mathematics

MDPI

Article

Generalization of Neural Networks on Second-Order Hypercomplex Numbers

Stanislav Pavlov [1,2,3,†], Dmitry Kozlov [1,*,†], Mikhail Bakulin [1,†], Aleksandr Zuev [1], Andrey Latyshev [1] and Alexander Beliaev [1]

[1] NN AI Team, Huawei Russian Research Institute, St. Maksima Gorkogo, 117, Nizhny Novgorod 603006, Russia
[2] Department of Informatics, Mathematics and Computer Sciences, National Research University Higher School of Economics, St. Bolshaya Pecherskaya, 25/12, Nizhny Novgorod 603155, Russia
[3] Department of Informatics and Telecommunications, Volga State University of Water Transport, St. Nesterova, 5, Nizhny Novgorod 603005, Russia
* Correspondence: kozlov.dmitry@huawei.com
† These authors contributed equally to this work.

Abstract: The vast majority of existing neural networks operate by rules set within the algebra of real numbers. However, as theoretical understanding of the fundamentals of neural networks and their practical applications grow stronger, new problems arise, which require going beyond such algebra. Various tasks come to light when the original data naturally have complex-valued formats. This situation is encouraging researchers to explore whether neural networks based on complex numbers can provide benefits over the ones limited to real numbers. Multiple recent works have been dedicated to developing the architecture and building blocks of complex-valued neural networks. In this paper, we generalize models by considering other types of hypercomplex numbers of the second order: dual and double numbers. We developed basic operators for these algebras, such as convolution, activation functions, and batch normalization, and rebuilt several real-valued networks to use them with these new algebras. We developed a general methodology for dual and double-valued gradient calculations based on Wirtinger derivatives for complex-valued functions. For classical computer vision (CIFAR-10, CIFAR-100, SVHN) and signal processing (G2Net, MusicNet) classification problems, our benchmarks show that the transition to the hypercomplex domain can be helpful in reaching higher values of metrics, compared to the original real-valued models.

Keywords: deep learning; hypercomplex neural networks; complex numbers; dual numbers; double numbers; hypercomplex norm; hypercomplex batch normalization

MSC: 68T07

Citation: Pavlov, S.; Kozlov, D.; Bakulin, M.; Zuev, A.; Latyshev, A.; Beliaev, A. Generalization of Neural Networks on Second-Order Hypercomplex Numbers. *Mathematics* **2023**, *11*, 3973. https://doi.org/10.3390/math11183973

Academic Editors: Miguel Atencia and Daniela Danciu

Received: 30 August 2023
Revised: 13 September 2023
Accepted: 14 September 2023
Published: 19 September 2023

1. Introduction

The exploration of neural networks in complex algebra is a modern area of research. Many tasks in classification, signal pattern recognition, and the generation of new signals inherently present data in a complex format. Recent studies have shown that complex-valued neural networks outclass real-valued ones in terms of representational capacity [1] and learning speed [2]. Other publications aim to develop complex-valued layers and activation functions for neural networks (see the survey in [3]). The progress allows researchers to build effective models with better convergence and accuracy [4,5].

These promising results have inspired us to further advance the generalization of neural networks, extending to the dual $\mathbb{D}$ and double or complex-split $\mathbb{S}$ algebras. The basic mathematical operations for these numbers are as follows:

$$(x_1 + \tau y_1) \pm (x_2 + \tau y_2) = (x_1 \pm x_2) + \tau(y_1 \pm y_2)$$
$$(x_1 + \tau y_1) \cdot (x_2 + \tau y_2) = (x_1 x_2 + \sigma y_1 y_2) + \tau(x_1 y_2 + y_1 x_2)$$
$$\frac{x_1 + \tau y_1}{x_2 + \tau y_2} = \frac{x_1 x_2 - \sigma y_1 y_2}{x_2^2 - \sigma y_2^2} + \tau \frac{x_2 y_1 - x_1 y_2}{x_2^2 - \sigma y_2^2}$$
$$(x + \tau y)^* = x - \tau y$$

In addition to complex numbers, dual and double numbers have found applications in physics, e.g., screw theory [6] or relativistic cosmology [7]. Dual numbers also make it possible to automatically compute derivatives of functions [8–10]. Double-valued networks have been used for solving CV tasks [11]. Therefore, using hypercomplex numbers for deep learning is well justified. There have been studies on neural networks based on Clifford algebra [12], specifically on dual numbers [13], and room for research still remains. The goal of this paper is to generalize the approach of using second-order hypercomplex algebras for neural networks.

There is another reason for our interest in dual and double numbers. As real-valued layers expand to the hypercomplex algebras, the number of operations increases. The magnitude of this growth depends on the computational complexity of basic operations on the numbers. For example, one multiplication of two complex numbers is more expensive than four multiplications of real numbers. This influences the computational cost of the operators discussed in Section 2. Thus, the convolution of a complex-valued input $z = x + iy$ with a complex-valued filter matrix $W = A + iB$ can be rewritten as $Wz = (Ax - By) + i(Ay + Bx)$, which implies four real-valued matrix multiplications (see Figure 1). The dual-valued convolution of a dual filter $W = A + \varepsilon B$ and a dual input $z = x + \varepsilon y$ evaluates $Wz = Ax + \varepsilon(Ay + Bx)$ because $\varepsilon^2 = 0$. This type of convolution only has three real-valued matrix multiplications (see Figure 1), which gains a +25% theoretical performance speed-up, compared to the complex-valued convolution, and interconnects x and y in the process. As for double numbers, the cost of the naive implementation of the convolution is the same as for complex numbers because $(A + jB) \cdot (x + jy) = (Ax + By) + j(Bx + Ay)$. However, the features of double numbers enable variable substitution that allows changing to a so-called diagonal basis. Under that new setup, a double-valued convolution is effectively only two real-valued convolutions instead of four. Section 3 of this paper is dedicated to the optimizations of hypercomplex networks, including the diagonal representation of double numbers (Section 3.3). In this paper, we will mainly use the original representation of double numbers, for the sake of following the universal convention.

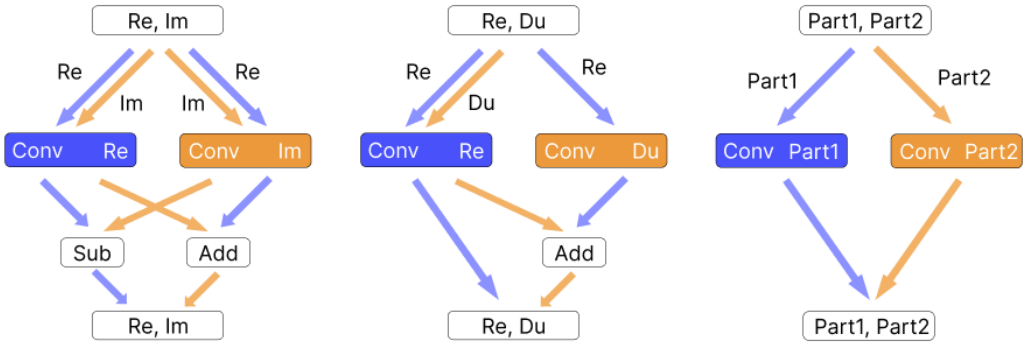

Figure 1. Computational complexity of convolution in the complex, dual, and double algebras.

It is worth mentioning that double numbers in the diagonal representation have a significant flaw: the inability to connect x and y. However, the influence of this problem can be mitigated by a special batch normalization operator and/or an activation function for double numbers, which would make them connect. This adjustment is very important. As the experiments with real-valued networks have shown [14], an appropriate batch

normalization helps accelerate the training process and reach better accuracy. We will further discuss the components of hypercomplex-valued neural networks in detail.

In Section 4 of this paper, we show that the adaptation of neural networks to the algebra of hypercomplex numbers allows us to improve precision. The important conclusion here is that the proposed types of neural networks, being barely explored so far, hold great potential for further research.

2. Methodology

This section is dedicated to the definitions of key elements of a hypercomplex neural network.

2.1. MindSpore Hypercomplex Data Representation

Until recently, only complex operators have been supported in popular frameworks, such as TensorFlow and PyTorch. The list of supported operations only contains basic manipulations with complex-valued tensors, as well as convolutional and linear layers. As of now, the situation has changed dramatically. MindSpore has become the first platform to support all the core layers needed to build second-order hypercomplex neural networks.

MindSpore allows researchers to create complex, dual, and double tensors, and perform all basic operations, such as slicing, concatenation, conjugation, and accessing both components of tensors. Convolution and linear layers are supported by MindSpore for all types of hypercomplex algebras of the second order. Other essential operators for deep learning, such as activation functions and algebra-specific batch normalization, are also integrated into MindSpore.

Within this framework, the hypercomplex tensor is represented as a two-channel real-valued tensor. Tensors of this shape are used as arguments and all operators return values in the same format. The peculiarities of every type of algebra are considered by a specific realization of network operators inside MindSpore, which interlinks the two channels of the tensors.

2.2. Hypercomplex Operators

The implementation of the operators mentioned above is based on our proposed method of generalizing the process of constructing building blocks for hypercomplex neural networks.

2.2.1. Convolution

Convolution is one of the most important operations in neural networks. In order to explain convolution in the hypercomplex algebra, we exploit the matrix representation of second-order hypercomplex numbers that use real numbers. The algebras of complex, dual, and double numbers $u = x + \tau y$ are all known to be isomorphic to the corresponding algebras of the second-order real matrices of the following form: $\begin{pmatrix} x & y \\ \sigma y & x \end{pmatrix}$. Recall that $\tau^2 = \sigma$. Therefore, the convolution of a hypercomplex filter $W = W_x + \tau W_y$ and a hypercomplex input $u = x + \tau y$ can be expressed as follows:

$$W * u \sim \begin{pmatrix} W_x & W_y \\ \sigma W_y & W_x \end{pmatrix} * \begin{pmatrix} x & y \\ \sigma y & x \end{pmatrix} = \begin{pmatrix} W_x * x + \sigma W_y * y & W_x * y + W_y * x \\ \sigma(W_x * y + W_y * x) & W_x * x + \sigma W_y * y \end{pmatrix}.$$

As mentioned earlier, we need to make three real-valued multiplications in the dual algebra ($W_x * x$, $W_y * x$, and $W_x * y$), but four in the cases of complex and double numbers ($W_x * x$, $W_y * x$, $W_x * y$, and $W_y * x$). For double numbers, however, changing variables, as shown in Section 3.3, enables moving to the diagonal representation, where the algebra becomes isomorphic to the algebra of real matrices of the following form: $\frac{1}{2}\begin{pmatrix} x+y & x-y \\ x-y & x+y \end{pmatrix}$, and the convolution only requires two convolutions in real numbers.

2.2.2. Linear Layer

Linear (fully connected, dense) layers are commonly used in neural networks as the head blocks in the CV (computer vision) domain and as the main operators in NLP (natural language processing) models. To generalize linear layers for the two-dimensional algebras, we use the matrix representation of second-order hypercomplex numbers. In the end, a linear layer with hypercomplex inputs and weights is equivalent to the superposition of real-valued linear layers:

$$\mathbb{H}L(W, B, u) = \mathbb{R}L(W_x, B_x, u_x) + \tau^2\mathbb{R}L(W_y, 0, u_y) + \tau(\mathbb{R}L(W_x, 0, u_y) + \mathbb{R}L(W_y, B_y, u_x)),$$

where $\mathbb{R}L(w, b, x)$ stands for a real-valued linear layer, with w, b, and x denoting weights, bias, and the input, respectively.

2.2.3. Average Pooling

The average pooling operation implies calculating the arithmetic mean for each patch of the feature map. This means collapsing every $n \times n$ square of the feature map into its average value. This is equivalent to convolution, with the stride being equal to the kernel size, where kernel weights are real numbers that are equal to $\frac{1}{n^2}$. Because the kernel $W = W_x + \tau W_y$ in this particular case is purely real, $W_y = 0$, and the convolution formula can be simplified as follows:

$$W * u = W * (x + \tau y) = W_x * (x + \tau y) = W_x * x + \tau(W_x * y).$$

From this expression, average pooling is equivalent to two real-valued average pooling operations, with each independently applied to every component of the input data $u = x + \tau y$:

$$\mathbb{H}AvgPool(u) = \mathbb{R}AvgPool(x) + \tau\mathbb{R}AvgPool(y).$$

2.2.4. ReLU

Activation functions are used to introduce non-linearity into neural networks. There are multiple activation functions based on real numbers, and greater varieties are based on hypercomplex numbers. Among the real-valued activations, there is a family of *ReLU*-type functions, which do not suffer from the vanishing gradient problem. This property makes them quite stable. Functions of this type are used in the complex algebra as well. For example, the authors of [15] applied *ReLU* to the real and imaginary parts separately. In this paper, we extend this definition and apply it to other algebras:

$$\mathbb{H}ReLU(u) = \mathbb{R}ReLU(x) + \tau\mathbb{R}ReLU(y).$$

This activation approach is not perfect. First, in contrast to $\mathbb{R}ReLU$ on real numbers, $\mathbb{H}ReLU(u)$ may not be equal to 0 or u. Second, the real and imaginary parts are processed independently and, therefore, are not fully interlinked. Despite that, the method shows good experimental results, and its obvious advantage is simplicity.

2.3. Norm of Hypercomplex Numbers

In deep learning, the ability to properly update model parameters is important for training a model with better accuracy on a test dataset. The weights may not be efficiently updated if elements of intermediate tensors are not normalized due to the gradient explosion or vanishing problem. In real-valued models, the solution is the batch normalization procedure. The idea is to balance tensor elements based on the variance of the batch. To develop a counterpart operator for neural networks based on second-order algebras, we must find an equivalent of the absolute value function. Here, we define a concept of

the norm (or modulus) of hypercomplex numbers. In mathematical or physical literature, the modulus of a complex number is traditionally defined as

$$\mathbb{C}|z| = \sqrt{zz^*} = \sqrt{(x+iy)(x-iy)} = \sqrt{x^2+y^2}. \tag{1}$$

However, the generalization of this method to other types of hypercomplex numbers does not work quite well:

$$\mathbb{H}|u| = \sqrt{uu^*} = \sqrt{(x+\tau y)(x-\tau y)} = \sqrt{x^2-\sigma y^2} = \begin{cases} |x|, \ u \in \mathbb{D} \\ \sqrt{x^2-y^2}, \ u \in \mathbb{S} \end{cases}$$

This result is hardly applicable to our purpose. The dual norm does not depend on the dual part. In the case of double numbers, the function is not well-defined for half of the elements. In [16], the authors derive the norms of dual numbers that are free from the mentioned downsides. Here, we extend the idea to develop the formula for all hypercomplex numbers of the second order, and make sure that it replicates the expression of the standard norm of complex numbers (1). The approach is based on the matrix representation $A = \begin{pmatrix} x & y \\ \sigma y & x \end{pmatrix}$ of a hypercomplex number $u = x + \tau y$. We derive the norm of this matrix and associate it with the norm of a hypercomplex number. For the matrix norm, we use the following definition:

$$\|A\|^2 = \sup\left\{ \|At\|_2^2 : t \in \mathbb{R}^{2\times 1}, \|t\|_2^2 = 1 \right\}$$

After some straightforward mathematical manipulations (see Appendix A), we obtain the following formula for the hypercomplex number norm:

$$\mathbb{H}\|u\| = \begin{cases} \sqrt{x^2+y^2}, \ u \in \mathbb{C} \\ \left|\frac{y}{2}\right| + \sqrt{x^2 + \left(\frac{y}{2}\right)^2}, \ u \in \mathbb{D}. \\ |x| + |y|, \ u \in \mathbb{S} \end{cases} \tag{2}$$

To show the differences among the proposed norms for the hypercomplex algebras, we depict $\|u\|$ equal to 2, 4, or 6 with contour lines in Figure 2. It is shown that the contour lines of dual and double norms are closer to the center than the corresponding contour lines for complex numbers. Different norm expressions have different effects on batch normalization for the dual, complex, and double algebras.

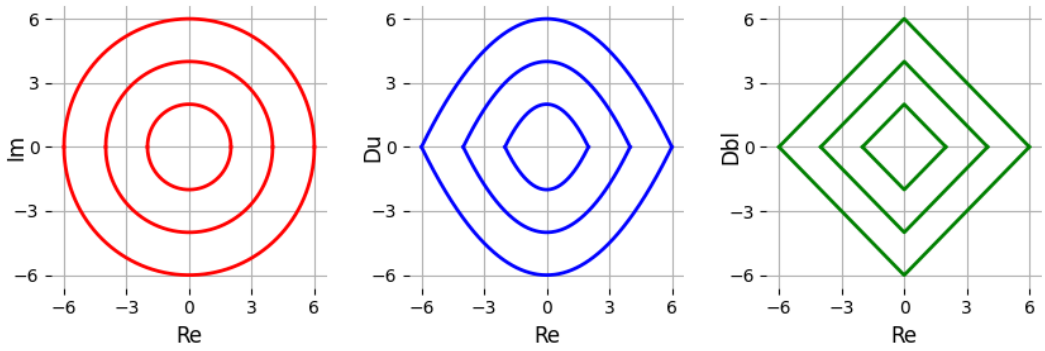

Figure 2. Illustration of the contour lines corresponding to the constant values of the complex, dual, and double number norms.

2.4. Batch Normalization

As mentioned in the previous section, batch normalization is very important for a successful training process and convergence to high-metric values. In its original form [14],

it can only be applied to the real-valued models. This approach includes the normalization of each dimension of the k-dimensional input:

$$\check{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}.$$

After this transformation, the mean of $\check{x}^{(k)}$ is equal to zero, and the standard variance is equal to one. However, it is better to make the mean and variance trainable parameters. In practice, it is achieved by performing additional scaling and shifting:

$$\mathbb{R}BN[\check{x}^{(k)}] = \check{\gamma}^{(k)}\check{x}^{(k)} + \check{\beta}^{(k)}.$$

The technique of batch normalization has been extended to complex inputs [15]. The authors suggested transforming the inputs using a covariance matrix so that the real and imaginary parts do not correlate. However, we cannot generalize this method explicitly; this is because—for dual numbers—the pseudo-covariance C cannot be equal to zero:

$$\begin{aligned}\mathbb{D}C = E[(z-\mu)^2] = E[((z_x - \mu_x) + \varepsilon(z_y - \mu_y))^2] = \\ = E[(z_x - \mu_x)^2] + \varepsilon(2 \cdot E[(z_x - \mu_x)(z_y - \mu_y)]),\end{aligned}$$

with at least the real parts always being greater than zero. Therefore, we need to find an alternative way to generalize batch normalization to all types of hypercomplex algebras of the second order. In this paper, we present a method based on the norm concept of hypercomplex numbers, as derived in the previous section, and compare the proposed method for complex numbers with the idea presented in [15].

First, we define the mean value in the traditional channel-wise way, which is the same for all algebras:

$$\mu^{(k)} = E[u^{(k)}] = \mu_x^{(k)} + \tau\mu_y^{(k)}.$$

Then we define the variance that is specific to the type of hypercomplex algebra:

$$Var[u^{(k)}] = \frac{1}{m-1}\sum_{i=1}^{m} \|u_i^{(k)} - \mu^{(k)}\|^2.$$

where $\|\cdot\|$ is determined by (2). Then we transform the input as follows:

$$\hat{u}^{(k)} = \frac{u^{(k)} - \mu^{(k)}}{\sqrt{Var[u^{(k)}] + \delta}},$$

where δ is a small number needed to avoid division by zero. The last step is hypercomplex channel-wise scaling and shifting:

$$\mathbb{H}BN[\hat{u}^{(k)}] = \hat{\gamma}^{(k)}\hat{u}^{(k)} + \hat{\beta}^{(k)}, \tag{3}$$

where $\hat{\gamma}^{(k)}$ and $\hat{\beta}^{(k)}$ are the hypercomplex weights and bias, accordingly.

In the specific case of complex numbers:

$$\mathbb{C}Var[z] = E[(z-\mu)(z-\mu)^*] = E[(x-\mu_x)^2 + (y-\mu_y)^2] = \Gamma(z).$$

So, dividing the centered input by $\sqrt{\mathbb{C}Var[z]}$ provides the covariance $\Gamma(\hat{z}) = 1$. Note that, in general, pseudo-covariance is not equal to zero:

$$C(\hat{z}) = E[\hat{z}^2] = E[(\hat{x} + i\hat{y})^2] = E[\hat{x}^2 - \hat{y}^2] + i(2 \cdot E[\hat{x}\hat{y}]) \neq 0.$$

Since $\hat{x}$ and $\hat{y}$ may correlate, $E[\hat{x}\hat{y}] \neq 0$. Despite that, as shown in Section 4, using this method results in good metric values. We emphasize that MindSpore is the only framework that has implemented this mathematically justified batch normalization technique.

We conduct an ablation study of the proposed batch normalization technique. For this purpose, we construct a hypercomplex LeNet-5 model and vary the normalization. The following approaches are considered: real-valued channel-wise normalization, covariance-based normalization from [15], and algebra-aware normalization (3) based on the proposed hypercomplex norm (2).

We show that all normalization types lead to about the same results on the MNIST dataset after 20 epochs (see Table 1). Nevertheless, the proposed hypercomplex normalization is more beneficial because it can be further fused with a previous convolutional layer (Section 3.4).

Table 1. Ablation study of normalization techniques.

Type of BN	Algebra		
	Complex	**Dual**	**Double**
Covariance	98.12	96.88	98.02
Channel-wise	98.53	98.25	98.40
Complex	98.50	98.38	98.38
Dual	98.35	98.23	98.28
Double	98.26	98.26	98.20

2.5. Backward Propagation of Loss Function Gradient

An important part of neural network training is gradient calculation. Computing the loss function gradient relies on the backpropagation algorithm, which exploits the chain rule. In this paper, we explore the gradient propagation problem in the algebra of hypercomplex numbers. The classic definition of the derivative of a function $f(u) = f(x + \tau y) = v(x, y) + \tau w(x, y)$ of a hypercomplex argument $u = x + \tau y$, where v and w are real-valued functions, is

$$f'(u) = \lim_{\Delta u \to 0} \frac{f(u + \Delta u) - f(u)}{\Delta u}.$$

This limit can only exist if it is well-defined when Δu approaches zero along the real axis ($\Delta u = \Delta x$) or the imaginary axis ($\Delta u = \tau \Delta y$). In either case, it should provide the same result. Equating these special cases, we obtain the general equivalents of Cauchy–Riemann equations:

$$\frac{\partial v}{\partial x} = \frac{\partial w}{\partial y}, \qquad \frac{\partial v}{\partial y} = \tau^2 \frac{\partial w}{\partial x} = \sigma \frac{\partial w}{\partial x}.$$

Functions that satisfy these equations are called holomorphic. In practice, this is a strong restriction, and most of the existing operators do not satisfy the Cauchy–Riemann criteria. To overcome this, we use the approach invented by Wirtinger for complex numbers [17]. It applies variable substitution in order to rewrite a complex-variable function $f(z)$ as a holomorphic function of two arguments $f(z, z^*)$. This idea was applied in [18] to derive the gradient of double numbers. Here, we extend this approach to all second-order algebras:

$$x = \frac{u + u^*}{2}, \; y = \frac{u - u^*}{2\tau}.$$

For complex and double numbers, we can easily eliminate $\frac{1}{\tau}$ by multiplying both the numerator and denominator by τ and we end up with $\tau^2 = \sigma \in \mathbb{R}$ in the denominator. However, we cannot perform the same trick for dual numbers because—in that case—σ is equal to zero. Thereby, we must keep ε in the denominator. We must also keep the expressions

with ε^2 in the numerator, because they will contribute to the component with ε if they cancel one of the ε with $\frac{1}{\varepsilon}$.

Here, we clarify what exactly must be calculated in order to update hypercomplex weights. With complex-valued networks, researchers usually treat the real x_n and imaginary y_n parts of weights z_n as separate real-valued channels and update them using the real-valued derivative of a loss function:

$$\begin{cases} x_{n+1} = x_n - \alpha \frac{\partial L}{\partial x} \\ y_{n+1} = y_n - \alpha \frac{\partial L}{\partial y} \end{cases} \Rightarrow z_{n+1} = z_n - \alpha \left(\frac{\partial L}{\partial x} + i \frac{\partial L}{\partial y} \right)$$

We generalize these calculations for all hypercomplex numbers of the second order, considering:

$$u_{n+1} = u_n - \alpha \cdot \left(\frac{\partial L}{\partial x} + \tau \frac{\partial L}{\partial y} \right)$$

We define the expression inside the parenthesis as a hypercomplex gradient and calculate it via the gradient of a hypercomplex operator f (see Appendix B) as follows:

$$\frac{\partial L}{\partial x} + \tau \frac{\partial L}{\partial y} = \frac{\partial L}{\partial f} \left(\frac{\partial f}{\partial x} + \tau \frac{\partial f}{\partial y} \right) + \left(\frac{\partial L}{\partial f} \right)^* \left(\frac{\partial f}{\partial x} - \tau \frac{\partial f}{\partial y} \right)^*$$

It is noteworthy that $\frac{\partial L}{\partial x} + \tau \frac{\partial L}{\partial y}$ can be expressed in terms of u and u^*, i.e.:

$$\frac{\partial f}{\partial x} + \tau \frac{\partial f}{\partial y} = \begin{cases} 2 \cdot \frac{\partial f}{\partial u^*}, \ u \in \mathbb{C} \\ 2 \cdot \frac{\partial f}{\partial u}, \ u \in \mathbb{S} \\ \frac{\partial f}{\partial u} + \frac{\partial f}{\partial u^*} + \varepsilon^2 \left(\frac{\partial f}{\partial u} - \frac{\partial f}{\partial u^*} \right), \ u \in \mathbb{D} \end{cases}$$

With dual numbers, we keep the part with ε^2, despite the fact that it must be equal to zero, as stated above.

We implemented this formula for dual algebra. This method yields results consistent with the calculations from two real-valued derivatives. Our experiments show that the training time is approximately the same as well. However, the performance can be improved if operations on hypercomplex numbers are implemented as instructions on the hardware level.

3. Optimizations

In this section, we discuss technical methods to speed up the execution of testing and/or training models based on hypercomplex numbers. While this is an important engineering challenge, it is not as crucial as achieving the best possible accuracy. We focus on searching for equivalent transformations that speed up hypercomplex operators and do not affect the network output or metric values. These methods are not universal, so the final acceleration depends on the model architecture.

3.1. Hypercomplex Convolution Grouping

We express a hypercomplex convolution through a certain number of real-valued convolutions (Section 2.2). For complex- and dual-valued convolutions, we have to conduct extra addition and/or subtraction (see Figure 1). It is possible to "insert" these arithmetic operations into the real-valued convolutions. In order to do that, we concatenate the components of the input and perform a similar operation for the weights (see Figure 3). This approach increases memory usage and necessitates tensor duplication, using the best-optimized solution. Note that, in the case of the complex-valued convolution, we need to invert the imaginary part of the weights when we calculate the real part of the output. Thus, we implement subtraction. Although the negation operation must be cheap in terms

of execution time, it is possible, if needed, to store the inverted weights in advance to speed up the inference at the cost of a larger model size.

3.2. Karatsuba's Algorithm

Karatsuba's algorithm is a well-known technique used for the optimization of complex number multiplication. It can also be applied to convolutions and linear complex-valued operators. For example, a linear operator can be rewritten as three real-valued linear operators:

$$\mathbb{C}L(w_x + iw_y, x + iy, b_x + ib_y) = L_1 - L_2 + i(L_3 - L_1 - L_2),$$

where $L_1 = \mathbb{R}L(w_x, x, b_x)$, $L_2 = \mathbb{R}L(w_y, y, 0)$, and $L_3 = \mathbb{R}L(w_x + w_y, x + y, b_x + b_y)$. Fewer matrix multiplications come at the expense of a greater number of additions and subtractions. Therefore, Karatsuba's algorithm performs better for convolutions with large batch sizes and weight widths, where multiplications are more expensive.

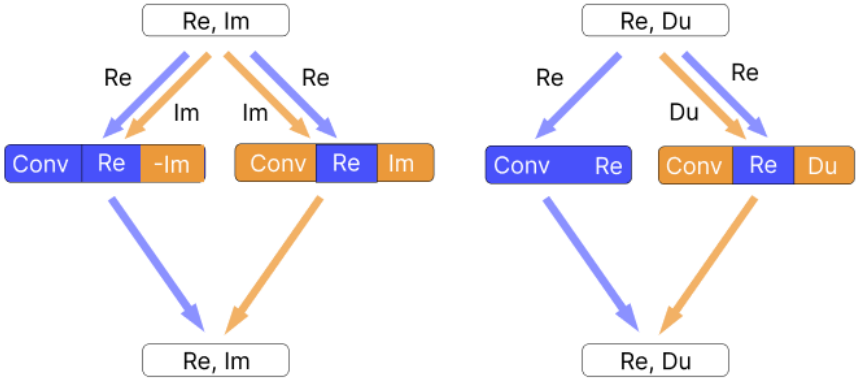

Figure 3. Convolution grouping in the complex and dual algebras.

3.3. Diagonal Representation of Double Numbers

In Section 1, we briefly mentioned the representation of double numbers in the diagonal basis. Here is its mathematical definition:

$$x' = x + y, \quad y' = x - y$$
$$z = x + jy = x'\frac{1+j}{2} + y'\frac{1-j}{2}$$

The computation complexity of multiplication is halved in this form:

$$(A\frac{1+j}{2} + B\frac{1-j}{2}) \cdot (x\frac{1+j}{2} + y\frac{1-j}{2}) = Ax\frac{1+j}{2} + By\frac{1-j}{2}$$

Section 2.2 shows the isomorphism between hypercomplex numbers and matrices of the second order. Then, a convolution of double numbers in the diagonal form can be expressed as follows:

$$W * u \sim \frac{1}{2}\begin{pmatrix} W_x + W_y & W_x - W_y \\ W_x - W_y & W_x + W_y \end{pmatrix} * \frac{1}{2}\begin{pmatrix} x + y & x - y \\ x - y & x + y \end{pmatrix} =$$
$$= \frac{1}{2}\begin{pmatrix} W_x * x + W_y * y & W_x * x - W_y * y \\ W_x * x - W_y * y & W_x * x + W_y * y \end{pmatrix},$$

which implies the convolution calculation as two component-wise real-valued convolutions $W_x * x$ and $W_y * y$. Thus, we define new variables as $j_1 = \frac{1+j}{2}$ and $j_2 = \frac{1-j}{2}$, so that $z = j_1 x + j_2 y$. Batch normalization can be updated accordingly. Moving from the regular

representation to the diagonal one is identical to the affine transformation of the plane of double numbers. The norm of a double number $z = j_1 x + j_2 y$ is expressed as $\max(|x|, |y|)$.

As discussed before, the downside of using the diagonal representation is that x and y are poorly interlinked and are almost equivalent to two independent real numbers. The batch normalization based on the norm of double numbers partially solves this issue. Average pooling and linear functions are calculated for both components independently, because they are special cases of convolution. The relationship between two components is introduced through non-linear activation functions. The ReLU implementation

$$\mathbb{S}ReLU(x + jy) = \mathbb{R}ReLU(x) + j\mathbb{R}ReLU(y)$$

works well with the classic representation of double numbers. It makes sense to reuse this by transforming the intermediate tensor to the classic representation before applying the component-wise activation. In the end, the formula looks as follows:

$$\mathbb{S}ReLU(j_1 x + j_2 y) = j_1 \left[\mathbb{R}ReLU(\frac{x+y}{2}) + \mathbb{R}ReLU(\frac{x-y}{2}) \right] + \\ + j_2 \left[\mathbb{R}ReLU(\frac{x+y}{2}) - \mathbb{R}ReLU(\frac{x-y}{2}) \right].$$

This activation function requires more operations compared to the base ReLU implementation. However, in combination with the enhanced performance of the convolution, it allows the model to save precision with significant acceleration.

3.4. Equivalent Transformations

Real-valued convolutional neural networks can be accelerated by means of a fusing convolution with a subsequent batch normalization layer into a convolution with new parameters, as described by the following equations:

$$W_{fused} = \frac{\alpha}{\sqrt{Var}} \times W, \tag{4}$$

$$b_{fused} = \frac{\alpha}{\sqrt{Var}} \times (b - \mu) + \beta, \tag{5}$$

where W_{fused} and b_{fused} are weights and biases of the fused convolution, W and b are weights and biases of the original convolution, μ, Var, α, β are the mean value, variance, scale, and shift of the batch normalization layer, respectively. The $\times$ operator is an element-wise multiplication.

This transformation leads to improved inference time and memory conservation. In the case of real models, some frameworks have already implemented this optimization. For example, the ONNX simplifier does that automatically for models in the ONNX graph format.

Hypercomplex operators require more resources, which makes this optimization even more critical. However, so far, there is no automated tool that could apply it to hypercomplex models. For example, an attempt to export a model to ONNX and use the ONNX simplifier would fail. Currently, the ONNX simplifier cannot fuse a sequence consisting of hypercomplex convolutional and batch normalization layers. So, we introduce a new layer called CBN (Conv+BN), which behaves differently depending on the mode. During training, it behaves as the original sequence of two layers, and in the evaluation mode, it works as a convolution with fused parameters (Figure 4). These parameters are pre-calculated according to Formulas (4) and (5), assuming that all multiplications follow the rules of arithmetic of hypercomplex numbers.

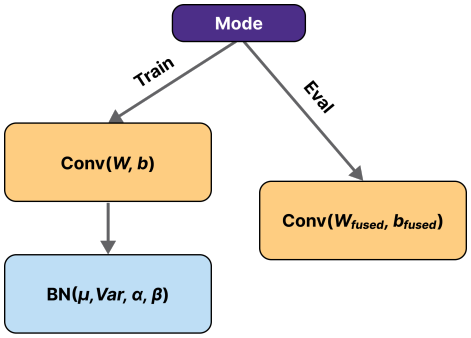

Figure 4. Scheme of hypercomplex CBN layer behaviors in different modes.

Our CBN layer is implemented in MindSpore to improve the inference times of hypercomplex models.

4. Results

To explore the generalization of neural networks on hypercomplex numbers, we develop a demonstrative network to predict values of noised hypercomplex functions. In addition, we conduct a series of experiments with hypercomplex models for computer vision and signal processing tasks. We compare the results of these networks with the results of real-valued models of the same architecture.

4.1. Hypercomplex Toy Net

We start with a demonstrative neural network designed for predicting the values of noisy functions of a hypercomplex argument. Our toy network consists of two linear layers and a sigmoid-type activation function (see Appendix C) in between. This architecture remains the same for all the algebras, but for every algebra, we use its implementation of the operators.

In order to show the advantage of hypercomplex models, we compare their results with the result of a real-valued model of the same architecture. The mean square error (MSE) is used as a loss function. We train these four models for 1000 epochs with the same number of parameters to predict values of two functions: $Ai(u)$—the Airy function of the first kind and $J_3(u)$—the Bessel function of the first kind of the third order. We also add noise with a normal distribution to the values of functions in a training set (and a testing set). The results are shown in Table 2.

Table 2. Toy net—value of the loss function.

Algebra	$Ai(u)$			$J_3(u)$		
	$u \in \mathbb{C}$	$u \in \mathbb{D}$	$u \in \mathbb{S}$	$u \in \mathbb{C}$	$u \in \mathbb{D}$	$u \in \mathbb{S}$
Real	0.026	0.045	0.013	0.018	0.017	0.015
Complex	**0.008**	0.015	0.015	**0.009**	0.012	0.011
Dual	0.017	**0.009**	0.017	0.013	**0.009**	0.012
Double	0.023	0.050	**0.010**	0.017	0.018	**0.010**

It is clear from Figure 5 and Table 2 that the lowest mean square error is achieved by a neural network belonging to the same type of hypercomplex number as the function argument. For example, a complex-valued function is best predicted by a complex-valued model, and so on. Thus, we conclude that networks based on hypercomplex numbers can learn dependencies or patterns that lie in the corresponding algebra.

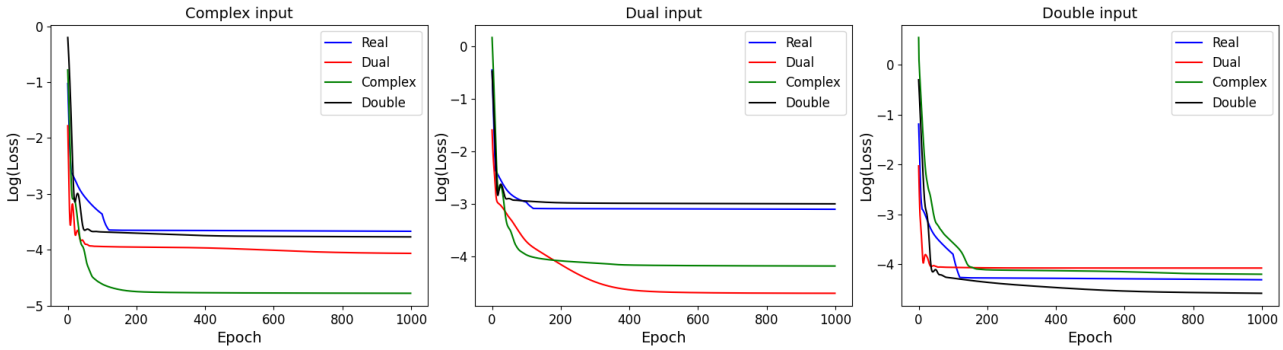

Figure 5. Timeline of the MSE test loss function with respect to the number of epochs for the value prediction of the Airy function in the complex, dual, and double algebras via our real and hypercomplex models.

4.2. Classical CV Problems

Before we move on to solving CV classification problems, it is necessary to clarify the process of data preparation. To convert a real-valued image to the hypercomplex format in this research, we use a more general method compared to the one proposed in [5]. In [5], the authors convert a real-valued image to a complex form by $[R, G, B] \Rightarrow [R + iG, G + iB]$, claiming that this type of encoding captures the channel correlations and hue shift. In this paper, we generalize this idea and use the following transformation $[R, G, B] \Rightarrow [R + \tau G, G + \tau B, B + \tau R]$, which we call the color combination. However, there is another possibility to generate an imaginary component by processing the real input data using several layers [15].

We take ResNet18 architecture [19] as the basis of our model that is generalized to hypercomplex algebras of the second order by replacing real-valued operators with their counterparts in hypercomplex algebras. We use stochastic gradient descent with 0.9 momentum to optimize real-valued loss functions by treating the real and hypercomplex parts as separate real-valued channels. To transform hypercomplex features into the real ones, we concatenate both components of the output into a double-sized real-valued tensor before feeding it to the terminating fully connected layer. The cross-entropy loss between the input and the target is used as a criterion for these problems. The total number of epochs is 200 for every experiment. The learning rate is scheduled by the cosine annealing rule, starting from 0.1. The size of a mini-batch is 8. Every model is trained 5 times with different seed numbers, and the median value of the accuracy is taken to negate the effects of fluctuation. Our image classification results on CIFAR-10, CIFAR-100, and SVHN are presented in Table 3.

Table 3. The accuracy (%) of our models for classical CV problems.

Algebra	**Dataset**		
	CIFAR-100	**CIFAR-10**	**SVHN**
Real	78.63	95.33	96.47
Dual	79.09	95.51	**96.60**
Complex	**79.23**	**95.66**	**96.60**
Double	78.88	95.51	96.45

One can observe in Table 3 that all models based on the second-order numbers achieve higher accuracy compared to the real one. In addition, the complex-valued neural network shows better metric values compared to other hypercomplex models.

Table 4 demonstrates how remarkably inference time is reduced by using the CBN equivalent transformation described in Section 3.4. The effect is so dramatic that, from now on, it is worth considering models with CBN only. Figures from the CBN column of Table 4 are used as the baseline for the assessment of other optimizations, which are discussed

below. These figures also indicate that transferring from a real model to a hypercomplex one significantly increases computational complexity. Table 5 shows the effect of optimization on the second-order hypercomplex models.

Table 4. Effect of batch normalization fusion on the ResNet18 inference time (ms).

Algebra	Device			
	CPU		GPU	
	No Fusion	With CBN	No Fusion	With CBN
Real	8.56	**6.54**	1.72	**1.42**
Complex	34.97	**26.49**	6.73	**5.13**
Dual	28.00	**20.78**	5.45	**4.17**
Double	34.83	**26.62**	6.67	**5.08**

Software as follows: Python 3.8.10, MindSpore 1.10.0; Hardware as follows: CPU: Intel(R) Core (TM) i9-11900KF @ 3.50 GHz, GPU: NVIDIA GeForce RTX 3090.

Table 5. Average inference time (ms) of ResNet18 models.

Algebra	Optimization	GPU Inference Time, ms
Real	Baseline	1.42
Dual	Baseline	4.17
	Conv Grouping	**4.04**
Complex	Baseline	5.13
	Karatsuba	**4.79**
	Conv Grouping	4.91
Double	Baseline	5.08
	Diagonal	**3.29**

Software as follows: Python 3.8.10, MindSpore 1.10.0; Hardware as follows: CPU: Intel(R) Core (TM) i9-11900KF @ 3.50 GHz, GPU: NVIDIA GeForce RTX 3090.

The worst performance is shown by the naive double-valued network, but the transition to the diagonal representation accelerates it dramatically. Still, the double-valued network is not an optimal model because it shows the worst accuracy among all hypercomplex models (see Table 3). As for the complex- and dual-valued networks, grouping convolutions helps to improve the inference time, and for the complex-valued network, the implementation of Karatsuba's algorithm helps even more.

Our main idea is that dual-valued neural networks, as well as complex-valued ones, allow different channels to interact with each other due to the internal connection between real and dual parts. This inner complexity should be exploited; possibly, more sophisticated input preprocessing can lead to even better accuracy for these more complicated networks.

4.3. Gravitational Waves Detection

This part is dedicated to the solution of a signal detection task using hypercomplex networks for the G2Net dataset. This dataset consists of simulated noised signals that are similar to gravitational waves recorded by a system of three ground-based laser interferometers: LIGO Hanford, LIGO Livingston, and Virgo [20–22]. Generally, gravitational waves are emitted during cosmic events, such as the merging of black holes [20]. The G2Net dataset contains records of emulated events of the same nature.

To classify the original signal, we preprocess data to an image and run it through our neural networks. The aim of preprocessing is to build a representative frequency map of the original signal. The CQT algorithm [23] is considered efficient for analyzing gravitational waves.

This preprocessing transfers the time series into the frequency portrait (see Figure 6), which is treated as an image in further steps. Thus, the variations of frequency characteristics of the signal at certain moments are reflected as visual features, such as specific

shapes and colors, in the resulting image. To classify the image obtained after CQT, we again use the ResNet18 model, whose operators are changed to the corresponding ones in ad hoc algebras.

The model is optimized using the stochastic gradient descent algorithm with a momentum of 0.9 and weight decay of 5×10^{-5} for L_2 regularization. The regularization is enhanced by adding dropout with a value of 0.2. The scheduler for the learning rate uses the cosine annealing policy with $Tmax = 6$, and the initial value of lr is set to 5×10^{-2}. Training lasts 30 epochs.

From Table 6, one can see that all models on the hypercomplex algebra outclass the real-valued model in both accuracy and AUC ROC (area under the receiver operating characteristic curve) metrics. The best result is achieved by a dual-valued neural network.

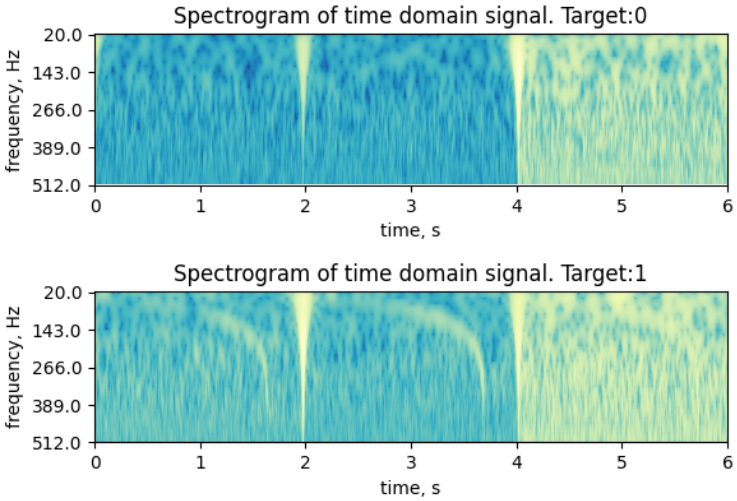

Figure 6. Frequency portrait obtained after CQT in the presence (target = 1) and absence (target = 0) of gravitational waves.

Table 6. Average values of metrics (%) of our models for gravitational wave detection.

Algebra	Accuracy, %	AUC ROC
Real	76.45	0.82
Complex	78.73	0.84
Dual	**79.24**	**0.85**
Double	77.41	0.84

4.4. Music Transcription Task

In this section, we present the results of training hypercomplex networks on an automatic music transcription task. The experiments are performed on the MusicNet dataset [24]. In order to convert the input signal into a complex-valued tensor, we use the Fourier transformation. We reuse the DeepConvNet architecture developed in [15], with its building blocks replaced with the corresponding layers designed for specific algebras.

Table 7 shows that hypercomplex networks significantly outclass the same model on real numbers. The best figures are shown by a dual-valued network.

Table 7. DeepConvNet—average precision.

Algebra	Avg Precision, %
Real	70.0
Complex	73.2
Dual	**73.4**
Double	73.2

Next, we present the results of inference time measurement on GPU (see Table 8). The first column presents the measured inference time of the baseline model, and the second one shows the inference time of the same models after optimizations.

Table 8. DeepConvNet—average GPU inference time (ms).

Algebra	Not Optimized, ms	Optimized, ms
Real	0.66	**0.53**
Complex	2.26	**1.39**
Dual	1.88	**1.36**
Double	2.26	**1.53**

Software as follows: Python 3.8.10, MindSpore 1.10.0; Hardware as follows: CPU: Intel(R) Core (TM) i9-11900KF @ 3.50 GHz, GPU: NVIDIA GeForce RTX 3090.

The optimizations of the complex-valued network include convolution grouping and Karatsuba's algorithm for the most expensive linear operator. For dual numbers, we only use convolution grouping, which provides a small performance gain. The double-valued operators are redesigned to take advantage of the diagonal representation. In all cases, batch normalization operators are fused with the preceding convolutions. The experiments are performed with a batch size of 8. The dual-valued model outperforms all other second-order hypercomplex models. The double-valued network, despite its diagonal representation, yields the poorest results due to the significant overhead associated with activation functions.

5. Discussion

5.1. Summary

Improving model accuracy is the cornerstone of many modern research studies in the field of deep learning. This target can be achieved by different methodologies, such as dataset expansion, changing the training procedure, variation of neural network architecture, and so on. In this study, we offer a take-and-go recipe on how to enhance the model's generalization ability. Namely, one needs to replace real-valued layers with the corresponding complex, dual, or double counterparts, without any modifications to the overall architecture. Generally speaking, it is impossible to predict which kind of algebra will be better for a particular task. To overcome this obstacle, we define a universal approach to developing operators based on any kind of hypercomplex algebra of the second order. This allows researchers to design an architecture of neural networks without heavy dependence on a specific kind of algebra.

In our experiments on the CV, G2Net, and MusicNet tasks, complex and dual-valued models have shown the best results. Nevertheless, neural network construction is a multi-criteria problem. If inference and training times are important, then dual-valued models are often preferable, whereas the complex-valued ones can provide extra accuracy when the timing is not so critical.

5.2. Limitations and Outlook

A limitation of the present study is our sole method of converting real-valued data to hypercomplex numbers for CV problems. It is possible that suboptimal data preprocessing resulted in small gains on these tasks. We plan to explore new methods of real input transformation.

Another unsolved issue is how to identify the type of algebra—for a particular task—that would have the best performance among the hypercomplex algebras of the second order. We will be looking for some heuristic methods that predict the best algebra without training the models with all types of algebras.

In conclusion, the presented methodology exhibits significant promise in enhancing the efficiency of transformer-based architectures. Also, we plan to develop a hypercomplex network compression methodology because we did not find any compression algorithms for non-real models.

Author Contributions: Conceptualization, D.K. and S.P.; methodology, D.K. and A.Z.; software, M.B., A.L. and A.B.; validation, M.B., A.L. and A.B.; formal analysis, S.P., D.K. and A.Z.; investigation, D.K.; funding acquisition, S.P.; project administration, S.P.; data curation, M.B., A.L. and A.B.; writing—original draft preparation, D.K., M.B. and S.P.; writing—review and editing, M.B., A.Z. and S.P.; visualization, M.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are available in a publicly accessible repository that does not issue DOIs. Publicly available datasets were used in this study. These data can be found here: CIFAR-100, CIFAR-10: [https://www.cs.toronto.edu/~kriz/cifar.html (accessed on 29 August 2023)], SVHN: [http://ufldl.stanford.edu/housenumbers/ (accessed on 29 August 2023)], MusicNet: [https://zenodo.org/record/5120004 (accessed on 29 August 2023)], G2Net: [https://www.kaggle.com/competitions/g2net-gravitational-wave-detection/data (accessed on 29 August 2023)]. Our optimized hypercomplex operators on MindSpore are available at [https://gitee.com/c_34/MindSpore/tree/update_hc/MindSpore/python/MindSpore/hypercomplex (accessed on 29 August 2023)].

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Deriving the Hypercomplex Norm Formula

We associate the norm of a hypercomplex number $u = x + \tau y$ with the norm of its matrix representation $A = \begin{pmatrix} x & y \\ \sigma y & x \end{pmatrix}$. Thus, we have

$$\|u\| = \sqrt{sup\{\|At\|^2 \mid t \in \mathbb{R}^{2\times 1}, \|t\| = 1\}}.$$

For the matrix that corresponds to the hypercomplex number $u = x + \tau y$, the norm expression is as follows:

$$\begin{aligned} \|At\|_2^2 &= \left[\begin{pmatrix} x & y \\ \sigma y & x \end{pmatrix} \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} \right]^T \times \left[\begin{pmatrix} x & y \\ \sigma y & x \end{pmatrix} \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} \right] = \\ &= (xt_1 + yt_2)^2 + (xt_2 + \sigma y t_1)^2 = \begin{bmatrix} t_1^2 + t_2^2 = 1 \\ t_1 = \sin\varphi \\ t_2 = \cos\varphi \end{bmatrix} = \\ &= x^2 + y^2(\cos^2\varphi + \sigma^2 \sin^2\varphi) + xy \sin 2\varphi(1+\sigma) \end{aligned}$$

In order to find the extremum of this function, we equate the derivative to zero and solve φ:

$$f'(\varphi) = y^2 \sin 2\varphi(\sigma^2 - 1) + 2xy \cos 2\varphi(1+\sigma) = 0$$

$$\tan 2\varphi = \frac{2x}{y(1-\sigma)} \Rightarrow \begin{cases} cos2\varphi = \dfrac{\pm y(1-\sigma)}{\sqrt{4x^2 + y^2(1-\sigma)^2}} \\ sin2\varphi = \dfrac{\mp 2x}{\sqrt{4x^2 + y^2(1-\sigma)^2}} \end{cases}$$

Eventually, we obtain the maximum of the following function:

$$\|u\|^2 = x^2 + \frac{y^2(1+\sigma^2)}{2} + |y|(1+\sigma)\sqrt{x^2 + y^2\left(\frac{1-\sigma}{2}\right)^2}.$$

From this, it is easy to show that

$$\|u\| = \frac{|y|(1+\sigma)}{2} + \sqrt{x^2 + y^2\left(\frac{1-\sigma}{2}\right)^2}.$$

Particular cases of complex, dual, double numbers ($\sigma = -1, 0, 1$) lead to Equation (2). It is noticeable that with complex numbers ($\sigma = -1$), we obtain the traditional definition of complex norm (1). In addition, we modify the double norm formula $\mathbb{S}\|x + jy\| = |x| + |y|$ to the case of the diagonal representation $x' = x + y$, $y' = x - y$. In the diagonal representation, $x' = x + y$, $y' = x - y$, and then $S\|x + jy\| = \left|\frac{x'+y'}{2}\right| + \left|\frac{x'-y'}{2}\right| = \max(|x'|, |y'|)$.

Appendix B. Deriving the Hypercomplex Gradient Formula

Given that $u = x + \tau y$, where $x, y \in \mathbb{R}$, we can express x and y as follows:

$$x = \frac{u + u^*}{2},\ y = \frac{u - u^*}{2\tau}$$

From here,

$$\frac{\partial L}{\partial u} = \frac{\partial L}{\partial x} \cdot \frac{\partial x}{\partial u} + \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial u} = \frac{1}{2} \cdot \left(\frac{\partial L}{\partial x} + \frac{1}{\tau} \cdot \frac{\partial L}{\partial y}\right)$$

where L is a loss function. Similarly,

$$\frac{\partial L}{\partial u^*} = \frac{1}{2} \cdot \left(\frac{\partial L}{\partial x} - \frac{1}{\tau} \cdot \frac{\partial L}{\partial y}\right)$$

As we assume that L is a real function, and x, y are real by definition,

$$\frac{\partial L}{\partial u^*} = \left(\frac{\partial L}{\partial u}\right)^*$$

Therefore,

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial u} + \left(\frac{\partial L}{\partial u}\right)^* \quad \frac{\partial L}{\partial y} = \tau\left(\frac{\partial L}{\partial u} - \left(\frac{\partial L}{\partial u}\right)^*\right)$$

For a hypercomplex operator $f(u) = f(x + \tau y) = v(x, y) + \tau w(x, y)$, where v and w are both real functions:

$$\begin{aligned}
\frac{\partial L}{\partial x} + \tau\frac{\partial L}{\partial y} &= \left(\frac{\partial L}{\partial v} \cdot \frac{\partial v}{\partial x} + \frac{\partial L}{\partial w} \cdot \frac{\partial w}{\partial x}\right) + \tau\left(\frac{\partial L}{\partial v} \cdot \frac{\partial v}{\partial y} + \frac{\partial L}{\partial w} \cdot \frac{\partial w}{\partial y}\right) = \\
&= \frac{\partial L}{\partial v}\left(\frac{\partial v}{\partial x} + \tau\frac{\partial v}{\partial y}\right) + \frac{\partial L}{\partial w}\left(\frac{\partial w}{\partial x} + \tau\frac{\partial w}{\partial y}\right) = \\
&= \left(\frac{\partial L}{\partial f} + \left(\frac{\partial L}{\partial f}\right)^*\right)\left(\frac{\partial v}{\partial x} + \tau\frac{\partial v}{\partial y}\right) + \tau\left(\frac{\partial L}{\partial f} - \left(\frac{\partial L}{\partial f}\right)^*\right)\left(\frac{\partial w}{\partial x} + \tau\frac{\partial w}{\partial y}\right) = \\
&= \frac{\partial L}{\partial f}\left[\left(\frac{\partial v}{\partial x} + \tau\frac{\partial w}{\partial x}\right) + \tau\left(\frac{\partial v}{\partial y} + \tau\frac{\partial w}{\partial y}\right)\right] + \\
&\quad + \left(\frac{\partial L}{\partial f}\right)^*\left[\left(\frac{\partial v}{\partial x} - \tau\frac{\partial w}{\partial x}\right) + \tau\left(\frac{\partial v}{\partial y} - \tau\frac{\partial w}{\partial y}\right)\right] = \\
&= \frac{\partial L}{\partial f}\left(\frac{\partial f}{\partial x} + \tau\frac{\partial f}{\partial y}\right) + \left(\frac{\partial L}{\partial f}\right)^*\left(\frac{\partial f^*}{\partial x} + \tau\frac{\partial f^*}{\partial y}\right) = \\
&= \frac{\partial L}{\partial f}\left(\frac{\partial f}{\partial x} + \tau\frac{\partial f}{\partial y}\right) + \left(\frac{\partial L}{\partial f}\right)^*\left(\frac{\partial f}{\partial x} - \tau\frac{\partial f}{\partial y}\right)^*
\end{aligned}$$

Appendix C. Functions of Second-Order Hypercomplex Variables

The following is the definition of the sigmoid function for the hypercomplex algebras of the second order:

$$\mathbb{H}\sigma(u) = \frac{1}{1+e^{-u}} = \frac{1}{1+e^{-(x+\tau y)}} = \begin{cases} \frac{1+e^{-x}cosy}{1+2e^{-x}\cos y+e^{-2x}} + i\frac{e^{-x}siny}{1+2e^{-x}\cos y+e^{-2x}}, & u \in \mathbb{C} \\ \frac{1}{1+e^{-x}} + \varepsilon y\frac{e^{-x}}{(1+e^{-x})^2}, & u \in \mathbb{D} \\ j_1\frac{1}{1+e^{-\frac{x+y}{2}}} + j_2\frac{1}{1+e^{\frac{y-x}{2}}}, & u \in \mathbb{S} \end{cases}$$

References

1. Nitta, T. The Computational Power of Complex-Valued Neuron. In Proceedings of the Artificial Neural Networks and Neural Information Processing—ICANN/ICONIP 2003, Istanbul, Turkey, 26–29 June 2003; Kaynak, O., Alpaydin, E., Oja, E., Xu, L., Eds.; Springer: Berlin/Heidelberg, Germany, 2003; pp. 993–1000.
2. Arjovsky, M.; Shah, A.; Bengio, Y. Unitary Evolution Recurrent Neural Networks. In Proceedings of the 33rd International Conference on Machine Learning 2015, Lille, France, 6–11 July 2015.
3. Bassey, J.; Qian, L.; Li, X. A Survey of Complex-Valued Neural Networks. *arXiv* **2021**, arXiv:2101.12249. [CrossRef]
4. Chakraborty, R.; Xing, Y.; Yu, S.X. SurReal: Complex-Valued Learning as Principled Transformations on a Scaling and Rotation Manifold. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *33*, 940–951. [CrossRef] [PubMed]
5. Singhal, U.; Xing, Y.; Yu, S.X. Co-Domain Symmetry for Complex-Valued Deep Learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) 2022, New Orleans, LA, USA, 18–24 June 2022; pp. 681–690.
6. Dimentberg, F.M. *The Screw Calculus and Its Applications in Mechanics*; WP-AFB: Dayton, OH, USA, 1968.
7. del Castillo, G.T. Applications of the double and the dual numbers. The Bianchi models. *Rev. Mex. De Física E* **2020**, *17*, 146. [CrossRef]
8. Baydin, A.G.; Pearlmutter, B.A.; Radul, A.A. Automatic differentiation in machine learning: A survey. *arXiv* **2015**, arXiv:1502.05767.
9. Kiran, R.; Khandelwal, K. Automatic implementation of finite strain anisotropic hyperelastic models using hyper-dual numbers. *Comput. Mech.* **2014**, *55*, 229–248. [CrossRef]
10. Forth, S.; Hovland, P.; Phipps, E.; Utke, J.; Walther, A. (Eds.) *Recent Advances in Algorithmic Differentiation*; Springer: Berlin/Heidelberg, Germany, 2012; Volume 87. [CrossRef]
11. Anderson, T. *Split-Complex Convolutional Neural Networks*; 2017; p. 7. Available online: http://cs229.stanford.edu/proj2017/final-reports/5231880.pdf (accessed on 20 August 2023).
12. Buchholz, S.; Sommer, G. On Clifford neurons and Clifford multi-layer perceptrons. *Neural Netw.* **2008**, *21*, 925–935. [CrossRef] [PubMed]
13. Okawa, Y.; Nitta, T. Learning Properties of Feedforward Neural Networks Using Dual Numbers. In Proceedings of the 2021 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), Tokyo, Japan, 14–17 December 2021; pp. 187–192.
14. Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of the 32nd ICML 2015, Lille, France, 6–11 July 2015; pp. 448–456.
15. Trabelsi, C.; Bilaniuk, O.; Zhang, Y.; Serdyuk, D.; Subramanian, S.; Santos, J.F.; Mehri, S.; Rostamzadeh, N.; Bengio, Y.; Pal, C.J. Deep Complex Networks. In Proceedings of the International Conference on Learning Representations 2018, Vancouver, BC, Canada, 30 April–3 May 2018.
16. Kozlov, D.; Pavlov, S.; Zuev, A.; Bakulin, M.; Krylova, M.; Kharchikov, I. Dual-valued Neural Networks. In Proceedings of the 2022 18th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), Madrid, Spain, 29 November–2 December 2022; pp. 1–8. [CrossRef]
17. Wirtinger, W. Zur formalen Theorie der Funktionen yon mehr komplexen Veriinderliehen. *Math. Ann.* **1926**, *97*, 357–375. [CrossRef]
18. Nitta, T.; Kuroe, Y. Hyperbolic Gradient Operator and Hyperbolic Back-Propagation Learning Algorithms. *IEEE Trans. Neural Netw. Learn. Syst.* **2017**, *29*, 1689–1702. [CrossRef] [PubMed]
19. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778. [CrossRef]
20. Abbott, B.; Abbott, R.; Abbott, T.; Abernathy, M.R.; Acernese, F.; Ackley, K.; Adams, C.; Adams, T.; Addesso, P.; Adhikari, R.X.; et al. Observation of Gravitational Waves from a Binary Black Hole Merger. *Phys. Rev. Lett.* **2016**, *116*, 061102. [CrossRef] [PubMed]
21. Shannon, R.M.; Ravi, V.; Lentati, L.T.; Lasky, P.D.; Hobbs, G.; Kerr, M.; Manchester, R.N.; Coles, W.A.; Levin, Y.; Bailes, M.; et al. Gravitational waves from binary supermassive black holes missing in pulsar observations. *Science* **2015**, *349*, 1522–1525. [CrossRef] [PubMed]
22. Thorne, K.S. Gravitational Waves from Compact Bodies. In *Symposium-International Astronomical Union*; Cambridge University Press: Cambridge, UK, 1995. [CrossRef]

23. Brown, J. Calculation of a constant Q spectral transform. *J. Acoust. Soc. Am.* **1991**, *89*, 425–434. [CrossRef]
24. Thickstun, J.; Harchaoui, Z.; Kakade, S.M. Learning features of music from scratch. *arXiv* **2017**, arXiv:1611.09827.