Complex-Valued vs. Real-Valued Neural Networks for Classification Perspectives: An Example on Non-Circular Data

J Agustin A Barrachina, Chengfang Ren, Christele Morisseau, Gilles Vieillard, Jean-Philippe Ovarlez

Complex-Valued vs. Real-Valued Neural Networks for Classification Perspectives: An Example on Non-Circular Data

J. A. Barrachina,[12*] **C. Ren,**[1] **C. Morisseau,**[2] **G. Vieillard,**[2] **J.-P. Ovarlez**[12]

[1] SONDRA, CentraleSupélec, Université Paris-Saclay, rue Joliot Curie, 91192 Gif-sur-Yvette, France
[2] DEMR, ONERA, Université Paris-Saclay, Chemin de la Hunière, 91120 Palaiseau, France

Abstract

The contributions of this paper are twofold. First, we show the potential interest of Complex-Valued Neural Network (CVNN) on classification tasks for complex-valued datasets. To highlight this assertion, we investigate an example of complex-valued data in which the real and imaginary parts are statistically dependent through the property of non-circularity. In this context, the performance of fully connected feed-forward CVNNs is compared against a real-valued equivalent model. The results show that CVNN performs better for a wide variety of architectures and data structures. CVNN accuracy presents a statistically higher mean and median and lower variance than Real-Valued Neural Network (RVNN). Furthermore, if no regularization technique is used, CVNN exhibits lower overfitting. The second contribution is the release of a Python library (Barrachina 2019) using *Tensorflow* as back-end that enables the implementation and training of CVNNs in the hopes of motivating further research on this area.

1 Introduction

In the Machine Learning community, most Neural Networks are developed for processing real-valued features (voice signals, RGB images, videos, etc.). The signal processing community, however, has a higher interest in developing theory and techniques over complex fields. Indeed, complex-valued signals are encountered in a large variety of applications such as biomedical sciences, physics, communications and radar. All these fields use signal processing tools (Schreier and Scharf 2010), which are usually based on complex filtering operations (Discrete Fourier Transform, Wavelet Transform, Wiener Filtering, Matched Filter, etc.). CVNNs appear as a natural choice to process and to learn from these complex-valued features since the operation performed at each layer of CVNNs can be interpreted as complex filtering. Notably, CVNNs are more adapted to extract phase information (Hirose and Yoshida 2012), which could be helpful, *e.g.*, for retrieving Doppler frequency in radar signals, classifying polarimetric SAR data (Hänsch and Hellwich 2009; Zhao et al. 2019), etc.

Although CVNN has been proven to achieve better results than its real-valued counterpart for particular structures of complex-valued data (Hirose and Yoshida 2012; Hirose 2012, 2009; Hänsch and Hellwich 2009), the difficulties in implementing CVNN models in practice have slowed down the field from growing further (Mönning and Manandhar 2018). In this paper, we address this issue by providing a *Python* library to deal with the implementation of CVNN models. The code also offers statistical indicators, e.g., loss and accuracy box plots (McGill, Tukey, and Larsen 1978; Chambers 2018), to compare the performance of CVNN models with its real-valued counterpart.

There are no rules in practice to prioritize CVNN over RVNN for complex-valued datasets. Performance highly depend on the characteristics of the complex-valued dataset. Indeed, merely taking real data as input does not profit from using CVNN (Mönning and Manandhar 2018). We propose in this paper to analyze the influence of the non-circular statistical property on the performance of both CVNN and RVNN networks. We show that particular structures of such complex data, as, for example, phase information or statistical correlation between real and imaginary parts, can notably benefit from using CVNN compared to its real-valued equivalent. Under this context, CVNN is potentially an attractive network to obtain better classification performance on complex datasets.

The paper is organized as follows: In Section 2, we discuss the mathematical background of CVNN to then continue with the circularity property for a random variable. Section 3 discusses the feed-forward network architecture and the programming code used for these experiments. Section 4 illustrates the comparison of statistical performance obtained for each network. In particular, the sensitivity of CVNN and RVNN results are evaluated either by changing the dataset characteristics or the network hyper-parameters.

Although CVNN is an acronym that involves numerous complex-valued neural network architectures, in this work, we will always be referring to fully connected feed-forward ones. This convention was chosen to be coherent with the existing bibliography (Hirose and Yoshida 2012; Hirose 2012, 2009; Hänsch and Hellwich 2009; Amin et al. 2011; Hirose 2013).

2 Mathematical Background

A natural way to build CVNN consists in extending RVNN for handling complex-valued neurons. The latter implies that

*The authors would like to thank the Délégation Générale de l'Armement (DGA) for funding.

the weights for connecting neurons and the hidden activation functions should be complex-valued. In contrast, the loss function remains real-valued to minimize an empirical risk during the learning process. Despite the architectural change for handling complex-valued inputs, the main challenge of CVNN is the way to train such neural networks.

2.1 Wirtinger derivation

When training CVNNs, the weights are updated using a complex gradient, so the back-propagation operation becomes complex-valued. In complex analysis, *Liouville's theorem* states that every bounded holomorphic function is constant (Fine and Rosenberger 1997, pp.70-71), implying that the loss and activation functions should be either constant or unbounded.

Wirtinger calculus (Wirtinger 1927) generalizes the notion of complex derivative for *non-holomorphic* functions. It states that given a complex function $f(z)$ of a complex variable $z = x + j\,y \in \mathbb{C}, (x, y) \in \mathbb{R}^2$, the partial derivatives with respect to z and $\overline{z}$ respectively are:

$$\frac{\partial f}{\partial z} \triangleq \frac{1}{2}\left(\frac{\partial f}{\partial x} - j\,\frac{\partial f}{\partial y}\right), \frac{\partial f}{\partial \overline{z}} \triangleq \frac{1}{2}\left(\frac{\partial f}{\partial x} + j\,\frac{\partial f}{\partial y}\right). \tag{1}$$

Wirtinger calculus enables one to work with *non-holomorphic* functions, providing an alternative method for computing the gradient. Following references (Amin et al. 2011) and (Hirose, Amin, and Murase 2013), the complex gradient is then defined as:

$$\nabla_z f = 2\,\frac{\partial f}{\partial \overline{z}}. \tag{2}$$

Also, for any real-valued loss function $\mathcal{L} : \mathbb{C} \to \mathbb{R}$, the complex derivative of the composition of $\mathcal{L}$ with any complex function $g : \mathbb{C} \to \mathbb{C}$ with $g(z) = r(z) + j\,s(z)$ is given by the following so-called chain rule:

$$\frac{\partial \mathcal{L} \circ g}{\partial \overline{z}} = \frac{\partial \mathcal{L}}{\partial r}\frac{\partial r}{\partial \overline{z}} + \frac{\partial \mathcal{L}}{\partial s}\frac{\partial s}{\partial \overline{z}}. \tag{3}$$

Using equations 2 and 3 it is possible to train CVNN.

2.2 Circularity

The importance of circularity for CVNNs has already been mentioned in (Hirose 2012, 2013). Let us denote the vector $\mathbf{u} \triangleq [X, Y]^T$ as the real random vector built by stacking the real and imaginary parts of a complex random variable $Z = X + j\,Y$. The probability density function (pdf) of Z can be identified with the pdf of $\mathbf{u}$. The *variance* of Z is defined by:

$$\sigma_Z^2 \triangleq \mathbb{E}\left[|Z - \mathbb{E}[Z]|^2\right] = \sigma_X^2 + \sigma_Y^2, \tag{4}$$

where σ_X^2 and σ_Y^2 are respectively the *variance* of X and Y. The latter does not bring information about the *covariance*:

$$\sigma_{XY} \triangleq \mathbb{E}\left[(X - \mathbb{E}[X])\,(Y - \mathbb{E}[Y])\right], \tag{5}$$

but this information can be retrieved thanks to the *pseudo-variance* (Ollila 2008; Picinbono 1996):

$$\tau_Z \triangleq \mathbb{E}\left[(Z - \mathbb{E}[Z])^2\right] = \sigma_X^2 - \sigma_Y^2 + 2\,j\,\sigma_{XY}. \tag{6}$$

The circularity quotient ϱ_Z is then: $\varrho_Z = \tau_Z/\sigma_Z^2$. If Z has a vanishing *pseudo-variance*, $\tau_Z = 0$, or equivalently, $\varrho_Z = 0$, it is said to be second-order circular. The correlation coefficient is defined as

$$\rho = \frac{\sigma_{XY}}{\sigma_X \sigma_Y}. \tag{7}$$

In this work, complex non-circular random datasets are generated and classified, with two non-exclusive possible sources of non-circularity: X and Y have unequal variances or X and Y are correlated.

3 Proposed Neural Networks

3.1 Library

Although it has been proven several times that CVNN achieves better generalization than RVNN (Hirose and Yoshida 2012), the latter has been often favored over the former due to difficulties in the implementation (Mönning and Manandhar 2018). Indeed, the two most popular Python libraries for developing deep neural networks, Pytorch and Tensorflow do not fully support the creation of complex-valued models. Therefore, one of the contributions of this work is to provide a library, published in conjunction with this paper (Barrachina 2019), that enables the full implementation of fully-connected feed-forward CVNNs using *Tensorflow* as back-end.

Even though *Tensorflow* does not fully support the implementation of a CVNN, it has one significant benefit: It enables the use of complex-valued data types for the automatic differentiation (autodiff) algorithm (Hoffmann 2016) to calculate the complex gradients. *Tensorflow*'s autodiff implementation calculates the gradient for $f : \mathbb{C} \to \mathbb{C}$ as:

$$\nabla_z f = \overline{\frac{\partial f}{\partial z} + \frac{\partial \overline{f}}{\partial z}}. \tag{8}$$

It is straightforward to prove that (2) and (8) are equivalent for a real-valued loss function $f : \mathbb{C} \to \mathbb{R}$ (Kreutz-Delgado 2009).

Libraries to develop CVNNs do exist, the most important one of them being probably the code published in (Trabelsi et al. 2017). However, we have decided not to use this library since the latter uses *Theano* as back-end, which is no longer maintained. Another code was published on GitHub that uses *Tensorflow* and *Keras* to implement CVNN (Dramsch and Contributors 2019). However, as *Keras* does not support complex-valued numbers, the published code simulates complex operations always using real-valued datatypes. Therefore, the user has to transform its complex-valued data to a real-valued equivalent before using this library.

The library used to generate the results of this paper was tested through several well known real-valued applications. Probably the most relevant test was the adaptation of *Tensorflow*'s online code example Training a neural network on MNIST with Keras since the model architecture is very similar to the one used in this paper. The network architecture has been coded with our library. Statistically, the results obtained with our network were equivalent to the one given

by *Tensorflow*'s example. They can be found in the *GitHub* repository inside the *examples* subfolder.

For more information about the library published in this paper, please refer to the documentation.

3.2 Model Architecture

Activation function One of the essential characteristics of CVNN is its activation functions, which should be non-linear and complex-valued. Additionally, the latter is desired to be piece-wise smooth to compute the gradient. The extension to complex field offers many possibilities to design an activation function. Among them, two main types are proposed by extending real-valued activation functions (Kuroe, Yoshid, and Mori 2003):

- Type-A: $g_A(f) = g_R\left(\mathrm{Re}(f)\right) + j\, g_I(\mathrm{Im}(f))$,
- Type-B: $g_B(f) = g_r(|f|)\, e^{j\, g_\phi(\phi(f))}$,

where $f : \mathbb{C} \to \mathbb{C}$ is a complex function and g_R, g_I, g_r, g_ϕ are all real-valued functions[1]. The most popular activation functions, sigmoid, hyperbolic tangent (tanh) and Rectified Linear Unit (ReLU), are extensible using Type-A or Type-B approach.

Normally, g_ϕ is left as a linear mapping (Kuroe, Yoshid, and Mori 2003; Hirose 2012). Under this condition, using ReLU activation function for σ_r has a limited interest since the latter makes g_B converge to a linear function, limiting Type-B ReLU usage. Nevertheless, ReLU has increased in popularity over the others as it has proved to learn several times faster than equivalents with saturating neurons (Krizhevsky, Sutskever, and Hinton 2012). Consequently, we will adopt, in Section 4, the Type-A ReLU, which is a non-linear function, as CVNN hidden layers activation functions.

The image domain of the output layer depends on the set of data labels. For classification tasks, real-valued integers or binary numbers are frequently used to label each class. Therefore the output layer's activation function should be real-valued. For this reason, we use *softmax* (normalized exponential) as the output activation function, which maps the magnitude of complex-valued input to $[0; 1]$, so the image domain is homogeneous to a probability.

Number of layers Even though the tendency is to make the models as deep as possible for Convolutional Neural Networks (CNN), this is not the case for fully-connected feed-forward neural networks, also known as MultiLayer Perceptron (MLP). For these models, one hidden layer (1HL) is usually sufficient for the vast majority of problems (Hornik, Stinchcombe, and White 1989; Stinchcombe and White 1989). Although some authors may argue that two hidden layers (2HL) may be better than one (Thomas et al. 2017), all authors seem to agree that there is currently no theoretical reason to use a MLP with more than two hidden layers (Heaton 2008, p. 158)

References (Heaton 2008) and (Kulkarni and Joshi 2015) recommend the neurons of the hidden layer to be between the size of the input layer and the output layer. Therefore, two models will be used as default in section 4, one with a single hidden layer of size 64 and one with two hidden layers of shape 100 and 40 for the first and second hidden layers respectively. In order to prevent the models from overfitting, dropout regularization technique (Srivastava et al. 2014) is used on 1HL and 2HL hidden layers. Both CVNN and RVNN are trained with a dropout factor of 0.5.

Equivalent RVNN To define an equivalent RVNN, the strategy used in (Hirose 2012) is adopted, separating the input z into two real values (x, y) where $z = x + j\, y$, giving the network a double amount of inputs. The same is done for the number of neurons in each hidden layer. Although this strategy keeps the same amount of features in hidden layers, it provides a higher capacity for the RVNN with respect to the number of real-valued training parameters (Mönning and Manandhar 2018).

Loss function and optimizer Mean square error and Cross-Entropy loss functions are mostly used for RVNN to solve regression and classification problems, respectively. The loss remains the same for CVNN since the training phase still requires the minimization over a real-valued loss function. We currently limit our optimizer to the well-known standard Stochastic Gradient Descent (SGD). The default learning rate used in this work is 0.01 as being *Tensorflow*'s default (v2.1) for its SGD implementation.

Weights initialization For weights initialization, Glorot uniform (also known as Xavier uniform) (Glorot and Bengio 2010) is used, and all biases start at zero as those are *Tensorflow*'s current (v2.1) default initialization methods for dense layers. Glorot initialization generates weight values according to the uniform distribution in (Glorot and Bengio 2010, eq.16) where its boundaries depend on both input and output sizes of the initialized layer. Note that CVNN has halved the number of input and output neurons compared to an equivalent RVNN, meaning the initial values of the real and the imaginary part of its weights are generated using a slightly larger uniform distribution boundary comparing to the real case. Therefore, CVNN might be penalized by starting with a higher initial loss value before any training.

4 Experimental Results

4.1 Dataset setup

As mentioned previously, to respect the equivalence between RVNN and CVNN, we use in the following input vectors of size 128 (resp. 256) for CVNNs (resp. RVNN). Each element of the feature vector is generated according to a non-circular Complex Normal distribution $\mathcal{CN}(0, \sigma_Z^2, \tau_Z)$. Two sources of non-circularity could occur in practice: $\sigma_X \neq \sigma_Y$ and/or $\rho \neq 0$, or equivalently $\tau_Z \neq 0$. Therefore, we propose to evaluate the classification performance of CVNN and RVNN for three types of datasets presented in Table 1.

Figure 1 shows an example of two classes for dataset A with a feature size of 128. It is possible to see that most features are coincident even if they are from different classes, meaning that several points will yield no information or even confuse the classification algorithm.

[1] Although not with the same notation, these two types of complex-valued activation functions are also discussed in Section 3.3 of (Hirose 2012)

	Data A		Data B		Data C	
Class	1	2	1	2	1	2
ρ	0.5	-0.5	0	0	0.5	-0.5
σ_X^2	1	1	1	2	1	2
σ_Y^2	1	1	2	1	2	1
τ_Z	j	$-j$	-1	1	$j-1$	$1-j$
ϱ_Z	$\frac{j}{2}$	$-\frac{j}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$	$\frac{j-1}{3}$	$\frac{1-j}{3}$

Table 1: Dataset characteristics

It is important to note that the distinction between classes is entirely contained in the relationship between the real and imaginary parts. This means that removing, for example, the imaginary part of the dataset will result in both classes being statistically identical, and therefore, rendering the classification impossible.

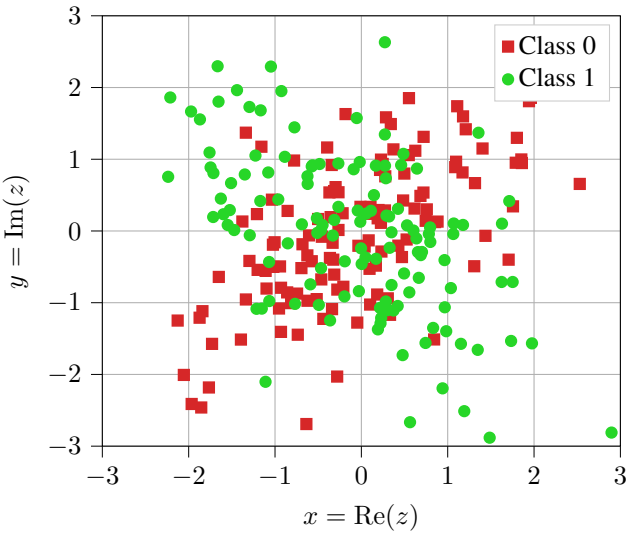

Figure 1: Dataset A example

To evaluate the difficulty of classifying this dataset, a Maximum Likelihood Estimation of τ_Z was implemented with the *prior* knowledge of the underlying Gaussian distributions used to generate the dataset. The data are then classified using a threshold on the estimate of τ_Z. The accuracy of this classifier gives an upper bound of the optimal accuracy. For a low correlation coefficient, for example $\rho = 0.1$, this parametric classifier only achieves around 85% accuracy.

4.2 1HL and 2HL baseline results

To ensure that the models do not fall short of data, 10000 samples of each class were generated using 80% for the train set and the remaining 20% for testing. Accuracy and loss of both CVNN and RVNN, defined previously in section 3.2, are statistically evaluated over 1000 Monte-Carlo trials. Each trial contained 150 epochs with a batch size of 100. This number was chosen observing that after 150 epochs, the accuracy and loss presented almost no amelioration.

Only the 2HL case is illustrated in Fig. 2 as their results are more favorable to the RVNN model. CVNN loss starts higher but decreases faster than RVNN. Both losses behave well without significant indication of overfitting. Additionally, the test accuracy of CVNN trials stays above 95%. The test accuracy of RVNN is lower than the CVNN one and presents more outliers.

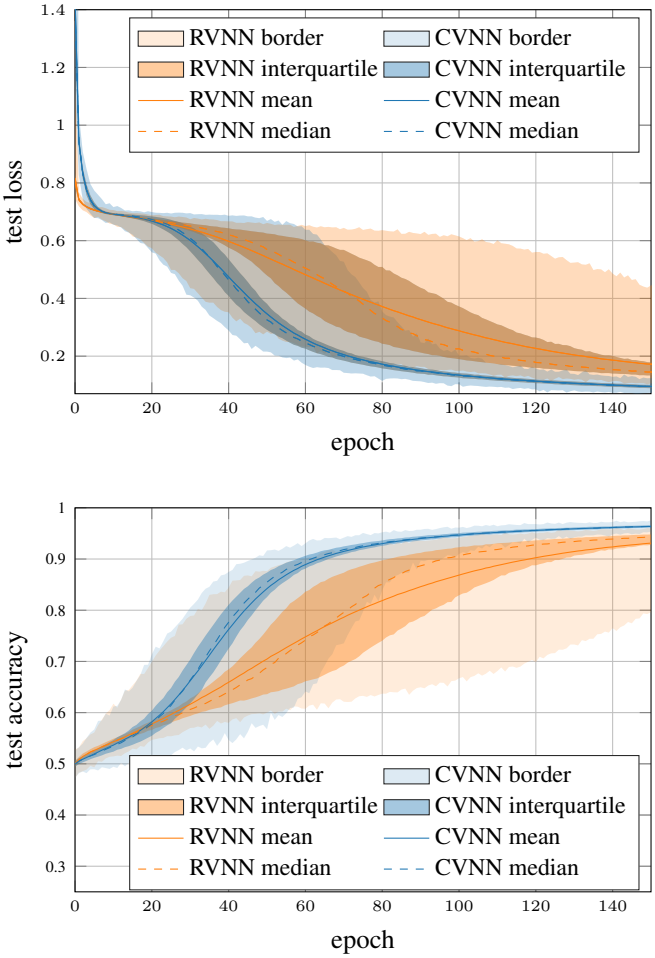

Figure 2: Test loss and accuracy for 2HL CVNN and RVNN with a dropout of 50%

Table 2 summarizes the test accuracy of 1HL and 2HL models for all three different datasets. The median error is computed as $1.57\,\mathrm{IQR}/\sqrt{n}$ (McGill, Tukey, and Larsen 1978), where IQR is the Inter-Quartile Range, and n is the number of trials. According to (Chambers 2018), using this error definition, if median values do not overlap there is a 95% confidence that their values differ.

Because the results are skewed, there is a big difference between mean and median accuracy, as the mean is less robust to outliers. For this reason, the median would be a better measure of central tendency in this paper's simulations. For the complex-valued model, the outliers tend to be the bad

		Data A		Data B		Data C	
		CVNN	RVNN	CVNN	RVNN	CVNN	RVNN
1HL	median	95.00 ± 0.03	75.58 ± 0.69	84.38 ± 0.05	58.73 ± 0.07	96.95 ± 0.02	82.90 ± 0.78
	mean	94.98 ± 0.02	76.71 ± 0.27	84.28 ± 0.03	58.89 ± 0.05	96.96 ± 0.01	82.76 ± 0.26
	IQR	$94.73 - 95.23$	$69.20 - 82.93$	$83.83 - 84.83$	$58.10 - 59.48$	$96.78 - 97.18$	$75.49 - 91.08$
	full range	$93.60 - 96.08$	$64.05 - 93.15$	$76.15 - 86.60$	$55.68 - 59.48$	$96.00 - 97.90$	$67.35 - 95.88$
2HL	median	97.03 ± 0.03	92.90 ± 0.08	69.90 ± 0.88	59.03 ± 0.33	98.43 ± 0.02	96.10 ± 0.04
	mean	96.98 ± 0.02	92.37 ± 0.07	69.48 ± 0.32	59.10 ± 0.14	98.41 ± 0.01	96.02 ± 0.01
	IQR	$96.78 - 97.23$	$92.02 - 93.48$	$60.89 - 78.43$	$55.80 - 62.35$	$98.28 - 98.58$	$95.75 - 96.40$
	full range	$95.23 - 98.05$	$68.78 - 94.78$	$50.03 - 87.08$	$49.98 - 71.23$	$97.23 - 99.03$	$90.38 - 97.18$

Table 2: Test accuracy results (%)

cases, whereas, for the real model, they are the good cases. This can be verified by the mean being lower than the median for CVNN and higher than the median for RVNN.

For dataset B with 2HL, both models fail to achieve excellent results on average. Despite these poor performances, CVNN still proves to be superior to RVNN by far. For 1HL, CVNN achieves very high accuracy with a median of over 84%. Dataset C presents almost the same results as dataset A with some improvement for both architectures.

From these results, the merits of CVNNs are statistically justified by a higher accuracy than RVNNs with less overfitting and smaller variance.

In general, RVNN performed much better with 2HL than with 1HL. In this report we are trying to prove that CVNN outperforms RVNN regardless of the model architecture and hyper-parameters. To that end, even if all simulations to follow were done with both 1HL and 2HL, 2HL results will be prioritized, bearing in mind that 1HL cases were even more favorable to CVNN.

4.3 Case without dropout

The simulations were re-done with no dropout to test the models tendency to overfit. RVNN presented very high overfitting, as can be seen in figure 3.

The impact was so high that the test accuracy of RVNN drop from 92% to 66% for 2HL, whereas for CVNN it dropped less than 2% reaching a test accuracy of 96% for the case without dropout. The totality of the simulations mentioned in this report were also done without dropout. In general, dropout had very little impact on CVNNs performance and a huge amelioration for RVNN. In conclusion, CVNN has a very low tendency to over-fit, whereas for RVNN is decisive to use dropout or at least a regularization technique.

4.4 Phase and amplitude

Since the phase information could be relevant for classifying these datasets, polar-RVNN, is defined where the inputs are the amplitude and phase of data. This method is tested for datasets A and B with and without dropout.

Figure 4 shows the results for 2HL tested on dataset A with dropout. It can be seen that polar-RVNN highly improves compared to the conventional RVNN, showing higher mean accuracy but also much less variance, even

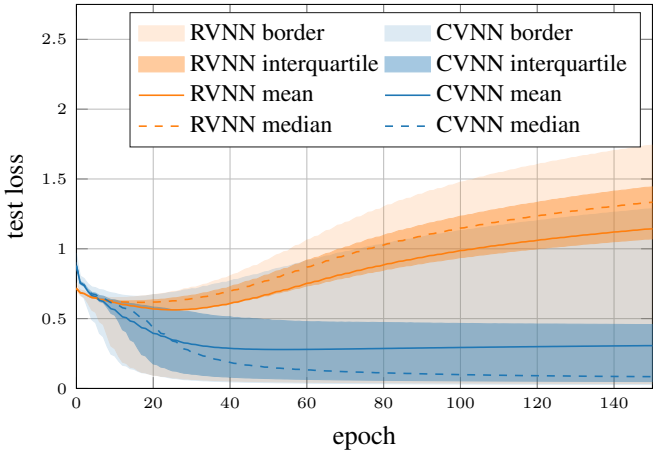

Figure 3: Test loss for 2HL CVNN and RVNN without dropout

lower than for CVNN. However, CVNN still outperforms both real models by a wide margin.

This higher performance of polar-RVNN against RVNN can be explained by the fact that dataset type A presents more relevant information in the phase. However, the opposite happens with dataset type B for which case, polar-RVNN completely fails to converge and achieves worst results than conventional RVNN for both 1HL and 2HL models, reason why results where omitted.

4.5 Parameter sensibility study

In this section, a swipe through several model architectures and hyper-parameters is done to assert that the results obtained are independent of specific parameters. These simulations are done for both 1HL and 2HL networks.

Other sensibility studies were done but are not presented in this work due to paper size limitations. They concern learning rate, dataset size, feature vector size and multiclasses for all combinations of 1HL and 2HL with and without dropout.

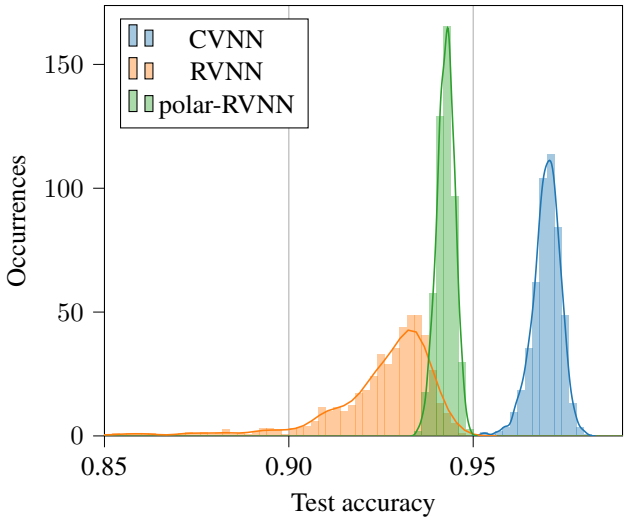

Figure 4: Test accuracy histogram of 2HL CVNN, polar-RVNN and RVNN on dataset A

Correlation coefficient Several correlation coefficients have been tested for 1HL and 2HL models. Figure 5 shows the accuracy of 2HL models tested on datasets similar to data A (see table 1), in which the correlation coefficient varies from 0.1 to 0.8. As expected, for small ρ, both networks fail to distinguish between classes with accuracy values barely above 50%. Note that both models cannot achieve more than 85% for this case, as it was explained in section 4.1. As $|\rho|$ rises, CVNN merits become evident. When $|\rho|$ is close to one, then the link between real and imaginary parts is strengthened, which facilitates the classification of the data for both models. Results for 1HL are even more favorable for CVNN.

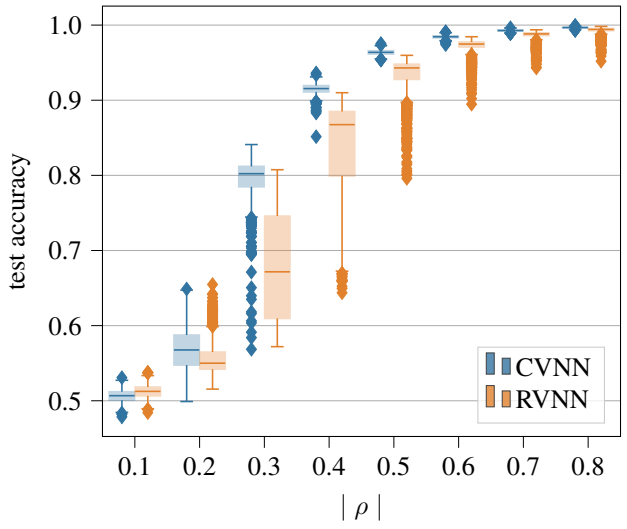

Figure 5: Test accuracy box plot for different values of correlation coefficient ρ for 2HL model with dropout

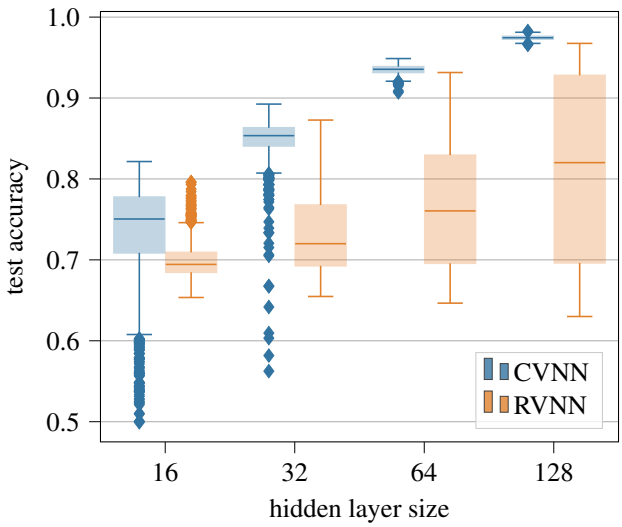

Figure 6: Test accuracy box plot for different 1HL size with dropout

Hidden layer size We evaluated the accuracy of 1HL for 4 sizes of the hidden layer. All these models were trained on dataset A. Figure 6 shows that the median accuracy of CVNNs is always higher than the one of RVNN, no matter the number of hidden neurons. However, we can note that CVNN has low accuracy outliers for sizes 16 and 32, whereas RVNN has not. This could be explained by RVNN having higher capacity, as explained in section 3.2. Fortunately, this behavior disappears when the hidden size is well dimensioned.

5 Conclusion

In this paper, we provided a end-to-end tool for the implementation of CVNNs to fully use the complex-valued characteristics of the data. It also allows a more straightforward comparison with real equivalent networks in order to motivate further analysis and use of CVNN (Barrachina 2019). Moreover we showed that CVNNs stand as attractive networks to obtain higher performances than conventional RVNNs on complex-valued datasets. The latter point was illustrated by several examples of non-circular complex-valued data which cover a large amount of data types that can be encountered in signal processing and radar fields. All statistical indicators showed that CVNN clearly outperforms RVNN showing higher accuracy, smaller variance and less overfitting, regardless the model architecture and hyperparameters. Conversely, the few cases where RVNN competes with CVNN, occurred when the dataset is small or the correlation coefficient $|\rho|$ is close to zero, rendering the discrimination from feature vectors nearly impossible. For these exceptions, neither CVNN and RVNN were actually of any practical use. Further analysis involving sensitivities to learning rates, feature vector size, multi-class dataset will be investigated to assert the generalisation of our results.

References

Amin, M. F.; Amin, M. I.; Al-Nuaimi, A. Y. H.; and Murase, K. 2011. Wirtinger Calculus Based Gradient Descent and Levenberg-Marquardt Learning Algorithms in Complex-Valued Neural Networks. In *Neural Information Processing*, 550–559. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-24955-6.

Barrachina, J. A. 2019. Complex-Valued Neural Networks (CVNN). https://github.com/NEGU93/cvnn.

Chambers, J. M. 2018. *Graphical methods for data analysis*. CRC Press.

Dramsch, J. S.; and Contributors. 2019. Complex-Valued Neural Networks in Keras with Tensorflow. doi:10.6084/m9.figshare.9783773. URL https://figshare.com/articles/Complex-Valued_Neural_Networks_in_Keras_with_Tensorflow/9783773/1.

Fine, B.; and Rosenberger, G. 1997. *The Fundamental Theorem of Algebra*. Springer Science & Business Media.

Glorot, X.; and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 249–256.

Hänsch, R.; and Hellwich, O. 2009. Classification of polarimetric SAR data by complex valued neural networks. In *Proc. ISPRS Hannover Workshop, High-Resolution Earth Imag. Geospatial Inf.*, volume 37.

Heaton, J. 2008. *Introduction to neural networks with Java*. Heaton Research, Inc.

Hirose, A. 2009. Complex-valued neural networks: The merits and their origins. In *2009 International joint conference on neural networks*, 1237–1244. IEEE.

Hirose, A. 2012. *Complex-valued neural networks*, volume 400. Springer Science & Business Media.

Hirose, A. 2013. *Complex-valued neural networks: Advances and applications*, volume 18. John Wiley & Sons.

Hirose, A.; Amin, F.; and Murase, K. 2013. Learning Algorithms in Complex-Valued Neural Networks using Wirtinger Calculus. In *Complex-Valued Neural Networks: Advances and Applications*, 75–102. IEEE. URL https://ieeexplore.ieee.org/document/6515223.

Hirose, A.; and Yoshida, S. 2012. 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.

Hoffmann, P. H. W. 2016. A hitchhiker's guide to automatic differentiation. *Numerical Algorithms* 72(3): 775–811.

Hornik, K.; Stinchcombe, M.; and White, H. 1989. Multilayer Feedforward Networks Are Universal Approximators. *Neural Networks* 2(5): 359–366. ISSN 0893-6080. doi:10.1016/0893-6080(89)90020-8.

Kreutz-Delgado, K. 2009. The complex gradient operator and the CR-calculus. *arXiv preprint arXiv:0906.4835* .

Krizhevsky, A.; Sutskever, I.; and Hinton, G. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Kulkarni, P.; and Joshi, P. 2015. *Artificial intelligence: building intelligent systems*. PHI Learning Pvt. Ltd.

Kuroe, Y.; Yoshid, M.; and Mori, T. 2003. On activation functions for complex-valued neural networks: existence of energy functions. In *Artificial Neural Networks and Neural Information Processing, ICANN/ICONIP 2003*, 985–992. Springer.

McGill, R.; Tukey, J. W.; and Larsen, W. A. 1978. Variations of box plots. *The American Statistician* 32(1): 12–16.

Mönning, N.; and Manandhar, S. 2018. Evaluation of Complex-Valued Neural Networks on Real-Valued Classification Tasks. *arXiv preprint arXiv:1811.12351* .

Ollila, E. 2008. On the circularity of a complex random variable. *IEEE Signal Processing Letters* 15: 841–844.

Picinbono, B. 1996. Second-order complex random vectors and Normal distributions. *IEEE Transactions on Signal Processing* 44(10): 2637–2640. ISSN 1941-0476. doi:10.1109/78.539051.

Schreier, P. J.; and Scharf, L. L. 2010. *Statistical Signal Processing of Complex-Valued Data*. Cambridge University Press.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The journal of machine learning research* 15: 1929–1958. ISSN 1532-4435.

Stinchcombe, M.; and White, H. 1989. Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. In *IJCNN International Joint Conference on Neural Networks*.

Thomas, A. J.; Petridis, M.; Walters, S. D.; Gheytassi, S. M.; and Morgan, R. E. 2017. Two Hidden Layers Are Usually Better than One. In *Engineering Applications of Neural Networks*, Communications in Computer and Information Science, 279–290. Cham: Springer International Publishing. ISBN 978-3-319-65172-9. doi:10.1007/978-3-319-65172-9_24.

Trabelsi, C.; Bilaniuk, O.; Zhang, Y.; Serdyuk, D.; Subramanian, S.; Santos, J. F.; Mehri, S.; Negar, R.; Bengio, Y.; and Pal, C. J. 2017. Deep complex networks. *arXiv preprint arXiv:1705.09792* .

Wirtinger, W. 1927. Zur formalen Theorie der Funktionen von mehr komplexen Veränderlichen. *Mathematische Annalen* 97(1): 357–375. ISSN 1432-1807. doi:10.1007/BF01447872. URL https://doi.org/10.1007/BF01447872.

Zhao, J.; Datcu, M.; Zhang, Z.; Xiong, H.; and Yu, W. 2019. Contrastive-Regulated CNN in the Complex Domain: A Method to Learn Physical Scattering Signatures From Flexible PolSAR Images. *IEEE Transactions on Geoscience and Remote Sensing* 57(12): 10116–10135.