Deep Quaternion Networks

Chase Gaudet
cjg7182@louisiana.edu

Anthony Maida
maida@louisiana.edu

December 10, 2017

Abstract

The field of deep learning has seen significant advancement in recent years. However, much of the existing work has been focused on real-valued numbers. Recent work has shown that a deep learning system using the complex numbers can be deeper for a set parameter budget compared to its real-valued counterpart. In this work, we explore the benefits of generalizing one step further into the hyper-complex numbers, quaternions specifically, and provide the architecture components needed to build deep quaternion networks. We go over quaternion convolutions, present a quaternion weight initialization scheme, and present algorithms for quaternion batch-normalization. These pieces are tested by end-to-end training on the CIFAR-100 data set to show the improved convergence to a real-valued network.

1 Introduction

There have been many advances in deep neural network architectures in the past few years. One such improvement is a normalization technique called batch normalization [IS15] that standardizes the activations of layers inside a network using minibatch statistics. It has been shown to regularize the network as well as provide faster and more stable training. Another improvement comes from architectures that add so called shortcut paths to the network. These shortcut paths connect later layers to earlier layers typically, which allows for the stronger gradients to propagate to the earlier layers. This method can be seen in Highway Networks [SGS15] and Residual Networks [HZRS16]. Other work has been done to find new activation functions with more desirable properties. One example is the exponential linear unit (ELU) [CUH15], which attempts to keep activations standardized. All of the above methods are combating the vanishing gradient problem [Hoc91] that plagues deep architectures. With solutions to this problem appearing it is only natural to move to a system that will allow one to construct deeper architectures with as low a parameter cost as possible.

Other work in this area has explored the use of complex and hyper-complex numbers, which are a generalization of the complex, such as quaternions. Using complex numbers in recurrent neural networks (RNNs) has been shown to increase learning speed and provide a more noise robust memory retrieval mechanism [ASB16, DWU$^+$16, WPH$^+$16]. The first formulation of complex batch normalization and complex weight initialization is presented by [TBS$^+$17] where they achieve some state of the art results on the MusicNet data set. Hyper-complex numbers are less explored in neural networks, but have seen use in manual image and signal processing techniques [Bül99, SE00, BS01]. Examples of using quaternion values in networks is mostly limited to architectures that take in quaternion inputs or predict quaternion outputs, but do not have quaternion weight values [Ris06, KGC15]. There are some more recent examples of building models that use quaternions represented as real-values. In [PMB$^+$16] they used a quaternion multi-layer perceptron (QMLP) for document understanding and [MINM17] uses a similar approach in processing multi-dimensional signals.

Building on [TBS$^+$17] our contribution in this paper is to formulate quaternion convolution, batch normalization, and weight initialization. There arises some difficulty over complex batch normalization that we had to overcome as their is no analytic form for our inverse square root matrix.

2 Motivation and Related Work

The ability of quaternions to effectively represent spatial transformations and analyze multi-dimensional signals makes them promising for applications in artificial intelligence.

One common use of quaternions is for representing rotation into a more compact form. PoseNet [KGC15] used a quaternion as the target output in their model where the goal was to regress the $6-$DOF camera pose from a single RGB image. The ability to encode rotations may make a quaternion network more robust to rotational variance.

Quaternion representation has also been used in signal processing. The amount of information in the phase of an image has been shown to be sufficient to recover the majority of information encoded in its magnitude by Oppenheim and Lin [OL81]. The phase also encodes information such as shapes, edges, and orientations. Quaternions can be represented as a 2 x 2 matrix of complex numbers, which gives them a group of phases potentially holding more information compared to a single phase. Bulow and Sommer [BS01] used the higher complexity representation of quaternions by extending Gabor's complex signal to a quaternion one which was then used for texture segmentation. Another use of quaternion filters is shown in [SE00] where they introduce a new class of filter based on convolution with hyper-complex masks, and present three color edge detecting filters. These filters rely on a three-space rotation about the grey line of RGB space and when applied to a color image produce an almost greyscale image with color edges where the original image had a sharp change of color. A quaternionic extension of feed forward neural network, for processing multi-dimensional signals, is shown in [MINM17]. They expect that quaternion neurons operate on multi-dimensional signals as single entities, rather than real-valued neurons that deal with each element of signals independently. A convolutional neural network (CNN) should be able to learn a powerful set of quaternion filters for more impressive tasks.

Another large motivation is discussed in [TBS+17], which is that complex numbers are more efficient and provide more robust memory mechanisms compared to the reals [Bül99, SE00, BS01]. They continue that residual networks have a similar architecture to associative memories since the residual shortcut paths compute their residual and then sum it into the memory provided by the identity connection. Again, given that quaternions can be represented as a complex group, they may provide an even more efficient and robust memory mechanisms.

3 Quaternion Network Components

This section will include the work done to obtain a working deep quaternion network. Some of the longer derivations are shown in the appendix.

3.1 Quaternion Representation

In 1833 Hamilton proposed complex numbers $\mathbb{C}$ be defined as the set $\mathbb{R}^2$ of ordered pairs (a, b) of real numbers. He then began working to see if triplets (a, b, c) could extend multiplication of complex numbers. In 1843 he discovered a way to multiply in four dimensions instead of three, but the multiplication lost commutativity. This construction is now known as quaternions. Quaternions are composed of four components, one real part, and three imaginary parts. Typically denoted as

$$\mathbb{H} = \{a + bi + cj + dk \ : \ a, b, c, d \in \mathbb{R}\} \tag{1}$$

where a is the real part, (i, j, k) denotes the three imaginary axis, and (b, c, d) denotes the three imaginary components. Quaternions are governed by the following arithmetic:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{2}$$

which leads to the noncommutative multiplication rules

$$ij = k, \ jk = i, \ ki = j, \ ji = -k, \ kj = -i, \ ik = -j \tag{3}$$

Since we will be performing quaternion arithmetic using reals it is useful to embed $\mathbb{H}$ into a real-valued representation. There exists an injective homomorphism from $\mathbb{H}$ to the matrix ring $M(4, \mathbb{R})$ where $M(4, \mathbb{R})$ is a 4x4 real matrix.

The 4 x 4 matrix can be written as

$$\begin{bmatrix} a & -b & -c & -d \\ b & a & -d & c \\ c & d & a & -b \\ d & -c & b & a \end{bmatrix} = a \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + b \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix} + \\ c \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix} + d \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}. \quad (4)$$

This representation of quaternions is not unique, but we will stick to the above in this paper. It is also possible to represent $\mathbb{H}$ as $M(2, \mathbb{C})$ where $M(2, \mathbb{C})$ is a 2 x 2 complex matrix.

With our real-valued representation a quaternion real-valued $2D$ convolution layer can be expressed as follows. Say that the layer has N feature maps such that N is divisible by 4. We let the first $N/4$ feature maps represent the real components, the second $N/4$ represent the i imaginary components, the third $N/4$ represent the j imaginary components, and the last $N/4$ represent the k imaginary components.

3.2 Quaternion Differentiability

In order for the network to perform backpropagation the cost function and activation functions used must be differentiable with respect to the real, i, j, and k components of each quaternion parameter of the network. As the complex chain rule is shown in [TBS$^+$17], we provide the quaternion chain rule which can be seen in 7.1 of the Appendix.

3.3 Quaternion Convolution

Convolution in the quaternion domain is done by convolving a quaternion filter matrix $\mathbf{W} = \mathbf{A} + i\mathbf{B} + j\mathbf{C} + k\mathbf{D}$ by a quaternion vector $\mathbf{h} = \mathbf{w} + i\mathbf{x} + j\mathbf{y} + k\mathbf{z}$. Performing the convolution by using the distributive property and grouping terms one gets

$$\begin{aligned} \mathbf{W} * \mathbf{h} = & (\mathbf{A} * \mathbf{w} - \mathbf{B} * \mathbf{x} - \mathbf{C} * \mathbf{y} - \mathbf{D} * \mathbf{z}) + \\ & i(\mathbf{A} * \mathbf{x} + \mathbf{B} * \mathbf{w} + \mathbf{C} * \mathbf{z} - \mathbf{D} * \mathbf{y}) + \\ & j(\mathbf{A} * \mathbf{y} - \mathbf{B} * \mathbf{z} + \mathbf{C} * \mathbf{w} + \mathbf{D} * \mathbf{x}) + \\ & k(\mathbf{A} * \mathbf{z} + \mathbf{B} * \mathbf{y} - \mathbf{C} * \mathbf{x} + \mathbf{D} * \mathbf{w}). \end{aligned} \quad (5)$$

Using a matrix to represent the components of the convolution we have:

$$\begin{bmatrix} \mathscr{R}(\mathbf{W} * \mathbf{h}) \\ \mathscr{I}(\mathbf{W} * \mathbf{h}) \\ \mathscr{J}(\mathbf{W} * \mathbf{h}) \\ \mathscr{K}(\mathbf{W} * \mathbf{h}) \end{bmatrix} = \begin{bmatrix} \mathbf{A} & -\mathbf{B} & -\mathbf{C} & -\mathbf{D} \\ \mathbf{B} & \mathbf{A} & -\mathbf{D} & \mathbf{C} \\ \mathbf{C} & \mathbf{D} & \mathbf{A} & -\mathbf{B} \\ \mathbf{D} & -\mathbf{C} & \mathbf{B} & \mathbf{A} \end{bmatrix} * \begin{bmatrix} \mathbf{w} \\ \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{bmatrix} \quad (6)$$

An example is shown in Fig. 1.

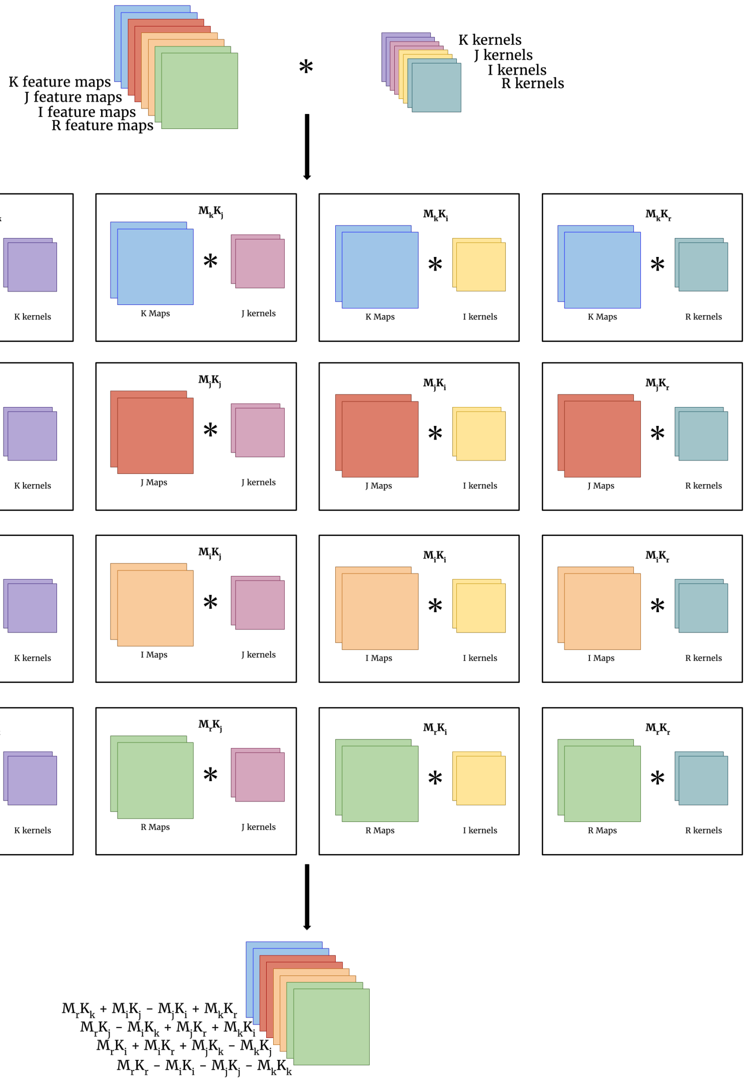

Figure 1: An illustration of quaternion convolution.

3.4 Quaternion Batch-Normalization

Batch-normalization [IS15] is used by the vast majority of all deep networks to stabilize and speed up training. It works by keeping the activations of the network at zero mean and unit variance. The original formulation of batch-normalization only works for real-values. Applying batch normalization to complex or hyper-complex numbers is more difficult, one can not simply translate and scale them such that their mean is 0 and their variance is 1. This would not give equal variance in the multiple components of a complex or hyper-complex number. To overcome this for complex numbers a whitening approach is used [TBS+17], which scales the data by the square root of their variances along each of the two principle components. We use the same approach, but must whiten 4D vectors.

However, an issue arises in that there is no nice way to calculate the inverse square root of a 4 x 4 matrix. It turns out that the square root is not necessary and we can instead use the Cholesky decomposition on our covariance matrix. The details of why this works for whitening can be found in 7.2 of the Appendix. Now our whitening is accomplished by multiplying the $\mathbf{0}$-centered data $(\mathbf{x}-\mathbb{E}[\mathbf{x}])$ by $\mathbf{W}$:

$$\tilde{x} = \mathbf{W}(\mathbf{x} - \mathbb{E}[\mathbf{x}]) \tag{7}$$

where $\mathbf{W}$ is one of the matrices from the Cholesky decomposition of $\mathbf{V}^{-1}$ where $\mathbf{V}$ is the covariance matrix given by:

$$\begin{aligned}\mathbf{V} &= \begin{bmatrix} V_{rr} & V_{ri} & V_{rj} & V_{rk} \\ V_{ir} & V_{ii} & V_{ij} & V_{ik} \\ V_{jr} & V_{ji} & V_{jj} & V_{jk} \\ V_{kr} & V_{ki} & V_{kj} & V_{kk} \end{bmatrix} \\ &= \begin{bmatrix} \mathrm{Cov}(\mathscr{R}\{\mathbf{x}\},\mathscr{R}\{\mathbf{x}\}) & \mathrm{Cov}(\mathscr{R}\{\mathbf{x}\},\mathscr{I}\{\mathbf{x}\}) & \mathrm{Cov}(\mathscr{R}\{\mathbf{x}\},\mathscr{J}\{\mathbf{x}\}) & \mathrm{Cov}(\mathscr{R}\{\mathbf{x}\},\mathscr{K}\{\mathbf{x}\}) \\ \mathrm{Cov}(\mathscr{I}\{\mathbf{x}\},\mathscr{R}\{\mathbf{x}\}) & \mathrm{Cov}(\mathscr{I}\{\mathbf{x}\},\mathscr{I}\{\mathbf{x}\}) & \mathrm{Cov}(\mathscr{I}\{\mathbf{x}\},\mathscr{J}\{\mathbf{x}\}) & \mathrm{Cov}(\mathscr{I}\{\mathbf{x}\},\mathscr{K}\{\mathbf{x}\}) \\ \mathrm{Cov}(\mathscr{J}\{\mathbf{x}\},\mathscr{R}\{\mathbf{x}\}) & \mathrm{Cov}(\mathscr{J}\{\mathbf{x}\},\mathscr{I}\{\mathbf{x}\}) & \mathrm{Cov}(\mathscr{J}\{\mathbf{x}\},\mathscr{J}\{\mathbf{x}\}) & \mathrm{Cov}(\mathscr{J}\{\mathbf{x}\},\mathscr{K}\{\mathbf{x}\}) \\ \mathrm{Cov}(\mathscr{K}\{\mathbf{x}\},\mathscr{R}\{\mathbf{x}\}) & \mathrm{Cov}(\mathscr{K}\{\mathbf{x}\},\mathscr{I}\{\mathbf{x}\}) & \mathrm{Cov}(\mathscr{K}\{\mathbf{x}\},\mathscr{J}\{\mathbf{x}\}) & \mathrm{Cov}(\mathscr{K}\{\mathbf{x}\},\mathscr{K}\{\mathbf{x}\}) \end{bmatrix}\end{aligned} \tag{8}$$

where $\mathrm{Cov}(\cdot)$ is the covariance and $\mathscr{R}\{x\}$, $\mathscr{I}\{x\}$, $\mathscr{J}\{x\}$, and $\mathscr{K}\{x\}$ are the real, i, j, and k components of $\mathbf{x}$ respectively.

Real-valued batch normalization also uses two learned parameters, β and γ. Our shift parameter $\boldsymbol{\beta}$ must shift a quaternion value so it is a quaternion value itself with real, i, j, and k as learnable components. The scaling parameter $\boldsymbol{\gamma}$ is a symmetric matrix of size matching $\mathbf{V}$ given by:

$$\gamma = \begin{pmatrix} V_{rr} & V_{ri} & V_{rj} & V_{rk} \\ V_{ri} & V_{ii} & V_{ij} & V_{ik} \\ V_{rj} & V_{ij} & V_{jj} & V_{jk} \\ V_{rk} & V_{ik} & V_{jk} & V_{kk} \end{pmatrix} \tag{9}$$

Because of its symmetry it has only ten learnable parameters. The variance of the components of input $\tilde{\mathbf{x}}$ are variance 1 so the diagonal of $\boldsymbol{\gamma}$ is initialized to $1/\sqrt{16}$ in order to obtain a modulus of 1 for the variance of the normalized value. The off diagonal terms of $\boldsymbol{\gamma}$ and all components of $\boldsymbol{\beta}$ are initialized to 0. The quaternion batch normalization is defined as:

$$\mathrm{BN}(\tilde{\mathbf{x}}) = \gamma\tilde{\mathbf{x}} + \beta \tag{10}$$

3.5 Quaternion Weight Initialization

The proper initialization of weights is vital to convergence of deep networks. In this work we derive our quaternion weight initialization using the same procedure as Glorot and Bengio [GB10] and He et al. [HZRS15].

To begin we find the variance of a quaternion weight:

$$\begin{aligned} W &= |W|e^{(\cos\phi_1 i + \cos\phi_2 j + \cos\phi_3 k)\theta} \\ &= \mathscr{R}\{W\} + \mathscr{I}\{W\} + \mathscr{J}\{W\} + \mathscr{K}\{W\}. \end{aligned} \tag{11}$$

where $|W|$ is the magnitude, θ and ϕ are angle arguments, and $\cos^2\phi_1 + \cos^2\phi_2 + \cos^2\phi_3 = 1$ [PT02].

Variance is defined as

$$\mathrm{Var}(W) = \mathbb{E}[|W|^2] - (\mathbb{E}[W])^2, \tag{12}$$

but since W is symmetric around 0 the term $(\mathbb{E}[W])^2$ is 0. We do not have a way to calculate $\mathrm{Var}(W) = \mathbb{E}[|W|^2]$ so we make use of the magnitude of quaternion normal values $|W|$, which follows an independent normal distribution with four degrees of freedom (DOFs). We can then calculate the expected value of $|W|^2$ to find our variance

$$\mathbb{E}[|W|^2] = \int_{\infty}^{\infty} x^2 f(x)\ dx = 4\sigma^2 \tag{13}$$

where $f(x)$ is the four DOF distribution shown in the Appendix.

And since $\mathrm{Var}(W) = \mathbb{E}[|W|^2]$, we now have the variance of W expressed in terms of a single parameter σ:

$$\mathrm{Var}(W) = 4\sigma^2. \tag{14}$$

To follow the Glorot and Bengio [GB10] initialization we have $\mathrm{Var}(W) = 2/(n_{in} + n_{out})$, where n_{in} and n_{out} are the number of input and output units respectivly. Setting this equal to (14) and solving for σ gives $\sigma = 1/\sqrt{2(n_{in} + n_{out})}$. To follow He et al. [HZRS15] initialization that is specialized for rectified linear units (ReLUs) [NH10], then we have $\mathrm{Var}(W) = 2/n_{in}$, which again setting equal to (14) and solving for σ gives $\sigma = 1/\sqrt{2n_{in}}$.

As shown in (11) the weight has components $|W|$, θ, and ϕ. We can initialize the magnitude $|W|$ using our four DOF distribution defined with the appropriate σ based on which initialization scheme we are following. The angle components are initialized using the uniform distribution between $-\pi$ and π where we ensure the constraint on ϕ.

4 Experimental Results

Our experiments only covered image classification using both the CIFAR-10 and CIFAR-100 benchmarks. We use the same architecture as the large model in [TBS$^+$17], which is a 110 layer Residual model similar to the one in [HZRS16]. There is one difference between the real-valued network and the ones used for both the complex and hyper-complex valued networks. Because the datasets are all real-valued the network must learn the imaginary or quaternion components.

Architecture	CIFAR-10	CIFAR-100
[HZRS16]	6.37	-
[TBS^{+}17] Complex	5.60	27.09
Quarernion	**5.44**	**HOLDER**

Table 1: Classification error on CIFAR-10 and CIFAR-100. Note that [HZRS16] is a 110 layer residual network, [TBS^{+}17] is 118 layer complex network with the same design as the prior except additional initial layers to extract complex mappings.

We use the same technique as [TBS^{+}17] where there is an additional block immediately after the input which will learn the hyper-complex components

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

Since all datasets that we work with are real-valued, we present a way to learn their imaginary components to let the rest of the network operate in the complex plane. We learn the initial imaginary component of our input by performing the operations present within a single real-valued residual block

This means that to maintain the same approximate parameter count the number of convolution kernels for the complex network was increased. We however did not increase the number of convolution kernels for the quaternion trials so any increase in model performance comes from the quaternion filters and at a lower hardware budget.

The architecture consists of 3 stages of repeating residual blocks where at the end of each stage the images are downsized by strided convolutions. Each stage also doubles the previous stage's number of convolution kernels. The last layers are a global average pooling layer followed by a single fully connected layer with a softmax function is used to classify the input as either one of the 10 classes in CIFAR-10 or one of the 100 classes in CIFAR-100.

We also followed their training procedure of using the backpropagation algorithm with Stochastic Gradient Descent with Nesterov momentum [Nes] set at 0.9. The norm of the gradients are clipped to 1 and a custom learning rate scheduler is used. The learning scheduler is the same used in [TBS^{+}17] for a direct comparison in performance. The learning rate is initially set to 0.01 for the first 10 epochs and then set to 0.1 from epoch 10-100 and then cut the learning rate by a factor of 10 at epochs 120 and 150. Table 1 presents our results along side the real and complex valued networks. Our quaternion model outperforms the real and complex networks on both datasets on a smaller parameter budget.

5 Conclusions

We have extended upon work looking into complex valued networks by exploring quaternion values. We presented the building blocks required to build and train deep quaternion networks and used them to test residual architectures on two common image classification benchmarks. We show that they have competitive performance by beating both the real and complex valued networks with less parameters. Future work will be needed to test quaternion networks for tasks outside of image classification such as semantic segmentation or audio processing.

6 Acknowledgments

We would like to thank James Dent of the University of Louisiana at Lafayette Physics Department for helpful discussions. We also thank Fugro for research time on this project.

References

[ASB16] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128, 2016.

[BS01] Thomas Bulow and Gerald Sommer. Hypercomplex signals-a novel extension of the analytic signal to the multidimensional case. *IEEE Transactions on signal processing*, 49(11):2844–2852, 2001.

[Bül99] Thomas Bülow. *Hypercomplex spectral signal representations for the processing and analysis of images.* Universität Kiel. Institut für Informatik und Praktische Mathematik, 1999.

[CUH15] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[DWU$^+$16] Ivo Danihelka, Greg Wayne, Benigno Uria, Nal Kalchbrenner, and Alex Graves. Associative long short-term memory. *arXiv preprint arXiv:1602.03032*, 2016.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[Hoc91] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91, 1991.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[KGC15] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015.

[KLS17] Agnan Kessy, Alex Lewin, and Korbinian Strimmer. Optimal whitening and decorrelation. *The American Statistician*, (just-accepted), 2017.

[MINM17] Toshifumi Minemoto, Teijiro Isokawa, Haruhiko Nishimura, and Nobuyuki Matsui. Feed forward neural network with random quaternionic neurons. *Signal Processing*, 136:59–68, 2017.

[Nes] Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2).

[NH10] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[OL81] Alan V Oppenheim and Jae S Lim. The importance of phase in signals. *Proceedings of the IEEE*, 69(5):529–541, 1981.

[PMB$^+$16] Titouan Parcollet, Mohamed Morchid, Pierre-Michel Bousquet, Richard Dufour, Georges Linarès, and Renato De Mori. Quaternion neural networks for spoken language understanding. In *Spoken Language Technology Workshop (SLT), 2016 IEEE*, pages 362–368. IEEE, 2016.

[PT02] Robert Piziak and Danny Turner. The polar form of a quaternion. 2002.

[Ris06] Adityan Rishiyur. Neural networks with complex and quaternion inputs. *arXiv preprint cs/0607090*, 2006.

[SE00] Stephen J Sangwine and Todd A Ell. Colour image filters based on hypercomplex convolution. *IEE Proceedings-Vision, Image and Signal Processing*, 147(2):89–93, 2000.

[SGS15] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.

[TBS$^+$17] Chiheb Trabelsi, Olexa Bilaniuk, Dmitriy Serdyuk, Sandeep Subramanian, João Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and Christopher J Pal. Deep complex networks. *arXiv preprint arXiv:1705.09792*, 2017.

[WPH$^+$16] Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 4880–4888, 2016.

7 Appendix

7.1 The Generalized Quaternion Chain Rule for a Real-Valued Function

Let L be a real valued loss function and q be a quaternion variable such that $q = a + i\, b + j\, c + k\, d$ where $a, b, c, d \in \mathbb{R}$ then,

$$\begin{aligned}\nabla_L(q) = \frac{\partial L}{\partial q} &= \frac{\partial L}{\partial a} + i\, \frac{\partial L}{\partial b} + j\, \frac{\partial L}{\partial c} + k\, \frac{\partial L}{\partial d} \\ &= \frac{\partial L}{\partial \mathbb{R}(q)} + i\, \frac{\partial L}{\partial \mathbb{I}(q)} + j\, \frac{\partial L}{\partial \mathbb{J}(q)} + k\, \frac{\partial L}{\partial \mathbb{K}(q)} \\ &= \mathbb{R}(\nabla_L(q)) + \mathbb{I}(\nabla_L(q)) + \mathbb{J}(\nabla_L(q)) + \mathbb{K}(\nabla_L(q))\end{aligned} \tag{15}$$

Now let $g = m + i\,n + j\,o + k\,p$ be another quaternion variable where q can be expressed in terms of g and $m, n, o, p \in \mathbb{R}$ we then have,

$$
\begin{aligned}
\nabla_L(q) &= \frac{\partial L}{\partial g} = \frac{\partial L}{\partial m} + i\,\frac{\partial L}{\partial n} + j\,\frac{\partial L}{\partial o} + k\,\frac{\partial L}{\partial p} \qquad (16)\\
&= \frac{\partial L}{\partial a}\frac{\partial a}{\partial m} + \frac{\partial L}{\partial b}\frac{\partial b}{\partial m} + \frac{\partial L}{\partial c}\frac{\partial c}{\partial m} + \frac{\partial L}{\partial d}\frac{\partial d}{\partial m}\\
&+ i\,\left(\frac{\partial L}{\partial a}\frac{\partial a}{\partial n} + \frac{\partial L}{\partial b}\frac{\partial b}{\partial n} + \frac{\partial L}{\partial c}\frac{\partial c}{\partial n} + \frac{\partial L}{\partial d}\frac{\partial d}{\partial n}\right)\\
&+ j\,\left(\frac{\partial L}{\partial a}\frac{\partial a}{\partial o} + \frac{\partial L}{\partial b}\frac{\partial b}{\partial o} + \frac{\partial L}{\partial c}\frac{\partial c}{\partial o} + \frac{\partial L}{\partial d}\frac{\partial d}{\partial o}\right)\\
&+ k\,\left(\frac{\partial L}{\partial a}\frac{\partial a}{\partial p} + \frac{\partial L}{\partial b}\frac{\partial b}{\partial p} + \frac{\partial L}{\partial c}\frac{\partial c}{\partial p} + \frac{\partial L}{\partial d}\frac{\partial d}{\partial p}\right)\\
&= \frac{\partial L}{\partial a}\left(\frac{\partial a}{\partial m} + i\,\frac{\partial a}{\partial n} + i\,\frac{\partial a}{\partial o} + i\,\frac{\partial a}{\partial p}\right)\\
&+ \frac{\partial L}{\partial b}\left(\frac{\partial b}{\partial m} + i\,\frac{\partial b}{\partial n} + i\,\frac{\partial b}{\partial o} + i\,\frac{\partial b}{\partial p}\right)\\
&+ \frac{\partial L}{\partial c}\left(\frac{\partial c}{\partial m} + i\,\frac{\partial c}{\partial n} + i\,\frac{\partial c}{\partial o} + i\,\frac{\partial c}{\partial p}\right)\\
&+ \frac{\partial L}{\partial d}\left(\frac{\partial d}{\partial m} + i\,\frac{\partial d}{\partial n} + i\,\frac{\partial d}{\partial o} + i\,\frac{\partial d}{\partial p}\right)\\
&= \frac{\partial L}{\partial \mathbb{R}(q)}\left(\frac{\partial a}{\partial m} + i\,\frac{\partial a}{\partial n} + i\,\frac{\partial a}{\partial o} + i\,\frac{\partial a}{\partial p}\right)\\
&+ \frac{\partial L}{\partial \mathbb{I}(q)}\left(\frac{\partial b}{\partial m} + i\,\frac{\partial b}{\partial n} + i\,\frac{\partial b}{\partial o} + i\,\frac{\partial b}{\partial p}\right)\\
&+ \frac{\partial L}{\partial \mathbb{J}(q)}\left(\frac{\partial c}{\partial m} + i\,\frac{\partial c}{\partial n} + i\,\frac{\partial c}{\partial o} + i\,\frac{\partial c}{\partial p}\right)\\
&+ \frac{\partial L}{\partial \mathbb{K}(q)}\left(\frac{\partial d}{\partial m} + i\,\frac{\partial d}{\partial n} + i\,\frac{\partial d}{\partial o} + i\,\frac{\partial d}{\partial p}\right)\\
&= \mathbb{R}(\nabla_L(q))\left(\frac{\partial a}{\partial m} + i\,\frac{\partial a}{\partial n} + i\,\frac{\partial a}{\partial o} + i\,\frac{\partial a}{\partial p}\right)\\
&+ \mathbb{I}(\nabla_L(q))\left(\frac{\partial b}{\partial m} + i\,\frac{\partial b}{\partial n} + i\,\frac{\partial b}{\partial o} + i\,\frac{\partial b}{\partial p}\right)\\
&+ \mathbb{J}(\nabla_L(q))\left(\frac{\partial c}{\partial m} + i\,\frac{\partial c}{\partial n} + i\,\frac{\partial c}{\partial o} + i\,\frac{\partial c}{\partial p}\right)\\
&+ \mathbb{K}(\nabla_L(q))\left(\frac{\partial d}{\partial m} + i\,\frac{\partial d}{\partial n} + i\,\frac{\partial d}{\partial o} + i\,\frac{\partial d}{\partial p}\right)
\end{aligned}
$$

7.2 Whitening a Matrix

Let $\mathbf{X}$ be an n x n matrix and $\text{cov}(\mathbf{X}) = \mathbf{\Sigma}$ is the symmetric covariance matrix of the same size. Whitening a matrix linearly decorrelates the input dimensions, meaning that whitening transforms $\mathbf{X}$ into $\mathbf{Z}$ such that $\text{cov}(\mathbf{Z}) = \mathbf{I}$ where $\mathbf{I}$ is the identity matrix [KLS17]. The matrix $\mathbf{Z}$ can be written as:

$$\mathbf{Z} = \mathbf{W}(\mathbf{X} - \mu) \tag{17}$$

where $\mathbf{W}$ is an n x n 'whitening' matrix. Since $\text{cov}(\mathbf{Z}) = \mathbf{I}$ it follows that:

$$\begin{aligned}
&\mathbb{E}[\mathbf{Z}\mathbf{Z}^T] = \mathbf{I} \\
&\mathbb{E}[\mathbf{W}(\mathbf{X} - \mu)(\mathbf{W}(\mathbf{X} - \mu))^T] = \mathbf{I} \\
&\mathbb{E}[\mathbf{W}(\mathbf{X} - \mu)(\mathbf{X} - \mu)^T\mathbf{W}^T] = \mathbf{I} \\
&\mathbf{W}\Sigma\mathbf{W}^T = \mathbf{I} \\
&\mathbf{W}\Sigma\mathbf{W}^T\mathbf{W} = \mathbf{W} \\
&\mathbf{W}^T\mathbf{W} = \boldsymbol{\Sigma}^{-1}
\end{aligned} \tag{18}$$

From (18) it is clear that the Cholesky decomposition provides a suitable (but not unique) method of finding $\mathbf{W}$.

7.3 Cholesky Decomposition

Cholesky decomposition is an efficient way to implement LU decomposition for symmetric matrices, which allows us to find the square root. Consider $\mathbf{AX} = \mathbf{b}$, $\mathbf{A} = [a_{ij}]_{n\times n}$, and $a_{ij} = a_{ji}$, then the Cholesky decomposition of $\mathbf{A}$ is given by $\mathbf{A} = \mathbf{L}\mathbf{L}'$ where

$$\mathbf{L} = \begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{bmatrix} \tag{19}$$

Let l_{ki} be the k^{th} row and i^{th} column entry of $\mathbf{L}$, then

$$l_{ki} = \begin{cases} 0, & k < i \\ \sqrt{a_{ii} - \sum_{j=1}^{i-1} l_{kj}^2}, & k = i \\ \frac{1}{l_{ii}}(a_{ki} - \sum_{j=1}^{i-1} l_{ij}l_{kj}), & i < k \end{cases}$$

7.4 4 DOF Independent Normal Distribution

Consider the four-dimensional vector $\mathbf{Y} = (S, T, U, V)$ which has components that are normally distributed, centered at zero, and independent. Then S, T, U, and V all have density functions

$$f_S(x;\sigma) = f_T(x;\sigma) = f_U(x;\sigma) = f_V(x;\sigma) = \frac{e^{-x^2/(2\sigma^2)}}{\sqrt{2\pi\sigma^2}}. \tag{20}$$

Let $\mathbf{X}$ be the length of $\mathbf{Y}$, which means $\mathbf{X} = \sqrt{S^2 + T^2 + U^2 + V^2}$. Then $\mathbf{X}$ has the cumulative distribution function

$$F_X(x;\sigma) = \int\int\int\int_{H_x} f_S(\mu;\sigma)f_T(\mu;\sigma)f_U(\mu;\sigma)f_V(\mu;\sigma)\ dA, \tag{21}$$

where H_x is the four-dimensional sphere

$$H_x = \Big\{(s,t,u,v) \ : \ \sqrt{s^2+t^2+u^2+v^2} < x\Big\}. \tag{22}$$

We then can write the integral in polar representation

$$\begin{aligned} F_X(x;\sigma) &= \frac{1}{4\pi^2\sigma^4}\int_0^\pi\int_0^\pi\int_0^{2\pi}\int_0^x r^3 e^{-r^2/(2\sigma^2)}\sin(\theta)\sin(\phi)\cos(\psi)\ dr\ d\theta\ d\phi\ d\psi \\ &= \frac{1}{2\sigma^4}\int_0^x r^3 e^{-r^2/(2\sigma^2)}\ dr. \end{aligned} \tag{23}$$

The probability density function of **X** is the derivative of its cumulative distribution function so we use the funamental theorem of calculus on (23) to finally arrive at

$$f_X(x;\sigma) = \frac{d}{dx}F_X(x;\sigma) = \ \frac{1}{2\sigma^4}x^3 e^{-x^2/(2\sigma^2)}. \tag{24}$$