Université de Montréal

Applications of Complex Numbers to Deep Neural Networks

par

Olexa Bilaniuk

Département de mathématiques et de statistique
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures et postdoctorales
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en informatique

August 31, 2018

Sommaire

Dans la dernière décennie, une heureuse confluence de matériel, de logiciels et de théorie ont permis à l'intelligence artificielle de connaître un renouveau: un "printemps" et qui, contrairement au passé, semblent avoir mené non pas à la déception d'un autre "hiver", mais à un "été" durable, rempli de réelles avances.

Une de ces récentes avances est l'entrée en scène de l'apprentissage véritablement "profond". Dans maintes applications les architectes de réseaux de neurones ont connu du succès en les approfondissant, et plus personne ne doute de l'utilité de représentations profondes, composées, hiérarchiques, apprises automatiquement à base d'exemples.

Mais il existe d'autres avenues, moins explorées, qui pourraient être utiles, comme l'emploi d'alternatives au système numérique le plus commun, les nombres réels: nombres à basse précision, nombres complexe, quaternions.... En 2017, moi-même et l'un de mes principaux collaborateurs discutâmes du manque d'intérêt accordé au traitement en nombres complexes et à l'analyse de signaux complexes ou aisément convertis en une série de nombres complexes grâce à la transformée de Fourier (1D, 2D, à court terme ou non). Puisque ce secteur semblait peu exploré, nous nous y sommes lancés et, au terme d'une année passée à relever des défis propres à l'architecture et l'initialisation d'un réseau de neurones n'employant que des nombres complexes, nous avons débouché sur des résultats prometteurs en vision informatique et en traitement de musique. Nous déjouons aussi les pièges d'une initialisation et d'une normalisation naïve de ce type de réseau de neurones avec des procédures adaptées.

Mots-clés: Intelligence artificielle, Apprentissage profond, Apprentissage automatique, réseaux de neurones, nombres complexes.

Summary

In the past decade, a convergence of hardware, software and theory have allowed artificial intelligence to experience a renewal: a "spring" that, unlike previous times, seems to have led not to a burst hype bubble and a new "AI winter", but to a lasting "summer", anchored by tangible advances in the field.

One of the key such advances is truly "deep" learning. In many applications, the architects of neural networks have had great success by deepening them, and there is now little doubt about the value of deep, composable, hierarchical, automatically-learned-from-examples representations.

But there exist other, less-well-explored avenues for research, such as alternatives to the real-valued number system most commonly used: low-precision, complex, quaternions.... In 2017, myself and one of my primary collaborators discussed the seeming lack of interest given to purely complex-valued processing of digital signals, either directly available in complex form or convertible to such using e.g. the Fourier Transform (1D, 2D, short-time or not). Since this area seemed under-explored, we threw ourselves into it and, after a year spent dealing with the challenges of neural networks with purely complex-valued internal representations, we obtained good results in computer vision and music spectrum prediction. We also expose the pitfalls of naively initializing and normalizing such complex-valued networks and solve them with custom formulations adapted for the use of complex numbers.

Keywords: Artificial intelligence, Deep learning, Machine learning, Neural networks, complex numbers.

Contents

List of Tables

List of Figures

List of Acronyms & Abbreviations

DSP	Traitement numérique du signal, de l'anglais *Digital Signal Processing*
CNN	Réseau de neurones convolutionel, de l'anglais *Convolutional Neural Network*
RNN	Réseau de neurones récurrent, de l'anglais *Recurrent Neural Network*
LSTM	Une architecture particulière de RNN, de l'anglais *Long-Short Term Memory*
MSE	Erreur quadratique moyenne, de l'anglais *Mean Square Error*
FFT	Transformée de Fourier rapide, de l'anglais *Fast Fourier Transform*
STFT	Transformée de Fourier à court terme, de l'anglais *Short-Time Fourier Transform*
ReLU	Type de non-linéarité commune, de l'anglais *Rectified Linear Unit*

Acknowledgements

I would first like to thank my supervisors, Yoshua Bengio and Roland Memisevic, for the opportunity and funding to study and research at the then-MILA (now Mila) laboratory, and the supervision they gave me. I would also like to thank my parents, Nykolai & Miroslawa Bilaniuk and my brother and sister Borys & Ksenia Bilaniuk for their support and visits in Montreal. Last but not least, I'd like to thank my colleagues Chiheb Trabelsi, Rosemary Nan Ke, Anirudh Goyal, Faruk Ahmed, Dimitry Serdyuk, Devansh Arpit, Akram Erraqabi and many others for supporting, pressuring, cajoling or dragging me through to the finish line.

Introduction

In recent years, deep neural networks have burst onto the scene of artificial intelligence, and since then firmly established themselves in many areas previously thought of as the exclusive domain of humans.

A watershed moment in AI was the crushing victory, in 2012, of neural networks in the famed computer vision challenge ILSVRC (ImageNet Large-Scale Visual Recognition Challenge) [1], the task of classifying correctly natural images into one of 1000 different object categories. A neural network, AlexNet [2], obtained a top-5 error rate of 15.3%, whereas its closest competitor obtained 26.2%. In the years since, neural networks have dominated this competition, and now plumb depths of as low as 5.79% [3], firmly establishing computer vision as a killer application of deep convolutional neural networks.

Vision is not the only human signal-processing task deep neural networks have been successfully applied to. Deep recurrent neural networks have successfully emulated various aspects of human hearing and speaking: source separation [4], speech recognition [5] and text-to-speech [6]. They have also been used for vision tasks fundamentally more complex than the mere classification of still images, for instance hand gesture recognition [7], which requires reasoning across both space and time.

All of these tasks have something in common: They are highly-refined examples of artificially-intelligent *digital signal processing* (DSP). They have something else in common: None employ complex numbers as their primary numeric system, even though complex numbers arise naturally in DSP. Instead, all use real numbers, and it is vanishingly rare to see neural networks using complex numbers natively.

Complex numbers pervade many branches of mathematics and engineering. One of them is digital signal processing, where complex numbers often appear in the analysis of a filter's properties, and the study of a signal's frequency content. Filters and signals abound in the

real world. Signals exist in many types: 1-dimensional (e.g. audio), 2-dimensional (e.g. still images), 3-dimensional (e.g. video and volumetric imaging), or even higher-dimensional in rare cases. In the light of complex numbers' obvious utility and yet near-total disuse, it felt appropriate for us to investigate natively complex-valued deep neural networks for various tasks.

The rest of the document is structured as follows. In Chapter 1, we cover the basics of complex numbers and digital signal processing, to provide some context. Chapter 2 then presents our unification of these tools into *Deep Complex Networks*. Lastly, the Conclusion provides a short recapitulation and indicates promising future directions.

Chapter 1

Essentials in Digital Signal Processing

The body of this work relies heavily on complex numbers and the application of a few concepts in digital signal processing to machine learning tasks on signals. We will therefore quickly review here complex numbers, then Fourier Transforms and some basic properties.

1.1. Complex Numbers

1.1.1. Definition

The set of complex numbers, $\mathbb{C}$, is an extension of the set of real numbers $\mathbb{R}$ to a second, *imaginary* axis. A complex number $x \in \mathbb{C}$ may be written as $x = a + bi$, where i is the *imaginary unit* with the property that $i^2 = -1$, $\Re(x) = a \in \mathbb{R}$ is called the *real part* and $\Im(x) = b \in \mathbb{R}$ is called the *imaginary part*. This is called the *Cartesian* or *rectangular* form of x.

Because complex numbers are two-dimensional objects, there exists an alternative, equivalent expression $x = r\angle\theta$, called the *polar* form, where r is the *radius*, *magnitude* or *amplitude* (the distance from the origin) and θ is the *phase*, *argument* or *angle* with respect to the real axis. The polar form is particularly useful in expressing rotations and scalings, and computing exponentials.

One can readily convert between the Cartesian and polar forms:

$$r\angle\theta = (r\cos\theta) + (r\sin\theta)i \tag{1.1.1}$$

$$a + bi = \sqrt{a^2 + b^2}\ \angle\ \text{atan2}(b,a) \tag{1.1.2}$$

where atan2(y,x) is the two-argument form of the mathematical arctangent function, atan(y/x) or $\tan^{-1}(y/x)$. atan2(y,x) returns the angle of rotation counter-clockwise about the origin from the positive x-axis towards the vector (x,y). atan2(y,x) is included as a separate routine in many math libraries because of the singularities and loss of precision that atan(y/x) exhibits as $x \to 0$, and consequently $y/x \to \pm\infty$.

$$\text{atan2}(y,x) = \begin{cases} \text{atan}(y/x) & \text{if } x > 0,\ y \geq 0 \\ \frac{\pi}{2} & \text{if } x = 0,\ y > 0 \\ \pi - \text{atan}(y/x) & \text{if } x < 0,\ y \geq 0 \\ \pi + \text{atan}(y/x) & \text{if } x < 0,\ y \leq 0 \\ \frac{3\pi}{2} & \text{if } x = 0,\ y < 0 \\ 2\pi - \text{atan}(y/x) & \text{if } x > 0,\ y \leq 0 \\ \text{Indeterminate} & \text{if } x = 0,\ y = 0 \end{cases} \tag{1.1.3}$$

The real numbers $\mathbb{R}$ are the special-case subset of complex numbers $\mathbb{C}$ where $\theta = 0$ or, equivalently, $b = 0$.

1.1.2. Arithmetic

Because of the imaginary unit, arithmetic with complex numbers includes a slight twist over real number arithmetic.

Addition and **subtraction** resemble that of a two-element vector:

$$(a + bi) + (c + di) = (a + c) + (b + d)i \tag{1.1.4}$$

$$(a + bi) - (c + di) = (a - c) + (b - d)i \tag{1.1.5}$$

Multiplication must take into account that i^2 is by definition equal to -1, a real number:

$$\begin{aligned} (a + bi) \cdot (c + di) &= (ac) + (adi) + (bci) + (bdi^2) \\ &= (ac - bd) + (ad + bc)i \end{aligned} \tag{1.1.6}$$

$$(r\angle\theta) \cdot (s\angle\phi) = rs\angle(\theta + \phi) \tag{1.1.7}$$

We highlight the elegance of polar-form multiplication: Angles *add* and magnitudes *multiply*.

1.2. Digital Signal Processing

Digital signal processing is a set of techniques for digital computer analysis, manipulation and synthesis of signals. Within the context of such a computer, these signals are necessarily *discrete* (sampled at time intervals, rather that continuous-valued) and *quantized* (stored in finite precision, rather than unlimited-precision real numbers) versions of a continuous analog signal from the outside world. Examples of such digital signals are Hi-Fi 16-bit, 44100Hz studio recordings, or 8-bit-depth, 24-megapixel RGB images taken by a good-quality camera.

We will cover only a few needed DSP techniques here, and for a more complete exposition the reader is urged to refer to good-quality DSP handbooks such as the *Digital Signal Processing Handbook* published by CRC Press [8].

1.2.1. Convolution

One of the most common digital processing techniques, both in signal processing in general and deep learning specifically, is *convolution*. Convolution is a binary, associative, commutative, linear operation of two functions or signals denoted by $\mathbf{a} * \mathbf{b}$. It yields $\mathbf{a}$, "filtered" in some sense by $\mathbf{b}$ (equivalently, $\mathbf{b}$ filtered by $\mathbf{a}$). Depending on the exact shape of $\mathbf{a}$ (respectively, $\mathbf{b}$), this may have various effects, including smoothing $\mathbf{a}$, sharpening $\mathbf{a}$, or perhaps detecting very specific patterns.

Convolution over discrete signals $\mathbf{a}$ and $\mathbf{b}$ is defined as

$$(a * b)_i = \sum_{j=-\infty}^{\infty} a_j \cdot b_{i-j} \tag{1.2.1}$$

which, in the case of $\mathbf{a}$ equal to zero outside of a finite interval $[-N, M]$, reduces to a finite sum

$$(a * b)_i = \sum_{j=-N}^{M} a_j \cdot b_{i-j} \tag{1.2.2}$$

This can be understood as the dot-product of $\mathbf{a}$ with a reversed sliding-window $\mathbf{b}$.

Convolution is one of the most important and heavily used operations in deep learning, and is the defining characteristic of Convolutional Neural Networks (CNNs). It also has a deep connection to multiplication, through the Discrete Fourier Transform (DFT).

1.2.2. Discrete Fourier Transform

1.2.2.1. *Definition*

The Discrete Fourier Transform is an invertible decomposition of a discrete-time, periodic complex-valued signal x_t of N samples into a unique linear combination of N complex-valued frequency components, X_k, called its *spectrum*. It is elegantly defined, in vector and scalar form, as:

$$\mathbf{X} = \mathcal{F}\mathbf{x} \tag{1.2.3}$$

$$\begin{aligned} X_k &= \sum_{t=0}^{T-1} \mathcal{F}_{kt} \cdot x_t \\ &= \sum_{t=0}^{T-1} e^{-\frac{2\pi i}{N} kt} \cdot x_t \end{aligned} \tag{1.2.4}$$

Intuitively, all periodic signals of length N are the weighted sum of N complex exponentials, each with a different frequency, amplitude and phase. The frequency associated with X_k is $f = \frac{k}{N}$, or k cycles per N samples; The amplitude of that complex exponential is the amplitude of X_k; And the phase of that complex exponential is the phase of X_k.

It should be noted here that even if the input signal $\mathbf{x}$ is entirely real, the output spectrum $\mathbf{X}$ need not be, unless the signal satisfies additional symmetries [8].

1.2.2.2. *Convolution Theorem*

The convolution theorem states that the convolution of two signals a, b is equivalent to the Inverse Fourier Transform of the product of the Fourier Transforms of a and b. In other words, convolution and pointwise multiplication are *dual* to each other under the Fourier Transform.

$$a * b = \mathcal{F}^{-1}\left\{\mathcal{F}\{a\} \odot \mathcal{F}\{b\}\right\} \tag{1.2.5}$$

$$a \odot b = \mathcal{F}^{-1}\left\{\mathcal{F}\{a\} * \mathcal{F}\{b\}\right\} \tag{1.2.6}$$

where $\odot$ is the Hadamard (pointwise) product.

The convolution theorem applies to the Discrete Fourier Transform. However, because of the DFT's assumption that its input is periodic, "convolution" is to be interpreted here as

circular convolution, a variation of Eq. 1.2.1 where all subscripts are taken mod$N$. If this is not desired, zero-padding can be used to avoid edge effects.

The DFT implemented naively requires $O(N^2)$ computation. However, there exists an $O(N \log N)$ algorithm to compute the DFT called the *Fast Fourier Transform*. The existence and efficiency of this algorithm makes many spectral analyzes and fast convolution/multiplication algorithms possible that otherwise would not have been.

1.2.2.3. *Short-Time Fourier Transform Variant*

The DFT is very useful, but does have some disadvantages in its purest form. Obtaining the DFT of a signal yields the entire spectrum of the signal, but simultaneously loses all space/time localization information. It is often desirable to look only at the frequency content within a specific area or time interval.

Moreover, **directly supplying a spectrum to a convolutional neural network is wasteful and pointless**. Indeed, convolution over the spectrum of a signal is equivalent, by the convolution theorem, to pointwise multiplication in the time/space-domain. Pointwise multiplication of the input data is of questionable utility, and not even an optimized FFT and convolution routine could compete with pointwise multiplication in the original time or space domain of the signal.

A compromise solution is to split the signal into space/time *blocks*, and perform the DFT on these blocks. Care must be taken when doing this.

Slicing out a block from the signal corresponds to pointwise multiplication by a rectangular *window*; The spectrum of the block will thus be corrupted by convolution with the rectangular window's spectrum. A solution with better spectral properties is to choose a different windowing function, such as the Hann, Hamming, or cosine window, that taper towards 0 at both ends of the block.

Given the window functions' tapering to 0, it is also common to have an overlap between blocks, so that signal samples near the edges of blocks are not ignored.

1.2.2.4. *Multidimensional Variants*

The Fourier Transforms in one dimension defined above can be generalized to two-dimensional and higher-dimensional spaces. It retains exactly the same properties, and

can be implemented with the same algorithms. A simple way to implement it is to perform the 1-D DFT along each dimension of the signal in turn.

For instance, the 2-D DFT over an $M \times N$ image can be evaluated by computing M 1-D, N-element DFTs along each row, producing a partially-transformed image, then applying the 1-DFT on this partially-transformed image N times along its M-element columns. This is equivalent to first performing the DFT along the N M-element columns and then along the M N-element rows.

Chapter 2

Deep Complex Networks

This paper was accepted at ICLR 2018 in the conference track. Its authors were:
Chiheb Trabelsi, Olexa Bilaniuk*, Ying Zhang, Dmitriy Serdyuk, Sandeep Subramanian, João Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio & Christopher J Pal*

Contribution *Chiheb Trabelsi and myself led the project. Both of us originated the idea of implementing complex-valued deep neural networks, wrote the bulk of the paper and with the help of Sandeep Subramanian coded the bulk of the Keras Python software package we made use of in our experiment. I wrote scripts for and ran most of the computer vision experiments. Chiheb suggested a whitening-based batch normalization for complex numbers, and I found its correct form. I drew in Asymptote the figure illustrating it (§2.7.6). and Other collaborators applied our software package to audio tasks like music transcription and speech spectrum prediction.*

2.1. Introduction

Recent research advances have made significant progress in addressing the difficulties involved in learning deep neural network architectures. Key innovations include normalization techniques [9, 10] and the emergence of gating-based feed-forward neural networks like Highway Networks [11]. Residual networks [12, 13] have emerged as one of the most popular and effective strategies for training very deep convolutional neural networks (CNNs). Both highway networks and residual networks facilitate the training of deep networks by providing shortcut paths for easy gradient flow to lower network layers thereby diminishing the effects of vanishing gradients [14]. He et al. [13] show that learning explicit residuals

of layers helps in avoiding the vanishing gradient problem and provides the network with an easier optimization problem. Batch normalization [9] demonstrates that standardizing the activations of intermediate layers in a network across a minibatch acts as a powerful regularizer as well as providing faster training and better convergence properties. Further, such techniques that standardize layer outputs become critical in deep architectures due to the vanishing and exploding gradient problems.

The role of representations based on complex numbers has started to receive increased attention, due to their potential to enable easier optimization [15], better generalization characteristics [16], faster learning [17, 18, 19] and to allow for noise-robust memory mechanisms [18]. Wisdom et al. [19] and Arjovsky et al. [17] show that using complex numbers in recurrent neural networks (RNNs) allows the network to have a richer representational capacity. Danihelka et al. [18] present an LSTM [20] architecture augmented with associative memory with complex-valued internal representations. Their work highlights the advantages of using complex-valued representations with respect to retrieval and insertion into an associative memory. In residual networks, the output of each block is added to the output history accumulated by summation until that point. An efficient retrieval mechanism could help to extract useful information and process it within the block.

In order to exploit the advantages offered by complex representations, we present a general formulation for the building components of complex-valued deep neural networks and apply it to the context of feed-forward convolutional networks and convolutional LSTMs. Our contributions in this paper are as follows:

(1) A formulation of complex batch normalization, which is described in Section 2.3.5;
(2) Complex weight initialization, which is presented in Section 2.3.6;
(3) A comparison of different complex-valued ReLU-based activation functions presented in Section 2.4.1;
(4) A state of the art result on the MusicNet multi-instrument music transcription dataset, presented in Section 2.4.2;
(5) A state of the art result in the Speech Spectrum Prediction task on the TIMIT dataset, presented in Section 2.4.3.

We perform a sanity check of our deep complex network and demonstrate its effectiveness on standard image classification benchmarks, specifically, CIFAR-10, CIFAR-100. We also

use a reduced-training set of SVHN that we call SVHN*. For audio-related tasks, we perform a music transcription task on the MusicNet dataset and a Speech Spectrum prediction task on TIMIT. The results obtained for vision classification tasks show that learning complex-valued representations results in performance that is competitive with the respective real-valued architectures. Our promising results in music transcription and speech spectrum prediction underscore the potential of deep complex-valued neural networks applied to acoustic related tasks[1] – We continue this paper with discussion of motivation for using complex operations and related work.

2.2. Motivation and Related Work

Using complex parameters has numerous advantages from computational, biological, and signal processing perspectives. From a computational point of view, Danihelka et al. [18] has shown that Holographic Reduced Representations [21], which use complex numbers, are numerically efficient and stable in the context of information retrieval from an associative memory. Danihelka et al. [18] insert key-value pairs in the associative memory by addition into a *memory trace*. Although not typically viewed as such, residual networks [12, 13] and Highway Networks [11] have a similar architecture to associative memories: each ResNet residual path computes a residual that is then inserted – by summing into the "memory" provided by the identity connection. Given residual networks' resounding success on several benchmarks and their functional similarity to associative memories, it seems interesting to marry both together. This motivates us to incorporate complex weights and activations in residual networks. Together, they offer a mechanism by which useful information may be retrieved, processed and inserted in each residual block.

Orthogonal weight matrices provide a novel angle of attack on the well-known vanishing and exploding gradient problems in RNNs. Unitary RNNs [17] are based on unitary weight matrices, which are a complex generalization of orthogonal weight matrices. Compared to their orthogonal counterparts, unitary matrices provide a richer representation, for instance being capable of implementing the discrete Fourier transform, and thus of discovering spectral representations. Unitary RNNs [17] show the potential of this type of recurrent neural

[1]The source code is located at `http://github.com/ChihebTrabelsi/deep_complex_networks`

networks on toy tasks. Wisdom et al. [19] provided a more general framework for learning unitary matrices and they applied their method on toy tasks and on a real-world speech task.

Using complex weights in neural networks also has biological motivations. Reichert and Serre [22] proposed a biologically-plausible deep network that allows one to construct richer and more versatile representations using complex-valued neuronal units. The complex-valued formulation allows one to express the neuron's output in terms of its firing rate and the relative timing of its activity. The amplitude of the complex neuron represents the former and its phase the latter. Input neurons that have similar phases are called *synchronous* as they add constructively, whereas *asynchronous* neurons add destructively and thus interfere with each other. This is related to the gating mechanism used in both deep feed-forward neural networks [11, 23, 6] and recurrent neural networks [20, 24, 25] as this mechanism learns to synchronize inputs that the network propagates at a given feed-forward layer or time step. In the context of deep gating-based networks, synchronization means the propagation of inputs whose controlling gates simultaneously hold high values. These controlling gates are usually the activations of a sigmoid function. This ability to take into account phase information might explain the effectiveness of incorporating complex-valued representations in the context of recurrent neural networks.

The phase component is not only important from a biological point of view but also from a signal processing perspective. It has been shown that the phase information in speech signals affects their intelligibility [26]. Oppenheim and Lim [27] showed that the amount of information present in the phase of an image is sufficient to recover the majority of the information encoded in its magnitude. In fact, phase provides a detailed description of objects as it encodes shapes, edges, and orientations.

Recently, Rippel, Snoek, and Adams [28] leveraged the Fourier spectral representation for convolutional neural networks, providing a technique for parametrizing convolution kernel weights in the spectral domain, and performing pooling on the spectral representation of the signal. However, the authors avoid performing complex-valued convolutions, instead building from real-valued kernels in the spatial domain. In order to ensure that a complex parametrization in the spectral domain maps onto real-valued kernels, the authors impose a conjugate symmetry constraint on the spectral-domain weights, such that when the *Inverse Fourier Transform* is applied to them, it only yields real-valued kernels.

As pointed out in Reichert and Serre [22], the use of complex-valued neural networks [29, 30, 31, 32, 33] has been investigated long before the earliest deep learning breakthroughs [34, 35, 36]. Recently [22, 37, 17, 18, 19] have tried to bring more attention to the usefulness of deep complex neural networks by providing theoretical and mathematical motivation for using complex-valued deep networks. However, to the best of our knowledge, most of the recent works using complex valued networks have been applied on toy tasks, with the exception of some attempts. In fact, [38, 39, 40] have used complex representation in vision tasks. [19] have also performed a real-world speech task consisting of predicting the log magnitude of the future *Short Time Fourier Transform* frames. In Natural Language Processing, [41, 42] have used complex-valued embeddings. Much remains to be done to develop proper tools and a general framework for training deep neural networks with complex-valued parameters.

Given the compelling reasons for using complex-valued representations, the absence of such frameworks represents a gap in machine learning tooling, which we fill by providing a set of building blocks for deep complex-valued neural networks that enable them to achieve competitive results with their real-valued counterparts on real-world tasks.

2.3. Complex Building Blocks

In this section, we present the core of our work, laying down the mathematical framework for implementing complex-valued building blocks of a deep neural network.

2.3.1. Representation of Complex Numbers

We start by outlining the way in which complex numbers are represented in our framework. A complex number $z = a + ib$ has a real component a and an imaginary component b. We represent the real part a and the imaginary part b of a complex number as logically distinct real valued entities and simulate complex arithmetic using real-valued arithmetic internally. Consider a typical real-valued $2D$ convolution layer that has N feature maps such that N is divisible by 2; to represent these as complex numbers, we allocate the *first* $N/2$ feature maps to represent the real components and the remaining $N/2$ to represent the imaginary ones. Thus, for a four dimensional weight tensor W that links N_{in} input feature maps to N_{out} output feature maps and whose kernel size is $m \times m$ we would have a weight tensor of size $(N_{out} \times N_{in} \times m \times m)/2$ complex weights.

2.3.2. Complex Convolution

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 $\mathbf{A}$ and $\mathbf{B}$ are real matrices and $\mathbf{x}$ and $\mathbf{y}$ are real vectors since we are simulating complex arithmetic using real-valued entities. As the convolution operator is distributive, convolving the vector $\mathbf{h}$ by the filter $\mathbf{W}$ we obtain:

$$\mathbf{W} * \mathbf{h} = \ (\mathbf{A} * \mathbf{x} - \mathbf{B} * \mathbf{y}) + i\,(\mathbf{B} * \mathbf{x} + \mathbf{A} * \mathbf{y}). \tag{2.3.1}$$

As illustrated in Figure 2.1a, if we use matrix notation to represent real and imaginary parts of the convolution operation we have:

$$\begin{bmatrix} \Re(\mathbf{W} * \mathbf{h}) \\ \Im(\mathbf{W} * \mathbf{h}) \end{bmatrix} = \begin{bmatrix} \mathbf{A} & -\mathbf{B} \\ \mathbf{B} & \mathbf{A} \end{bmatrix} * \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}. \tag{2.3.2}$$

2.3.3. Complex Differentiability

In order to perform backpropagation in a complex-valued neural network, a sufficient condition is to have a cost function and activations that are *differentiable* with respect to the real and imaginary parts of each complex parameter in the network. See Section 2.7.3 in the Appendix for the complex chain rule.

By constraining activation functions to be *complex differentiable* or *holomorphic*, we restrict the use of possible activation functions for a complex valued neural networks (for further details about holomorphism please refer to Section 2.7.2 in the appendix). Hirose and Yoshida [16] show that it is unnecessarily restrictive to limit oneself only to holomorphic activation functions; Those functions that are differentiable with respect to the real part and the imaginary part of each parameter are also compatible with backpropagation. [17, 19, 18] have used non-holomorphic activation functions and optimized the network using regular, real-valued backpropagation to compute partial derivatives of the cost with respect to the real and imaginary parts.

Even though their use greatly restricts the set of potential activations, it is worth mentioning that holomorphic functions can be leveraged for computational efficiency purposes.

As pointed out in [43], using holomorphic functions allows one to share gradient values (because the activation satisfies the Cauchy-Riemann equations 2.7.1 and 2.7.2 in the appendix). So, instead of computing and backpropagating 4 different gradients, only 2 are required.

2.3.4. Complex-Valued Activations

2.3.4.1. *modReLU*

Numerous activation functions have been proposed in the literature in order to deal with complex-valued representations. Arjovsky et al. [17] have proposed modReLU, which is defined as follows:

$$\text{modReLU}(z) = \text{ReLU}(|z| + b)\, e^{i\theta_z} = \begin{cases} (|z| + b)\frac{z}{|z|} & \text{if } |z| + b \geq 0, \\ 0 & \text{otherwise,} \end{cases} \tag{2.3.3}$$

where $z \in \mathbb{C}$, θ_z is the phase of z, and $b \in \mathbb{R}$ is a learnable parameter. As $|z|$ is always positive, a bias b is introduced in order to create a *"dead zone"* of radius b around the origin 0 where the neuron is inactive, and outside of which it is active. The authors have used modReLU in the context of unitary RNNs. Their design of modReLU is motivated by the fact that applying separate ReLUs on both real and imaginary parts of a neuron performs poorly on toy tasks. The intuition behind the design of modReLU is to preserve the pre-activated phase θ_z, as altering it with an activation function severely impacts the complex-valued representation. modReLU does not satisfy the Cauchy-Riemann equations, and thus is not holomorphic. We have tested modReLU in deep feed-forward complex networks and the results are given in Table 2.1.

2.3.4.2. $\mathbb{C}$*ReLU and z*ReLU

We call Complex ReLU (or $\mathbb{C}$ReLU) the complex activation that applies separate ReLUs on both of the real and the imaginary part of a neuron, i.e:

$$\mathbb{C}\text{ReLU}(z) = \text{ReLU}(\Re(z)) + i\,\text{ReLU}(\Im(z)). \tag{2.3.4}$$

$\mathbb{C}$ReLU satisfies the Cauchy-Riemann equations when both the real and imaginary parts are at the same time either strictly positive or strictly negative. This means that $\mathbb{C}$ReLU satisfies the Cauchy-Riemann equations when $\theta_z \in]0,\, \pi/2[$ or $\theta_z \in]\pi,\, 3\pi/2[$. We have tested $\mathbb{C}$ReLU in deep feed-forward neural networks and the results are given in Table 2.1.

It is also worthwhile to mention the work done by Guberman [44], where a ReLU-based complex activation that satisfies the Cauchy-Riemann equations everywhere except for the set of points $\{\Re(z) > 0, \Im(z) = 0\} \cup \{\Re(z) = 0, \Im(z) > 0\}$ is used. The activation function has similarities to $\mathbb{C}$ReLU. We will refer to the activation in [44] as zReLU. It is defined as follows:

$$z\text{ReLU}(z) = \begin{cases} z & \text{if } \theta_z \in [0, \pi/2], \\ 0 & \text{otherwise,} \end{cases} \tag{2.3.5}$$

We have tested zReLU in deep feed-forward complex networks and the results are given in Table 2.1.

2.3.5. Complex Batch Normalization

Deep networks generally rely upon Batch Normalization [9] to accelerate learning. In some cases batch normalization is essential to optimize the model. The standard formulation of Batch Normalization applies only to real values. In this section, we propose a batch normalization formulation that can be applied for complex values.

To standardize an array of complex numbers to the standard normal complex distribution, it is not sufficient to translate and scale them such that their mean is 0 and their variance 1. This type of normalization does not ensure equal variance in both the real and imaginary components, and the resulting distribution is not guaranteed to be circular; it will be elliptical, potentially with high eccentricity.

We instead choose to treat this problem as one of whitening 2D vectors, which implies scaling the data by the square root of their variances along each of the two principal components. This can be done by multiplying the $\mathbf{0}$-centered data $(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])$ by the inverse square root of the 2×2 covariance matrix $\boldsymbol{V}$:

$$\tilde{\boldsymbol{x}} = (\boldsymbol{V})^{-\frac{1}{2}} (\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}]),$$

where the covariance matrix $\boldsymbol{V}$ is

$$\boldsymbol{V} = \begin{pmatrix} V_{rr} & V_{ri} \\ V_{ir} & V_{ii} \end{pmatrix} = \begin{pmatrix} \text{Cov}(\Re\{\boldsymbol{x}\}, \Re\{\boldsymbol{x}\}) & \text{Cov}(\Re\{\boldsymbol{x}\}, \Im\{\boldsymbol{x}\}) \\ \text{Cov}(\Im\{\boldsymbol{x}\}, \Re\{\boldsymbol{x}\}) & \text{Cov}(\Im\{\boldsymbol{x}\}, \Im\{\boldsymbol{x}\}) \end{pmatrix}.$$

The square root and inverse of 2×2 matrices has an inexpensive, analytical solution, and its existence is guaranteed by the positive (semi-)definiteness of $\boldsymbol{V}$. Positive definiteness of

$\boldsymbol{V}$ is ensured by the addition of $\epsilon \boldsymbol{I}$ to $\boldsymbol{V}$ (Tikhonov regularization). The mean subtraction and multiplication by the inverse square root of the variance ensures that $\tilde{\boldsymbol{x}}$ has standard complex distribution with mean $\mu = 0$, covariance $\Gamma = 1$ and pseudo-covariance (also called relation) $C = 0$. The mean, the covariance and the pseudo-covariance are given by:

$$\begin{aligned} \mu &= \mathbb{E}\left[\tilde{\boldsymbol{x}}\right] \\ \Gamma &= \mathbb{E}\left[\left(\tilde{\boldsymbol{x}} - \mu\right)\left(\tilde{\boldsymbol{x}} - \mu\right)^*\right] = V_{rr} + V_{ii} + i\left(V_{ir} - V_{ri}\right) \\ C &= \mathbb{E}\left[\left(\tilde{\boldsymbol{x}} - \mu\right)\left(\tilde{\boldsymbol{x}} - \mu\right)\right] = V_{rr} - V_{ii} + i\left(V_{ir} + V_{ri}\right). \end{aligned} \tag{2.3.6}$$

The normalization procedure allows one to decorrelate the imaginary and real parts of a unit. This has the advantage of avoiding co-adaptation between the two components, which reduces the risk of overfitting [45, 46].

Analogously to the real-valued batch normalization algorithm, we use two parameters, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$. The shift parameter $\boldsymbol{\beta}$ is a complex parameter with two learnable components (the real and imaginary means). The scaling parameter $\boldsymbol{\gamma}$ is a 2×2 positive semi-definite matrix with only three degrees of freedom, and thus only three learnable components. In much the same way that the matrix $(\boldsymbol{V})^{-\frac{1}{2}}$ normalized the variance of the input to 1 along both of its original principal components, so does $\boldsymbol{\gamma}$ scale the input along desired new principal components to achieve a desired variance. The scaling parameter $\boldsymbol{\gamma}$ is given by:

$$\boldsymbol{\gamma} = \begin{pmatrix} \gamma_{rr} & \gamma_{ri} \\ \gamma_{ir} & \gamma_{ii} \end{pmatrix}.$$

As the normalized input $\tilde{\boldsymbol{x}}$ has real and imaginary variance 1, we initialize both γ_{rr} and γ_{ii} to $1/\sqrt{2}$ in order to obtain a modulus of 1 for the variance of the normalized value. γ_{ri}, $\Re\{\boldsymbol{\beta}\}$ and $\Im\{\boldsymbol{\beta}\}$ are initialized to 0. The complex batch normalization is defined as:

$$\mathrm{BN}\left(\tilde{\boldsymbol{x}}\right) = \boldsymbol{\gamma}\,\tilde{\boldsymbol{x}} + \boldsymbol{\beta}. \tag{2.3.7}$$

We use running averages with momentum to maintain an estimate of the complex batch normalization statistics during training and testing. The moving averages of V_{ri} and $\boldsymbol{\beta}$ are initialized to 0. The moving averages of V_{rr} and V_{ii} are initialized to $1/\sqrt{2}$. The momentum for the moving averages is set to 0.9.

2.3.6. Complex Weight Initialization

In a general case, particularly when batch normalization is not performed, proper initialization is critical in reducing the risks of vanishing or exploding gradients. To do this, we follow the same steps as in Glorot and Bengio [47] and He, Zhang, Ren, and Sun [48] to derive the variance of the complex weight parameters.

A complex weight has a polar form as well as a rectangular form

$$W = |W|e^{i\theta} = \Re\{W\} + i\,\Im\{W\}, \tag{2.3.8}$$

where θ and $|W|$ are respectively the argument (phase) and magnitude of W.
Variance is the difference between the *expectation of the squared magnitude* and the *square of the expectation*:

$$\mathrm{Var}(W) = \mathbb{E}\left[WW^*\right] - \left(\mathbb{E}\left[W\right]\right)^2 = \mathbb{E}\left[|W|^2\right] - \left(\mathbb{E}\left[W\right]\right)^2,$$

which reduces, in the case of W symmetrically distributed around 0, to $\mathbb{E}\left[|W|^2\right]$. Although we have not yet derived the value of $\mathrm{Var}(W) = \mathbb{E}\left[|W|^2\right]$, we do know a related quantity, $\mathrm{Var}(|W|)$, because the magnitude of complex normal values, $|W|$, follows the Rayleigh distribution (Chi-distributed with two degrees of freedom (DOFs)). This quantity is

$$\mathrm{Var}(|W|) = \mathbb{E}\left[|W||W|^*\right] - \left(\mathbb{E}\left[|W|\right]\right)^2 = \mathbb{E}\left[|W|^2\right] - \left(\mathbb{E}\left[|W|\right]\right)^2. \tag{2.3.9}$$

Putting them together:

$$\mathrm{Var}(|W|) = \mathrm{Var}(W) - \left(\mathbb{E}\left[|W|\right]\right)^2, \text{ and } \mathrm{Var}(W) = \mathrm{Var}(|W|) + \left(\mathbb{E}\left[|W|\right]\right)^2.$$

We now have a formulation for the variance of W in terms of the variance and expectation of its magnitude, both properties analytically computable from the Rayleigh distribution's single parameter, σ, indicating the *mode*. These are:

$$\mathbb{E}\left[|W|\right] = \sigma\sqrt{\frac{\pi}{2}}, \quad \mathrm{Var}(|W|) = \frac{4-\pi}{2}\sigma^2.$$

The variance of W can thus be expressed in terms of its generating Rayleigh distribution's single parameter, σ, thus:

$$\mathrm{Var}(W) = \frac{4-\pi}{2}\sigma^2 + \left(\sigma\sqrt{\frac{\pi}{2}}\right)^2 = 2\sigma^2. \tag{2.3.10}$$

If we want to use the Glorot and Bengio [47] initialization, which ensures that the variances of the input and output are the same, then we would have $\mathrm{Var}(W) = 2/(n_{in} + n_{out})$, where n_{in} and n_{out} are the number of input and output units respectively. In that case,

$\sigma = 1/\sqrt{n_{in} + n_{out}}$. If we want to use the He et al. [48] initialization, which is specialized for ReLU-based networks, then $\text{Var}(W) = 2/n_{in}$, in which case $\sigma = 1/\sqrt{n_{in}}$.

The magnitude of the complex parameter W is then drawn from the Rayleigh distribution with the appropriate mode σ, while the phase is drawn from the uniform distribution between $-\pi$ and π. By multiplying the real magnitude W by the complex phasor $e^{i\theta}$ (equation 2.3.8), we combine them into an appropriate initialization value for the complex parameter.

In all the experiments that we report, we use variants of this initialization that additionally leverage the independence property of unitary matrices. As noted by Cogswell et al. [45], Srivastava et al. [46], and Tompson, Goroshin, Jain, LeCun, and Bregler [49], learning decorrelated features is beneficial, as it yields better generalization and faster learning. This motivates initializing the weights with appropriately-scaled (semi-)unitary matrices (a matrix U not necessarily square s.t. either $U^H U = I$ or $UU^H = I$). The scaling factor required is $\sqrt{\text{He}_{\text{var}}/\text{Var}(W)}$ or $\sqrt{\text{Glorot}_{\text{var}}/\text{Var}(W)}$ where $\text{Glorot}_{\text{var}}$ and He_{var} are respectively equal to $2/(n_{in} + n_{out})$ and $2/n_{in}$. We thereby achieve both our aim to make kernels maximally independent of each other and appropriately scaled for complex-valued models. For real-valued models, the analogous initialization uses scaled (semi-)orthogonal matrices.

2.3.7. Complex Convolutional Residual Network

A deep convolutional residual network of the nature presented in [12, 13] consists in three *stages*, each comprising several *residual blocks*. Within a stage, feature maps maintain the same shape. At the end of a stage, the feature maps are downsampled by a factor of 2 and the number of convolution filters is doubled. The sizes of the convolution kernels are always set to 3×3. The stage's *residual blocks* contain two convolution layers each. The contents of one such residual block in the real and complex settings are illustrated in the appendix Figure 2.1b.

In the complex-valued setting, the majority of the architecture remains identical to the one presented in [13], with only a few subtle differences. First, since all datasets that we work with have real-valued inputs, we present a way to learn their imaginary components to let the rest of the network operate in the complex plane. We learn the initial imaginary component of our input by performing the operations present within a single real-valued

residual block

$$BN \rightarrow ReLU \rightarrow Conv \rightarrow BN \rightarrow ReLU \rightarrow Conv\ .$$

Using this residual block yielded better results empirically than assuming that the input image has null imaginary part. The parameters of this real-valued residual block are trained by backpropagation of errors from the task-specific loss function. Second, we perform a $Conv \rightarrow BN \rightarrow Activation$ operation on the obtained complex input before feeding it to the first stage of the residual network proper. We also perform the same operation on the real-valued network's input, instead of $Conv \rightarrow Maxpooling$ as in He et al. [13]. Third, we subtly alter the way in which we perform a projection at the end of a stage in our network. We concatenate the output of the last residual block with the output of a 1×1 convolution applied on it with the same number of filters used throughout the stage and downsample by a factor of 2. In contrast, He et al. [13] perform a 1×1 convolution with twice the number of feature filters in the current stage to both downsample the feature maps spatially and double them in number.

2.4. Experimental Results

In this section, we present empirical results from using our model to perform image, music classification and spectrum prediction. We present our model's architecture followed by the results we obtained on natural image classification datasets CIFAR-10, CIFAR-100, and SVHN, as well as the results on automatic music transcription on the MusicNet benchmark and speech spectrum prediction on TIMIT.

2.4.1. Image Recognition

We adopt an architecture inspired by [13]. The latter will also serve as a baseline to compare against. We train comparable real-valued Neural Networks using the standard ReLU activation function. We have tested our complex models with the $\mathbb{C}$ReLU, zReLU and modRelu activation functions. We use a cross entropy loss for both real and complex models. A global average pooling layer followed by a single fully connected layer with a softmax function is used to classify the input as belonging to one of 10 classes in the CIFAR-10 and SVHN datasets and 100 classes for CIFAR-100.

We consider architectures that trade off model depth (number of residual blocks per stage) and width (number of convolutional filters in each layer) given a fixed parameter budget. Specifically, we build three different models - wide and shallow (WS), deep and narrow (DN) and in-between (IB). In a model that has roughly 1.7 million parameters, our WS architecture for a complex network starts with 12 complex filters (24 real filters) per convolution layer in the initial stage and 16 residual blocks per stage. The DN architecture starts with 10 complex filters and 23 blocks per stage while the IB variant starts with 11 complex filters and 19 blocks per stage. The real-valued counterpart has also 1.7 million parameters. Its WS architecture starts with 18 real filters per convolutional layer and 14 blocks per stage. The DN architecture starts with 14 real filters and 23 blocks per stage and the IB architecture starts with 16 real filters and 18 blocks per stage.

All models (real and complex) were trained using the backpropagation algorithm with Stochastic Gradient Descent with Nesterov momentum [50] set at 0.9. We also clip the norm of our gradients to 1. We tweaked the learning rate schedule used in [13] in both the real and complex residual networks to extract small performance improvements in both. We start our learning rate at 0.01 for the first 10 epochs to warm up the training and then set it at 0.1 from epoch 10-100 and then anneal the learning rates by a factor of 10 at epochs 120 and 150. We end the training at epoch 200.

Table 2.1 presents our results for image classification on CIFAR-10 and CIFAR-100. In addition, we consider a reduced version of the Street View House Numbers (SVHN) dataset, which we call SVHN*. For computational reasons, we use only the required 73,257 "training" images of Street View House Numbers (SVHN), and not the "extra" dataset. We still test on all 26,032 "test" images. For all the tasks and for both the real- and complex-valued models, the WS architecture has yielded the best performances. This is in agreement with Zagoruyko and Komodakis [3], who observed that wider and shallower residual networks perform better than their deeper and narrower counterpart. On CIFAR-10 and SVHN*, the real-valued representation performs slightly better than its complex counterpart. On CIFAR-100, the complex representation outperforms the real one. In general, the obtained results for both representation are quite comparable. To understand the effect of using either real or complex representation for a given task, we designed hybrid models that combine both. Table 2.2 contains the results for hybrid models. We can observe in Table 2.2 that in cases

Table 2.1. Classification error on CIFAR-10, CIFAR-100 and SVHN* using different complex activations functions (zReLU, modReLU and $\mathbb{C}$ReLU). WS, DN and IB stand for the wide and shallow, deep and narrow and in-between models respectively. The prefixes R & C refer to the real and complex valued networks respectively. Performance differences between the real network and the complex network using $\mathbb{C}$ReLU are reported between their respective best models. All models are constructed to have roughly 1.7M parameters except the modReLU models which have roughly 2.5M parameters. modReLU and zReLU were largely outperformed by $\mathbb{C}$ReLU in the reported experiments. Due to limited resources, we haven't performed all possible experiments as the conducted ones are already conclusive. A "-" is filled in front of an unperformed experiment.

Arch	CIFAR-10			CIFAR-100			SVHN*		
	zReLU	modReLU	$\mathbb{C}$ReLU	zReLU	modReLU	$\mathbb{C}$ReLU	zReLU	modReLU	$\mathbb{C}$ReLU
CWS	11.71	23.42	6.17	-	50.38	**26.36**	80.41	7.43	3.70
CDN	9.50	22.49	6.73	-	50.64	28.22	80.41	-	3.72
CIB	11.36	23.63	5.59	-	48.10	28.64	4.98	-	3.62
	ReLU			ReLU			ReLU		
RWS	**5.42**			27.22			**3.42**		
RDN	6.29			27.84			3.52		
RIB	6.07			27.71			4.30		
DIFF	-0.17			+0.86			-0.20		

where complex representation outperformed the real one (wide and shallow on CIFAR-100), combining a real-valued convolutional filter with a complex batch normalization improves the accuracy of the real-valued model. However, the complex-valued model still outperforms the hybrid. In cases where real-valued models outperformed the complex ones (wide and shallow on CIFAR-10 and SVHN*), replacing a complex batch normalization with a regular batch normalization increased the accuracy of the complex model. Despite that replacement, the real-valued model performs better in terms of accuracy for such tasks. In general, these experiments show that the difference in efficiency between the real and complex models varies by dataset, task and architecture.

Table 2.2. Classification error on CIFAR-10, CIFAR-100 and SVHN* using different normalization strategies. NCBN, CBN and BN stand for a Naive variant of the complex batch-normalization, complex batch-normalization and regular batch normalization respectively. (R) & (C) refer to the use of the real- and complex-valued convolution respectively. The complex models use ℂReLU as activation. All models are constructed to have roughly 1.7M parameters. 5 out of 6 experiments using the naive variant of the complex batch normalization failed with the apparition of NaNs during training. As these experiments are already conclusive and due to limited resources, we haven't conducted other experiments for the NCBN model. A "-" is filled in front of an unperformed experiment.

Arch	CIFAR-10			CIFAR-100			SVHN*		
	NCBN(C)	CBN(R)	BN(C)	NCBN(C)	CBN(R)	BN(C)	NCBN(C)	CBN(R)	BN(C)
WS	-	**5.47**	6.32	27.29	**26.63**	27.89	NAN	3.80	**3.52**
DN	-	5.89	6.71	NAN	27.13	28.83	NAN	3.54	3.58
IB	-	5.66	6.83	NAN	26.99	29.89	NAN	3.74	3.56

Ablation studies were performed in order to investigate the importance of the 2D whitening operation that occurs in the complex batch normalization. We replaced the complex batch normalization layers with a naive variant (NCBN) which, instead of left multiplying the centered unit by the inverse square root of its covariance matrix, just divides it by its complex variance. Here, this naive variant of CBN is Mimicking the regular BN by not taking into account correlation between the elements in the complex unit. The Naive variant of the Complex Batch Normalization performed very poorly; In 5 out of 6 experiments, training failed with the appearance of NaNs (See Section 2.7.6 for the explanation). By way of contrast, all 6 complex-valued Batch Normalization experiments converged. Results are given in Table 2.2.

Another ablation study was undertaken to compare ℂReLU, modReLU and zRELU. Again the differences were stark: All ℂReLU experiments converged and outperformed both modReLU and zReLU, both which variously failed to converge or fared substantially worse. We think that modRelu didn't perform as well as ℂReLU due to the fact that consecutive layers in a feed-forward net do not represent time-sequential patterns, and so, they might

need to drop some phase information. Results are reported in Table 2.1. More discussion about phase information encoding is presented in section 2.7.7.

2.4.2. Automatic Music Transcription

In this section we present results for the automatic music transcription (AMT) task. The nature of an audio signal allows one to exploit complex operations as presented earlier in the paper. The experiments were performed on the MusicNet dataset [51]. For computational efficiency we resampled the original input from the original 44.1kHz to the reduced 11kHz using the algorithm described in [52]. This sampling rate is sufficient to recognize frequencies present in the dataset while reducing the computational cost dramatically. We modeled each of the 84 notes that are present in the dataset with independent sigmoids (due to the fact that notes can fire simultaneously). We initialized the bias of the last layer to the value of -5 to reflect the distribution of silent/non-silent notes. As in the baseline, we performed experiments on the raw signal and the frequency spectrum. For complex experiments with the raw signal, we considered its imaginary part equal to zero. When using the spectrum input we used its complex representation (instead of only the magnitudes, as usual for AMT) for both real and complex models. For the real model, we considered the real and imaginary components of the spectrum as separate channels.

The model we used for raw signals is a shallow convolutional network similar to the model used in the baseline, with the size reduced by a factor of 4 (corresponding to the reduction in sampling rate). The convolutional filter size was 512 samples (spanning about 12ms) with a stride of 16. The model for the spectral input is similar to the VGG model [53]. The first layer has filter with size of 7 and is followed by 5 convolutional layers with filters of size 3. The final convolution block is followed by a fully connected layer with 2048 units. The latter is followed, in its turn, by another fully connected layer with 84 sigmoidal units. In all of our experiments we use an input window of 4096 samples or its corresponding FFT (which corresponds to the 16,384 window used in the baseline) and predicted notes in the center of the window. All networks were optimized with the Adam optimizer [54]. We start our learning rate at 10^{-3} for the first 10 epochs and then anneal it by a factor of 10 at epochs 100, 120 and 150. We end the training at epoch 200. For the real-valued models, we have used ReLU as the activation. For complex-valued models we have used $\mathbb{C}$ReLU as the activation.

Table 2.3. MusicNet experiments. *FS* is the sampling rate. *Params* is the total number of parameters. We report the average precision (AP) metric that is the area under the precision-recall curve.

Architecture	FS	Params	AP %
Shallow, Real	11kHz		66.1
Shallow, Complex	11kHz		66.0
Shallow, Thickstun et al. [51]	44.1kHz	-	67.8
Deep, Real	11kHz	10.0M	69.6
Deep, Complex	11kHz	8.8M	**72.9**

The complex network was initialized using the unitary He initialization scheme described in Section 2.3.6. For the real-valued network, we have used the analogous real-valued orthogonal He initialization. The complex batch normalization of Section 2.3.5 was used. Following Thickstun et al. [51] we used recordings with ids '2303', '2382' and '1819' as the test subset, and additionally we created a validation subset by randomly choosing from the training set recording ids '2131', '2384', '1792', '2514', '2567' and '1876'. The validation subset was used for model selection and early stopping. The remaining 321 files were used for training. The results are summarized in Table 2.3. We achieve a performance comparable to the baseline with the shallow convolutional network. Our VGG-based deep real-valued model reaches 69.6% average precision on the downsampled data. With significantly fewer parameters than its real counterpart, the VGG-based deep complex model achieves 72.9% average precision, which is state of the art to the best of our knowledge. See Figures 2.2 and 2.3 in the appendix for precision-recall curves and a sample of the output of the model.

2.4.3. Speech Spectrum Prediction

We apply both a real Convolutional LSTM [55] and a complex Convolutional LSTM on speech spectrum prediction task (See section 2.7.5 in the appendix for the details of the real and complex Convolutional LSTMs). In this task, the model predicts the magnitude spectrum. It implicitly infers the real and imaginary components of the spectrum at time $t+1$, given all the spectrum (imaginary part and real components) up to time t, which is slightly

Table 2.4. Speech Spectrum Prediction on TIMIT test set. *CConv-LSTM* denotes the Complex Convolutional LSTM.

Model	#params	MSE(validation)	MSE(test)
LSTM [19]	≈ 135k	16.59	16.98
Full-capacity uRNN [19]	≈ 135k	14.56	14.66
Conv-LSTM (our baseline)	≈ 88k	11.10	12.18
CConv-LSTM (ours)	≈ 88k	**10.78**	**11.90**

different from [19]. The real and imaginary components are considered as separate channels in both model. We evaluate the model with mean-square-error (MSE) on log-magnitude to compare with Wisdom et al. [19]. The experiments are conducted on a downsampled (8kHz) version of the TIMIT dataset. Following the steps in [19], raw audio waves are transformed into the frequency domain via the short-time Fourier transform (STFT) with a Hann window of 256 samples and a window hop of 128 samples (50% overlap). We use a training set with 3690 utterances, a validation set with 400 utterances and the standard test set with 192 utterance.

To match the number of parameters for both model, the Convolutional LSTM has 84 real-valued feature maps while the complex model has 60 complex-valued feature maps (120 real feature maps in total). The Adam optimizer [54] with a fixed learning rate of 1e-4 is used in both experiments. We initialize the complex model with the unitary Glorot initialization and the real model with the orthogonal Glorot initialization. The result is shown in Table 2.4 and the learning curve is shown in Figure 2.4. Our baseline model has achieved the state of the art and the complex convolutional LSTM model performs better than the baseline in terms of MSE and speed of convergence.

2.5. CONCLUSIONS

We have presented key building blocks required to train complex valued neural networks, such as complex batch normalization and complex weight initialization. We have also explored a wide variety of complex convolutional network architectures, including some yielding competitive results for image classification and state of the art results for a music transcription task and speech spectrum prediction. We hope that our work will stimulate further

investigation of complex valued networks for deep learning models and their application to more challenging tasks such as generative models for audio and images.

2.6. ACKNOWLEDGEMENTS

We are grateful to Roderick Murray-Smith, Jörn-Henrik Jacobsen, Jesse Engel and all the students at MILA, especially Jason Jo, Anna Huang and Akram Erraqabi, for helpful feedback and discussions. We also thank the developers of Theano [56] and Keras [57]. We are grateful to Samsung and the Fonds de Recherche du Québec – Nature et Technologie for their financial support. We would also like to acknowledge NVIDIA for donating a DGX-1 computer used in this work.

2.7. APPENDIX

In practice, the complex convolution operation is implemented as illustrated in Fig.2.1a where M_I, M_R refer to imaginary and real feature maps and K_I and K_R refer to imaginary and real kernels. $M_I K_I$ refers to result of a real-valued convolution between the imaginary kernels K_I and the imaginary feature maps M_I.

2.7.1. MusicNet illustrations

2.7.2. Holomorphism and Cauchy–Riemann Equations

Holomorphism, also called *analyticity*, ensures that a complex-valued function is *complex differentiable* in the neighborhood of every point in its domain. This means that the derivative, $f'(z_0) \equiv \lim_{\Delta z \to 0}[\frac{(f(z_0)+\Delta z)-f(z_0)}{\Delta z}]$ of f, exists at every point z_0 in the domain of f where f is a complex-valued function of a complex variable $z = x + i\,y$ such that $f(z) = u(x,y) + i\,v(x,y)$. u and v are real-valued functions. One possible way of expressing Δz is to have $\Delta z = \Delta x + i\,\Delta y$. Δz can approach 0 from multiple directions (along the real axis, imaginary axis or in-between). However, in order to be complex differentiable, $f'(z_0)$ must be the same complex quantity regardless of direction of approach. When Δz approaches 0 along the real axis, $f'(z_0)$ could be written as:

$$\begin{aligned} f'(z_0) &\equiv \lim_{\Delta z \to 0}\left[\frac{(f(z_0)+\Delta z) - f(z_0)}{\Delta z}\right] \\ &= \lim_{\Delta x \to 0}\lim_{\Delta y \to 0}\left[\frac{\Delta u(x_0,y_0) + i\,\Delta v(x_0,y_0)}{\Delta x + i\,\Delta y}\right] \\ &= \lim_{\Delta x \to 0}\left[\frac{\Delta u(x_0,y_0) + i\,\Delta v(x_0,y_0)}{\Delta x + i\,0}\right]. \end{aligned} \tag{2.7.1}$$

When Δz approaches 0 along the imaginary axis, $f'(z_0)$ could be written as:

$$\begin{aligned} &= \lim_{\Delta y \to 0}\lim_{\Delta x \to 0}\left[\frac{\Delta u(x_0,y_0) + i\,\Delta v(x_0,y_0)}{\Delta x + i\,\Delta y}\right] \\ &= \lim_{\Delta y \to 0}\left[\frac{\Delta u(x_0,y_0) + i\,\Delta v(x_0,y_0)}{0 + i\,\Delta y}\right] \end{aligned} \tag{2.7.2}$$

Satisfying equations 2.7.1 and 2.7.2 is equivalent of having $\frac{\partial f}{\partial z} = \frac{\partial u}{\partial x} + i\,\frac{\partial v}{\partial x} = -i\,\frac{\partial u}{\partial y} + \frac{\partial v}{\partial y}$. So, in order to be complex differentiable, f should satisfy $\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}$ and $\frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}$. These are called the Cauchy–Riemann equations and they give a necessary condition for f to be complex differentiable or "*holomorphic*". Given that u and v have continuous first partial derivatives, the Cauchy-Riemann equations become a sufficient condition for f to be holomorphic.

2.7.3. The Generalized Complex Chain Rule for a Real-Valued Loss Function

If L is a real-valued loss function and z is a complex variable such that $z = x + i\,y$ where $x, y \in \mathbb{R}$, then:

$$\nabla_L(z) = \frac{\partial L}{\partial z} = \frac{\partial L}{\partial x} + i\frac{\partial L}{\partial y} = \frac{\partial L}{\partial \Re(z)} + i\frac{\partial L}{\partial \Im(z)} = \Re(\nabla_L(z)) + i\Im\left(\nabla_L(z)\right). \tag{2.7.3}$$

Now if we have another complex variable $t = r + i\,s$ where z could be expressed in terms of t and $r, s \in \mathbb{R}$, we would then have:

$$\begin{aligned}
\nabla_L(t) = \frac{\partial L}{\partial t} &= \frac{\partial L}{\partial r} + i\frac{\partial L}{\partial s} \\
&= \frac{\partial L}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial r} + i\left(\frac{\partial L}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial s}\right) \\
&= \frac{\partial L}{\partial x}\left(\frac{\partial x}{\partial r} + i\frac{\partial x}{\partial s}\right) + \frac{\partial L}{\partial y}\left(\frac{\partial y}{\partial r} + i\frac{\partial y}{\partial s}\right) \\
&= \frac{\partial L}{\partial \Re(z)}\left(\frac{\partial x}{\partial r} + i\frac{\partial x}{\partial s}\right) + \frac{\partial L}{\partial \Im(z)}\left(\frac{\partial y}{\partial r} + i\frac{\partial y}{\partial s}\right) \\
&= \Re(\nabla_L(z))\left(\frac{\partial x}{\partial r} + i\frac{\partial x}{\partial s}\right) + \Im(\nabla_L(z))\left(\frac{\partial y}{\partial r} + i\frac{\partial y}{\partial s}\right).
\end{aligned} \tag{2.7.4}$$

2.7.4. Computational Complexity and FLOPS

In terms of computational complexity, the convolutional operation and the complex batchnorm are of the same order as their real counterparts. However, as a complex multiplication is 4 times more expensive than its real counterpart, all complex convolutions are 4 times more expensive as well.

Additionally, the complex BatchNorm is not implemented in cuDNN and therefore had to be simulated with a sizeable sequence of elementwise operations. This leads to a ballooning of the number of nodes in the compute graph and to inefficiencies due to lack of effective operation fusion. A dedicated cuDNN kernel will, however, reduce the cost to little more than that of the real-valued BatchNorm.

Ignoring elementwise operations, which constitute a negligible fraction of the floating-point operations in the neural network, we find that for all architectures in 2.1 and for all of CIFAR10, CIFAR100 or SVHN, the inference cost in real FLOPS per example is roughly identical. It is ~ 265 MFLOPS for the $\mathbb{R}$-valued variant and ~ 1030 MFLOPS for the $\mathbb{C}$-valued variant of the architecture, approximately quadruple.

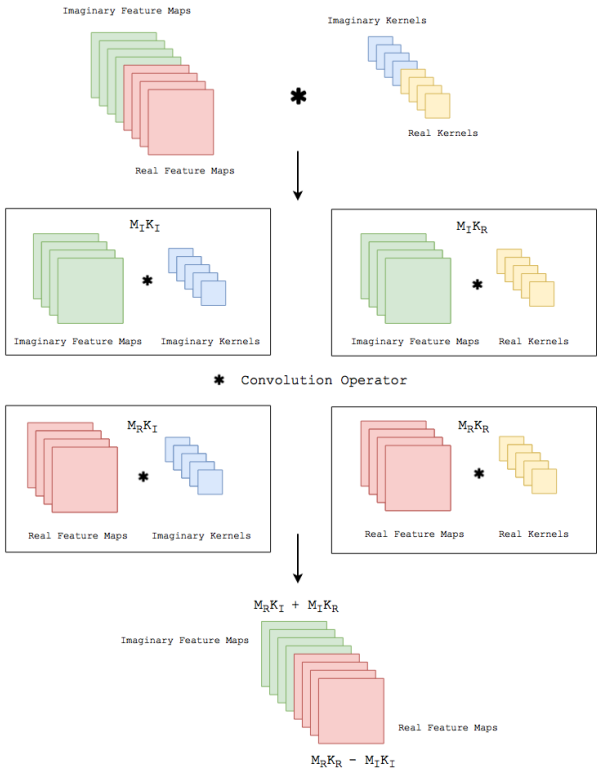
2.7.5. Convolutional LSTM

A Convolutional LSTM is similar to a fully connected LSTM. The only difference is that, instead of using matrix multiplications to perform computation, we use convolutional operations. The computation in a real-valued Convolutional LSTM is defined as follows:

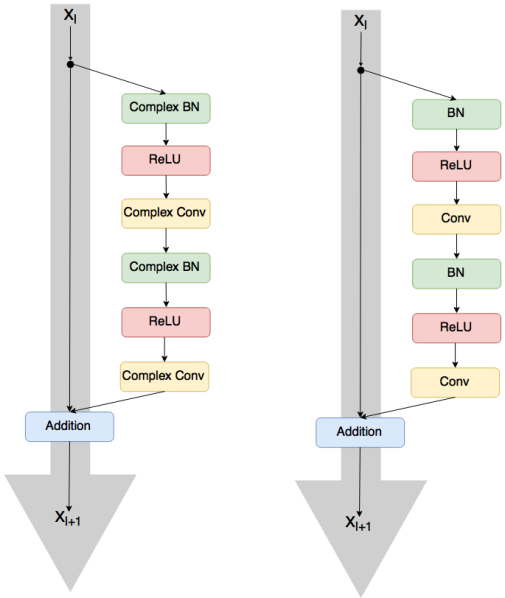
$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_{xi} * \mathbf{x}_t + \mathbf{W}_{hi} * \mathbf{W}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_{xf} * \mathbf{x}_t + \mathbf{W}_{hf} * \mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{c}_t &= \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tanh(\mathbf{W}_{xc} * \mathbf{x}_t + \mathbf{W}_{hc} * \mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_{xo} * \mathbf{x}_t + \mathbf{W}_{ho} * \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \circ \tanh(\mathbf{c}_t)
\end{aligned}
\tag{2.7.5}
$$

Where σ denotes the sigmoidal activation function, $\circ$ the elementwise multiplication and $*$ the real-valued convolution. $\mathbf{i}_t$, $\mathbf{f}_t$, $\mathbf{o}_t$ represent the vector notation of the input, forget and output gates respectively. $\mathbf{c}_t$ and $\mathbf{h}_t$ represent the vector notation of the cell and hidden states respectively. the gates and states in a ConvLSTM are tensors whose last two dimensions are spatial dimensions. For each of the gates, $\mathbf{W}_{xgate}$ and $\mathbf{W}_{hgate}$ are respectively the input and hidden kernels.

For the Complex Convolutional LSTM, we just replace the real-valued convolutional operation by its complex counterpart. We maintain the real-valued elementwise multiplication. The sigmoid and tanh are both performed separately on the real and the imaginary parts.



(a) An illustration of the complex convolution operator.



(b) A complex convolutional residual network (left) and an equivalent real-valued residual network (right).

Figure 2.1. Complex convolution and residual network implementation details.

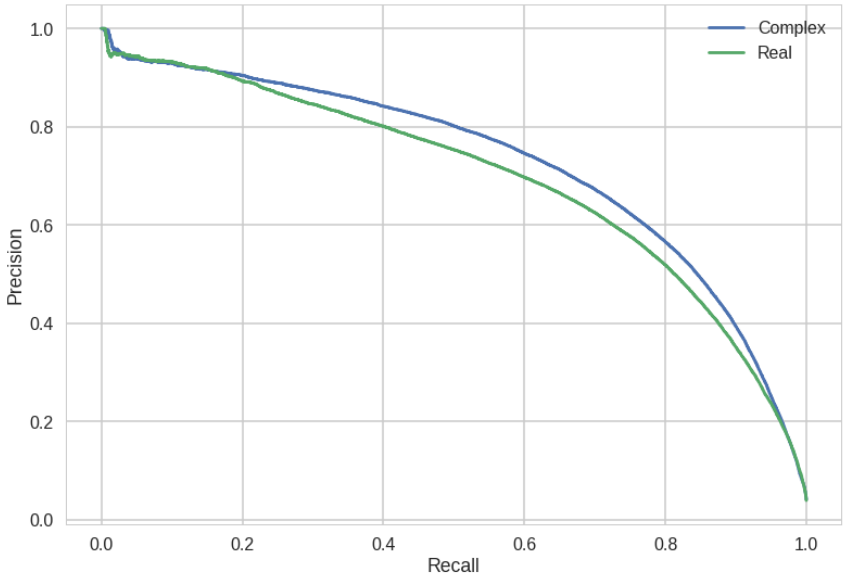


Figure 2.2. Precision-recall curve

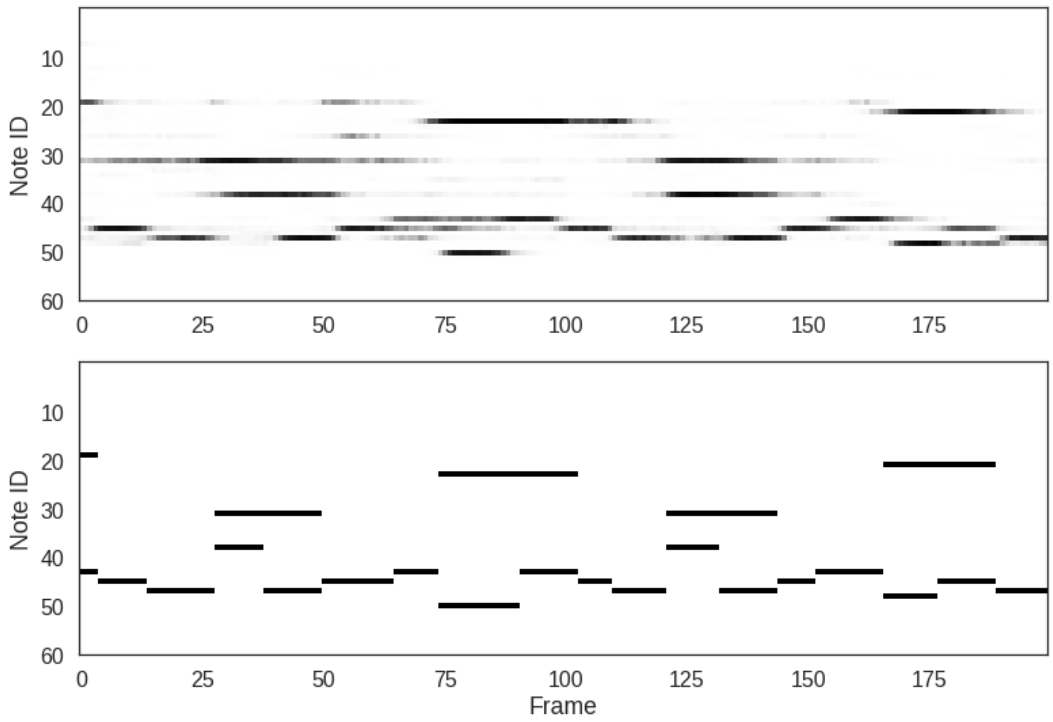


Figure 2.3. Predictions (Top) vs. ground truth (Bottom) for a music segment from the test set.

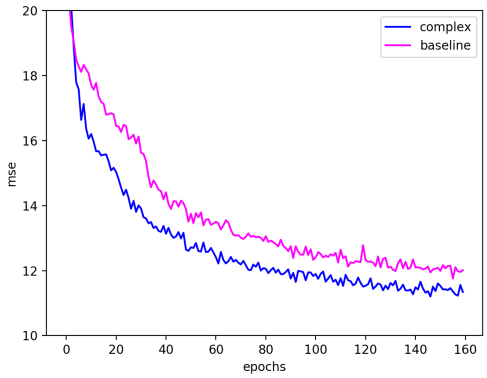


Figure 2.4. Learning curve for speech spectrum prediction from dev set.

2.7.6. Complex Standardization and Internal Covariate Shift

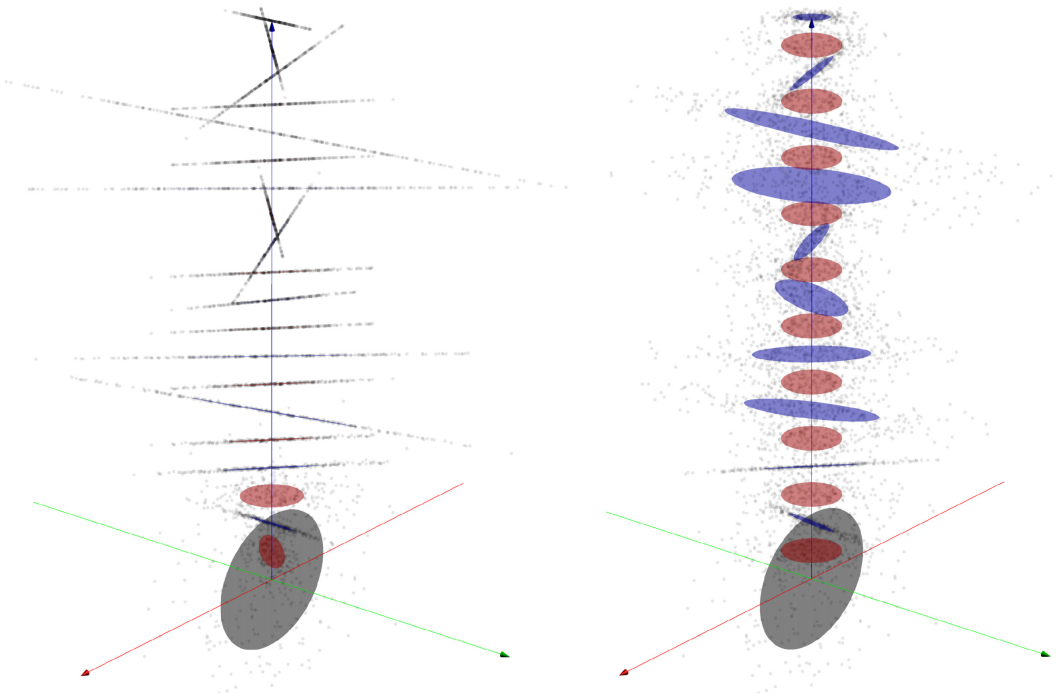
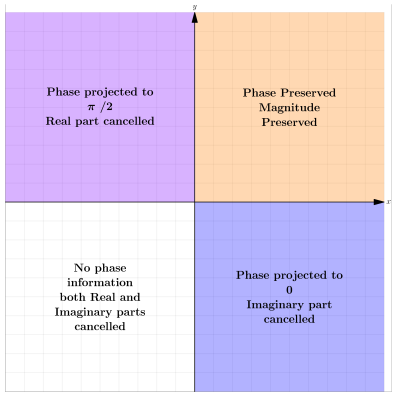
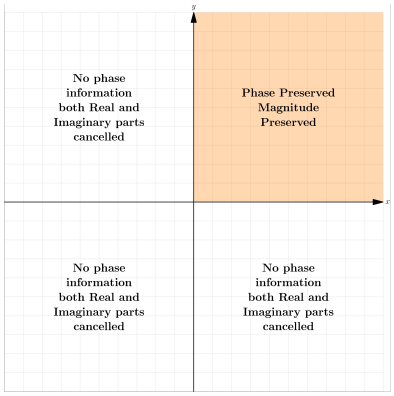


Figure 2.5. Depiction of Complex Standardization in Deep Complex Networks. *At left*, Naive Complex Standardization (division by complex standard deviation); *At right*, Complex Standardization (left-multiplication by inverse square root of covariance matrix between $\Re$ and $\Im$). The 250 input complex scalars are at the bottom, with $\Re(v)$ plotted on x (red axis) and $\Im(v)$ plotted on y (green axis). Deeper representations correspond to greater z (blue axis). *The gray ellipse* encloses the input scalars within 1 standard deviation of the mean. *Red ellipses* enclose all scalars within 1 standard deviation of the mean after "standardization". *Blue ellipses* enclose all scalars within 1 standard deviation of the mean after left-multiplying all the scalars by a random 2×2 linear transformation matrix. With the naive standardization, the distribution becomes progressively more elliptical with every layer, eventually collapsing to a line. This ill-conditioning manifests itself as NaNs in the forward pass or backward pass. With the complex standardization, the points' distribution is always successfully re-circularized.

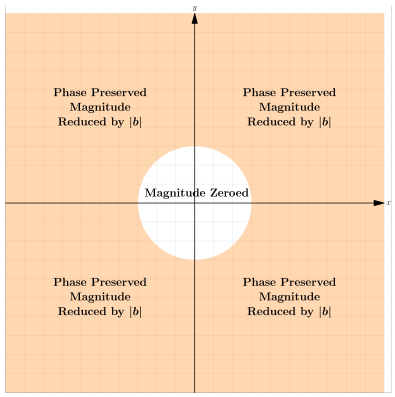
2.7.7. Phase Information Encoding



(a) ℂReLU



(b) zReLU



(c) modReLU

Figure 2.6. Phase information encoding for each of the activation functions tested for the Deep Complex Network. The x-axis represents the real part and the y-axis represents the imaginary part; The bottom figure corresponds to the case where $b < 0$ for modReLU. The radius of the white circle is equal to $|b|$. In case where $b \geq 0$, the whole complex plane would be preserving both phase and magnitude information and the whole plane would have been colored with orange. Different colors represents different encoding of the complex information in the plane. We can see the for both zReLU and modReLU, the complex representation is discriminated into two regions, i.e., the one that preserves the whole complex information (colored in orange) and the one that cancels it (colored in white). However, ℂReLU discriminates the complex information into 4 regions where in two of which, phase information is projected and not canceled. This allows ℂReLU to discriminate information more easily with respect to phase information than the other activation functions. For both zReLU and modReLU, we can see that phase information may be preserved explicitly through a number of layers when these activation functions are operating in their linear regime, prior to a layer further up in a network where the phase of an input lies in a zero region. ℂReLU has more flexibility manipulating phase as it can either set it to zero or $\pi/2$, or even delete the phase information (when both real and imaginary parts are canceled) at a given level of depth in the network.

Conclusion

In the introduction, we remarked on the utility of complex numbers in digital signal processing, and raised an important question: Why is it that so few neural networks use complex numbers for signal processing, if they are such a natural medium for expressing the frequency content of a signal?

We then demonstrated that natively complex-valued deep neural networks can match or outperform real-valued networks on image and audio tasks, which indicates they should be considered seriously in the future for general-purpose, deep-neural-network-based signal-processing tasks.

In the process of doing so, we have identified several traps that may have led to the failure of past efforts to use complex numbers. Specifically, due to the properties of complex numbers, deep neural networks need adapted weights initialization procedures, a significantly modified batch normalization layer, and careful attention to the choice of non-linearity.

Using our proposed complex weight initializations, deep neural networks using complex numbers avoid exploding or vanishing gradients, just as the Glorot and He initializations once did for real-valued neural networks.

We empirically demonstrate that our Complex Batch Normalization layer solves a pathology of complex numbers, namely the tendency for their distribution to collapse down to a highly elliptical distribution, thus avoiding among others numerical problems (like `NaN`-raising) that otherwise plague the training of complex-valued networks.

And we show that complex numbers are especially sensitive to the choice of non-linearity. We find that the non-linearity we term $\mathbb{C}$ReLU is an excellent choice for deep feedforward neural networks and find, *contra* existing literature, that modReLU and zReLU are poor ones.

Deep Complex Networks thus open the door to a modest and incremental, but still significant, gain in many sub-fields of artificial intelligence concerned with digital signal processing. It promises even more significant gains where the input data is naturally complex-valued, such as quantum particle physics and wireless signal/radar return processing.

Bibliography

[1] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[3] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[4] Po-Sen Huang, Minje Kim, Mark Hasegawa-Johnson, and Paris Smaragdis. Deep learning for monaural speech separation. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1562–1566. IEEE, 2014.

[5] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[6] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016.

[7] Pavlo Molchanov, Shalini Gupta, Kihwan Kim, and Jan Kautz. Hand gesture recognition with 3d convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–7, 2015.

[8] Douglas B. Williams and Vijay Madisetti, editors. *Digital Signal Processing Handbook*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1997. ISBN 0849385725.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[10] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv preprint arXiv:1602.07868*, 2016.

[11] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in Neural Information Processing Systems*, pages 2377–2385, 2015.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *arXiv preprint arXiv:1603.05027*, 2016.

[14] Sepp Hochreiter. *Untersuchungen zu dynamischen neuronalen Netzen*. PhD thesis, Institut für Informatik, chair Prof. Brauer, Technische Universität München, 1991.

[15] T Nitta. On the critical points of the complex-valued neural network. In *Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP'02.*, volume 3, pages 1099–1103. IEEE, 2002.

[16] Akira Hirose and Shotaro Yoshida. Generalization characteristics of complex-valued feedforward neural networks in relation to signal coherence. *IEEE Transactions on Neural Networks and Learning Systems*, 23(4):541–551, 2012.

[17] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. *arXiv preprint arXiv:1511.06464*, 2015.

[18] Ivo Danihelka, Greg Wayne, Benigno Uria, Nal Kalchbrenner, and Alex Graves. Associative long short-term memory. *arXiv preprint arXiv:1602.03032*, 2016.

[19] Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 4880–4888, 2016.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9 (8):1735–1780, 1997.

[21] Tony A Plate. Holographic reduced representation: Distributed representation for cognitive structures. *Clinical & Laboratory Standards Institute Publications*, 2003.

[22] David P Reichert and Thomas Serre. Neuronal synchrony in complex-valued deep networks. *arXiv preprint arXiv:1312.6115*, 2013.

[23] Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances In Neural Information Processing Systems*, pages 4790–4798, 2016.

[24] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[25] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. *arXiv preprint arXiv:1607.03474*, 2016.

[26] Guangji Shi, Maryam Modir Shanechi, and Parham Aarabi. On the importance of phase in human speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 14 (5):1867–1874, 2006.

[27] Alan V Oppenheim and Jae S Lim. The importance of phase in signals. *Proceedings of the IEEE*, 69(5):529–541, 1981.

[28] Oren Rippel, Jasper Snoek, and Ryan P Adams. Spectral representations for convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 2449–2457, 2015.

[29] George M Georgiou and Cris Koutsougeras. Complex domain backpropagation. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 39(5):330–334, 1992.

[30] Richard S Zemel, Christopher KI Williams, and Michael C Mozer. Lending direction to neural networks. *Neural Networks*, 8(4):503–512, 1995.

[31] Taehwan Kim and Tülay Adalı. Approximation by fully complex multilayer perceptrons. *Neural Computation*, 15(7):1641–1666, 2003.

[32] Akira Hirose. *Complex-valued neural networks: Theories and applications*, volume 5. World Scientific, 2003.

[33] Tohru Nitta. Orthogonality of decision boundaries in complex-valued neural networks. *Neural Computation*, 16(1):73–97, 2004.

[34] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[35] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in Neural Information Processing Systems*, 19:153, 2007.

[36] Christopher Poultney, Sumit Chopra, Yann L Cun, et al. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems*, pages 1137–1144, 2007.

[37] Joan Bruna, Soumith Chintala, Yann LeCun, Serkan Piantino, Arthur Szlam, and Mark Tygert. A mathematical motivation for complex-valued convolutional networks. *arXiv preprint arXiv:1503.03438*, 2015.

[38] Edouard Oyallon and Stéphane Mallat. Deep roto-translation scattering for object classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2865–2873, 2015.

[39] Mark Tygert, Arthur Szlam, Soumith Chintala, Marc'Aurelio Ranzato, Yuandong Tian, and Wojciech Zaremba. Scale-invariant learning and convolutional networks. *arXiv preprint arXiv:1506.08230*, 2015.

[40] Daniel E Worrall, Stephan J Garbin, Daniyar Turmukhambetov, and Gabriel J Brostow. Harmonic networks: Deep translation and rotation equivariance. *arXiv preprint arXiv:1612.04642*, 2016.

[41] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International Conference on Machine Learning*, pages 2071–2080, 2016.

[42] Théo Trouillon and Maximilian Nickel. Complex and holographic embeddings of knowledge graphs: A comparison. *arXiv preprint arXiv:1707.01475*, 2017.

[43] Andy M Sarroff, Victor Shepardson, and Michael A Casey. Learning representations using complex-valued nets. *arXiv preprint arXiv:1511.06351*, 2015.

[44] Nitzan Guberman. On complex valued convolutional neural networks. *arXiv preprint arXiv:1602.09046*, 2016.

[45] Michael Cogswell, Faruk Ahmed, Ross Girshick, Larry Zitnick, and Dhruv Batra. Reducing overfitting in deep networks by decorrelating representations. *arXiv preprint arXiv:1511.06068*, 2015.

[46] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[47] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AIstats*, volume 9, pages 249–256, 2010.

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[49] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 648–656, 2015.

[50] Yurii Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. 1983.

[51] John Thickstun, Zaid Harchaoui, and Sham Kakade. Learning features of music from scratch. In *Proc. ICLR*, 2016.

[52] Julius O Smith. Digital audio resampling. *Online http://www-ccrma.stanford.edu/∼jos/resample*, 2002.

[53] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. ICLR*, 2015.

[54] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[55] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems*, pages 802–810, 2015.

[56] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[57] François Chollet et al. Keras: Deep learning library for Theano and Tensorflow. *https://keras.io*, 2015.