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 2020/2021

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 implementing ...

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

- 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.
- most methods nowadays work like: drop phase info, take modulus or 2 independent channels, ignoring inherent relationships among them
- complex-valued deep learning grew very slowly during the years, due to a long list of factors that we will list during the thesis. there are inherently complex data (e.g. MRI signals) and also real data can be "moved" to the complex space (e.g. via Fourier transform)

1.2 Previous work

- [4]: Modern networks apply real-valued transformations on the data. Particularly, convolutions in convolutional neural networks discard phase information entirely. Many deterministic signals, such as seismic data or electrical signals, contain significant information in the phase of the signal. We explore complex-valued deep convolutional networks to leverage non-linear feature maps. While it has been shown that phase content can be restored in deep neural networks, we show how including phase information in feature maps improves both training and inference from deterministic physical data. Furthermore, we show that the reduction of parameters in a complex network outperforms larger real-valued networks.

1.3 Neuronal Synchrony

1.4 Thesis Organization

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 [5].

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 [7], 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 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.
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.

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

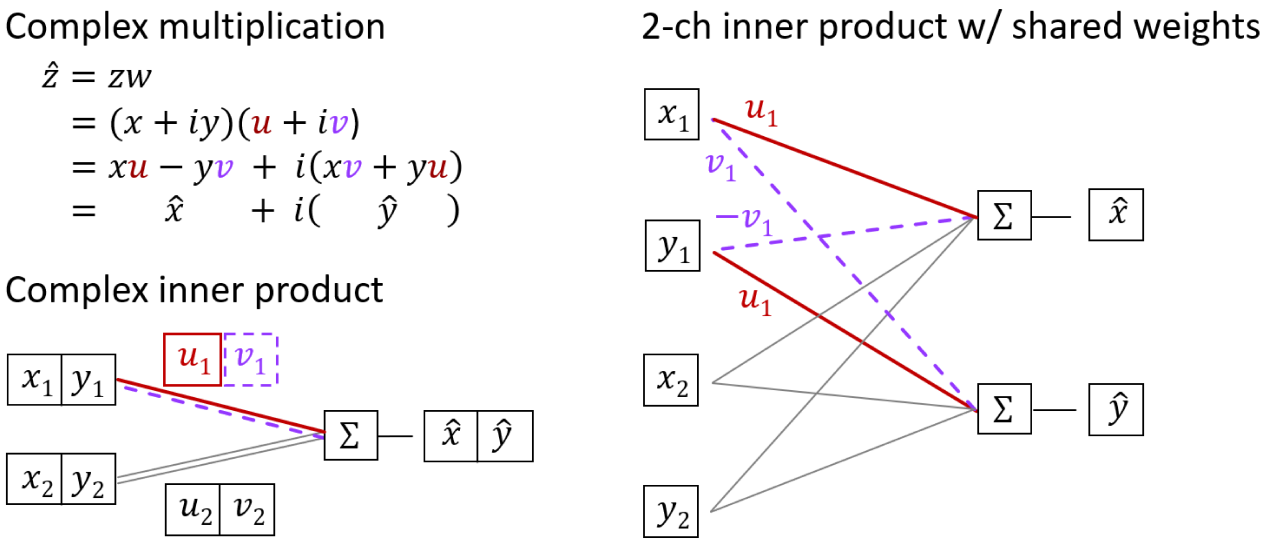

Figure 2.1: Visual representation of the multiplication and the corresponding implementation in a complex-valued network (bottom left). In order to replicate the same behavior on a real-valued network, the weights' components must be replicated, or shared, in the 2-channels layer (on the right). (source: [8])

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 2.1, 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 2.1 (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 2.2, 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), compared 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 (2.2, left), the only possible effects is a re-scaling of the input;
- in the complex plane (2.2, 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 (2.2, 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.

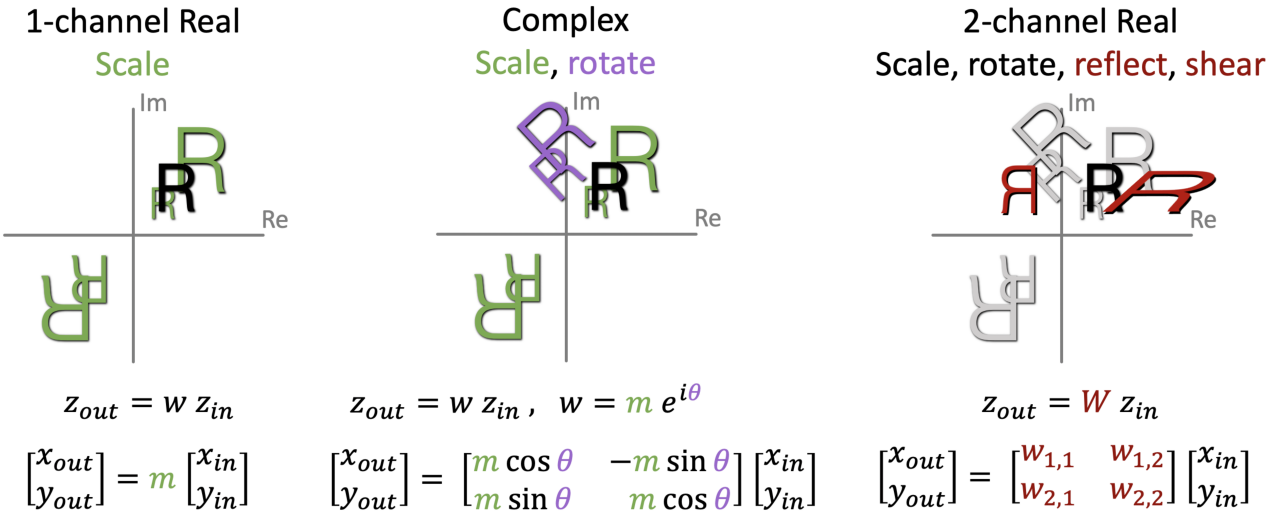

Figure 2.2: Visual representation of the complex multiplication (center) compared to its real-valued implementations (1-ch left, 2-ch right). (source: [8])

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.

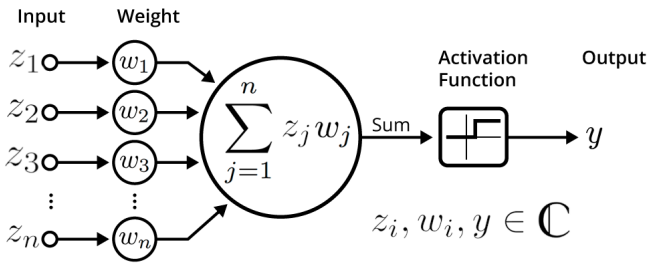

Figure 3.1: Fundamental unit (neuron) of a complex-valued neural network.

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 [7]. 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 [7], 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) [9]. 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 [10].
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, 10] 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.

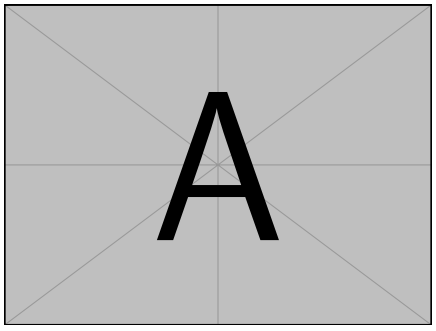

Figure 3.2: Complex Gradient Descent.

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.

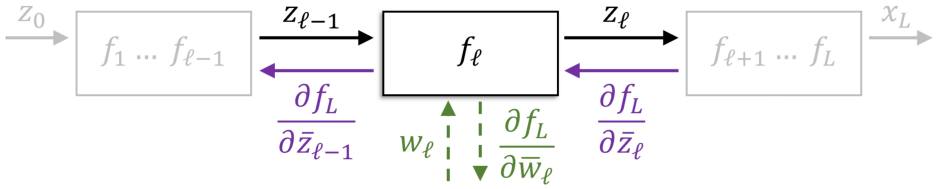

Figure 3.3: Forward and backward pass through a complex layer. (source [8])

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

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

Table 3.1: Comparison of backpropagation calculus. (source: [8])

3.3 it is effectively illustrated how input and derivatives flow through the layers during the forward and backward pass, respectively.
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 [11] or by He [12] (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:

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

  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

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

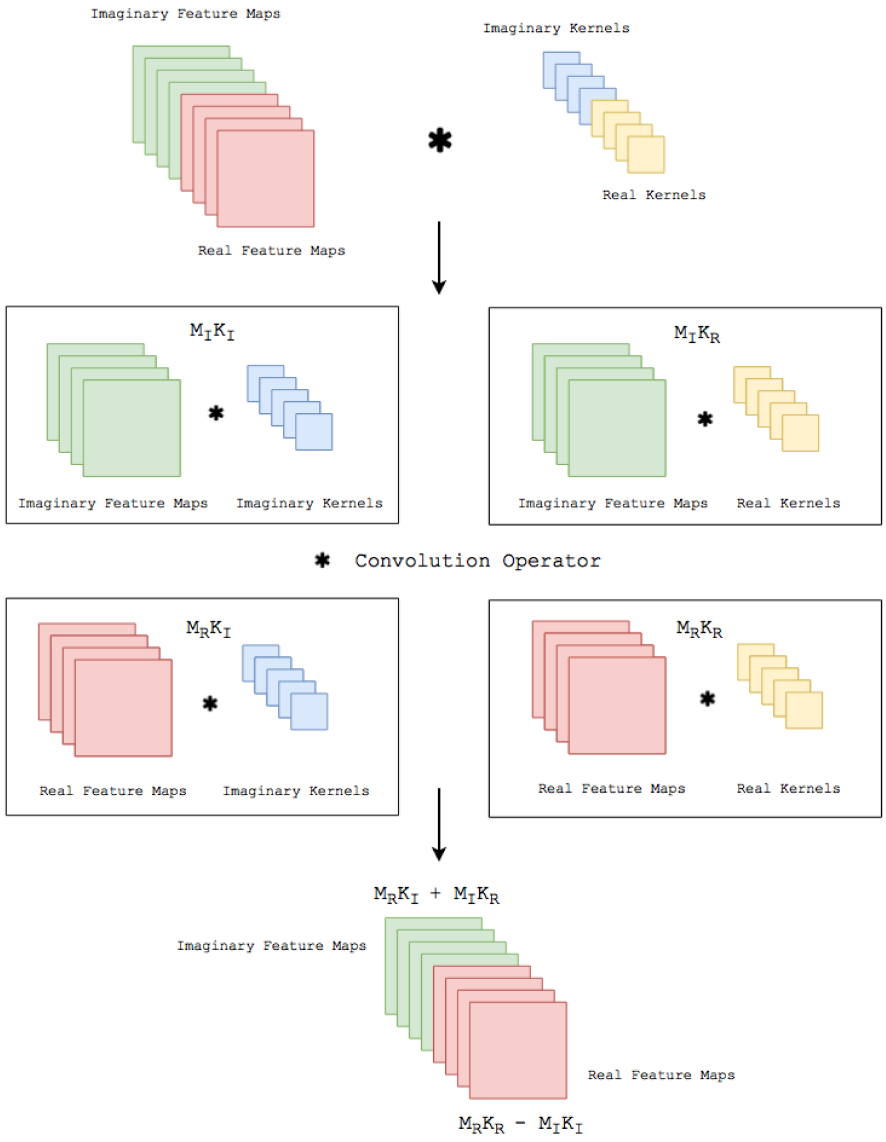

Figure 3.4: Implementation details of the Complex Convolution (by [1]).

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

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

[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).

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 [13].
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.

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 `hypoerbolic 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. 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.

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 $\{\mathrm{Re}(z) > 0,\ \mathrm{Im}(z) = 0\} \cup \{\mathrm{Re}(z) = 0,\ \mathrm{Im}(z) > 0\}$.

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` [8]. 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.
We will see, in our applications, that the cardioid effectively allows complex networks to converge in a fast and also stable way.

A brief recap

Activation	Analytic Form	Reference
Sigmoid	$\sigma(z)$	//
Hyperbolic Tangent	$\tanh(z)$	//
Split-Sigmoid	$\sigma(\mathrm{Re}(z)) + i\sigma(\mathrm{Im}(z))$	[7]
Split-Tanh	$\tanh(\mathrm{Re}(z)) + i\tanh(\mathrm{Im}(z))$	[7]
$\mathbb{C}$ReLU	$ReLU(\mathrm{Re}(z)) + iReLU(\mathrm{Im}(z))$	[1]
zReLU	z if $z \in [0, \pi/2]$, 0 otherwise	[14]
Siglog	$\sigma(\log \|z\|)e^{-i\theta_z}$	[15]
iGauss	$\left(1 - e^{-\frac{z\bar{z}}{2\sigma^2}}\right)\frac{z}{\|z\|}$	[8]
modReLU	$ReLU(\|z\| + b)e^{i\theta_z}$	[16]
Cardioid	$\frac{1}{2}\left(1 + \cos(\angle z)\right) z$	[8]

Table 3.2: Recal 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 3.5: 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

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] 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.

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

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

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

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

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

[10] 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.

[11] 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.

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

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

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

[15] 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.

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

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

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

[23] 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.

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

[25] 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.

[26] 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.

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

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

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

Appendix A

Mathematical Proofs

A.1 Complex Weights Initialization [1]

For complex-valued deep learning traditional approaches of Glorot [11] and He [12], 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 [11], 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] source

- With the He initialization [12], 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.