory is constituted of four different workbenches in which different experiments can be conduced. In each workbench, a lab-scale rotating simulation equipment is installed, constituted by two working axis, one representing the healthy configuration while the other will be modified in order to simulate a damaged engine, in one of its components. Tons of sensors have been collocated all around the mechanical system in order to measure physical quantities of interest (vibration, temperature, current, etc.) for both the *healthy-axis* and the *faulty-axis*.
So far, in the experiments, Bonfiglioli managed to reproduce up to five different kind of faults, that were artificially simulated using mechanical tricks:

- *unbalance*: the rotor of one of the engines was unbalanced via two masses of 3g;
- *bearing damage*: not specified in their reports, as they just states that bearings have been damaged radially;
- *solar*: the solar of the gear reducer was damaged via an electric pen;
- *pinion*: the input bearing of the gear reduced was damaged;
- *crown*: the crown of the gear reducer was damaged via an electric pen.

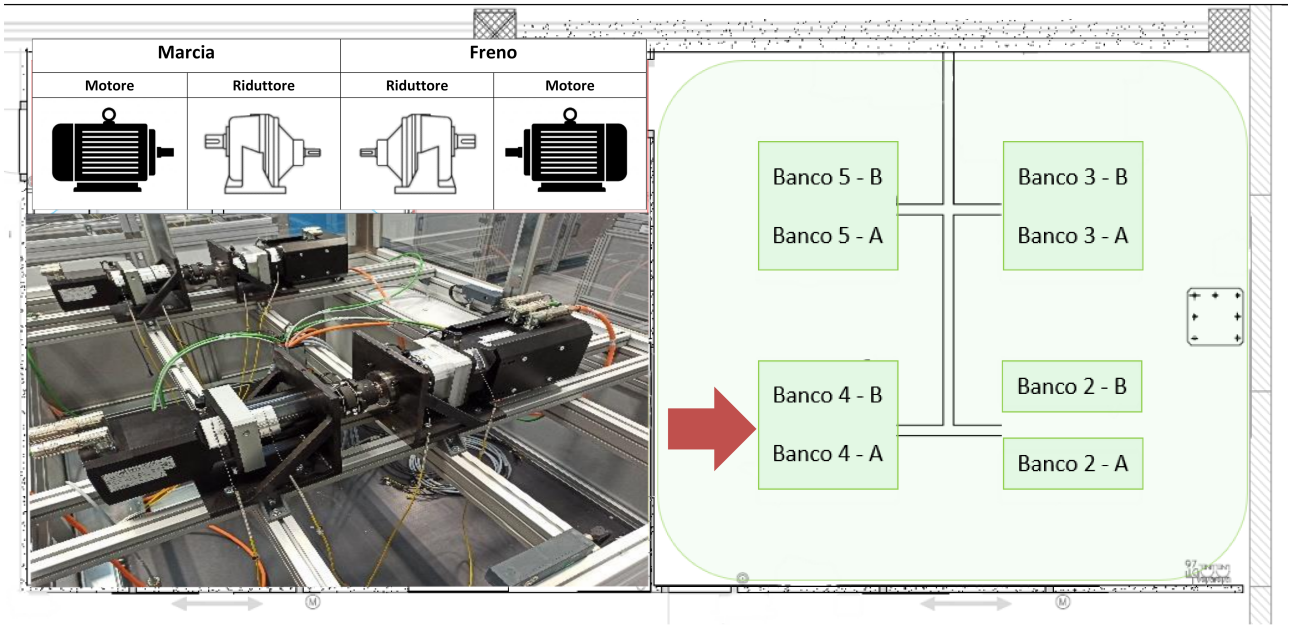


Figure 5.4: Schematic representation of the experimental workbench used by Bonfiglioli.

Just to provide a visual point of view, in figure 5.5 we reported the images of some simulated faults. In order to have data more heterogeneous, able to represent the engine behavior under different con-

Figure 5.5: Some of the faults simulated by Bonfiglioli on the mechanical components of the system.

ditions, and to improve the generalization capabilities of future models trained over them, Bonfiglioli setup a *working cycle*, during which measurements were taken. This cycle is periodic with a period of 24 hours, in general repeated up to five times to collect most of the datasets. It is characterized by the engine varying its speed and torque with a quite regular behavior following a piecewise constant function, also to guarantee stationary conditions during the acquisition phases. Data were not, in fact, collected continuously during this cycle: 30 or 60 seconds long windows were fixed along this period, during which sensors were actively measuring. An example of working cycle has been reported in figure 5.6. Vibrations signals, the ones we are mainly interested in, were collected thanks to accelerometers attached to the components of the engine (motor, brake and multiplier) they wanted to test; as explained above, each acquisition was 30/60 seconds long, with a sample rate of 25600 Hz: sampling was performed because a vibration is a periodic signal in the time domain, and most fault signals have periodicity too. Therefore, sampling can be used to examine the consistency and continuity of each condition [24]. The signal segmentation for sampling is usually based on the rotational frequency of the rotor. Generally, in fact, in rotating equipment, the rotational frequency is the most dominant component, and the majority of fault contributions appear in its harmonic form.

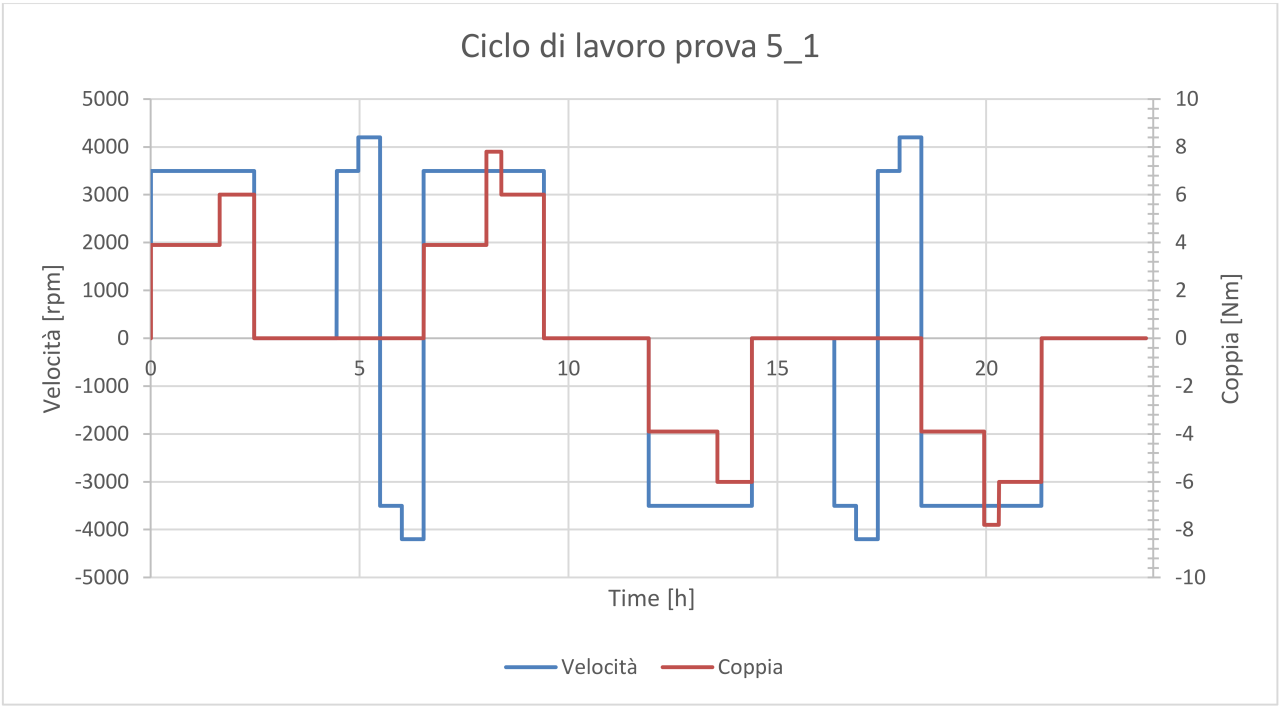


Figure 5.6: Example of working cycle (from Prova-5) of the Bonfiglioli experimental setup.

5.2.2 Datasets

In these last months, the Bonfiglioli laboratory have setup different experiments, in order to provide data relative to different kind of engines and for different types of simulated faults. For each of those runs, containing 120 hours of measurements, a dataset in `.h5` format has been provided, with the values registered during a working cycle. Those datasets had an internal structure quite mazy, and so we had to rely on a custom library, provided by FBK researchers, to load and process the data. Each sample contains several entries, related to the different variables measured during the process: beyond vibration signals collected through the accelerometers, Bonfiglioli decided to monitor also the temperature of some components, voltage and current flowing through them, together with the phase information of the inverter. Additionally, for the experiments realized more recently, they used an additional component in their setup, a *multiplier*, providing a second vibration signal.
For what concern our work, we are interested only in the vibration signals. In table 5.1 are reported, with some technical details, the dataset we have used in our analysis.

N	Name	Subname	Engine A	Duration (h)	Fault
4	Prova 4 - BN90 - HF Sbilanciato	4_2	BN90	120	Unbalance
5	Prova 5 - MP105 - HF Cuscinetto	5_3	BMD118	120	Bearing
6	Prova 6 - BN90 - HF Cuscinetto	6_3	BN90	100	Bearing
10	Prova 10 - MP105 - HF Solare	10_1	BMD118	120	Solar
10	Prova 10 - MP105 - FH Solare	10_4	BMD118	120	Solar
11	Prova 11- S30 - HF Cuscinetto	11_1	BN112	120	Bearing
13	Prova 13 - S30 - HF Pignone	13_1	BN112	120	Pinion
15	Prova 15 - S30 - HF Corona	15_1	BN112	78	Crown
15	Prova 15 - S30 - HF Corona	15_2	BN112	48	Crown

Table 5.1: Summary of the main technical features of the datasets provided by Bonfiglioli and used in our analysis.

5.2.3 Baseline Classification Approach

For completeness, we would like to briefly describe also the baseline approach, the one actually followed by FBK researchers, even if in this work it wasn't either considered.
The procedure actually implemented in the project is quite similar to the one presented as the state-of-the-art. The idea behind is, in fact, fundamentally the same: given a dataset of vibration acquisitions, these are splitted into subsignals of a given fixed size (usually 3 seconds), preprocessed and sent through a "feature extractor" that takes from them the information useful for the classification. Different kind of extractions are possible: up to now you can choose among MFCCs, ordinary Short Time Fourier Transform and power spectrograms, all of them with a wide margin of customization. The main library used for this is feature-extraction part is `scipy`.
All the information got thanks to this step is then sent through a neural network classifier, that in this case is a Residual Network [1], with a fundamental block constituted by two $3x3$ convolutions and a few regularization layers.
With respect to the procedure we are going to implement, this is much more efficient from the point of view of the memory complexity (especially if you select the MFCCs), but it requires an architecture that is much larger. We will prove that we can still manage to obtain very good results even with a much smaller number of parameters.

5.2.4 Complex Spectrogram Classification

As anticipated, the approach we decided to follow is slightly different. In sound classification tasks, the advantage provided by convolutional neural networks has been widely demonstrated, in recent years, providing a valid alternative to the standard procedure of exploiting high-level features (like MFCCs) [].
In our implementation, we first split the data acquisitions into signals of fixed length: we found that a window of 1.0 second was representing a nice tradeoff between dataset size and resolution, preserving, at the same time, the eventual periodicity of the wave. We then proceed computing, for each signal in the dataset, the corresponding **complex spectrogram** (essentially a STFT), i.e. a plot associating amplitude and phase in a coordinate system that foresees the time on the horizontal axis and the frequency in the vertical one. In the end, instead of losing all the phase information considering the power spectrum, we keep the complex data and exploit the new complex-valued deep learning framework we have just developed. Complex convolutional networks, in particular, seems to suggests that a model sensitive to the phase structure could provide very nice results.
From a practical perspective, the spectrogram extraction step was preformed using the corresponding functions provided by `Torchaudio` (and based on `librosa`): we believe that their implementation is better, at least for classification purposes, with respect to its alternative that relies on `scipy`, mainly because of efficiency and normalizations reasons.
The main difficulty in this implementation consists into finding a proper resolution, both in time and frequency, for the signal to effectively point out its discriminative characteristics: we found that, for a 1 second long vibration, setting the `n_fft` parameter to 300 and leaving the other to their default values was guaranteeing nice results. As a general rule, however, we feel to empirically suggest that a nice combination of parameters is the one creating a spectrogram that is square, or at most rectangular with longer time dimension, since, apparently, these cases were the ones behaving better with our models.
In figure **??** we left an example of the data we are going to classify. Since complex data are quite difficult to plot, we relied on a modified "colorize" function, that inherits from `mpmath` a mapping from the polar representation (m, θ) of complex numbers to triplets of colors in the `hls` representation. We know that the result is not very good aesthetically, but that's what we could achieve in continuity with amplitude and phase spectra.

[1] ResNet

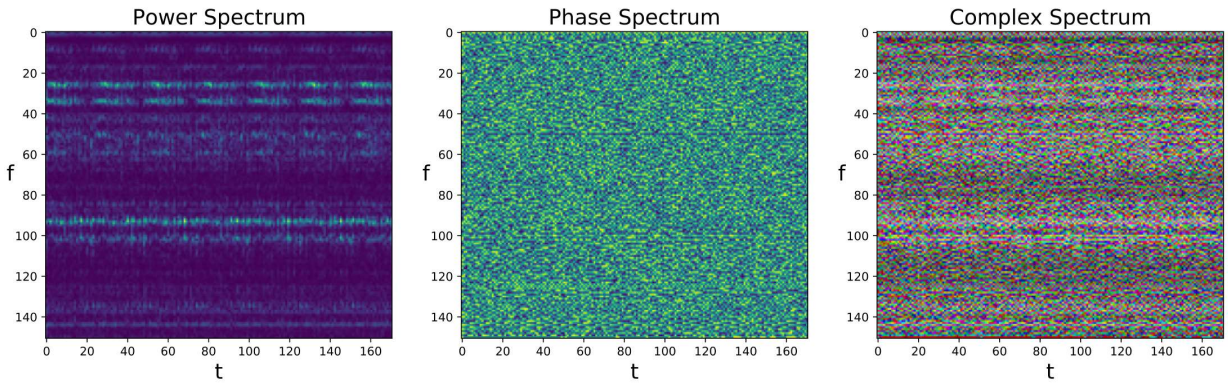


Figure 5.7: Example of spectrograms derived from a signal in the Bonfiglioli dataset. For completeness, we represented also the power spectrum (right) and the phase spectrum (center). The leftmost one is instead our representation of the complex spectrum.

5.3 Bonfiglioli - Results

Before presenting the effective results of this classification task, a couple of words should be spent presenting the analysis setup. As for the PhaseMNIST dataset **??**, we decided to employ two different networks: a purely complex-valued architecture and its corresponding real-valued counterpart, that treats real and imaginary parts of the input as two independent channels and, for this reason, has twice the trainable parameters of the first. Those architectures are based on the canonical forms of Alexnet and VGG16, with a few convolutional layers stacked on top of a multi-layer perceptron. In figure 5.8 we can visualize the models we have effectively used. It is interesting to notice that the networks are quite small (1092 and 2188 parameters, respectively), and the reason is that a few works ([13]) suggested that the hidden potential of complex-valued deep learning arises when models depend on a small number of parameters and are so more sensible to overfitting. More specifically, according to Dramsch et. al., if a real-valued models with two channels is enough large, than it is able to reconstruct internally the relative phase information dropped at the beginning.

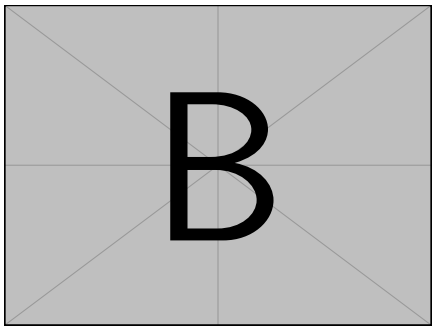


Figure 5.8: Complex and real-valued architecture implemented for the Bonfiglioli classification problem.

An important role is covered by the 3×3 average pooling layers, that are fundamental to reduce the dimensionality of the inputs. However, the only important difference among the complex and the real architectures, beyond the data type employed, is the activation function: `cardioid` for the first and ordinary `ReLU` for the second one.
Regarding the training setup, we followed a quite ordinary approach: since this is a classification problem we relied on the **crossentropy** (on the output's magnitude) as loss function and on the **categorical accuracy** to evaluate the performances of a model. The optimizer is a complex-valued version of `Adam` with an exponentially decaying learning rate (to improve the convergence at the beginning and the precision in the late) following the rule $lr(x) = 10^{-2} * (0.9)^{(x/100)}$. No further data preprocessing was needed (especially because spectrograms are already normalized) and labels were one-hot encoded. We used a $75\% - 25\%$ splitting to determine training and test sets.

The first results the we want to present are the classification scores obtained over the datasets listed in table 5.1. For all of them we extracted the vibration signals measured by the accelerometers installed on the engines, that are divided into two classes, "healthy" (label 0) and "faulty" (label 1), depending on the axis they belonged to. Recall, also, that for some of them (basically all but 4_2 and 6_3) we could use two signals (motor and multiplier) rather then just one (motor only): in order to verify that the classification could effectively benefit of multiplier vibrations, for those datasets we repeated the analysis twice, one with and one without this kind of measurements. We then proceed extracting the complex spectrograms from those signals: setting the segmentation window to 1.0 second and the parameter `n_fft` to 300, we obtain a dataset of images of dimension $(1, 151, 171)$ (just like the ones in figure 5.7). In the cases with the multiplier, we simply consider the two spectrograms as two independent channels of the same image: in this way, the inputs will have shape $(2, 151, 171)$.
In table 5.2 we summarized the accuracy and loss obtained for all the single validation datasets provided by Bonfiglioli, via binary classification for 100 epochs.

Dataset	**Samples**	**Fault type**	**Accuracy (%)**	**Loss** $(\cdot 10^{-3})$
4_2	3508	Unbalance	100 - 100	0.014 - 0.052
5_3	6071	Bearing	99.61 - 99.61	7.311 - 8.481
5_3 w/ multi	6071	Bearing	99.80 - 99.74	5.014 - 7.398
6_3_1	700	Bearing	100 - 100	0.018 - 0.042
6_3_2	2102	Bearing	100 - 100	0.004 - 0.008
10_1	5966	Solar	99.40 - 99.53	6.895 - 6.300
10_1 w/ multi	5966	Solar	99.48 - 99.41	6.710 - 7.638
10_4	5979	Solar	100 - 99.22	3.445 - 10.345
10_4 w/ multi	5979	Solar	100 - 99.04	5.142 - 11.538
11_1	3744	Bearing	100 - 100	0.001 - 0.002
11_1 w/ multi	3744	Bearing	100 - 100	0.001 - 0.028
13_1	3750	Pinion	100 - 100	0.008 - 0.014
13_1 w/ multi	3750	Pinion	100 - 100	0.001 - 0.007
15_1	2511	Crown	100 - 100	0.001 - 0.040
15_1 w/ multi	2511	Crown	100 - 100	0.039 - 2.025
15_2	1498	Crown	100 - 100	0.027 - 0.937
15_2 w/ multi	1498	Crown	100 - 100	0.056 - 0.164

Table 5.2: Binary classification of single Bonfiglioli datasets (real - complex).

5.3.1 Multiclass Classification

After performing all the binary classification tasks on the single datasets provided, is time to verify if our model are effectively able to distinguish also among different types of fault.
We setup, then, a multiclass classification problem, with 6 possible labels: *healthy*, *unbalance*, *bearing*, *solar*, *pinion*, *crown*. In order to have a properly balanced dataset, with all those classes being equally represented, and also to reduce the memory complexity of this task, we decided to adopt only one cycle of measurements for each dataset considered, i.e. instead of taking all the 120h of 4_2 we took only the first 24h. And this is a reasonable solution to guarantee equilibrium in the distribution of labels, given the heterogeneous collections of data provided. The equity, in reality, is not total: each dataset is composed by one half of healthy signals, and so this class will be represented, in the final distribution, five times more respect to the others. We could have bypassed this problem simply by taking subsets of healthy data every time, but, apparently, the results have not been affected so much, and so we left the proportions as they were. In the end this "comprehensive" dataset is constituted by 4612 samples (complex spectrogram).
Back to the classification, the training setup and the architectures are essentially the same as before: the only difference is that now we have 6 different classes, and so the last layer of both networks should then have 6 units rather than 2. In figure 5.9 we can observe and analyze the training loops'

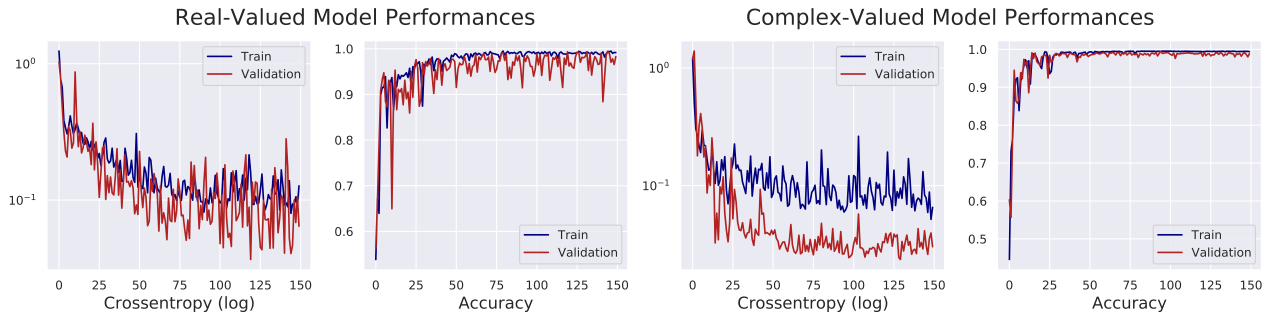


Figure 5.9: Training loops of real and complex-valued models for the Bonfiglioli multiclass classification problem.

scores of both the real and the complex-valued models over this comprehensive dataset. Differently from the situations before, this time it seems that something emerged. The first contrast that comes to our eyes is of course the stability: for the complex network we see the train and validation losses properly converging together, while, the real case, especially for the test line, is characterized by a lot of fluctuations. In the end, however, we see that the final performances are not so different: in table **??** we reported a few statistical estimators of the results, while in figure 5.10 we can check the confusion matrices of both models. Just like before, we are nearby the perfect classification, with all

	Accuracy	**Precision**	**Recall**	**F1Score**
Healthy	0.986 - 0.990	1.000 - 0.988	0.973 - 0.992	0.987 - 0.990
Unbalance	0.991 - 0.999	0.900 - 0.989	1.000 - 1.000	0.947 - 0.994
Solar	0.997 - 0.994	1.000 - 0.954	0.979 - 1.000	0.990 - 0.977
Bearing	0.992 - 0.997	0.941 - 1.000	1.000 - 0.972	0.969 - 0.986
Pinion	1.000 - 0.997	1.000 - 1.000	1.000 - 0.964	1.000 - 0.982
Crown	1.000 - 0.997	1.000 - 1.000	1.000 - 0.966	1.000 - 0.983

Table 5.3: Statistical estimators of the Bonfiglioli analysis (real - complex).

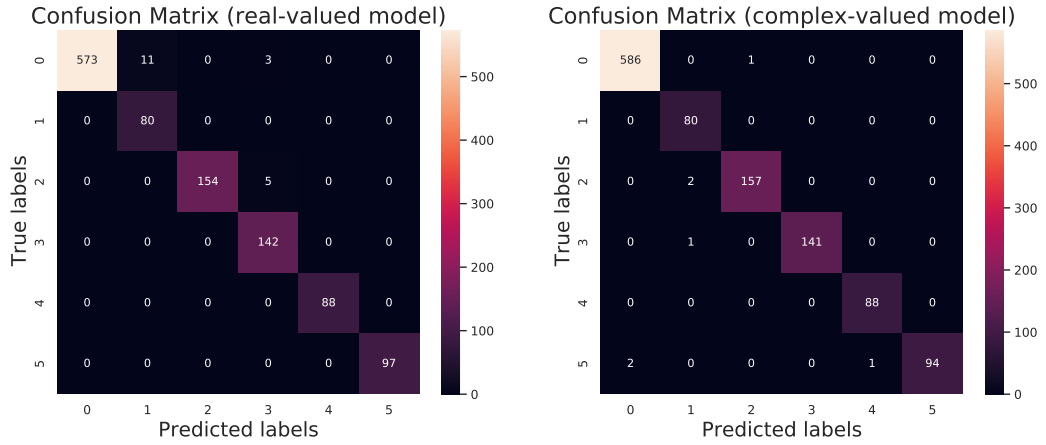


Figure 5.10: Confusion matrices of the Bonfiglioli multiclass classification problem.

the statistics almost reaching saturation. For this reason we believe that in this kind of situations, a qualitative analysis of the process' stability can be the better discriminator among two equivalent approaches.

5.3.2 Class Activation Maps

Several works have shown that the units and filters of convolutional layers in neural networks actually behaves as real object detectors, despite no supervision on the location of those objects was provided.

However, this ability is lost once the input is flattened to be sent through fully-connected layers, that is what happens also in our architectures. In order to preserve the locality of features learned by the convolutions, an interesting technique have been developed. First of all, we need to remove from the network all the linear layers, leaving the last convolution in direct contact with the output layer (that has as many units as the number of classes of the problem). Then, to replace the flattening layer, we rely on **Global Average Pooling** (GAP) [25]: a method developed first as a structural regularizer, but that was then found able to retain its remarkable localization ability until the final layer. Zhou et. al. [26] proposed a procedure for generating **Class Activation Maps** (CAM), that are visual representation of this locality, since they allow to determine the discriminative regions used by a CNN to identify a particular category. To summarize it, global average pooling returns the spatial average of the feature map for each channel of the last convolutional layer; a weighted sum of this values is then used to generate the final output.
Formally, let $f_k(x, y)$ be the activation of unit k in the last convolutional layer at spatial location (x, y). The GAP result for unit k is $F_k = \sum_{x,y} f_k(x, y)$. Thus, for a given class c, the input to the final layer is $\sum_k w_k^c F_k$, where w_k^c is the weight corresponding to class c for unit k. Essentially, w_k^c indicates the importance of F_k for class c. The class activation map for the class c is then defined as

$$M_c(x, y) = \sum_k w_k^c f_k(x, y)$$

Intuitively, we expect each unit to be activated by some visual pattern within its receptive field. The class activation map is simply a weighted linear sum of the presence of these visual patterns at different spatial locations. Upsampling those activation maps, we can identify, for each input sample, the local region that the network effectively use to determine the category.
Another important aspect of this approach is its extreme efficiency: removing all the fully-connected layers we are drastically reducing the number of learnable parameters, speeding up the train and increasing the flexibility of the model.
Back to our original problem, we implemented the CAM detection phase in the previous complex-valued architecture: we simply had to remove the final part (the multi-layer perceptron), and add a global average pooling. Running the training loop again with this "mutilated" architecture, we still managed to obtain a nice convergence, and very good accuracy values ($> 95\%$ for both the training and the test sets), even if the new model has just 456 parameters.
In figure 5.11 we represented the CAMs for each class of the Bonfiglioli dataset, in contrast with the average power spectrum that is sent through the model. Those images have been realized extracting, for each sample in the validation set, its class activation map, i.e. a linear combination of the final 8 feature maps multiplied by the weights vector corresponding to the predicted class.

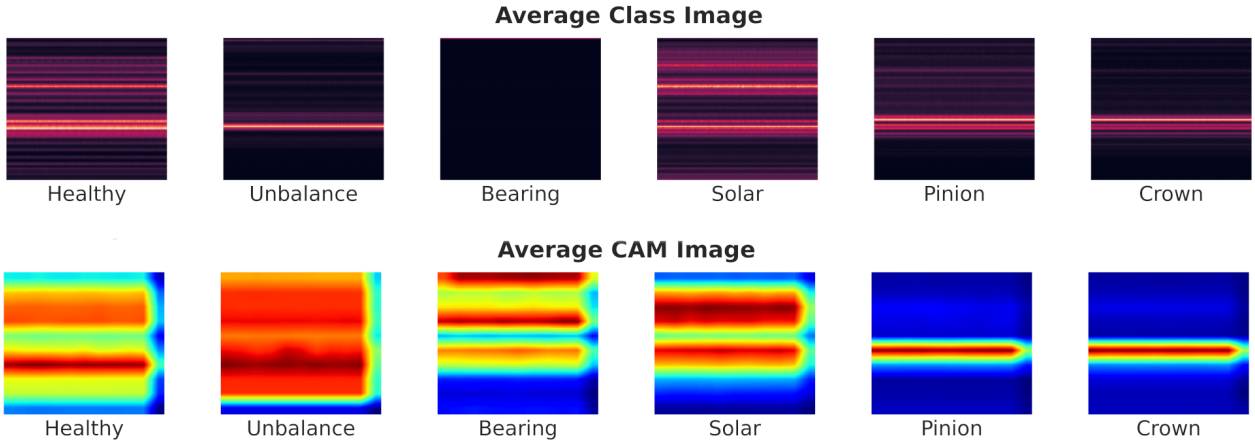


Figure 5.11: Average spectrograms and class activation maps of the Bonfiglioli validation set.

Looking at those images it is quite clear why we obtained so high performances up to now. Both the spectrograms and the activation maps are quite different from class to class, and it seems that we can

easily distinguish among them, even without the help of the networks.

5.4 Mendelay Data

The last dataset we are going to analyze for this chapter is the Mendeley Dataset [27]: a collection of vibration signals collected from bearings with different health conditions and under different time-varying speed conditions. The problem is again an healthy-faulty classification task with two different "bad" classes: faulty with an inner race defect and faulty with an outer race defect.
The experiments have been realized over a machinery fault simulator with the setup shown in 5.12. The shafts is driven by a motor which rotational speed is controlled by an AC Drive. Two ball bearings

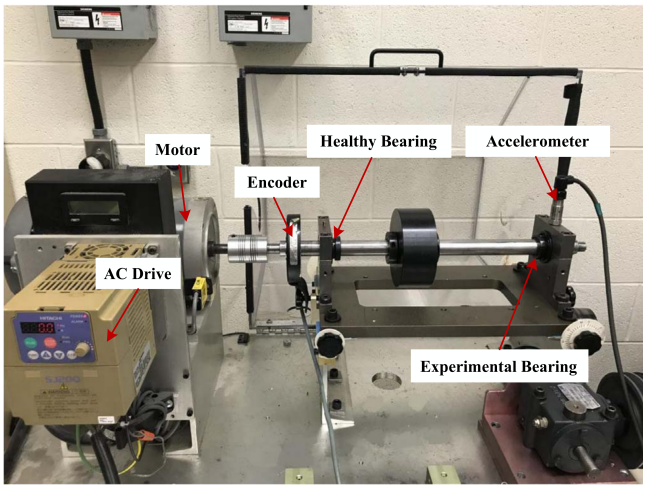


Figure 5.12: Experimental setup for Mendeley data.

are installed to support the shaft: the left one is healthy, while the one on the right is the experimental bearing, replaced during each experiment with a component of a different health condition. Again, vibration data are collected thanks to an accelerometer fixed on the housing of the experimental bearings, while an incremental encoder is used to collect the rotational speed values.
These datasets were collected and reorganized with the purpose of evaluating the effectiveness of methods developed for bearing fault diagnosis under time varying speed conditions. This last point is the main difference with most of vibration datasets available from actual literature, that have been collected under constant speed condition (as Bonfiglioli's). The complex-valued models we have developed so far, however, were not properly designed to work on such dynamic signals, but this still represents an opportunity to stress and push to the limit the approach we have designed.

5.4.1 Dataset Structure

Given the various general purposes for which this dataset was designed (i.e. condition monitoring and analysis of bearings' frequency characteristics under time-varying rotational speed conditions), its internal structure is slightly intricate. First of all, each sample has two channels, one for vibration data and the other for rotational speed data. Each signal is sampled at 200000 Hz and the sampling duration is 10 s. There whole collection of signals is divided into 36 different datasets, depending on two experimental settings:

- *health condition* (healthy, faulty with inner race defect, faulty with outer race defect);
- *speed varying condition* (increasing speed, decreasing speed, increasing then decreasing speed and decreasing then increasing speed).

Therefore, there are 12 different cases for configuration, each of them constituted of 3 trials, to ensure the authenticity of the data, that explain the 36 subsets. Raw signals and corresponding spectrograms are, in the end, visually similar to the ones of Bonfiglioli **??**. Data in the second channel, containing speed information, can be properly visualized via their spectrograms, looking at the frequency changes in the signal through the time (figure 5.13). For the data generation part, two parameters still needs

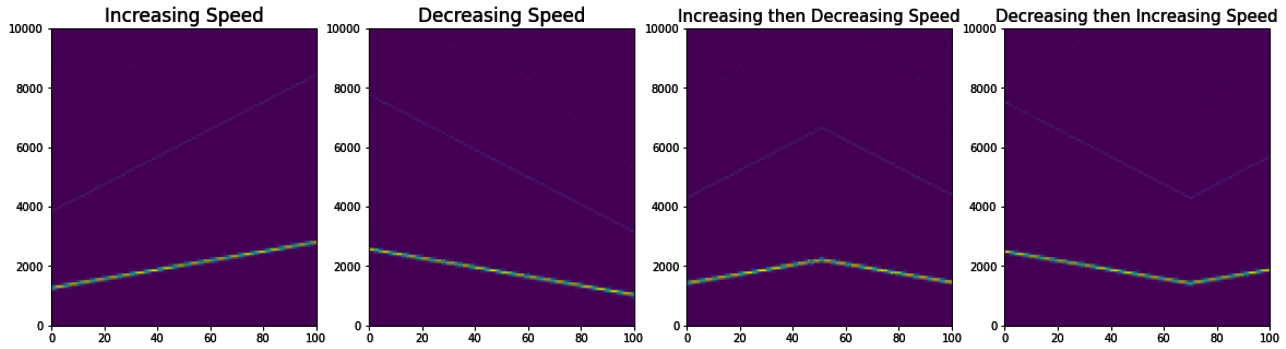


Figure 5.13: Spectrograms of the four varying-speed conditions of Mendelay data.

to be fixed:

- the `n_fft` of `torchaudio.transform.Spectrogram`, representing the resolution of the spectrogram;
- the "split rate" parameter, determining the number of subsegments in which each signal from the dataset has to be split.

The latter turns out to be quite important in this problem: recall, in fact, that each vibration registered in the dataset is 10 seconds long, with a sample rate of 200000 Hz, and so it is constituted by a total of $2 \cdot 10^6$ points, and so is quite long. Furthermore, the dataset itself is quite small: only $36 \cdot 3 = 108$ signals, number that could be increased just splitting them.

5.4.2 Mendeley Classification

We start by fixing the length of the subsignals equal to the sample rate (200000 points): as discussed in **??**, it is quite common that fault signals have periodicity, and we believe that the sampling rate provided has been chosen to properly detect this feature. Now, we have a dataset of 360, one second long, vibrations. Train-test splitting is again $75\% - 25\%$.
Even if they were not designed to deal with vibration signals registered in non-stationary conditions (with varying speed), we decided to rely on the same architecture built for Bonfiglioli analysis **??**.
We trained both models on the Mendelay dataset with all the setup described above, and the results are briefly summarized in picture 5.14. Those are very bad results: the training errors and accuracy,

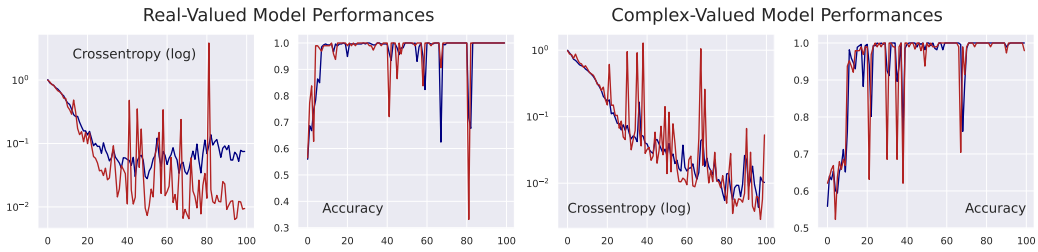


Figure 5.14: Performances achieved over the Mendelay dataset constructing one-second long sub-signals.

especially in the real-valued case, behaves nicely, but the models look like having bad generalization performances, given the high instability that we can notice on the validation lines. Now, since the architecture is already quite small, and we have already exploited the canonical techniques to prevent overfitting (normalization, dropout) what we can do is acting directly on the dataset.
Other tests realized have proved that using only the files that belong to the same speed class (increasing, decreasing, increasing then decreasing or decreasing then increasing), so keeping only monotonic or non-monotonic speed samples together, we noticed that the overfit and the instability were more limited.
Under this perspective, we believe that further reducing the length of input signals, even before constructing the spectrograms, we can manage to reduce also the effects due to high variations in

rotational speed. To prove it, we repeated the train considering samples 0.25 seconds long. In this way we have much more samples (3600) in the dataset. For completeness, it is better to specify that

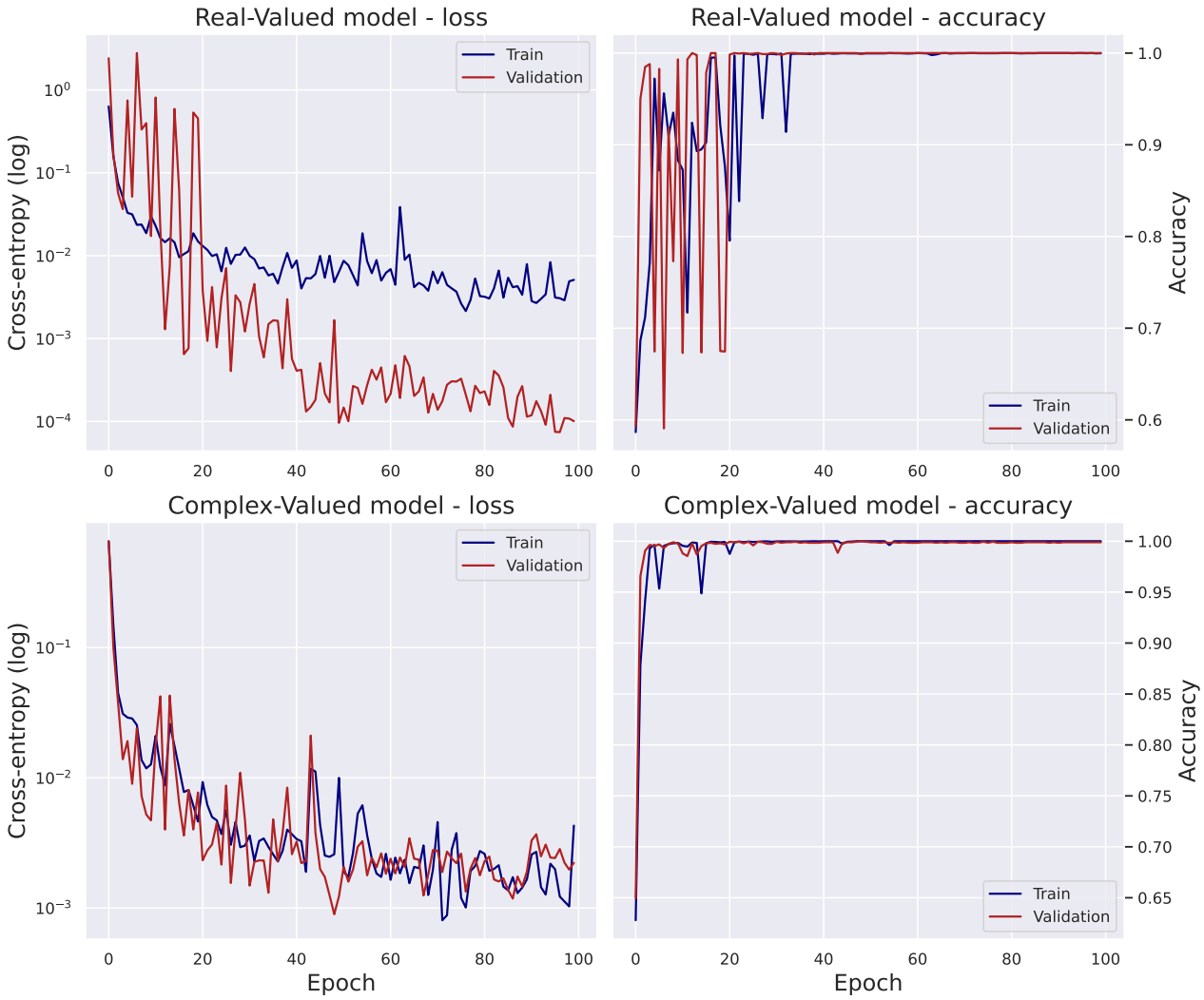


Figure 5.15: Performances achieved over the Mendelay dataset constructing 0.25 seconds long sub-signals.

to obtain the results shown in figure 5.15 we established a decay schedule for learning rate, in the hope of a faster, but more accurate, convergence.
Finally, this is a really nice behavior, with both models reaching the convergence and the perfect classification (100%) accuracy. This means that our supposition was probably right, and a suitable approach to signal classification in case of time-varying speed conditions, is to fragment the original wave into smaller samples to reduced the effects of this non-stationarity. Of course, we believe that a necessary hypotheses for this to happen is some kind of monotony in the rotational speed variation: also in the third and fourth cases 5.13, we could clearly distinguish two regions in which the velocity behavior was monotonic and amenable to the first two. Probably, an higher irregularity in the rotational speed would have made also the classification more unstable and challenging, but we would need further more datasets to verify it.

To conclude this part, we want to try stressing the generalization performances of our models. This can be done by first dividing the data files based on the four speed conditions (increasing, decreasing, increasing then decreasing, decreasing then increasing), training the networks over just one of them, and computing the classification accuracy over the others. In this way we can effectively verify which is the impact of non-stationary rotational speed conditions in the training process. The results, reported in table 5.4 are generally good: for almost all the combinations both networks managed to converge,

100.00	99.72	98.89	97.18	99.94	99.00	99.50	99.34
100.00	99.94	79.70	77.93	99.89	96.18	96.29	73.56
100.00	100.00	88.05	82.13	98.34	98.23	100.00	89.77
100.00	99.67	88.38	94.63	97.90	94.08	99.94	98.56

Table 5.4: Generalization performances of real and complex-valued models over the Mendelay dataset.

obtaining nearly the perfect classification accuracy. The cells colored in red represents the cases in which the network barely overfit and wasn't able to generalize properly, with the validation loss growing instead of decreasing during the training. There are two cases in which both models failed, and one seeing the only the complex network to do so (even with a nice accuracy, in the end). What is more interesting, instead, are the cells colored in orange: in those situations, limited to real-valued models, the loss behaved correctly, but the accuracy reached was quite limited, at least with respect to the score offered by its complex-valued counterpart. Furthermore, except for one case (the red one), all the combinations shown the complex architecture to reach higher accuracy, proving again that these kind of models are less likely to overfit and, in general, guarantees better generalization performances.

Chapter 6

Domain Adaptation

Conclusions and Future Works

During this thesis work, we collected the main instruments, extents and algorithms proposed during the years and merged them in a unique and working complex-valued deep learning framework. We tried to be as coherent as possible, remaining, at the same time, rigorous from a mathematical perspective, discussing also the main obstacles that researchers encountered during this year. The high number of tests realized, that confirmed the quality of our work, was possible thanks to the Python library we have written, that is characterized by an efficiency that the most common machine learning algorithms actually do not support for complex-valued data types.
Even if we didn't managed to get any definitive proof that complex models works better that their real-valued two-channels counterparts, as the theory seems to suggest, we had some clues about their higher robustness to overfitting and their increased generalization capabilities, that are the same impressions that similar works had during the years.
We want to insist also on the computational results achieved in the last chapter of part I, that is probably the most interesting find during this work, at least from a theoretical point of view: we found that complex and real-valued models behave differently in case of data distribution with a strong component of non-circularity. We believe that this aspect deserves further studies.

Bibliography

[1] C. Trabelsi, O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. J. Pal, "Deep complex networks." `https://arxiv.org/abs/1705.09792`, 2018.

[2] D. G. Messerschmitt, "Stationary points of a real-valued function of a complex variable," Tech. Rep. UCB/EECS-2006-93, EECS Department, University of California, Berkeley, Jun 2006.

[3] H. Li and T. Adali, "Complex-valued adaptive signal processing using nonlinear functions," *EURASIP J. Adv. Sig. Proc.*, vol. 2008, 12 2008.

[4] D. P. Reichert and T. Serre, "Neuronal synchrony in complex-valued deep networks." `https://arxiv.org/abs/1312.6115`, 2014.

[5] T. Nitta, "An extension of the back-propagation algorithm to complex numbers," *Neural Networks vol. 10 iss. 8*, vol. 10, nov 1997.

[6] K. Kreutz-Delgado, "The complex gradient operator and the cr-calculus." `https://arxiv.org/abs/0906.4835`, 2009.

[7] F. Amin, M. Amin, A. Y. H. Al Nuaimi, and K. Murase, "Wirtinger calculus based gradient descent and levenberg-marquardt learning algorithms in complex-valued neural networks," vol. 7062, pp. 550–559, 11 2011.

[8] T. Kim and T. Adali, "Fully complex multi-layer perceptron network for nonlinear signal processing," *VLSI Signal Processing*, vol. 32, pp. 29–43, 08 2002.

[9] S. Scardapane, S. V. Vaerenbergh, A. Hussain, and A. Uncini, "Complex-valued neural networks with non-parametric activation functions." `https://arxiv.org/abs/1802.08026`, 2018.

[10] P. Virtue, S. X. Yu, and M. Lustig, "Better than real: Complex-valued neural nets for mri fingerprinting." `https://arxiv.org/abs/1707.00070`, 2017.

[11] H. Akira, *Complex-Valued Neural Networks: Advances and Applications*. Wiley-IEEE Press, 2013.

[12] N. Guberman, "On complex valued convolutional neural networks." `https://arxiv.org/abs/1602.09046`, 2016.

[13] J. S. Dramsch, M. Lüthje, and A. N. Christensen, "Complex-valued neural networks for machine learning on non-stationary physical data," vol. 146, p. 104643, Jan 2021.

[14] A. Ziller, D. Usynin, M. Knolle, K. Hammernik, D. Rueckert, and G. Kaissis, "Complex-valued deep learning with differential privacy." `https://arxiv.org/abs/2110.03478`, 2021.

[15] J. A. Barrachina, C. Ren, C. Morisseau, G. Vieillard, and J.-P. Ovarlez, "Complex-valued vs. real-valued neural networks for classification perspectives: An example on non-circular data." `https://arxiv.org/abs/2009.08340v2`, 2021.

[16] R. S. Elias M. Stein, *Complex analysis Vol.2*. Princeton lectures in analysis 2, Princeton University Press, 2003.

[17] E. Ollila, "On the circularity of a complex random variable," *IEEE Signal Processing Letters*, vol. 15, pp. 841–844, 2008.

[18] P. Virtue, *Complex-valued Deep Learning with Applications to Magnetic Resonance Image Synthesis*. PhD thesis, EECS Department, University of California, Berkeley, Aug 2019.

[19] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, 01 2010.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," 2015.

[21] M. Cogswell, F. Ahmed, R. Girshick, L. Zitnick, and D. Batra, "Reducing overfitting in deep networks by decorrelating representations," 2016.

[22] G. Georgiou and C. Koutsougeras, "Complex domain backpropagation," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 39, no. 5, pp. 330–334, 1992.

[23] M. Arjovsky, A. Shah, and Y. Bengio, "Unitary evolution recurrent neural networks," *CoRR*, vol. abs/1511.06464, 2015.

[24] S. Lee, H. yu, H. Yang, I. Song, J. Choi, J. Yang, G. Lim, K.-S. Kim, B. Choi, and J. Kwon, "A study on deep learning application of vibration data and visualization of defects for predictive maintenance of gravity acceleration equipment," *Applied Sciences*, vol. 11, p. 1564, 02 2021.

[25] M. Lin, Q. Chen, and S. Yan, "Network in network," 2014.

[26] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," *CoRR*, vol. abs/1512.04150, 2015.

[27] H. Huang and N. Baddour, "Bearing vibration data collected under time-varying rotational speed conditions," *Data in Brief*, vol. 21, pp. 1745–1749, 2018.

[28] M. Ragab, Z. Chen, M. Wu, H. Li, C.-K. Kwoh, R. Yan, and X. li, "Adversarial multiple-target domain adaptation for fault classification," *IEEE Transactions on Instrumentation and Measurement*, vol. PP, pp. 1–1, 07 2020.

[29] P. Gardner, L. Bull, N. Dervilis, and K. Worden, "Overcoming the problem of repair in structural health monitoring: Metric-informed transfer learning," *Journal of Sound and Vibration*, vol. 510, p. 116245, 2021.

[30] H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, and M. Marchand, "Domain-adversarial neural networks." `https://arxiv.org/abs/1412.4446`, 2015.

[31] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks." `https://arxiv.org/abs/1505.07818`, 2016.

[32] J. Shen, Y. Qu, W. Zhang, and Y. Yu, "Wasserstein distance guided representation learning for domain adaptation." `https://arxiv.org/abs/1707.01217`, 2018.

[33] N. Schlömer, "cplot: Plot complex functions," Nov. 2021. If you use this software, please cite it as below.

[34] T. Farris, Frank A.; Needham, "Visual complex analysis.," *American Mathematical Monthly vol. 105 iss. 6*, vol. 105, jun 1998.

[35] N. Schlömer, "cplot: Plot complex functions."

Appendix A

Mathematical Proofs

A.1 Complex Weights Initialization [1]

For complex-valued deep learning traditional approaches of Glorot [19] and He [20], are no more suitable to be used for weights initialization. So we need to re-derive, or at least adapt, those efficient methods also for the complex domain.
Given a generic neural network layer, let's call $\mathbf{w} \in \mathbb{C}$ its set of complex-valued weights, that we prefer written in polar form:

$$\mathbf{w} = \|\mathbf{w}\| e^{i\theta}$$

The variance of this set is defined as

$$\mathrm{Var}(\mathbf{w}) = \mathbb{E}\left[\mathbf{w}\bar{\mathbf{w}}\right] - \left(\mathbb{E}\left[\mathbf{w}\right]\right)^2 = \mathbb{E}\left[\|\mathbf{w}\|^2\right] - \left(\mathbb{E}\left[\mathbf{w}\right]\right)^2$$

that, if the parameters are symmetrically distributed around 0, reduces to $\mathbb{E}\left[\|\mathbf{w}\|^2\right]$.
In order to compute these quantities, we rely on the fact that the magnitude of a standard complex normally distributed variable follows the Rayleigh distribution [1], i.e. basically a Chi-Square with two degrees of freedom. Just for knowledge, its probability density function depends only on a single parameter and writes

$$f(x;\sigma) = \frac{x}{\sigma^2} e^{-x^2/(2\sigma^2)}$$

So we can find a relation among the variances of $\mathbf{w}$ and its magnitude:

$$\mathrm{Var}(\|\mathbf{w}\|) = \mathrm{Var}\,\mathbf{w} - \left(\mathbb{E}\left[\|\mathbf{w}\|\right]\right)^2 \qquad \longrightarrow \qquad \mathrm{Var}(\mathbf{w}) = \mathrm{Var}(\|\mathbf{w}\|) + \left(\mathbb{E}\left[\|\mathbf{w}\|\right]\right)^2$$

But now, that we know the analytical distribution followed by $\|\mathbf{w}\|$, we can derive also the two addends of the sum above:

$$\mathbb{E}\left[\|\mathbf{w}\|\right] = \sigma\sqrt{\frac{\pi}{2}}, \qquad \mathrm{Var}\,\|\mathbf{w}\| = \frac{4-\pi}{2}\sigma^2$$

The variance of $\mathbf{w}$ can thus be expressed in terms of its generating Rayleigh distribution's single parameter σ:

$$\mathrm{Var}(\mathbf{w}) = \frac{4-\pi}{2}\sigma^2 + \left(\sigma\sqrt{\frac{\pi}{2}}\right)^2 = 2\sigma^2$$

How can we determine σ?

- Following the Xavier initialization [19], we would exploit a normal distribution (or a truncated-normal) with variance $\mathrm{Var}(\mathbf{w}) = 2/(n_{in} + n_{out})$, with n_{in} and n_{out} being the number of input and output units, respectively. For continuity, we have now to set

$$\sigma = 1/\sqrt{n_{in} + n_{out}}$$

[1] Rayleigh Distribution

- With the He initialization [20], instead, we used still a normal distribution, but with a variance depending only on input units, i.e. $\text{Var}(\mathbf{w}) = 2/n_{in}$, for which we have to set correspondingly

$$\sigma = 1/\sqrt{n_{in}}$$

The magnitude of the complex parameters is then initialized using a Rayleigh distribution with an appropriate σ, while their phase (that never appeared in the equations) can be set uniformly in $[-\pi, \pi]$.

A.2 Stationary points of a real-valued function of a complex variable [2]

Let $f(z) : \mathbb{C} \to \mathbb{R}$ be a real-valued function of a complex variable z, and let's say that we want to find the extreme points of f (i.e. the values of z for which f is maximum or minimum) exploiting the differential calculus.
As explained also in 2.2, differentiability in the complex plane is a quite strong assumption, and there are many function for which the stationarity condition in a point z_0,

$$\left.\frac{\partial f}{\partial z}\right|_{z=z_0}$$

cannot even be computed, because the limit in 2.1.3 is not the same from any direction.

A first approach to avoid the complex differentiability is to reformulate the problem in terms of two real variables, i.e. the real and imaginary parts of $z = x+iy$. Writing (with a minor abuse of notation) $f(z) = f(x, y)$, now the stationarity condition for a point z_0 becomes:

$$\left.\frac{\partial f}{\partial x}\right|_{z=z_0} = \left.\frac{\partial f}{\partial y}\right|_{z=z_0} = 0$$

While this method works, it is cumbersome because it involves the extra step of substituting $x + iy$ for z. Thus, we seek an equivalent method that works directly with the complex variable.

Assume now that f can be represented as $f(z) \equiv g(z, \bar{z})$, where g is an analytic function of two complex variables z and $\bar{z}$ that can be freely differentiated with respect to its arguments (because of the assumption of analyticity). Now, substituting $g(z, \bar{z})$ for $f(z)$, we can apply the chain rule of differential calculus to the stationarity conditions:

$$\frac{\partial f}{\partial x} = \frac{\partial g}{\partial z}\frac{\partial z}{\partial x} + \frac{\partial g}{\partial \bar{z}}\frac{\partial \bar{z}}{\partial x} = 0 \qquad \frac{\partial f}{\partial y} = \frac{\partial g}{\partial z}\frac{\partial z}{\partial y} + \frac{\partial g}{\partial \bar{z}}\frac{\partial \bar{z}}{\partial y} = 0$$

Evaluating the four derivatives, this becomes

$$\frac{\partial f}{\partial x} = \frac{\partial g}{\partial z} + \frac{\partial g}{\partial \bar{z}} = 0 \qquad \frac{\partial f}{\partial y} = i\frac{\partial g}{\partial z} - i\frac{\partial g}{\partial \bar{z}} = 0$$

which has a unique solution

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial \bar{z}} = 0$$

We have uncovered that the stationary point can be found by taking the partial derivatives of $f(z)$ with respect to both z and $\bar{z}$, considering them as independent variables, and setting those derivatives to zero.

Notice that, when f is real-valued, the complex condition yield redundant information. In this case, in fact, both $\partial f/\partial x$ and $\partial f/\partial y$ must be real-valued, and so

$$\frac{\partial f}{\partial x} = \overline{\left(\frac{\partial f}{\partial x}\right)} \qquad \frac{\partial f}{\partial y} = \overline{\left(\frac{\partial f}{\partial y}\right)}$$

Plugging in the earlier expressions for these derivatives, we can solve them arriving at the conclusion that $\partial f/\partial z = \partial f/\partial \bar{z}$. This establish the redundancy and allows to rewrite the stationarity condition for a real-valued complex function:

$$\frac{\partial f}{\partial \bar{z}} = 0$$

These results are easily extended to the case of vectors of complex variables. Let $\mathbf{z} = [z_1, z_2, \ldots, z_n]^T$ and assume that $g(\mathbf{z}, \bar{\mathbf{z}})$ is an analytic function of the complex vector $\mathbf{z}$ as well as its conjugate. Then the condition fo a stationary points becomes

$$\nabla_{\mathbf{z}} g = \mathbf{0} \qquad \nabla_{\bar{\mathbf{z}}} g = \mathbf{0}$$

or only the latter if g is real-valued.

A.3 Steepest complex gradient descent [3]

Once derived necessary and sufficient conditions for a certain $z_0 \in \mathbb{C}$ to be a stationary point of a real-valued function $f(z) : \mathbb{C} \to \mathbb{R}$, it is time to derive also the best optimization procedure. Using Wirtinger calculus we managed to overcome the strong requirements necessary to complex differentiability, but now we have twice the derivatives to compute ($\partial f/\partial z$ and $\partial f/\partial \bar{z}$). So, which direction should our gradient descent algorithm follow, in order to properly optimize f?

Let's start defining the gradient vector $\nabla_{\mathbf{z}} = [\partial/\partial z_1 \, , \partial/\partial z_2 \, , \ldots , \partial/\partial z_n]$ for the vector $\mathbf{z} = [z_1, z_2, \ldots, z_n]$ with $z_k = z_{k,Re} + i z_{k,Im}$ in order to write the first order Taylor series expansion for a function $g(\mathbf{z}, \bar{\mathbf{z}}) : \mathbb{C}^n \times \mathbb{C}^n \to \mathbb{R}$,

$$\Delta g = \langle \Delta \mathbf{z}, \nabla_{\bar{\mathbf{z}}} g \rangle + \langle \Delta \bar{\mathbf{z}}, \nabla_{\mathbf{z}} g \rangle = 2 \operatorname{Re} \{ \langle \Delta \mathbf{z}, \nabla_{\bar{\mathbf{z}}} g \rangle \}$$

where $\langle \cdot, \cdot \rangle$ is the canonical *inner product* in $\mathbb{C}^n$, and the last equality holds because g is real-valued. Using the Cauchy-Schwarz inequality, it is easy to show that the first-order change in g will be maximized when $\Delta \mathbf{z}$ and the cogradient $\nabla_{\bar{\mathbf{z}}}$ are collinear. Hence, it is the gradient with respect to the conjugate of the variable that defines the direction of the maximum rate of change in the function with respect to $\mathbf{z}$, and not the ordinary gradient $\nabla_{\mathbf{z}}$.
Thus, the gradient optimization of g should use the update rule

$$\Delta \mathbf{z} = -\alpha \nabla_{\bar{\mathbf{z}}} g$$

as this form leads to a non-positive increment given by $\Delta g = -2\alpha \|\nabla_{\bar{\mathbf{z}}}\|^2$, while the same rule but exploiting the other gradient would result in an update of $\Delta g = -2\alpha \operatorname{Re} \{ \langle \nabla_{\bar{\mathbf{z}}} g, \nabla_{\mathbf{z}} g \}$, which are not guaranteed to be non-positive.

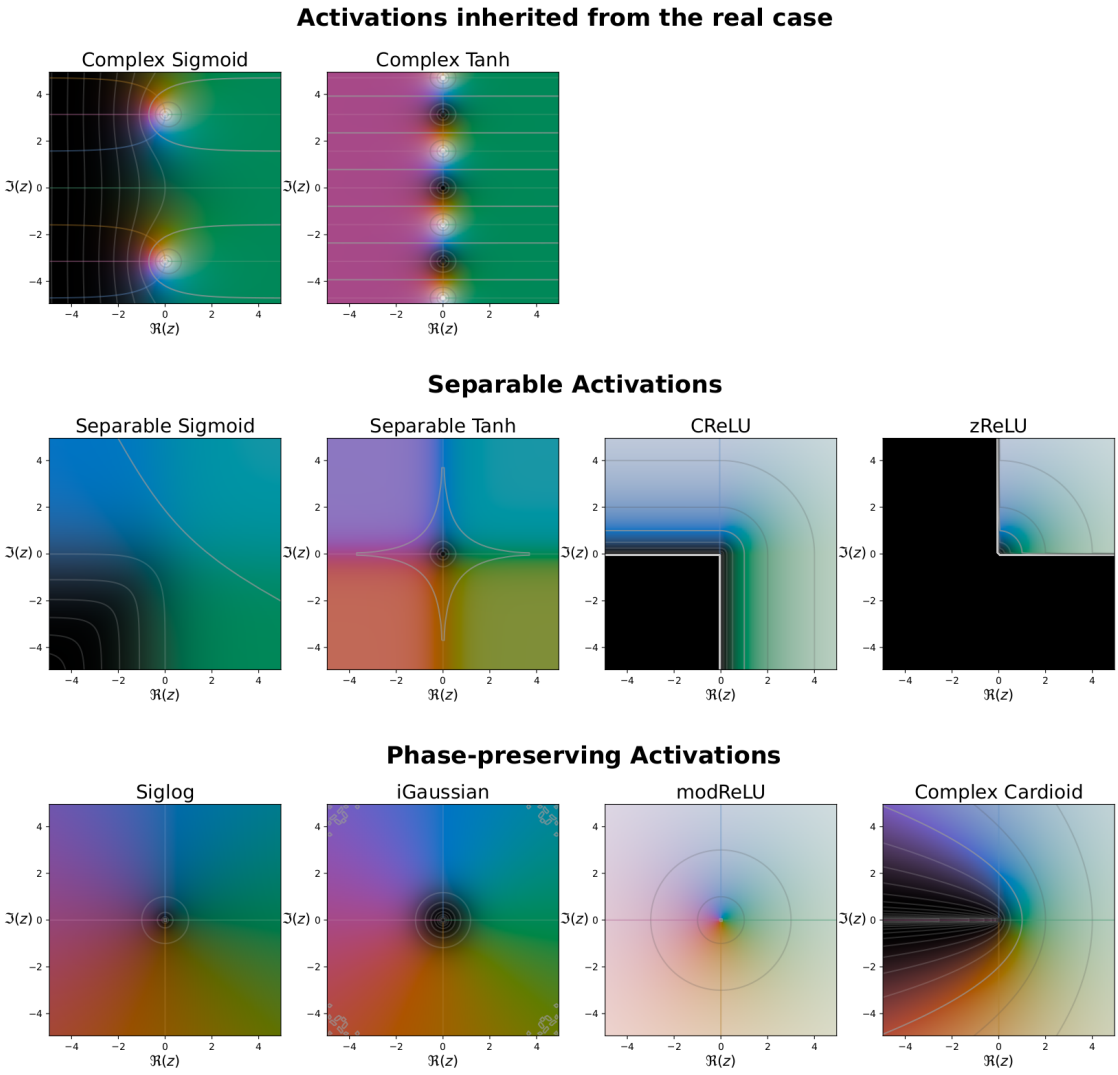
Appendix B

Activation Functions in the Complex Domain

This appendix has been realized in order to give a visual overview of the complex-valued activation functions discussed in section 3.4.
However, providing a complex representation of a function with values in $\mathbb{C}$ is a quite challenging task, but we managed to do it thanks to a library called `cplot` [35].
This library basically combines the three "historical" ways to plot complex functions:

- showing only the absolute value, eventually with a 3D plot;
- showing only the phase/argument in a color wheel;
- showing contour lines for both modulus and arguments.

By default, `cplot`, in fact, uses a perceptually uniform color space for the arguments: from green ($\theta = 0$) to blue ($\theta = \pi/2$) and from orange ($\theta = -\pi/2$) to pink ($\theta = \pi$). The modulus, instead, follows a gray scale: from the darker areas ($m \to 0$) to the lighter ones ($m \to \infty$). All of this is combined with a contour that emphasize the region with modulus 1.


Activations inherited from the real case
Complex Sigmoid
Complex Tanh
Separable Activations
Separable Sigmoid
Separable Tanh
CReLU
zReLU
Phase-preserving Activations
Siglog
iGaussian
modReLU
Complex Cardioid
ℑ(z)
ℜ(z)