DEEP COMPLEX NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

At present, the vast majority of building blocks, techniques, and architectures for deep learning are based on real-valued operations and representations. However, recent work on recurrent neural networks and older fundamental theoretical analysis suggests that complex numbers could have a richer representational capacity and could also facilitate noise-robust memory retrieval mechanisms. Despite their attractive properties and potential for opening up entirely new neural architectures, complex-valued deep neural networks have been marginalized due to the absence of the building blocks required to design such models. In this work, we provide the key atomic components for complex-valued deep neural networks and apply them to convolutional feed-forward networks. More precisely, we rely on complex convolutions and present algorithms for complex batch-normalization, complex weight initialization strategies for complex-valued neural nets and we use them in experiments with end-to-end training schemes. We demonstrate that such complex-valued models are competitive with their real-valued counterparts. We test deep complex models on several computer vision tasks, on music transcription using the MusicNet dataset and on Speech spectrum prediction using TIMIT. We achieve state of the art performance on these audio-related tasks.

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 (Ioffe and Szegedy, 2015; Salimans and Kingma, 2016) and the emergence of gating-based feed-forward neural networks like Highway Networks (Srivastava et al., 2015). Residual networks (He et al., 2015a; 2015b) 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 (Hochreiter 1991). He et al. (2016) 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 (Ioffe and Szegedy, 2015) 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 (Nitta, 2002), better generalization characteristics (Hirose and Yoshida 2012), faster learning (Arjovsky et al., 2015; Danihelka et al., 2016; Wisdom et al., 2016) and to allow for noise-robust memory mechanisms (Danihelka et al., 2016). Wisdom et al. (2016) and Arjovsky et al. (2015) show that using complex numbers in recurrent neural networks (RNNs) allows the network to have a richer representational capacity. Danihelka et al. (2016) present an LSTM (Hochreiter and Schmidhuber, 1997) 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. Our contributions in this paper are as follows:

1. A formulation of complex batch normalization, which is described in Section 3.5.
2. Complex weight initialization, which is presented in Section 3.6.
3. A state-of-the-art result on the MusicNet multi-instrument music transcription dataset, presented in Section 4.2.

We demonstrate the effectiveness of deep complex networks on standard image classification benchmarks, specifically, CIFAR-10, CIFAR-100, and reduced-training set SVHN*. We also perform a music transcription task on the MusicNet dataset. 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 underscore the potential of deep complex-valued neural networks applied to acoustic related tasks.

2 Motivation and Related Work

Using complex parameters has numerous advantageous from computational, biological, and signal processing perspectives. From a computational point of view, Danihelka et al. (2016) has shown that Holographic Reduced Representations (Plate, 2003), which use complex numbers, are numerically efficient and stable in the context of information retrieval from an associative memory. Danihelka et al. (2016) insert key-value pairs in the associative memory by addition into a memory trace. Although not typically viewed as such, residual networks (He et al., 2015a; 2016) and Highway Networks (Srivastava et al., 2015) 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 (Arjovsky et al., 2015) 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. Arjovsky et al. (2015) show the potential of this type of recurrent neural networks on toy tasks. Wisdom et al. (2016) 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 motivation. Reichert and Serre (2013) have 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 (Srivastava et al., 2015; van den Oord et al., 2016a; b) and recurrent neural networks (Hochreiter and Schmidhuber, 1997; Cho et al., 2014; Zilly et al., 2016) 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 (Shi et al., 2006). This could potentially become an improvement point for better speech recognition systems (cite a couple of them) and specially for advancements in speech and audio synthesis models (van den Oord et al., 2016; Mehri et al., 2016; Sotelo et al., 2017; Wang et al., 2017; Taigman et al., 2017; Arik et al., 2017) Also Oppenheim and Lim (1981) show 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 et al. (2015) leveraged the Fourier spectral representation for convolutional neural networks, providing a technique for parameterizing 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 (2013), the use of complex-valued neural networks (Georgiou and Koutsougeras, 1992; Zemel et al., 1995; Kim and Adalı, 2003; Hirose, 2003; Nitta, 2004) has been investigated long before the earliest deep learning breakthroughs (Hinton et al., 2006; Bengio et al., 2007; Poultney et al., 2007). Recently Reichert and Serre (2013); Bruna et al. (2015); Arjovsky et al. (2015); Danihelka et al. (2016); Wisdom et al. (2016) 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, (Oyallon and Mallat, 2015; Tygert et al., 2015; Worrall et al., 2016) have used complex representation in vision tasks. Wisdom et al. (2016) 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, (Trouillon et al., 2016; Trouillon and Nickel, 2017) 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 perform similarly to, or better than, their real-valued counterparts on real-world tasks.

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.

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.

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 $W = A + iB$ by a complex vector $h = x + iy$ where $A$ and $B$ are real matrices and $x$ and $y$ are real vectors since we are simulating complex arithmetic using
real-valued entities. As the convolution operator is distributive, convolving the vector $h$ by the filter $W$ we obtain:

$$W * h = (A * x - B * y) + i(B * x + A * y).$$

(1)

As illustrated in [2], if we use matrix notation to represent real and imaginary parts of the convolution operation we have:

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

(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.

By constraining activation functions to be complex differentiable or holomorphic, we restrict the use of possible activation functions for a complex-valued neural network (For further details about holomorphism please refer to [5]. Hirose and Yoshida [2012] shows that it is unnecessarily restrictive to limit oneself only to holomorphic activation functions; Those functions that are can be differentiable with respect to the real part and the imaginary part of each parameter are also compatible with backpropagation. (Arjovsky et al., 2015; Wisdom et al., 2016; Danihelka et al., 2016) 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 Sarroff et al. (2015), using holomorphic functions allows one to share gradient values (because the activation satisfies the Cauchy-Riemann equations [5]). So, instead of computing and backpropagating 4 different gradients, only 2 are required.

### 3.4 Complex-Valued Activations

#### 3.4.1 ModReLU

Numerous activation functions have been proposed in the literature in order to deal with complex-valued representations. (Arjovsky et al., 2015) have proposed modReLU, which is defined as follows:

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

(3)

where $z \in \mathbb{C}$, $\theta_z$ 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 $\theta_z$, 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 [4].

#### 3.4.2 CReLU and zReLU

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

$$
\text{CReLU}(z) = \text{ReLU}(|z|) + i \text{ReLU}(|\Im(z)|).
$$

(4)

CReLU 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 $\mathbb{C}$ satisfies the Cauchy-Riemann equations when $\theta_z \in [0, \pi/2]$ or $\theta_z \in [\pi, 3\pi/2].$
It is also worthwhile to mention the work done by [Guberman, 2016] where a ReLU-based complex activation which satisfies the Cauchy-Riemann equations except for the set of points \( \{ \Re(z) > 0, \Im(z) = 0 \} \cup \{ \Re(z) = 0, \Im(z) > 0 \} \). The activation function has similarities to CReLU. We call [Guberman, 2016] activation as \( z \text{ReLU} \) and is defined as follows:

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

We have tested \( z \text{ReLU} \) in deep feed-forward complex networks and the results are given in [3].

### 3.5 Complex Batch Normalization

Deep networks generally rely upon Batch Normalization [Ioffe and Szegedy, 2015] 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 0-centered data \( (x - \mathbb{E}[x]) \) by the inverse square root of the covariance matrix \( \mathbf{V} \):

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

where the covariance matrix \( \mathbf{V} \) is

\[
\mathbf{V} = \begin{pmatrix}
  V_{rr} & V_{ri} \\
  V_{ri} & V_{ii}
\end{pmatrix} = \begin{pmatrix}
  \text{Cov}(\Re\{x\}, \Re\{x\}) & \text{Cov}(\Re\{x\}, \Im\{x\}) \\
  \text{Cov}(\Im\{x\}, \Re\{x\}) & \text{Cov}(\Im\{x\}, \Im\{x\})
\end{pmatrix}.
\]

The square root and inverse of \( 2 \times 2 \) matrices has an inexpensive, analytical solution, and its existence is guaranteed by the positive (semi-)definiteness of \( \mathbf{V} \). Positive definiteness of \( \mathbf{V} \) is ensured by the addition of \( \mathbf{I} \) to \( \mathbf{V} \) (Tikhonov regularization). The mean subtraction and multiplication by the inverse square root of the variance ensures that \( \tilde{x} \) 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:

\[
\begin{align*}
\mu &= \mathbb{E}[\tilde{x}] \\
\Gamma &= \mathbb{E}[(\tilde{x} - \mu)(\tilde{x} - \mu)^*] = V_{rr} + V_{ii} + i(V_{ir} - V_{ri}) \\
C &= \mathbb{E}[(\tilde{x} - \mu)(\tilde{x} - \mu)] = V_{rr} - V_{ii} + i(V_{ir} + V_{ri}).
\end{align*}
\]  

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 [Cogswell et al., 2015; Srivastava et al., 2014].

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

\[
\gamma = \begin{pmatrix}
  \gamma_{rr} & \gamma_{ri} \\
  \gamma_{ri} & \gamma_{ii}
\end{pmatrix}.
\]

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

\[
\text{BN}(\tilde{x}) = \gamma \tilde{x} + \beta.
\]
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 $\beta$ are initialized to 0. The moving averages of $V_{ri}$ and $V_{ii}$ are initialized to $1/\sqrt{2}$. The momentum for the moving averages is set to 0.9.

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 (2010) and He et al. (2015b) to derive the variance of the complex weight parameters.

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

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

where $\theta$ 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:

$$\text{Var}(W) = E[WW^*] - (E[|W|])^2 = E[|W|^2] - (E[|W|])^2,$$

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


Putting them together:

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

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, $\sigma$, indicating the mode. These are:

$$E||W|| = \sigma \sqrt{\frac{\pi}{2}}, \quad \text{Var}(|W|) = \frac{4 - \frac{\pi}{2}}{\sigma^2}.$$ (10)

The variance of $W$ can thus be expressed in terms of its generating Rayleigh distribution’s single parameter, $\sigma$, thus:

$$\text{Var}(W) = \frac{4 - \frac{\pi}{2}}{\sigma^2} + \left(\sigma \sqrt{\frac{\pi}{2}}\right)^2 = 2\sigma^2.$$ (10)

If we want to respect the Glorot and Bengio (2010) criterion which ensures that the variances of the input, the output and their gradients are the same, then we would have $\text{Var}(W) = 2/(n_{in} + n_{out})$, where $n_{in}$ and $n_{out}$ are the number of input and output units respectively. In such case, $\sigma = 1/\sqrt{n_{in} + n_{out}}$. If we want to respect the He et al. (2015b) initialization that presents an initialization criterion that is specific to ReLUs, then $\text{Var}(W) = 2/n_{in}$ which $\sigma = 1/\sqrt{n_{in}}$.

The magnitude of the complex parameter $W$ is then initialized using the Rayleigh distribution with the appropriate mode $\sigma$. We can see from equation (10) that the variance of $W$ depends on its magnitude and not on its phase. We then initialize the phase using the uniform distribution between $-\pi$ and $\pi$. By performing the multiplication of the magnitude by the phasor as is detailed in equation (8) we perform the complete initialization of the complex parameter.

In all the experiments that we report, we use variant of this initialization which leverages the independence property of unitary matrices. As it is stated in Cogswell et al. (2015), Srivastava et al. (2014), and Tompson et al. (2015), learning decorrelated features is beneficial for learning as it allows to perform better generalization and faster learning. This motivates us to achieve initialization by considering a (semi-)unitary matrix which is reshaped to the size of the weight tensor. Once this is done, the weight tensor is multiplied by $\sqrt{He_{\text{var}}/\text{Var}(W)}$ or $\sqrt{Glorot_{\text{var}}/\text{Var}(W)}$ where $Glorot_{\text{var}}$ and $He_{\text{var}}$ are respectively equal to $2/(n_{in} + n_{out})$ and $2/n_{in}$. In such a way we allow kernels to be independent from each other as much as possible while respecting the desired criterion.
Table 1: Model Architecture. S1, S2 and S3 filters are the number of convolution filters used in each layer in stages 1, 2 and 3. (S) denotes a small network while (L) denotes a large network.

<table>
<thead>
<tr>
<th>ARCHITECTURE</th>
<th>PARAMS</th>
<th>LAYERS</th>
<th>S1 FILTERS</th>
<th>S2 FILTERS</th>
<th>S3 FILTERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL (S)</td>
<td>860K</td>
<td>56</td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>COMPLEX (S)</td>
<td>860K</td>
<td>64</td>
<td>22</td>
<td>44</td>
<td>88</td>
</tr>
<tr>
<td>REAL (L)</td>
<td>1.71M</td>
<td>110</td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>COMPLEX (L)</td>
<td>1.70M</td>
<td>118</td>
<td>22</td>
<td>44</td>
<td>88</td>
</tr>
</tbody>
</table>

Table 2: Classification error on CIFAR-10, CIFAR-100 and SVHN. Note that He et al. (2016) uses a 110 layer model.

<table>
<thead>
<tr>
<th>ARCHITECTURE</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
<th>SVHN</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPLEX WIDER AND SHALLOWER</td>
<td>6.17</td>
<td>26.36</td>
<td>3.70</td>
</tr>
<tr>
<td>COMPLEX DEEPER AND NARROWER</td>
<td>6.73</td>
<td>28.22</td>
<td>3.72</td>
</tr>
<tr>
<td>COMPLEX IN BETWEEN</td>
<td>5.59</td>
<td>28.64</td>
<td>3.62</td>
</tr>
<tr>
<td>REAL WIDER AND SHALLOWER</td>
<td>5.42</td>
<td>27.22</td>
<td>3.42</td>
</tr>
<tr>
<td>REAL DEEPER AND NARROWER</td>
<td>6.29</td>
<td>27.84</td>
<td>3.52</td>
</tr>
<tr>
<td>REAL IN BETWEEN</td>
<td>6.07</td>
<td>27.71</td>
<td>4.30</td>
</tr>
<tr>
<td>DIFF</td>
<td>-0.17</td>
<td>+0.86</td>
<td>-0.19</td>
</tr>
</tbody>
</table>

3.7 Complex Convolutional Residual Network

A deep convolutional residual network of the nature presented in [He et al. (2015a, 2016)] consists of three stages within which 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 and doubled. The sizes of the convolution kernels are always set to 3 x 3. Within a stage, there are several residual blocks which comprise 2 convolution layers each. The contents of one such residual block in the real and complex setting is illustrated in Appendix Figure [16].

In the complex valued setting, the majority of the architecture remains identical to the one presented in [He et al. (2016)] with a few subtle differences. Since all datasets that we work 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

\[\text{BN} \rightarrow \text{Activation} \rightarrow \text{Conv} \rightarrow \text{BN} \rightarrow \text{Activation} \rightarrow \text{Conv}\]

Using this learning block yielded better empirical results than assuming that the input image has a null imaginary part. The parameters of this real-valued residual block are trained by backpropagating errors from the task specific loss function. Secondly, we perform a \(\text{Conv} \rightarrow \text{BN} \rightarrow \text{Activation}\) operation on the obtained complex input before feeding it to the first residual block. We also perform the same operation on the real-valued network input instead of \(\text{Conv} \rightarrow \text{Maxpooling}\) as in [He et al. (2016)]. Inside, residual blocks, 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 1x1 convolution applied on it with the same number of filters used throughout the stage and subsample by a factor of 2. In contrast, [He et al. (2016)] perform a similar 1x1 convolution with twice the number of feature filters in the current stage to both downsample the feature maps spatially and double them in number.

4 Experimental Results

In this section, we present empirical results from using our model to perform image, music classification and spectrum prediction. First, we present our model’s architecture followed by the results we obtained on 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.

\(^*\)110 layer network (Table 2 of [He et al. (2016)])
Table 3: Classification error on CIFAR-10, CIFAR-100 and SVHN* using different complex activation functions ($\text{zReLU}$, $\text{modReLU}$ and $\text{CReLU}$). 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. DNC is reported when the model did not converge or produced NaNs due to instabilities in training. Performance differences between the real network and the complex network using $\text{CReLU}$ are reported between their respective best models. All models are constructed to have roughly 1.7M parameters.

<table>
<thead>
<tr>
<th>ARCH</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
<th>SVHN*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\text{zReLU}$</td>
<td>$\text{modReLU}$</td>
<td>$\text{CReLU}$</td>
</tr>
<tr>
<td>CWS</td>
<td>11.71</td>
<td>23.42</td>
<td>6.17</td>
</tr>
<tr>
<td>CDN</td>
<td>9.50</td>
<td>22.49</td>
<td>6.73</td>
</tr>
<tr>
<td>CIB</td>
<td>11.36</td>
<td>23.63</td>
<td>5.59</td>
</tr>
<tr>
<td>RWS</td>
<td>6.07</td>
<td>27.71</td>
<td></td>
</tr>
<tr>
<td>RDN</td>
<td>6.29</td>
<td>27.84</td>
<td></td>
</tr>
<tr>
<td>RIB</td>
<td>5.42</td>
<td>27.22</td>
<td></td>
</tr>
<tr>
<td>DIFF</td>
<td>-0.17</td>
<td>+0.86</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: 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, CBN 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 $\text{CReLU}$ are reported for the WS, DN & IB architectures. All models are constructed to have roughly 1.7M parameters.

<table>
<thead>
<tr>
<th>ARCH</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
<th>SVHN*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NCBN(C)</td>
<td>CBN(R)</td>
<td>BN(C)</td>
</tr>
<tr>
<td>WS</td>
<td>NAN</td>
<td>5.47</td>
<td>6.32</td>
</tr>
<tr>
<td>DN</td>
<td>NAN</td>
<td>5.66</td>
<td>-</td>
</tr>
<tr>
<td>IB</td>
<td>NAN</td>
<td>-</td>
<td>6.71</td>
</tr>
<tr>
<td>DIFF</td>
<td>-</td>
<td>-0.05</td>
<td>-0.73</td>
</tr>
</tbody>
</table>

4.1 Image Recognition

We adopt an architecture that is similar to [He et al. (2016)]. This will also serve as our 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 $\text{CReLU}$, $\text{zReLU}$ and $\text{modReLU}$ activation functions. We use a cross entropy loss for both real and complex models. The real and complex-valued networks have 3 stages within which the number and shape of feature maps and convolution kernels are preserved. At the end of each stage the feature maps are spatially downsam-pled linearly by a factor of 2 with a convolution and the number of convolution kernels are doubled thereby doubling the number of feature maps as well. 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 wide and shallow architecture for a complex network starts with 24 filters per convolution layer in the initial stage and 16 residual blocks per stage. The DN architecture has 20 filters with 23 blocks per stage while the IB variant has 22 filters with 19 blocks per stage.

All models (real and complex) were trained using the backpropagation algorithm with Stochastic Gradient Descent with Nesterov momentum [Nesterov, 1983] set at 0.9. We also clip the norm of our gradients to 1. We tweaked the learning rate schedule used in [He et al. (2016)] 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.

Table 2 presents our results on performing image classification on CIFAR-10, CIFAR-100 and SVHN. In addition, we also consider a truncated version of the Street View House Numbers (SVHN) dataset which we call SVHN*. For computational reasons, we use the required 73,257 training images of Street View House Numbers (SVHN). We still test on all 26,032 images. The "In between" model has yielded the best accuracy on CIFAR-10 and SVHN* for both the real and complex models. On CIFAR-100, the best result has been obtained on both the real and complex models using the wide and shallow architecture. 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 3 contains the results for hybrid models. We can observe in the tables [2][3] that in cases 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. In cases, where real-valued representation outperformed the complex one, replacing a complex batch normalization by a standard batch normalization increased the accuracy of the complex models. In general, these experiments show that the difference in efficiency between the real and complex models varies according to the dataset, to the task and to the architecture.

Ablation studies were performed to contrast real-valued and complex-valued Batch Normalization layers using the CIFAR-100 and SVHN datasets. The real-valued Batch Normalization performed very poorly: In 5 out of 6 experiments, training failed with the apparition of NaNs. By way of contrast, all 6 complex-valued Batch Normalization experiments converged.

Another ablation study was undertaken to compare ReLU, modReLU and holomorphic RELU. Again the differences were stark: All ReLU experiments converged and outperformed both modReLU and hReLU, both which variously failed to converge or fared substantially worse.

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 [Thickstun et al. 2016]. For computational efficiency we resampled the original input from 44.1kHz to 11kHz using the algorithm described in Smith (2002). This sampling rate is sufficient to recognize frequencies presented in the dataset while reducing computational cost dramatically. We modeled each of the 84 notes that are present in the dataset with independent sigmoids. 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 of the sampling rate). The filter size was 512 samples (about 12ms) with a stride of 16. The model for the spectral input is similar to the VGG model [Simonyan and Zisserman, 2015]. The first layer has filter with size of 7 and is followed by 5 convolutional layers with filters of size 3. 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.

The complex network was initialized using the unitary initialization scheme respecting the He criterion as described in Section 3.6. For the real-valued network, we have used the analogue initialization of the weight tensor. It consists of performing an orthogonal initialization with a gain of $\sqrt{2}$. The complex batch normalization was applied according to Section 3.5. Following [Thickstun et al. 2016] we used recordings with ids ‘2303’, ‘2382’, ‘1819’ as the test subset and additionally we created a validation subset using recording ids ‘2131’, ‘2384’, ‘1792’, ‘2514’, ‘2567’, ‘1876’ (randomly chosen from the training set). The validation subset was used for model selection and early stopping. The remaining 321 files were used for training. The results are summarized on Table 5.
Table 5: 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.

<table>
<thead>
<tr>
<th>ARCHITECTURE</th>
<th>FS</th>
<th>Params</th>
<th>AP, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHALLOW, REAL</td>
<td>11KHz</td>
<td></td>
<td>66.1</td>
</tr>
<tr>
<td>SHALLOW, COMPLEX</td>
<td>11KHz</td>
<td></td>
<td>66.0</td>
</tr>
<tr>
<td>SHALLOW, \cite{Thickstun2016}</td>
<td>44.1KHz</td>
<td>-</td>
<td>67.8</td>
</tr>
<tr>
<td>DEEP, REAL</td>
<td>11KHz</td>
<td>10.0M</td>
<td>69.6</td>
</tr>
<tr>
<td>DEEP, COMPLEX</td>
<td>11KHz</td>
<td>8.8M</td>
<td>72.9</td>
</tr>
</tbody>
</table>

Table 6: Speech Spectrum prediction test on TIMIT. \(CConv-LSTM\) denotes the complex convolutional LSTM.

<table>
<thead>
<tr>
<th>MODEL</th>
<th>#PARAMETERS</th>
<th>MSE(VALIDATION)</th>
<th>MSE(TEST)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM, \cite{Wisdom2016}</td>
<td>(\approx 135k)</td>
<td>16.59</td>
<td>16.98</td>
</tr>
<tr>
<td>FULL-CAPACITY (U)RNN, \cite{Wisdom2016}</td>
<td>(\approx 135k)</td>
<td>14.56</td>
<td>14.66</td>
</tr>
<tr>
<td>Conv-LSTM ( OUR BASELINE )</td>
<td>(\approx 113k)</td>
<td>10.97</td>
<td>12.74</td>
</tr>
<tr>
<td>CConv-LSTM (OURS)</td>
<td>(\approx100k)</td>
<td>10.74</td>
<td>12.44</td>
</tr>
</tbody>
</table>

We achieve a performance comparable to the baseline with the shallow convolutional network. The deep complex convolutional network, which has significantly less parameters than the real model, achieves \(72.9\%\) average precision which is the state of the art to the best of our knowledge. See the Appendix for precision-recall curves and a sample of the output of the model.

4.3 Speech Spectrum Prediction

We apply both a convolutional LSTM and a complex convolutional LSTM on speech spectrum prediction task. In this task, the model is to predict the imaginary part and the real part of the spectrum at time \(t + 1\), given all the spectrum (imaginary part and real part) up to time \(t\). We evaluate the model with mean-square-error (MSE) on log-magnitude to compare with the others. The experiments are conducted on downsampled (8kHz) TIMIT dataset. By following the steps in \cite{Wisdom2016}, raw audio waves are transformed into frequency domain via short-time Fourier transform (STFT) with a Hann analysis widnow of 256 samples and a window hop of 128 samples. We use a training set with 3690 utterances, a validation set with 400 utterances and a standard test set with 192 utterance.

To match the number of parameters for both model, the convolutional LSTM has 96 feature maps while the complex model has 128 feature maps. Adam \cite{Kingma2014} with a fixed learning rate \(1e-4\) is used in both experiments. We initialize the complex model with unitary initialization scheme and the other with golort initialization scheme. The result is shown in Table 6. Our baseline model has achieved the state of the art and the complex convolutional LSTM model speform iowemeshghtly better over the baseline.

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.
REFERENCES


吉村英次郎。*复合值神经网络: 理论与应用*, 第五卷。世界科学出版社, 2003。


APPENDIX

In practice, the complex convolution operation is implemented as illustrated in Fig. 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 \( M_I \).

(a) An illustration of the complex convolution operator. \hspace{1cm} (b) A complex convolutional residual network (left) and an equivalent real-valued residual network (right).

Figure 1: Complex convolution and residual network implementation details.

VISUALIZATIONS

(a) Training curve \hspace{1cm} (b) Testing curve

Figure 2: Training and test error curves for the small complex vs real valued network on CIFAR-10

Figures 3 and 4 were generated using the small complex resnet model trained on CIFAR-10.
Figure 3: (a) Stage 1 feature maps as real and imaginary pairs for each input, vs. (b) feature maps as magnitude and phase.

Figure 4: Stage 2 and 3 feature maps as real and imaginary pairs for each input

**MUSICNet illustrations**

Figure 5: Precision-recall curve

**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, $f'(z_0)$
lim_{\Delta z \to 0} \left[ \frac{(f(z_0) + \Delta z) - f(z_0)}{\Delta z} \right] 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 + iy such that \ f(z) = u(x, y) + i v(x, y). u and v are real-valued functions. One possible way of expressing \ \Delta z is to have \ \Delta z = \Delta x + i \Delta y. \ \Delta z 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 \ \Delta z approaches 0 along the real axis, \ f'(z_0) could be written as:

\[
\begin{align*}
f'(z_0) & \equiv \lim_{\Delta z \to 0} \left[ \frac{(f(z_0) + \Delta z) - f(z_0)}{\Delta z} \right] \\
& = \lim_{\Delta x \to 0} \lim_{\Delta y \to 0} \left[ \frac{\Delta u(x_0, y_0) + i \Delta v(x_0, y_0)}{\Delta x + i \Delta y} \right] \\
& = \lim_{\Delta x \to 0} \left[ \frac{\Delta u(x_0, y_0) + i \Delta v(x_0, y_0)}{\Delta x + 0} \right]. \quad (11)
\end{align*}
\]

When \ \Delta z approaches 0 along the imaginary axis, \ f'(z_0) could be written as:

\[
\begin{align*}
& = \lim_{\Delta y \to 0} \lim_{\Delta x \to 0} \left[ \frac{\Delta u(x_0, y_0) + i \Delta v(x_0, y_0)}{\Delta x + i \Delta y} \right] \\
& = \lim_{\Delta y \to 0} \left[ \frac{\Delta u(x_0, y_0) + i \Delta v(x_0, y_0)}{0 + i \Delta y} \right]. \quad (12)
\end{align*}
\]

Satisfying equations \[11\] and \[12\] is equivalent of having \ \frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} = i \left( -\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x} \right). \ 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.