Deep Complex Network

Francesco Starna Gianmarco Bracalello
Sapienza University of Rome
starna.1613660@studenti.uniroma1.it
bracalello.1551766@studenti.uniroma1.it

Abstract

In recent years, research on deep learning models have been done using essentially techniques and architectures based on real-valued operations. From a mathematical point of view it is ensured that complex numbers have a richer representation. Despite the clear advantage on this, developing complex models has never taken hold, due to the absence of prebuilt and stable models. In this paper we are going to present our solution, using Tensorflow [1], of the complex components, in order to build a complex-valued model. We also show that complex models can be applied in tasks such as image recognition and time series forecasting problems, obtaining competitive results.

1 Introduction

The objective of this work is to extend the residual networks (ResNet) in the field of complex numbers following the paper (Trabelsi et al., 2017) on deep complex networks. Residual networks are used to exploit the advantages of deep networks by providing shortcut paths which results are added to the output of each block reducing the effects of vanishing gradients. In order to reach this goal each layer of the network has been customized to deal with complex inputs, and applied to the context of feed-forward convolutional networks. Specifically, the complex-valued layers we implemented are:

- Complex Batch Normalization
- Complex Weight Initialization
- Complex Convolution
- Complex ReLU-based activation functions
- Imaginary Learning Block
- Complex Dropout

[1] code can be found at https://bit.ly/3iXvBip

Behind the scenes, all the computations are done in the real context, since most of the available tensor operations are not provided for dealing with complex inputs. We perform experiments on CIFAR10, CIFAR100 and the Weather Dataset, and the results have been compared with the real counterpart, obtaining some interesting outcomes.

2 Model

In this section we present the core of complex-valued deep models, going deeper towards the mathematical framework.

2.1 Complex Weight Initialization

Weight initialization is critical in reducing the risks of vanishing gradients. Complex numbers have a polar form and a rectangular form

$$W = |W|e^{i\theta} = R(W) + iI(W)$$

. The polar form is represented by e magnitude $|W|$ and the phase θ (argument) of the complex input. The magnitude follows the Rayleigh distribution (Raqab and Madi, 2011) with mode $\sigma = 1/\sqrt{n_{in}}$ where n_{in} are the input features, while the phase is initialized using the uniform distribution between $-\pi$ and π. The complex weights are computed multiplying the magnitude by the phasor $e^{i\theta}$ and finally the initialization is done in its real and imaginary part. The weight initialization process is used in the convolutions.

2.2 Complex Convolution

In order to convolve a complex input equivalently to the real-valued one, we have to perform the operation $W * h$ where, $W = A + iB$ is a complex filter matrix, with A and B real matrices, and $h = x + iy$ is a complex vector, with x and y are real vectors. Convolving the two we obtain

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

The custom layer is in fact made by two kernels, one performs the real convolution and the other one performs the imaginary convolution, dividing before the input in its real and imaginary part, and then building again the complex result.

2.3 Complex Batch Normalization

Batch normalization is essential to optimize the model training and performance, from its introduction by (Ioffe and Szegedy, 2015). The real normalization works translating and scaling the inputs such that their mean is 0 and their variance is 1. This does not ensure equal results when applied to complex numbers, given the presence of the imaginary part. We treat instead the problem by scaling the data by the square root of their variances, by multiplying the 0-centered data $(x - E[x])$ by the inverse square root of the 2x2 covariance matrix V

$$\tilde{x} = (V)^{-1/2}(x - E[x])$$

where the covariance matrix V is

$$\begin{pmatrix} Cov(\Re(x), \Re(x)) & Cov(\Re(x), \Im(x)) \\ Cov(\Im(x), \Re(x)) & Cov(\Im(x), \Im(x)) \end{pmatrix}$$

The inverse square root is guaranteed applying Tikhonov regularization (Calvetti and Reichel, 2003). This ensures $\tilde{x}$ has mean $\mu = 0$, covariance $\Gamma = 1$, pseudo-covariance $C = 0$. Analogously to the real-valued batch normalization, we also add two learning parameters, the scaling 2x2 matrix gamma and the shift beta, so that the operation becomes

$$BN(\tilde{x}) = \gamma\tilde{x} + \beta$$

The parameters of gamma are initialized to

$$\begin{pmatrix} 1/\sqrt{2} & 0 \\ 0 & 1/\sqrt{2} \end{pmatrix}$$

while beta is initialized to 0.

2.4 Complex-Valued Activation

To implement the complex network some complex activation functions are used and in this section we go through them analyzing their structures and parameters.

2.4.1 CRelu

This complex-valued activation function is simply the combination of two ReLUs on the real and the imaginary parts of the complex input and the output is a complex value.

$$\mathbb{C}ReLU(z) = ReLU(\Re(z)) + iReLU(\Im(z))$$

2.4.2 ZRelu

This activation function was introduced by (Guberman, 2016) where the ReLU is defined as the input if the argument of the inputs itself is in the interval between 0 and $\pi/2$. Dealing with tensors the condition can be checked by previously constructing two tensors as representing the extreme of the interval and then checking the actual condition.

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

2.4.3 ModRelu

This was proposed by (Arjovsky et al., 2015) and it is based on the fact that applying classic ReLUs on real and imaginary parts of the input performs bad in simple tasks. ModReLU is characterized by a dead zone of radius b around the origin, in this area the neuron is inactive. The design of this activation function is the following:

$$modReLU(z) = ReLU(|z| + b)e^{i\theta_z} =$$

$$= \begin{cases} (|z| + b)\frac{z}{|z|} & |z| + b \geq 0 \\ 0 & otherwise \end{cases}$$

In which θ represents the phase of the input and $|z|$ represents its magnitude and b is a learning parameter.

2.5 Complex Residual Network

The complex residual network follows the one presented in (He et al., 2015), and consists of 3 stages each one containing several residual blocks, depending on the architecture. At the end of each stage, the feature maps are downsampled by a factor of 2, using a 1x1 convolution with stride 2, and the number of filters are doubled. After the 3 stages, before feeding the linear classifier, we apply global average pooling.

2.5.1 Imaginary Part Learning

Since the input to the network is represented by real numbers, it is first 'complexed' by an imaginary learning block, which is a custom layer that implements one real-valued residual block of the type BN → ReLU → Conv → BN → ReLU → Conv. In this way, we give as input the real-valued data and we learn the imaginary representation of the input, giving as output its complex representation. This yields better results instead of assuming that all the inputs have null valued imaginary part. Before feeding the complex input to the first residual block, we apply a C-Conv → C-BN → C-Activation.

2.5.2 Residual Block

A complex residual block comprises the complex components described before: C-BN → C-Activation → C-Conv → C-BN → C-Activation → C-Conv. The output of each residual block is concatenated with its input after each residual block.

2.5.3 Dropout

During the training phase, we noticed that given the high number of epochs necessary to train the network, overfitting happened very often. We decided to build a complex dropout layer, that basically performs individual dropout to both the real and imaginary parts. The dropout is applied before each residual connection, so that the residual block becomes C-BN → C-Activation → C-Conv → C-BN → C-Activation → C-Conv → C-Dropout. The dropout layer gives higher performances on all the architectures.

2.6 Real-Valued Residual Network

The real counterpart is built according to He et al. with a few differences. First, Conv → BN → ReLU is applied before feeding the first residual block, instead of Conv → MaxPooling. Secondly, the number of blocks and filters is different, in order to build models with more or less the same number of parameters, to do a better comparison.

3 Experiments

3.1 Architecture

In order to test the performance of the complex network we consider the trade-off between the depth and the width of the model, yielding three different architectures: wide and shallow (WS), deep and narrow (DN) and in between (IB) both for the real and the complex network.

Regarding the real network, the WS architecture starts with 18 real filters per convolutional layer doubled each stage and 14 blocks per stage, the IB architecture uses 16 real filters with 18 blocks per stage, and the DN architecture starts with 14 real filters and 23 blocks per stage.

The complex counterpart is slightly different in a sense that the WS architecture starts with 12 complex filters per convolutional layer (24 real) and 14 residual blocks per stage, the IB architecture starts with 11 complex filters and 17 blocks per stage, and the DN architecture starts with 10 complex filters and 20 blocks per stage.

All the convolutional layers have 3x3 kernel, unlike the first convolution before feeding the residual blocks, that is a 7x7. The kernels are also regularized with l2 (Cortes et al., 2012) and initialized with he normal (He et al., 2015). All the models have been built with roughly 1.5M parameters, except the complex residual network, when ModRelu is considered as activation function, which increases the parameters to 2M.

3.2 Training

We trained all the networks using two optimizers: Adam (Kingma and Ba, 2017) and Stochastic Gradient Descent (Robbins, 2007). The real network was easy to train, while, due to GPU resources limits, the complex network required several training days to reach a good result, saving the model weights at some checkpoints.

3.2.1 Image Recognition

We tested the real and complex networks both on CIFAR10 and CIFAR100 and we optimized the cross entropy function for both. Before feeding the neural network we normalized the image data dividing each entry by 255. For the image recognition task, SGD optimizer, combined with Nesterov momentum set to 0.9 and gradient clipping, resulted slower, but reached the highest performance ensuring the convergence. We used a learning rate scheduler in order to speed up the training and to not get stuck in plateau. With regard to the real network we start the learning rate at 0.01 for the first 5 epochs to warm up, and then set it to 0.1 from epoch 5-30 and then anneal the learning rates by a factor of 10 at epochs 30, 40 and 50. We end the training at epoch 60. The complex counterpart does the same scheduling, with different epochs. Warm up until 10, then set the learning rate to 0.1 from 10-100 and then anneal it at epochs 100, 120

Architecture	Blocks	Filters	Activation	Image Recognition		Time Series	
				CIFAR10	CIFAR100	1h	24h
Wide-Shallow	14	[12,24,48]	ModRelu	0.767	0.524	0.902	0.892
			CRelu	0,801	0,613	0.934	**0.946**
			ZRelu	0,788	-	0,937	0.921
Real Valued	16	[18,36,72]	Relu	**0.873**	**0,709**	**0.940**	0.930
In-Between	17	[11,22,44]	ModRelu	0,740	0,503	0,919	0,925
			CRelu	0,783	0,605	0,935	**0,941**
			ZRelu	0,758	-	0,920	0,926
Real Valued	19	[16,32,64]	Relu	**0,851**	**0,617**	**0,936**	0,936
Deep-Narrow	20	[10,20,40]	ModRelu	0,703	0,510	0,905	0,941
			CRelu	0,794	0,626	**0,931**	0,937
			ZRelu	0,771	-	0,928	0,928
Real Valued	23	[14,28,56]	Relu	**0,866**	**0,678**	0,899	**0,949**

Table 1: Performance measures in accuracy of the different architectures on both tasks, image recognition and time series forecasting. The bold valued correspond to the highest score obtained per model and per task. A "-" is filled in front of an unperformed experiment, because of the appearance of NaN loss values.

and 150. According to the results, we can confirm that the real valued networks outperform a lot the complex valued ones on CIFAR10, while this gap is reduced on CIFAR100. Generally speaking, input data like images better fit real valued residual network, since the imaginary part learned does not improve the learning. Table 1 shows the results.

3.2.2 Time Series Forecasting

Time series analysis is the use of a model to predict future values based on previously observed ones. In particular we have taken the observation of the Weather Dataset, recorded by the Max Planck Institute for Biogeochemistry. This dataset contains 14 different features such as air temperature, atmospheric pressure, and humidity. These were collected every 10 minutes, beginning in 2003. For our work we used only the data collected between 2009 and 2016, and we have taken hourly recordings for efficiency. Before feeding the neural network we worked on the dataset on a few things: first, we used the pandas library for reading the csv file downloaded, then we analyzed and enhanced the features, and finally we created a window generator class for enabling the data windowing on the dataset, in order to train time series. Specifically, we noticed that some minimum and maximum values were maybe mistaken when recorded, so we modified them in order to avoid learning false data. We also changed some features: the wind velocity and the wind direction, have been changed to a wind vector, to better describe the feature, and we also introduced two new features based on periodicity, day and year, deleting the unuseful string format data. We finally normalized the data and splitted it in training, validation, and test data according to the 70%, 20%, 10% proportions.

For time series forecasting task, Adam optimizer, combined with gradient clipping, has proven better than SGD and faster in convergence. We tested our real and complex networks on two tasks: 1 hour prediction and 24 hours prediction. The loss function to be minimized is mean squared error. For the real network convergence was reached after only 20 epochs with base Adam and learning rate set to 0.001. The complex counterpart needed 40 epochs, when using CRelu and ZRelu, and 100-120 epochs when using ModRelu, depending on the architecture, with the same optimization parameters. The network behaves better in time series forecasting, in which training is faster and the final accuracy is higher, performing around 0.90 with peaks of 0.95 and valleys 0.88. In this case, the performances of the complex valued networks matche the real valued one, and in some cases it is even higher. This happens because time series forecasting is a more appropriate task for representing complex data. The results are shown in the Table 1.

4 Conclusion

We have presented a tensorflow implementation of some key building blocks for complex valued inputs, and we tested it on some general task such as image recognition and time series forecasting. Training deep neural networks requires time and also resources, and given our limits we have done the best we could, obtaining good results, on the various architectures presented. We really hope that our work on complex valued networks will be useful in the future when complex data will become available on other different tasks.

5 Acknowledgement

We want to thank professor Aurelio Uncini for delivering the Neural Network course, and the assistants Simone Scardapane and Danilo Comminiello for the availability and the hints given during the project development.

References

Martin Arjovsky, Amar Shah, and Yoshua Bengio. 2015. Unitary evolution recurrent neural networks.

Daniela Calvetti and Lothar Reichel. 2003. Tikhonov regularization of large linear problems. *BIT*, 43:263–283.

Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. 2012. L2 regularization for learning kernels. *CoRR*, abs/1205.2653.

Nitzan Guberman. 2016. On complex valued convolutional neural networks.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Diederik P. Kingma and Jimmy Lei Ba. 2017. Adam: a method for stochastic optimization.

Mohammad Raqab and Mohamed Madi. 2011. Generalized rayleigh distribution. pages 599–603.

H. Robbins. 2007. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407.

Chiheb Trabelsi, Olexa Bilaniuk, Ying Zhang, Dmitriy Serdyuk, Sandeep Subramanian, João Santos, Soroush Mehri, Negar Rostamzadeh, Y. Bengio, and Christopher Pal. 2017. Deep complex networks.