6-2023

A Comparison of Quaternion Neural Network Backpropagation Algorithms

Jeremiah Bill
Air Force Institute of Technology

Bruce A. Cox
Air Force Institute of Technology

Lance Champaign
Air Force Institute of Technology

Recommended Citation

Bill, J., Cox, B. A., & Champagne, L. (2023). A comparison of quaternion neural network backpropagation algorithms. Expert Systems with Applications, 232(1 December), Art. 120448. https://doi.org/10.1016/j.eswa.2023.120448

Expert Systems With Applications 232 (2023) 120448

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

A comparison of quaternion neural network backpropagation algorithms

Jeremiah Bill [*], Bruce A. Cox, Lance Champagne

Department of Operational Sciences, Air Force Institute of Technology, 2950 Hobson Way, Wright-Patterson AFB, 45433, OH, USA

A R T I C L E I N F O

Keywords:
Clifford algebra
Clifford neural networks
Design of experiments
Geometric algebra
GHR calculus
Hypercomplex numbers
Latin hypercube sampling
Quaternion neural networks
Space filling designs

A B S T R A C T

This research paper focuses on quaternion neural networks (QNNs) - a type of neural network wherein the weights, biases, and input values are all represented as quaternion numbers. Previous studies have shown that QNNs outperform real-valued neural networks in basic tasks and have potential in high-dimensional problem spaces. However, research on QNNs has been fragmented, with contributions from different mathematical and engineering domains leading to unintentional overlap in QNN literature. This work aims to unify existing research by evaluating four distinct QNN backpropagation algorithms, including the novel GHR-calculus backpropagation algorithm, and providing concise, scalable implementations of each algorithm using a modern compiled programming language. Additionally, the authors apply a robust Design of Experiments (DoE) methodology to compare the accuracy and runtime of each algorithm. The experiments demonstrate that the Clifford Multilayer Perceptron (CMLP) learning algorithm results in statistically significant improvements in network test set accuracy while maintaining comparable runtime performance to the other three algorithms in four distinct regression tasks. By unifying existing research and comparing different QNN training algorithms, this work develops a state-of-the-art baseline and provides important insights into the potential of QNNs for solving high-dimensional problems.

1. Introduction

Over the last several decades, research in artificial intelligence and machine learning (AI/ML) has progressed at a breakneck pace. The democratization of machine learning has led to state-of-the-art results and advances in a variety of high-dimensional scientific problem domains. As computing resources, data collection, and machine learning algorithms have increased, so too have the size and scale of problems addressed via machine learning methodologies. This work investigates various backpropagation algorithms utilized to train quaternion-valued neural networks; a line of research that seeks to reduce the dimensionality of neural networks through the inclusion of quaternion valued weights thereby improving the accuracy and generalizability of trained networks in high-dimensional problem domains.

The quaternions $\mathbb{H}$ are a 4-dimensional number system originally introduced in Hamilton (1844) as a direct extension of the complex numbers $\mathbb{C}$. While the complex numbers are ubiquitous in signal processing and electrical engineering applications due to the succinct representation of 2-dimensional vector rotations in the complex plane, the quaternions provide a similarly succinct representation of vector rotation in 3-dimensional space. Hence quaternions are often used in robotics, control theory, computer graphics, and even quantum computing (Kuipers, 1999). Quaternion-valued neural networks incorporate quaternions and quaternion algebra directly into a neural network structure to perform neural computation in high-dimensional problem domains. Such networks have demonstrated improvements in terms of network size, expressive power, and network accuracy in a wide variety of domains (Parcollet, Morchid, & Linarès, 2020).

In modern mathematics, quaternion algebra resides under the umbrella of what are known as hypercomplex number systems. The study of hypercomplex numbers has a disjointed history, as is evident in Fig. 1. Several hypercomplex number systems grew out of specific engineering and mathematical physics problems, such as the quaternions, spin algebras, and tensor algebras. Attempts to unify these systems under a common language of mathematics has also been disjointed, resulting in Grassmann Algebras, Clifford Algebras, and Geometric Algebras (Chappell et al., 2016). Each of these frameworks provides a common mathematical language that describes the algebraic structure and operations of hypercomplex numbers. The terms “Clifford algebra” and “geometric algebra” are generally used synonymously, but the confluence of terms in the literature can often be confusing.

1.1. Hypercomplex neural networks

In the field of neurocomputing, various hypercomplex number systems have been utilized in neural networks in an attempt to combat

* Corresponding author.
E-mail addresses: jeremiah.bill@afit.edu (J. Bill), bruce.cox@afit.edu (B.A. Cox), lance.champagne@afit.edu (L. Champagne).

https://doi.org/10.1016/j.eswa.2023.120448
Received 31 January 2023; Received in revised form 8 May 2023; Accepted 8 May 2023
Available online 25 June 2023

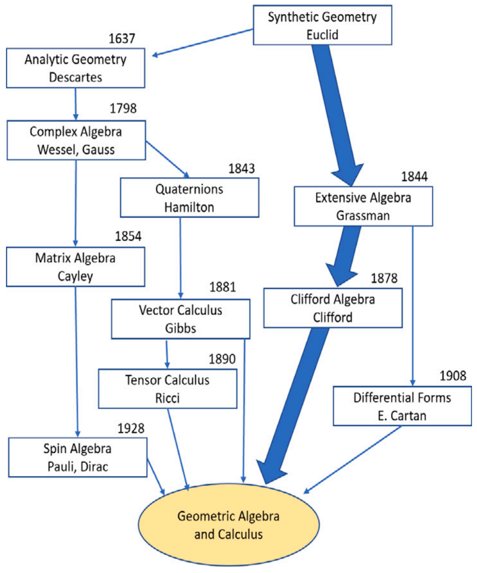


Fig. 1. Timeline of vector algebras recreated from Chappell, Iqbal, Hartnett, and Abbott (2016).

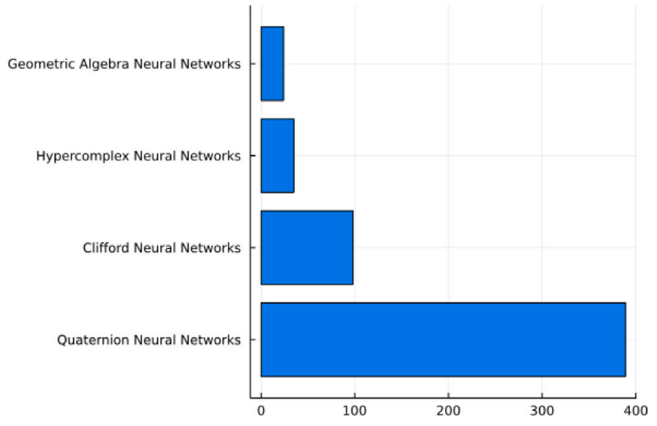


Fig. 2. Google Scholar search results by search term (as of August 2022).

the so-called "curse of dimensionality". These hypercomplex neural networks (HNNs) are neural network structures wherein the weights, biases, and inputs all consist of hypercomplex numbers as opposed to real numbers. Some examples include complex-valued neural networks (Hirose, 1992), quaternion-valued neural networks (Arena, Fortuna, Occhipinti, & Xibilia, 1994), and octonion-valued neural networks (Popa, 2016). Unfortunately, HNN research suffers from some of the same issues as that of hypercomplex numbers themselves. Contributions to HNN research have been published under various fields using a variety of index terms including Clifford-valued neural networks, geometric algebra neural networks, and architecture specific works such as quaternion-valued neural networks. As is evident in Fig. 2, quaternion-valued neural networks represent the lion's share of HNN research due to their broad applicability to specific engineering problems in computer vision (Kusamichi, Isokawa, Matsui, Ogawa, & Maeda, 2004; Shen, Zhang, Huang, Wei, & Zhang, 2020), robotics (Takahashi, 2018), and natural language processing (Parcollet et al., 2016; Tay et al., 2019).

Recent research in each of these fields has demonstrated that quaternion neural networks provide several benefits over traditional real-valued neural networks. In particular, since each quaternion element contains four real-valued components, RGB color image datasets can easily be cast into quaternion representations wherein each pixel is represented as a single pure quaternion as opposed to a 3-tuple of values. Parcollet, Morchid, and Linarès (2018) successfully use quaternion convolutional neural networks (QCNNs) to process color images in this manner, demonstrating that QCNNs trained with gray-scale only images were able to successfully reconstruct unseen color images during testing. The proposed network achieved higher reconstruction performance than a real-valued CNN while containing fewer overall parameters and weight connections.

In addition, quaternions are ubiquitous in robotics applications. Quaternion-based rigid body dynamic models avoid many of the pitfalls of other representations such as Euler angles (Wehage, 1984). In Cao, Li, and Zhong (2022), the authors construct a quaternion differential encoder–decoder network to predict 3D human motion. The authors demonstrate that quaternion networks successfully capture multi-order information in the quaternion space, providing for a better model of 3D motion than the real-valued model. The proposed network outperforms several advanced recurrent neural network models on several benchmark human motion datasets.

Moreover, a novel research field in machine learning combines the use of swarm intelligence and metaheuristic algorithms with machine learning methodologies (Bacanin et al., 2022; Malakar, Ghosh, Bhowmik, Sarkar, & Nasipuri, 2020). In Bill, Champagne, Cox, and Bihl (2021), the authors demonstrate that metaheuristic search techniques such as genetic algorithms can successfully train quaternion neural networks, bypassing the need for complex quaternion backpropagation rules in some simple settings. Finally, researchers have implemented a variety of specialized quaternion and Clifford-valued neural network architectures. Examples include quaternion and Clifford-valued recurrent neural networks (Liu, Zheng, Lu, Cao, & Rutkowski, 2020; Xia, Liu, Kou, & Wang, 2022), quaternion attention networks (Shahadat & Maida, 2021), and quaternion transformer networks (Tay et al., 2019). In each instance, the quaternion neural networks have shown improved performance over their real-valued counterparts in a variety of public and benchmark datasets.

While quaternion neural networks have become the most studied form of HNNs, the quaternions can be viewed as a special case of the geometric or Clifford algebras. Hence, any advancement that applies in general to Clifford-valued neural networks also applies to quaternion-valued neural networks. Even so, a comparison of recent QNN (Parcollet et al., 2020) and Geometric Algebra (Bayro-Corrochano, 2021; Breuils, Tachibana, & Hitzer, 2022; Hitzer, Nitta, & Kuroe, 2013) surveys reveals two distinct lines of research that are nearly disjoint. These disparate lines of research have led to a certain amount of unintentional overlap in HNN research. As an example, a brief review of QNN research reveals three distinct derivations of the quaternion backpropagation algorithm (Arena et al., 1994; Buchholz & Sommer, 2008; Nitta, 1995), each with slight differences. In addition, the novel Generalized Hamilton-Real (G$\mathbb{HR}$) calculus (Xu, Jahanchahi, Took, & Mandic, 2015) provides yet another quaternion backpropagation algorithm using a full quaternion calculus. While each of Arena et al. (1994), Buchholz and Sommer (2008), Nitta (1995) and Xu, Jahanchahi, et al. (2015) demonstrate the ability of QNNs trained with the four respective backpropagation algorithms to outperform equivalent real-valued neural networks on a variety of problems, no work has been done to compare and contrast the performance of these four training algorithms against each other.

This work provides such an evaluation and comparison of the current techniques used to train quaternion-valued neural network learning algorithms. In particular:

1. This work unifies the disparate lines of research between the geometric (Clifford) algebra community and the quaternion neural network community by formalizing the four current QNN backpropagation algorithms in use in the literature with common notation. This study places particular emphasis on the

differences between "split" backpropagation algorithms versus the full quaternion backpropagation algorithm which the novel $\mathbb{GHR}$ calculus provides.

2. This study offers a comprehensive comparison of the four backpropagation algorithms, highlighting the strengths and weaknesses of each update rule. To achieve this, smart sampling techniques and design of experiments methodologies are used to train tuned neural networks using each of the four methods on four distinct optimization test functions. Through multiple replications of the training and test process, the study demonstrates that the Clifford Multilayer Perceptron (CMLP) (Buchholz & Sommer, 2008) backpropagation algorithm provides a statistically significant improvement in neural network accuracy over the other three algorithms. Furthermore, the CMLP formulation offers the strongest theoretical results and the highest level of generality, which are discussed in Section 5.
3. Finally, this study provides implementations of each training method in a modern, dynamically-typed, just-in-time compiled programming language, Julia (Bezanson, Karpinski, Shah, & Edelman, 2012), with a robust scientific computing ecosystem. Notably, to the best of the authors' knowledge these implementations represent the first ever deep (> 1 hidden layer) implementations of the $\mathbb{GHR}$ calculus update rules and the first extension of adaptive gradient optimization methods to the $\mathbb{GHR}$ calculus learning algorithm. Additionally, these implementations can easily be extended to GPU computing environments using Julia's CUDA package (Besard, Foket, & De Sutter, 2018) or the Julia Flux (Innes, 2018; Innes et al., 2018) machine learning library for future work in state-of-the-art research.

The contributions of this work are threefold: first, this paper provides the first ever deep (i.e. greater than 1 hidden layer) implementation of $\mathbb{GHR}$-based backpropagation to train quaternion neural networks. Second, the authors show empirically, and verify via statistical testing, that even minor differences in algorithm implementations can have a significant impact on the performance of quaternion neural networks due to the non-commutative nature of quaternion multiplication. Finally, through rigorous statistical comparison tests, this paper establishes the CMLP algorithm as the current state-of-the-art training method for quaternion neural network regression problems, resulting in statistically significant improvements in network accuracy and improved generalizability versus other split and full calculus quaternion algorithms.

The rest of this article is organized as follows: Section 2 provides a brief background on the quaternions, quaternion neural networks, and the quaternion backpropagation algorithms in use in the literature. Section 3 presents a detailed overview of the experimental methodology used to assess the four algorithms while Section 4 outlines the results of the experiments. Finally, Section 5 provides several conclusions and recommendations for future work.

2. Preliminaries

This section provides background information on the quaternions, quaternion neural networks, Clifford Algebra, and the novel $\mathbb{GHR}$-calculus. The intent of this section is to provide a standalone introduction to quaternion algebra as it relates to neural networks. The information presented here generally follows the chronological development of quaternion neural network research, starting with an introduction to quaternion algebra which was first developed by Sir William Rowan Hamilton in 1844 (Hamilton, 1844), continuing through a discussion of the $\mathbb{GHR}$-calculus gradient optimization methods and the $\mathbb{GHR}$-based backpropagation algorithm.

2.1. Quaternion algebra

The quaternion numbers (denoted by $\mathbb{H}$) are a four-dimensional extension of the complex numbers. Each quaternion $\bar{q}$ consists of a real part and three imaginary parts, so that the quaternions form an isomorphism with $\mathbb{R}^4$ with basis elements 1, $\bar{i}$, $\bar{j}$, and $\bar{k}$:

$$\bar{q} = r + x\bar{i} + y\bar{j} + z\bar{k}. \tag{1}$$

Note that in the discussion that follows and throughout the rest of this paper, all quaternions are represented with bar notation, while scalars in $\mathbb{R}$ are represented with lowercase, unbolded letters. In addition, the single real and three imaginary components of a quaternion are represented with the variables (r, x, y, z), respectively.

Quaternions form a generalization of the complex numbers, wherein the three imaginary components $\bar{i}$, $\bar{j}$, and $\bar{k}$ follow the same construct as $\mathbf{i}$ in $\mathbb{C}$:

$$\bar{i}^{(2)} = \bar{j}^{(2)} = \bar{k}^{(2)} = -1. \tag{2}$$

However, the three imaginary basis components must also satisfy the following rules:

$$\bar{j}\bar{k} = -\bar{k}\bar{j} = \bar{i}, \tag{3}$$

$$\bar{k}\bar{i} = -\bar{i}\bar{k} = \bar{j}, \tag{4}$$

$$\bar{i}\bar{j} = -\bar{j}\bar{i} = \bar{k}. \tag{5}$$

These rules demonstrate that quaternion multiplication $\otimes$, known as the *Hamilton Product*, is non-commutative. The Hamilton Product is easily derived using the basis multiplication rules in Eqs. (3)–(5) and the distributive property. In reduced form, the Hamilton product of two quaternions $\bar{q}_1$ and $\bar{q}_2$ is defined as:

$$\begin{aligned}\bar{q}_1 \otimes \bar{q}_2 :=&(r_1r_2 - x_1x_2 - y_1y_2 - z_1z_2)\\ &+(r_1x_2 + x_1r_2 + y_1z_2 - z_1y_2)\bar{i}\\ &+(r_1y_2 - x_1z_2 + y_1r_2 + z_1x_2)\bar{j}\\ &+(r_1z_2 + x_1y_2 - y_1x_2 + z_1r_2)\bar{k}.\end{aligned} \tag{6}$$

In addition, the element-wise (Hadamard) product, is defined as:

$$\bar{q}_1 \odot \bar{q}_2 := r_1 \cdot r_2 + (x_1 \cdot x_2)\bar{i} + (y_1 \cdot y_2)\bar{j} + (z_1 \cdot z_2)\bar{k}. \tag{7}$$

Similarly, quaternion addition is defined using the element-wise addition operation. For two quaternions $\bar{q}_1, \bar{q}_2 \in \mathbb{H}$, the sum $\bar{q}_1 + \bar{q}_2$ is defined as,

$$\begin{aligned}\bar{q}_1 + \bar{q}_2 :=&(r_1 + r_2) + (x_1 + x_2)\bar{i}+\\ &(y_1 + y_2)\bar{j} + (z_1 + z_2)\bar{k}.\end{aligned} \tag{8}$$

The notion of a quaternion conjugate is analogous to that of a complex conjugate in $\mathbb{C}$. The conjugate of a quaternion $\bar{q} = r + x\bar{i} + y\bar{j} + z\bar{k}$ is given by $\bar{q}^* = r - x\bar{i} - y\bar{j} - z\bar{k}$. The norm of a quaternion is equivalent to the Euclidean norm in $\mathbb{R}$ and is given by

$$\|\bar{q}\| := \sqrt{\bar{q}\bar{q}^*} = \sqrt{r^2 + x^2 + y^2 + z^2}. \tag{9}$$

With this quaternion norm, one can also define a notion of distance $d(\bar{q}, \bar{p})$ between two quaternions $\bar{q}$ and $\bar{p}$ as

$$d(\bar{q}, \bar{p}) := \|\bar{q} - \bar{p}\|. \tag{10}$$

The quaternion norm is also used to define the multiplicative inverse of any quaternion:

$$\bar{q}^{(-1)} = \frac{\bar{q}^*}{\|\bar{q}\|^{(2)}}. \tag{11}$$

It is easy to verify that $\bar{q}^{(-1)}\bar{q} = \bar{q}\bar{q}^{(-1)} = 1$. In the special case where $\bar{q}$ is a unit quaternion (i.e., $\|\bar{q}\| = 1$), then $\bar{q}^{(-1)} = \bar{q}^*$.

2.2. Quaternion neural networks

The mathematical machinery described in Section 2.1 provides all of the necessary components to build a basic quaternion neural network wherein the inputs, weights, and biases of the network are all composed of quaternions as opposed to real numbers. The basic Quaternion Multilayer Perceptron (QMLP) was first presented by Arena et al. (1994) in 1994. The authors proposed a basic quaternion neural network model that generally mirrors the standard MLP with two major caveats.

First, Liouville's Theorem (Nelson, 1961), a fundamental result from Complex Analysis, indicates that any bounded entire function in the complex plane is constant. Hornik (1991) demonstrated that the two fundamental requirements for neural network activation functions are that they are "bounded and nonconstant" (Hornik, 1991). Hence, early work in Complex Multilayer Perceptrons revealed that traditional activation functions in the complex plane (and any higher-dimensional Clifford algebra) were problematic. As a workaround, Benvenuto and Piazza (1992) proposed a "split" activation function, defined in $\mathbb{C}$ as:

$$\bar{\sigma}(\cdot) = \sigma(\cdot) + i\sigma(\cdot), \tag{12}$$

where $\sigma(\cdot)$ represents any real-valued neural network activation function. Arena, Fortuna, Re, and Xibilia (1993) proved a universal approximation theorem for complex-valued neural networks using this split (or "component-wise") activation function construct, hence bypassing the issues posed by Liouville's Theorem for complex MLPs. Finally, in order to perform backpropagation using split activation functions, Benvenuto and Piazza (1992) proposed a "pseduo-gradient" update wherein the gradient is computed component-wise:

$$\dot{\bar{\sigma}}(\cdot) = \dot{\sigma}(\cdot) + i\dot{\sigma}(\cdot), \tag{13}$$

where $\dot{\sigma}(\cdot)$ represents the gradient of the real-valued function $\sigma(\cdot)$. Arena et al. (1994) extended these ideas directly to the quaternion domain, presenting a basic split quaternion backpropagation algorithm. Shortly thereafter, the authors presented a succinct proof of a Universal Approximation Theorem for QMLPs (Arena, Fortuna, Muscato, & Xibilia, 1997) similar to their proof for the complex-valued case (Arena et al., 1993) and Cybenko (1989)'s and Hornik, Stinchcombe, and White (1989)'s results for real-valued neural networks.

Nitta (1995) independently and concurrently proposed a QMLP model using the same split activation and pseudo-gradient construct as Arena et al. (1994). While Nitta (1995)'s QMLP model was identical in structure to Arena et al. (1994), it utilized a slightly different weight update rule in the proposed backpropagation algorithm that leveraged the quaternion conjugate of the network weights.

Finally, Buchholz and Sommer (2008) generalized the complex and quaternion backpropagation algorithms in the Clifford algebra literature, resulting in the Clifford Multilayer Perceptron (CMLP) neural network and associated backpropagation algorithm. The authors prove that CMLPs are universal approximators for any non-degenerate Clifford algebra. The presentations by Arena et al. (1994), Nitta (1995), and Buchholz's CMLP (Buchholz & Sommer, 2008) all contain slight differences and represent the three split backpropagation methods explored in this comparison. Recently, Xu, Zhang, and Mandic (2015) presented a full quaternion calculus, known as the generalized Hamilton-Real (G$\mathbb{HR}$) calculus. G$\mathbb{HR}$ calculus allows for a novel quaternion backpropagation algorithm that utilizes split activation functions but a full quaternion gradient in the weight update step. For brevity, Buchholz and Sommer (2008)'s CMLP algorithm is presented in full in Section 2.3 as the representative split backpropagation algorithm discussed in this paper. The G$\mathbb{HR}$ algorithm is presented in Section 2.4 and the Arena and Nitta backpropagation rules are presented and discussed in Appendix A.1.

2.3. Clifford algebra

Clifford algebras, which are often referred to synonymously as geometric algebras, provide a generalization of the algebras over $\mathbb{R}$, $\mathbb{C}$, $\mathbb{H}$, and several other hypercomplex algebras. While a full exposition of Clifford algebras is beyond the scope of this work, the interested reader should consult (Porteous et al., 1995) for an overview of the subject and Sommer (2013) for a review of several modern applications of Clifford algebras to fields such as robotics and computer vision.

In Buchholz and Sommer (2008), the authors present the Basic Clifford Neuron (BCN) model and accompanying Clifford Multilayer Perceptron (CMLP). In generality, the authors demonstrate that the CMLP model and backpropagation algorithm hold for neural networks created using any non-degenerate Clifford algebra, which includes $\mathbb{R}$, $\mathbb{C}$, $\mathbb{H}$, the split-biquaternions $\mathbb{H}\oplus\mathbb{H}$ (an 8-dimensional algebra), as well as many other high-dimensional algebras. In particular, with a quaternion-valued CMLP, the network structure and forward pass rules reduce to the following structure.

Notation:

- M: number of layers in the network
- l: layer index $0, 1, \ldots, M$
- N_l: number of neurons in the lth layer
- n: neuron index in a given layer
- $\bar{x}_n^{(l)} = r_{x_n^{(l)}} + x_{x_n^{(l)}}\bar{i} + y_{x_n^{(l)}}\bar{j} + z_{x_n^{(l)}}\bar{k}$: output of the nth neuron in the lth layer. Note that the output is a quaternion.
- $\bar{w}_{nm}^{(l)}$: the quaternion weight between the nth neuron of the lth layer and the mth neuron of the $(l-1)$-th layer
- $\bar{\theta}_n^{(l)}$: the quaternion bias term of the nth neuron in the lth layer

Forward Pass:
For $l = 1, \ldots, M$ and $n = 1, \ldots, N_l$

$$\bar{s}_n^{(l)} = \sum_{m=0}^{N_{l-1}} \bar{w}_{nm}^{(l)} \otimes \bar{x}_m^{l-1} + \bar{\theta}_n^{(l)}, \tag{14}$$

$$\bar{x}_n^{(l)} = \bar{\sigma}(\bar{s}_n^{(l)}), \tag{15}$$

where

$$\bar{\sigma}(\cdot) = \sigma(\cdot) + \sigma(\cdot)\bar{i} + \sigma(\cdot)\bar{j} + \sigma(\cdot)\bar{k} \tag{16}$$

represents the split activation function, with $\sigma(\cdot)$ being any real-valued activation function. In addition, the "pseudo-derivative" of $\bar{\sigma}(\cdot)$ is given by:

$$\dot{\bar{\sigma}}(\cdot) = \dot{\sigma}(\cdot) + \dot{\sigma}(\cdot)\bar{i} + \dot{\sigma}(\cdot)\bar{j} + \dot{\sigma}(\cdot)\bar{k} \tag{17}$$

Finally, the loss function is defined as the split error of the output layer:

$$\begin{aligned} E &= \frac{1}{2}\sum_{n=1}^{N_M} (\bar{y}_n - \bar{x}_n^{(M)})^2 \\ &= \frac{1}{2}\sum_{n=1}^{N_M} (r_{y_n} - r_{x_n})^2 + (x_{y_n} - x_{x_n})^2\bar{i} \\ &\qquad + (y_{y_n} - y_{x_n})^2\bar{j} + (z_{y_n} - z_{x_n})^2\bar{k} \end{aligned} \tag{18}$$

Error Backpropagation (CMLP):

To perform the backpropagation step, the error "pseudo-gradient" must first be defined as:

$$\frac{\partial E}{\partial \bar{w}_{nm}} = \frac{\partial E}{\partial r_{nm}} + \frac{\partial E}{\partial x_{nm}}\bar{i} + \frac{\partial E}{\partial y_{nm}}\bar{j} + \frac{\partial E}{\partial z_{nm}}\bar{k}, \tag{19}$$

wherein the partial derivatives of the loss function are computed component-wise for each quaternion weight and bias value in the network. Under this definition, the backpropagation rules for each layer $l = 1, \dots, M$ and each neuron $n = 1, \dots, N_l$ are given by:

$$\bar{\delta}_n^{(l)} = \begin{cases} \left(\bar{y}_n - \bar{x}_n^{(M)}\right), & l = M \\ \dot{\bar{\sigma}}\left(s_n^{(l)}\right) \odot \sum_{h=1}^{N_{l+1}} \bar{w}_{hn}^{*(l+1)} \otimes \bar{\delta}_h^{(l+1)}, & l \neq M \end{cases} \tag{20}$$

Similar to (A.2), the CMLP weights are updated utilizing a multiplication by the output of the previous layer:

$$\bar{w}_{nm}^{(l)} = \bar{w}_{nm}^{(l)} + \eta \cdot \bar{\delta}_n^{(l)} \otimes \bar{x}_m^{*(l-1)}, \tag{21}$$

$$\bar{\theta}_n^{(l)} = \bar{\theta}_n^{(l)} + \eta \cdot \bar{\delta}_n^{(l)}. \tag{22}$$

In matrix–vector form, the rules are:

$$\bar{\boldsymbol{\delta}}^{(l)} = \begin{cases} \bar{\mathbf{e}} = (\bar{\mathbf{y}} - \bar{\mathbf{x}}^{(M)}), & l = M \\ \dot{\bar{\sigma}}\left(\mathbf{s}^{(l)}\right) \odot \left[\left(\bar{\mathbf{W}}^{(l+1)}\right)^H \times \left(\bar{\boldsymbol{\delta}}^{(l+1)}\right)\right], & l \neq M \end{cases} \tag{23}$$

with weight updates given by:

$$\bar{\mathbf{W}}^{(l)} = \bar{\mathbf{W}}^{(l)} + \eta \cdot \bar{\boldsymbol{\delta}}^{(l)} \times \left(\bar{\mathbf{x}}^{(l-1)}\right)^H, \tag{24}$$

$$\bar{\boldsymbol{\theta}}^{(l)} = \bar{\boldsymbol{\theta}}^{(l)} + \eta \cdot \bar{\boldsymbol{\delta}}^{(l)}, \tag{25}$$

In Buchholz and Sommer (2008), the authors demonstrate how the conjugation operation $(\cdot)^*$ in $\mathbb{R}$ is simply the identity operator, hence the update rules as presented reduce exactly to standard backpropagation in $\mathbb{R}$. The same cannot be said for either the Arena et al. (1994) or Nitta (1995) rules due to differences in how the layers are indexed in the Arena et al. (1994) rules and the use of the pre-activation output in Eq. (A.8) in the Nitta (1995) update rules. Furthermore, Buchholz and Sommer (2008) proves a universal approximation theorem that holds for any non-degenerate Clifford algebra, which includes the quaternions and quaternion neural networks, but provides much stronger statements than the proofs presented in Arena et al. (1997).

2.4. GHR calculus

The generalized Hamilton-Real (G$\mathbb{HR}$) calculus is a novel calculus first proposed in 2015 by Xu, Jahanchahi, et al. (2015). The authors provided a second proof of the calculus in 2016 (Xu, Gao, & Mandic, 2016) as well as several critical results establishing a general optimization framework using the calculus in Xu, Xia, and Mandic (2016). Finally, Flamant, Miron, and Brie (2021) provided a complete framework for convex optimization in the quaternion domain using G$\mathbb{HR}$ calculus rules, including a full set of optimality conditions.

The G$\mathbb{HR}$ calculus rules leverage quaternion involutions and quaternion rotation to define a set of quaternion derivative rules. An involution is a function that is its own inverse. The quaternion conjugation rule defined in Section 2.1 is an example of an involution. In general, involutions play a vital role in quaternionic analysis, and Ell and Sangwine (2007) provides further details on some useful involutions.

In addition, quaternion rotation is perhaps the most used quaternion operation, and is defined as:

$$\bar{q}^{\bar{\mu}} := \bar{\mu}\bar{q}\bar{\mu}^{(-1)}. \tag{26}$$

If $\bar{\mu}$ is a pure quaternion (i.e., a quaternion with real part $r = 0$), then the rotation operation becomes an involution. As an example, rotation about each of the quaternion basis elements are the involutions shown below:

$$\begin{aligned} \bar{q}^{\bar{\imath}} &= -\bar{\imath}\bar{q}\bar{\imath} = r + x\bar{\imath} - y\bar{\jmath} - z\bar{k} \\ \bar{q}^{\bar{\jmath}} &= -\bar{\jmath}\bar{q}\bar{\jmath} = r - x\bar{\imath} + y\bar{\jmath} - z\bar{k} \\ \bar{q}^{\bar{k}} &= -\bar{k}\bar{q}\bar{k} = r - x\bar{\imath} - y\bar{\jmath} + z\bar{k}. \end{aligned} \tag{27}$$

Finally, note that for any quaternion $\bar{\mu} \in \mathbb{H}$, one can define a general orthogonal quaternion basis $\{1, \bar{\imath}^{\bar{\mu}}, \bar{\jmath}^{\bar{\mu}}, \bar{k}^{\bar{\mu}}\}$ where the following properties hold:

$$\bar{\imath}^{\bar{\mu}}\bar{\imath}^{\bar{\mu}} = \bar{\jmath}^{\bar{\mu}}\bar{\jmath}^{\bar{\mu}} = \bar{k}^{\bar{\mu}}\bar{k}^{\bar{\mu}} = \bar{\imath}^{\bar{\mu}}\bar{\jmath}^{\bar{\mu}}\bar{k}^{\bar{\mu}} = -1. \tag{28}$$

Using these principles, the G$\mathbb{HR}$ derivatives are defined as follows (note that since quaternion multiplication is non-commutative, both left and right derivatives must be defined):

Left G$\mathbb{HR}$ derivatives:

$$\frac{\partial f}{\partial \bar{q}^{\bar{\mu}}} = \frac{1}{4}\left(\frac{\partial f}{\partial r} - \frac{\partial f}{\partial x}\bar{\imath}^{\bar{\mu}} - \frac{\partial f}{\partial y}\bar{\jmath}^{\bar{\mu}} - \frac{\partial f}{\partial z}\bar{k}^{\bar{\mu}}\right) \in \mathbb{H} \tag{29}$$

$$\frac{\partial f}{\partial \bar{q}^{\bar{\mu}*}} = \frac{1}{4}\left(\frac{\partial f}{\partial r} + \frac{\partial f}{\partial x}\bar{\imath}^{\bar{\mu}} + \frac{\partial f}{\partial y}\bar{\jmath}^{\bar{\mu}} + \frac{\partial f}{\partial z}\bar{k}^{\bar{\mu}}\right) \in \mathbb{H} \tag{30}$$

Right G$\mathbb{HR}$ derivatives:

$$\frac{\partial_r f}{\partial \bar{q}^{\bar{\mu}}} = \frac{1}{4}\left(\frac{\partial f}{\partial r} - \bar{\imath}^{\bar{\mu}}\frac{\partial f}{\partial x} - \bar{\jmath}^{\bar{\mu}}\frac{\partial f}{\partial y} - \bar{k}^{\bar{\mu}}\frac{\partial f}{\partial z}\right) \in \mathbb{H} \tag{31}$$

$$\frac{\partial_r f}{\partial \bar{q}^{\bar{\mu}*}} = \frac{1}{4}\left(\frac{\partial f}{\partial r} + \bar{\imath}^{\bar{\mu}}\frac{\partial f}{\partial x} + \bar{\jmath}^{\bar{\mu}}\frac{\partial f}{\partial y} + \bar{k}^{\bar{\mu}}\frac{\partial f}{\partial z}\right) \in \mathbb{H}. \tag{32}$$

While Xu, Zhang, and Mandic (2015) provides a full derivation of the G$\mathbb{HR}$ calculus rules, Xu, Xia, and Mandic (2016) presents quaternion gradient and Hessian operators as well as several derived quaternion learning algorithms. In Xu, Xia, and Mandic (2016), the authors demonstrate the real-valued gradient of a function f is related to the quaternion-valued gradient via a simple invertible linear transformation. In addition, the authors present an explicit derivation of the backpropagation learning algorithm for a single-hidden-layer quaternion neural network using the G$\mathbb{HR}$ derivatives. In particular, given the error function E shown in Eq. (18), the quaternion gradient is given by:

$$\nabla_{\bar{\mathbf{q}}^*} E = -\frac{1}{2}\sum_{\bar{\mu}\in\{1,\bar{\imath},\bar{\jmath},\bar{k}\}} \left(\mathbf{E}^H_{\bar{\mathbf{q}}^{\bar{\mu}}}\bar{\mathbf{e}}\right)^{\bar{\mu}}, \tag{33}$$

where $\bar{\mathbf{e}}$ is the error vector $\bar{\mathbf{y}} - \bar{\mathbf{x}}^M$ from Eq. (18) and the $(\cdot)^H$ operation represents the Hermitian transpose of the quaternion error vector. This gradient can be expressed in a layer-by-layer fashion using the same notation as the previous backpropagation algorithms.

$$\bar{\boldsymbol{\delta}}^{(l)} = \begin{cases} \bar{\mathbf{e}} = (\bar{\mathbf{y}} - \bar{\mathbf{x}}^{(M)}), & l = M \\ \left(\bar{\mathbf{W}}^{(l+1)} \times \dot{\bar{\sigma}}\left(\mathbf{s}^{(l+1)}\right)\right)^H \times \bar{\boldsymbol{\delta}}^{(l+1)}, & l \neq M \end{cases} \tag{34}$$

The weight update rules for the ouput layer are:

$$\bar{\mathbf{W}}^{(l)} = \bar{\mathbf{W}}^{(l)} + \eta \cdot \bar{\boldsymbol{\delta}}^{(l)} \times \left(\bar{\mathbf{x}}^{(l-1)}\right)^H, \tag{35}$$

$$\bar{\boldsymbol{\theta}}^{(l)} = \bar{\boldsymbol{\theta}}^{(l)} + \eta \cdot \bar{\boldsymbol{\delta}}^{(l)}, \tag{36}$$

while the weight updates for the hidden layers are given by:

$$\bar{\mathbf{W}}^{(l)} = \bar{\mathbf{W}}^{(l)} + \eta \cdot \sum_{\bar{\mu}\in\{1,\bar{\imath},\bar{\jmath},\bar{k}\}} \left(\bar{\boldsymbol{\delta}}^{(l)}\right)^{\bar{\mu}} \times \left(\bar{\mathbf{x}}^{(l-1)}\right)^H, \tag{37}$$

$$\bar{\boldsymbol{\theta}}^{(l)} = \bar{\boldsymbol{\theta}}^{(l)} + \eta \cdot \sum_{\bar{\mu}\in\{1,\bar{\imath},\bar{\jmath},\bar{k}\}} \left(\bar{\boldsymbol{\delta}}^{(l)}\right)^{\bar{\mu}}. \tag{38}$$

Finally, note that the formal G$\mathbb{HR}$ derivative of the split activation function $\bar{\sigma}(\cdot)$, shown in Eq. (35), is distinct from the general split derivative used in the other three backpropagation algorithms. In Xu, Xia, and Mandic (2016), the authors prove that the G$\mathbb{HR}$ derivatives of any split activation function $\bar{\sigma}(\cdot)$ is given by:

$$\frac{\partial\bar{\sigma}(q)}{\partial q} = \frac{1}{4}\left(\frac{\partial\sigma(q)}{\partial r} - \frac{\partial\sigma(q)}{\partial x}\bar{\imath} - \frac{\partial\sigma(q)}{\partial y}\bar{\jmath} - \frac{\partial\sigma(q)}{\partial z}\bar{k}\right) \tag{39}$$

$$= \frac{1}{4}\left(\dot{\sigma}(r) + \dot{\sigma}(x) + \dot{\sigma}(y) + \dot{\sigma}(z)\right) \tag{40}$$

$$\frac{\partial\bar{\sigma}(q)}{\partial q^*} = \frac{1}{4}\left(\frac{\partial\sigma(q)}{\partial r} + \frac{\partial\sigma(q)}{\partial x}\bar{\imath} + \frac{\partial\sigma(q)}{\partial y}\bar{\jmath} + \frac{\partial\sigma(q)}{\partial z}\bar{k}\right) \tag{41}$$

$$= \frac{1}{4}\left(\dot{\sigma}(r) - \dot{\sigma}(x) - \dot{\sigma}(y) - \dot{\sigma}(z)\right). \tag{42}$$

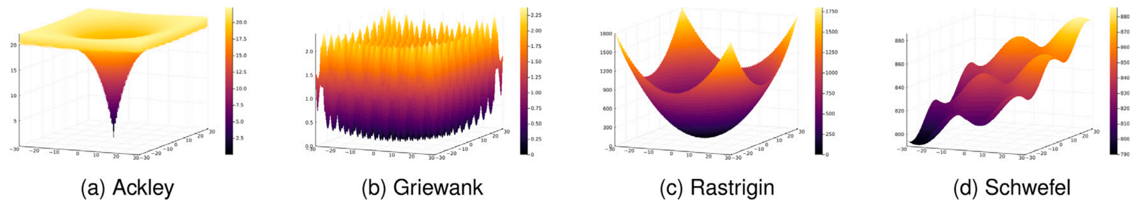


Fig. 3. Test functions.

Hence, the $\mathbb{GHR}$ derivatives of any split activation function are real-valued, thus simplifying several of the gradient calculations.

2.5. Space-filling designs and smart sampling techniques

The Design of Experiments (DoE) refers to a systematic method of experimentation which results in extracting as much information about a system as possible in the smallest number of experimental runs. This work utilizes a space-filling DoE to tune the hyperparameters of several neural network models. Space-filling designs are particularly well-suited for evaluating computer simulations or experiments (Myers, Montgomery, & Anderson-Cook, 2016) and provide a logical and systematic method of testing various combinations of hyperparameters without resorting to naive or expensive search techniques such as randomized search methods or grid search methods. To tune the hyperparameters of each network, the hyperparameters themselves are considered the factors of an experimental design. The goal, then, is to determine the optimal level of each factor that results in the best network performance.

Since computer experiments often exhibit relatively low run-to-run response variability, space-filling designs can be used to efficiently test for optimal parameter settings across the entire range of the parameter space (Montgomery, 2017). While there are a variety of different techniques for generating high quality space-filling designs, each method generally involves a combinatorial optimization problem to determine the optimal location of test points in the parameter space. Some examples of smart-sampling techniques and space filling designs include the Latin Hypercube Design (McKay, Beckman, & Conover, 1979), sphere-packing designs (Johnson, Moore, & Ylvisaker, 1990), maximum entropy designs (Shewry & Wynn, 1987), and Fast Flexible Filling (FFF) designs (Lekivetz & Jones, 2015). For an in-depth comparison of computer-generated test designs, see Johnson, Montgomery, and Jones (2011).

3. Methodology

This work compares the performance of the four quaternion backpropagation algorithms outlined in Section 2 by testing each algorithm's ability to approximate noisy nonlinear test functions. Function approximation is one of the most basic tasks in which neural networks excel. Cybenko (1989) and Hornik et al. (1989)'s universal approximation theorems demonstrate that real-valued neural networks with an appropriate topology can approximate any nonlinear function to an arbitrary degree of accuracy. The theorems in Arena, Baglio, Fortuna, and Xibilia (1995) and Buchholz and Sommer (2008) provide similar results for QMLPs and CMLPs, respectively.

Concretely, this work compares the four quaternion backpropagation algorithms by training a neural network with a fixed topology using each of the four backpropagation methods. Each network is trained and tested on four separate optimization test functions using individually tuned hyperparameters for each test function/backpropagation algorithm pairing. The topology used for each network in the comparison test is a $1 \to 5 \to 5 \to 5 \to 1$ Multilayer Perceptron model. This topology provides a sufficient number of neurons per layer for each quaternion network to learn some of the complex nonlinearities associated with each test function. In addition, the depth of the network allows for better comparisons between the runtime and scalability of each training method. To the best of the authors knowledge, this work presents the first ever deep (i.e., more than one hidden layer) implementation of the $\mathbb{GHR}$-calculus learning rules. The tuned models are compared using a variety of different metrics, including test set error, computational runtime, and training characteristics. Finally, the tuned model comparisons are validated using robust statistical tests performed on each metric.

Section 3.1 describes the four optimization test functions used in this study, while Section 3.2 describes the smart sampling techniques used to generate the training, test, and validation datasets. Section 3.3 presents two simple extensions of the Adagrad (Duchi, Hazan, & Singer, 2011) and Adadelta (Zeiler, 2012) adaptive gradient optimizers to the quaternion domain for use in the network comparisons. Section 3.4 presents a simple space-filling design used to tune the hyperparameters of each model. Finally, Section 3.5 details the metrics used to compare the performance of each network.

3.1. Test functions

In order to test the performance of each algorithm, the Ackley (2012), Griewank (1981), Schwefel (1993), and Rastrigin (1974) functions were selected from a library of standard optimization test functions. Each of these functions can be extended to an arbitrary number of input dimensions, making them ideal test cases for complex, quaternion, and hypercomplex neural networks.

In this work, each function was extended to five dimensions, i.e., each function utilized four input dimensions x_1, x_2, x_3, and x_4 and each quaternion neural network was trained to approximate the function output, $f(x_1, x_2, x_3, x_4)$. While it is impossible to picture the five-dimensional versions of each of these test functions, three-dimensional surface plots of each function are shown in Fig. 3. Even in three dimensions, the high degree of complexity and nonlinearity associated with each function are stark. Additionally, the four functions vary widely in terms of the size and scale of their surfaces over the test range. While the Ackley and Rastrigin functions appear to form smooth surfaces relative to the Griewank and Schwefel functions, in reality all four functions have highly corrugated surfaces. Zooming in on a smaller range reveals this for both functions.

For more information on each function as well as a host of other common optimization test functions, the interested reader is referred to Surjanovic and Bingham (2013).

3.2. Data generation

Each of the four functions described in Section 3.1 was sampled 30,000 times across a grid of points using an Optimal Latin Hypercube Sampling (OLHS) plan generated using Julia's LatinHypercubeSampling.jl package (Urquhart, Ljungskog, & Sebben, 2020). Similar to the space-filling designs discussed in Section 2.5, OLHS seeks to sample points evenly across the entire input space. The employment of the OLHS plan ensured good coverage of the datapoints across the four input dimensions, and initial experiments demonstrated superior training and generalization performance of models trained with OLHS sampled datasets as opposed to naive uniform random sampling.

Each dataset was split 70%/15%/15% between training, validation, and test sets, respectively. Each of the four input dimensions were standardized using the mean and standard deviation of each dimension from the training dataset. Finally, each input datapoint (x_1, x_2, x_3, x_4) was cast into a single quaternion using the following basic procedure:

$$\mathbf{x} = (x_1, x_2, x_3, x_4) \rightarrow \bar{q} = x_1 + x_2\bar{i} + x_3\bar{j} + x_4\bar{k}.$$

Following best practices for neural network training, each network was trained solely on the training dataset while the validation datasets were used to tune the hyperparameters of each network. Each network's performance on the validation dataset was also used to determine an early stopping condition for network training. Finally, after all networks were trained with tuned hyperparameters, the test set performance of each network was measured and recorded.

3.3. Adaptive gradient methods

In the real domain, a vast number of enhancements have been made to the original gradient descent algorithm. Improvements such as momentum optimization (Qian, 1999), adaptive gradient methods (Duchi et al., 2011), and regularization methods (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014) have all demonstrated consistent performance improvements over vanilla gradient descent. This work presents two simple extensions of the real-valued adaptive gradient methods Adagrad (Duchi et al., 2011) and Adadelta (Zeiler, 2012). Both methods allow for a more complete comparison of the four backpropagation algorithms and their performance using modern optimization techniques. While several works in the literature utilize adaptive learning methods with the pseudo-gradient backpropagation rules (Huang & Gai, 2020; Saoud, Al-Marzouqi, & Deriche, 2021; Yin et al., 2019), to the authors knowledge this work represents the first ever extension of adaptive step-size learning methods to the $\mathbb{GHR}$-calculus backpropagation.

Algorithm 1 Quaternion Adagrad update at time t

```
Require: Constant $\epsilon$
    for each parameter $\theta$
        compute gradient: $g_t$
        accumulate gradients: $G_t = \sum_{\tau=1}^{t} ||g_t||^2$
        update parameters: $\theta_t = \theta_{t-1} + \frac{\eta}{\sqrt{G_t+\epsilon}} \cdot g_t$
    end for
```

Algorithm 1 shows the general process for updating each weight and bias parameter using the quaternion Adagrad optimizer. The Adagrad routine relies on adaptively scaling the learning rate for each parameter, making it possible to extend Adagrad to the quaternion domain by simply accumulating the square of the norm of each gradient. Recall from Eq. (9) that the quaternion norm $\|\cdot\| : \mathbb{H} \rightarrow \mathbb{R}$, hence the Adagrad update in the quaternion domain scales the learning rate for each parameter by a value $G_t \in \mathbb{R}$. For both the Adagrad and Adadelta update rules, the constant ϵ is fixed at 1.00×10^{-9}

Algorithm 2 Quaternion Adadelta update at time t

```
Require: Constant $\epsilon$, decay rate $\rho$
Initialize accumulation variables: $E[G_0] = 0$
    for each parameter $\theta$
        compute gradient: $g_t$
        accum. gradients: $E[G_t^2] = \rho E[||g||^2]_{t-1} + (1-\rho)||g_t||^2$
        update parameters: $\theta_t = \theta_{t-1} + \frac{\eta}{\sqrt{E[G_t^2]+\epsilon}} \cdot g_t$
    end for
```

The Adadelta update rule shown in Algorithm 2 represents the sliding window Adadelta method, wherein gradient information from previous epochs is accumulated using a decay parameter of $\rho = 0.9$.

Table 1
Hyperparameter tuning results.

Architecture	Parameters	Ackley	Schwefel	Rastrigin	Griewank
Arena QMLP	LR	0.000285	0.006433	0.004264	0.000483
	Optimizer	Adadelta	Adagrad	Adadelta	Adadelta
	Activation	leakyrelu	leakyrelu	leakyrelu	swish
Nitta QMLP	LR	0.00311	0.00246	0.00451	0.00373
	Optimizer	Adadelta	Adadelta	Adadelta	Adadelta
	Activation	tanh	swish	tanh	sigmoid
CMLP	LR	0.00473	0.000483	0.000483	0.000483
	Optimizer	Adadelta	Adadelta	Adadelta	Adadelta
	Activation	sigmoid	swish	swish	swish
GHR	LR	0.00183	0.000285	0.00246	0.000483
	Optimizer	SGD	Adadelta	Adadelta	Adadelta
	Activation	tanh	leakyrelu	swish	swish

While the accumulated gradients in the Adagrad update can grow without bound, eventually causing the scaled learning rates to converge to zero, the sliding window approach of the Adadelta method prevents the accumulated gradients from growing too large.

3.4. Hyperparameter tuning

In addition to the smart-sampling techniques used to generate each experimental dataset (Section 3.2), this work employed a Fast Flexible Filling (FFF) design (Lekivetz & Jones, 2015) generated in JMP Pro 15 to tune the hyperparameters of a fixed neural network topology. The FFF test design was used to tune the hyperparameters for each backpropagation algorithm (the Arena QMLP, Nitta QMLP, CMLP, and $\mathbb{GHR}$ architectures) on each of the four test functions, resulting in a set of 16 optimal hyperparameters.

The FFF design employed three factors: the network Learning Rate, Optimizer, and hidden layer Activation Function. The Learning Rate was a continuous factor ranging from a minimum of 1×10^{-7} to a maximum of 1×10^{-2}. The two categorical factors, Optimizer and Activation Function, contained the following levels:

- *Optimizer*: Stochastic Gradient Descent (SGD), Adagrad, and Adadelta
- *Activation Function*: tanh, sigmoid, swish, and leakyrelu

The hyperparameter settings in each design point were run for 30 epochs on each network architecture and the lowest validation set error was recorded across each run as the response for each design point. The results of the hyperparameter tuning for each architecture are shown in Table 1. The design run table for the FFF design can be found in the Supplementary Materials for this paper.

3.5. Tuned network comparisons

Using the results from the hyperparameter tuning test, each network was re-initialized and re-trained using the optimal hyperparameter settings. Each network was trained for a maximum of 250 epochs, with a $PQ_{0.5}$ early stopping condition as defined in Prechelt (1998). Prechelt notes that such a criterion maximizes the average quality of solutions and provides some level of patience during unstable phases of training. Finally, to ensure that the best network parameters were returned upon termination of the training algorithm, each network was saved to disk in a binary format at each epoch when the validation set error surpassed the best validation set error seen up to that point. Upon termination of the training loop, the algorithm returned the cached network containing the optimal weight and bias values.

The test set performance of each saved model was then evaluated using the Mean Absolute Percentage Error (MAPE). The MAPE is calculated using the following formula:

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|\mathbf{y}_i - \hat{\mathbf{y}}_i|}{|\mathbf{y}_i|}, \tag{43}$$

where $\mathbf{y}_i$ represents each truth or target value and $\hat{\mathbf{y}}_i$ represents the predicted values (i.e., the output of each neural network). The MAPE allows for a scaled easily interpretable measure of test set performance across each network for each test function. This training/test process was repeated 30 times for each hyperparameter combination in order to generate summary statistics and statistical comparisons between each result.

In addition, the R^2 values for each algorithm on each dataset and the total training time for each replication were recorded. The R^2, or coefficient of determination, is calculated as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2}{\sum_{i=1}^{n}(\mathbf{y}_i - \bar{\mathbf{y}})^2}, \tag{44}$$

where $\mathbf{y}_i$ represents each truth or target value, $\hat{\mathbf{y}}_i$ represents the predicted values, and $\bar{\mathbf{y}}$ represents the mean of all target values $\mathbf{y}_i$. While the MAPE provides a measure of the accuracy and generalizability of each trained network, the R^2 value provides a measure of how well each neural network fits the underlying data distributions for each problem. Moreover, the computational runtime of each algorithm provides insight into each method's ability to scale to large, real-world problems. While several instances of pseudogradient methods have been applied to large datasets (see Parcollet, Morchid, Linares, & De Mori, 2018; Parcollet et al., 2019), to the best of the authors knowledge, no large-scale implementations have been developed for the G$\mathbb{HR}$-calculus backpropagation method and no comparisons have been made on the scalability of pseudogradient methods versus the G$\mathbb{HR}$ method.

Taken together, the R^2, MAPE, and runtime results provide a comprehensive picture of the overall performance of each algorithm. The MAPE measure is one of many metrics for assessing the accuracy of regression models. As a percentage measure, the MAPE always produces a value between 0 and 1, hence allowing for meaningful comparisons across the four algorithms as well as across the four test functions. Moreover, the R^2 values for each model provide critical insight into the amount of variance in each dataset explained by each regression model. Finally, performing multiple replications of each test function/backpropagation algorithm pairing allows for robust statistical comparisons between the results of each algorithm.

This study utilized a paired t-test to quantify the statistically significant differences between each algorithm's performance. The paired t-test contains three main assumptions:

- The observations being tested are all independent.
- The observations are approximately normally distributed.
- There are no outliers among the observations.

To ensure the independence of each observation, the random number streams for each experimental run were carefully controlled. Each test run i for $i \in (1, \dots, 30)$ was initialized with a unique non-overlapping random number stream. However, within each run i common random number streams were used to initialize each backpropagation algorithm as a variance reduction technique. Verification of the normality assumption and outlier assumption for each t-test, as well as the results from each test are summarized in Section 4 below.

4. Experimental results

All computer experiments were performed on a desktop workstation with 256 GB of RAM and an AMD Epyc 7402p 24-core processor running Ubuntu 22.04.1 LTS. All coding was performed in Julia v1.7.0. Many real-valued neural network routines from the Julia Flux machine learning package were extended to the quaternion domain with custom built training and optimization functions in Flux v0.12.8. Finally, the FFF computer designs were generated in JMP Pro v15 (SAS Institute Inc., 1989–2021).

The MAPE, R^2 and Runtime results for each test function are summarized in Table 2. The table displays the minimum, maximum, and mean values for each metric. The best mean value for each metric is highlighted in bold for each of the four test functions. The CMLP algorithm resulted in the best test set performance for the Ackley, Griewank, and Rastrigin test functions, with the lowest average MAPE and the highest R^2 values for each function. The G$\mathbb{HR}$ algorithm achieved slightly better R^2 and MAPE values on the Schwefel test function. However, the improvements that the G$\mathbb{HR}$ algorithm afforded over the CMLP algorithm on the Schwefel test function were not statistically significant, as discussed in Section 4.1.

The MAPE and R^2 values were computed using holdout data (i.e., the test set data) for each algorithm. Across all four test functions, the Nitta algorithm performed the worst in terms of both R^2 and MAPE. In each test case, the Nitta algorithm witnessed negative R^2 values, indicating final solutions with very poor fits to the underlying data distributions. On average, the Arena algorithm performed better than the Nitta algorithm on each of the four test functions but still lagged behind both the G$\mathbb{HR}$ algorithm and the CMLP algorithm.

Box and whisker plots of the MAPE, R^2, and runtime values for each test function are shown in Figs. 4–7. Each boxplot adheres to the following conventions: the box captures the 1st, 2nd, and 3rd quartiles of each dataset, where the 2nd quartile represents the median of the data. The whiskers extend out to the largest and smallest observations within 1.5 times the interquartile range (IQR), while outliers are shown as dots beyond each whisker. Finally, the mean for each dataset is shown as dashed horizontal line, and the dashed diagonal lines around the mean represent the standard deviation around the mean. In addition, scatter plots of the MAPE vs. R^2 values are shown for each test function. Since R^2 and MAPE both represent distinct scaled metrics of accuracy, these scatter plots present insightful representations of each algorithm's performance for each test function.

4.1. Statistical comparison testing

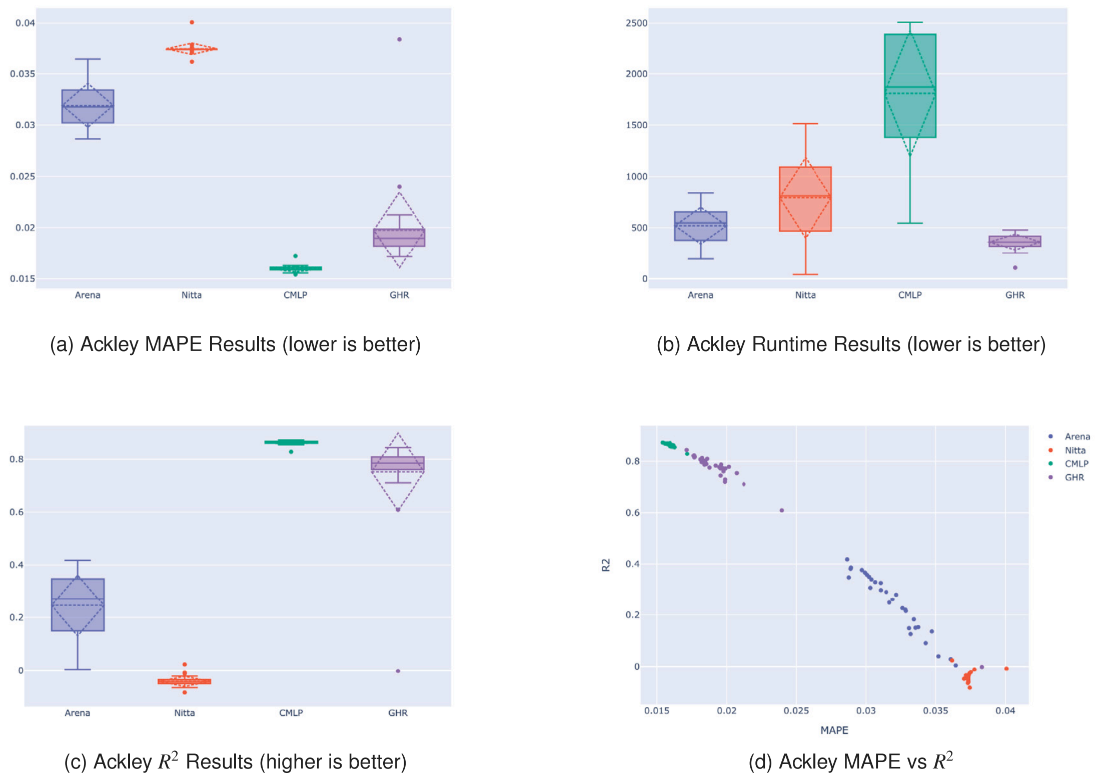
The results in Table 2 indicate that in three of the four test cases, the CMLP algorithm achieved the lowest average MAPE and highest average R^2 values. In the fourth case (the Schwefel function), the G$\mathbb{HR}$ achieved better MAPE and R^2 values by a very narrow margin. Hence, to test for statistical significance of the results, the means of each of the backpropagation methods were compared to the CMLP mean values as a baseline using a paired t-test and the Bonferroni multiple comparison method with an $\alpha = 0.1$.

Sawilowsky and Blair (1992) note that the paired t-test is robust against departures from normality in the data when the sample sizes between two populations under consideration are equal, when the sample sizes are fairly large (25 or 30), and when the tests are two-tailed. In addition, the paired t-test does not require an assumption of equal variance across the two populations being compared. However, the paired t-test does assume that there are no outliers in the data. To ensure that the test data adheres to the assumptions of the paired t-test, the Runtime, MAPE, and R^2 paired differences were carefully examined for each of the four test functions. Boxplots for the (CMLP – Nitta), (CMLP – Arena), and (CMLP – GHR) Runtime, MAPE, and R^2 values are shown in Fig. 8. Any outliers indicated on the boxplots (i.e. data points that were greater than or less than 1.5 times the interquartile range) were removed, and the subsequent paired differences were plotted on a normal quantile–quantile (QQ) plot, which provides an indication of the normality of the data. While some of the QQ plots indicated slight deviations from normality, the number of filtered data points (with outliers removed) for each test was greater than 25, thus aligning with the robustness requirements set forth in Sawilowsky and Blair (1992). QQ plots for the Ackley function R^2 paired differences are shown in Fig. 9, while the QQ plots for the other three functions can be found in the appendix.

The 90% confidence intervals and p-values for the statistical comparison tests are summarized in Table 3. As a reminder, using the Bonferroni multiple comparisons method, a value of $P < \frac{0.05}{3} = 0.01670$

Table 2
Function approximation results.

	MAPE			R^2			Runtime		
	Min	Max	Average	Min	Max	Average	Min	Max	Average
Ackley									
CMLP	0.015 40	0.017 19	**0.015 97**	0.8300	0.8739	**0.8645**	546.6	2507	1812
Arena	0.028 70	0.036 43	0.031 93	0.004 000	0.4174	0.2461	193.3	837.2	521.4
Nitta	0.036 20	0.040 07	0.037 42	−0.084 00	0.023 52	−0.041 64	42.37	1514	792.8
GHR	0.017 10	0.038 32	0.019 76	−0.002 000	0.8454	0.7525	10.38	478.4	**362.7**
Griewank									
CMLP	0.002 500	0.008 440	**0.004 711**	0.9996	1.0000	**0.9999**	491.8	1775	1167
Arena	0.1184	0.6240	0.3872	−0.005 000	0.9177	0.4530	24.58	831.6	**341.1**
Nitta	0.5209	0.6271	0.6042	−0.003 000	0.2742	0.038 12	123.8	2252	1621
GHR	0.005 600	0.029 90	0.012 26	0.9930	0.9998	0.9988	393.9	1751	1083
Schwefel									
CMLP	0.1490	0.1987	0.1856	0.1010	0.4723	0.2020	159.6	2739	1966
Arena	0.1977	0.2063	0.1987	0.040 00	0.1096	0.1010	2945	3272	3050
Nitta	0.1960	0.3789	0.2074	−3.286	0.1038	−0.034 98	28.39	451.4	**220.5**
GHR	0.1725	0.1973	**0.1853**	0.1050	0.3039	**0.2085**	1104	6087	5421
Rastrigin									
CMLP	0.1793	0.1825	**0.1807**	0.5040	0.5139	**0.5102**	147.1	1883	1114
Arena	0.1920	0.2753	0.2342	−0.019 00	0.3894	0.1749	55.48	259.9	**164.3**
Nitta	0.2456	0.2654	0.2621	−0.001 000	0.1003	0.016 56	31.07	2327	1404
GHR	0.1788	0.1891	0.1821	0.4900	0.5088	0.5015	53.15	541.4	434.1



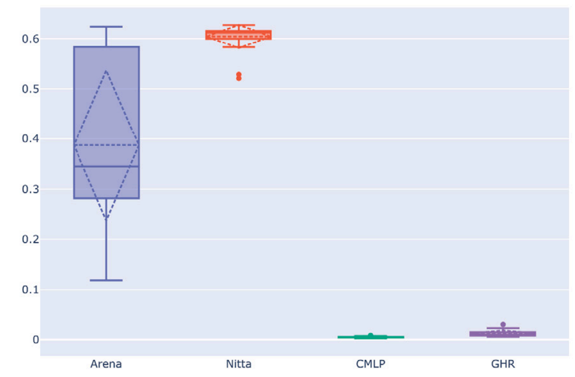
(a) Ackley MAPE Results (lower is better)
(b) Ackley Runtime Results (lower is better)
(c) Ackley R^2 Results (higher is better)
(d) Ackley MAPE vs R^2

Fig. 4. Function approximation results for the Ackley function.

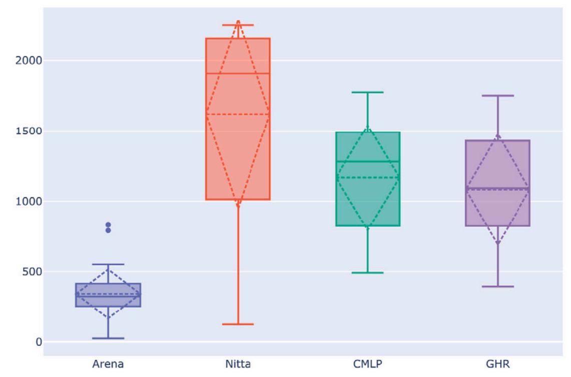
indicates that there is a statistically significant difference between the mean CMLP algorithm result and the listed algorithm's result in Table 3. Bolded values in the table indicate instances in which the CMLP algorithm witnessed a statistically significant improvement.

As Table 3 illustrates, the CMLP algorithm performed statistically significantly better in terms of MAPE and R^2 than the Nitta and Arena algorithms on all four test functions. In addition, the CMLP was significantly better than the $\mathbb{GHR}$ algorithm in three of the four test functions. On the Schwefel function, the performance between the two algorithms was statistically indistinguishable.

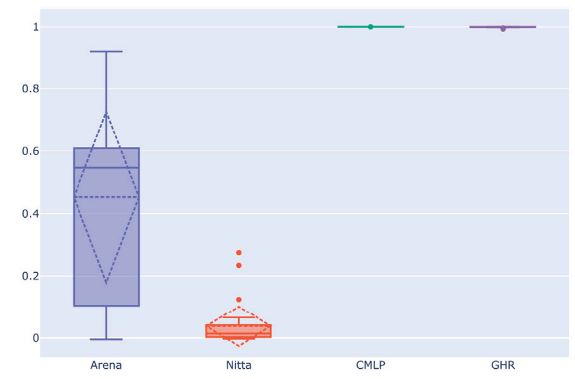
Regarding runtime, the four backpropagation methods did not show any clear trends. The total training time required to reach a stopping condition was measured in each instance. While the CMLP algorithm was never the outright winner in terms of runtime, it did perform significantly better than both the $\mathbb{GHR}$ and the Arena algorithm on the Schwefel test function, and significantly better than the Nitta algorithm on the Griewank function. Moreover, the CMLP had comparable runtime performance with the $\mathbb{GHR}$ algorithm on the Griewank test function and the Nitta algorithm on the Rastrigin test function. In all other cases, the CMLP witnessed significantly worse runtime performance. The authors suspect that this is likely due to the fact that the CMLP algorithm achieved continued improvements in training throughout the 250 epoch training window for each test function, whereas the other three algorithms failed to progress in meaningful



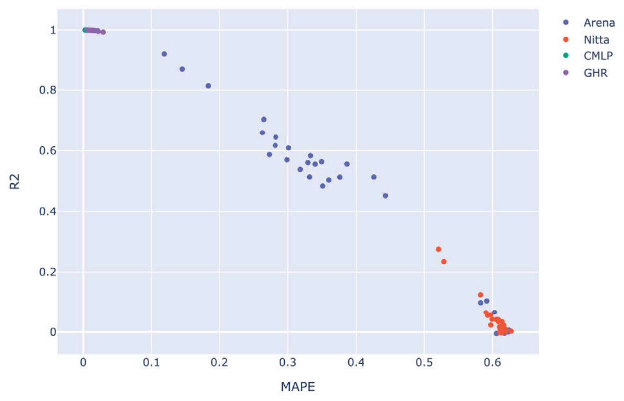
(a) Griewank MAPE Results (lower is better)



(b) Griewank Runtime Results (lower is better)



(c) Griewank R^2 Results (higher is better)



(d) Griewank MAPE vs R^2

Fig. 5. Function approximation results for the Griewank function.

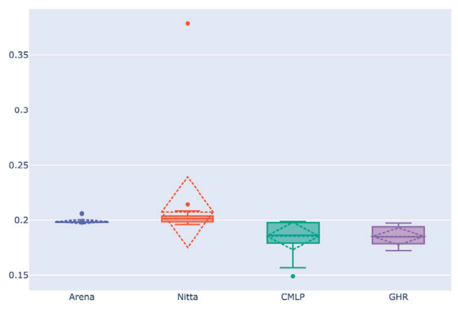
Table 3
Statistical comparison test results.

	CMLP - GHR		CMLP - Nitta		CMLP - Arena	
	90% C.I.	p-value	90% C.I.	p-value	90% C.I.	p-value
Ackley						
Runtime (s)	(1189, 1709)	3.642×10^{-13}	(726.0, 1312)	1.441×10^{-8}	(1021, 1560)	1.405×10^{-11}
MAPE	**(−0.003 433, −0.002 531)**	$\mathbf{1.678 \times 10^{-14}}$	**(−0.021 57, −0.021 37)**	$\mathbf{1.027 \times 10^{-52}}$	**(−0.016 85, −0.015 08)**	$\mathbf{5.817 \times 10^{-27}}$
R^2	**(0.065 44, 0.094 14)**	$\mathbf{1.024 \times 10^{-12}}$	**(0.9059, 0.9174)**	$\mathbf{1.475 \times 10^{-49}}$	**(0.5708, 0.6659)**	$\mathbf{5.374 \times 10^{-23}}$
Griewank						
Runtime (s)	(−95.36, 264.1)	0.3029	**(−832.3, −209.3)**	**0.000 835 9**	(649.8, 1003)	2.339×10^{-11}
MAPE	**(−0.008 848, −0.004 996)**	$\mathbf{9.264 \times 10^{-9}}$	**(−0.6101, −0.6025)**	$\mathbf{1.550 \times 10^{-49}}$	**(−0.4443, −0.3207)**	$\mathbf{2.666 \times 10^{-14}}$
R^2	**(0.000 356 3, 0.000 864 3)**	$\mathbf{1.175 \times 10^{-5}}$	**(0.9722, 0.9897)**	$\mathbf{1.606 \times 10^{-45}}$	**(0.4332, 0.6606)**	$\mathbf{1.250 \times 10^{-11}}$
Schwefel						
Runtime (s)	**(−3727, −3375)**	$\mathbf{1.462 \times 10^{-22}}$	(2074, 2254)	6.185×10^{-25}	**(−869.8, −610.8)**	$\mathbf{4.810 \times 10^{-12}}$
MAPE	(−0.003 507, 0.007 502)	0.4234	**(−0.021 86, −0.009 955)**	$\mathbf{1.922 \times 10^{-6}}$	**(−0.018 26, −0.007 927)**	$\mathbf{4.037 \times 10^{-6}}$
R^2	(−0.062 81, 0.024 02)	0.3259	**(0.078 77, 0.1700)**	$\mathbf{1.386 \times 10^{-6}}$	**(0.060 30, 0.1416)**	$\mathbf{5.512 \times 10^{-6}}$
Rastrigin						
Runtime (s)	(528.9, 896.3)	1.967×10^{-9}	(−583.4, 1.622)	0.034 25	(808.1, 1164)	6.670×10^{-13}
MAPE	**(−0.002 051, −0.000 207 8)**	**0.010 49**	**(−0.084 10, −0.082 35)**	$\mathbf{2.413 \times 10^{-42}}$	**(−0.063 49, −0.043 62)**	$\mathbf{8.237 \times 10^{-13}}$
R^2	**(0.006 652, 0.009 746)**	$\mathbf{1.945 \times 10^{-12}}$	**(0.4974, 0.5097)**	$\mathbf{1.032 \times 10^{-40}}$	**(0.2798, 0.3908)**	$\mathbf{4.979 \times 10^{-14}}$

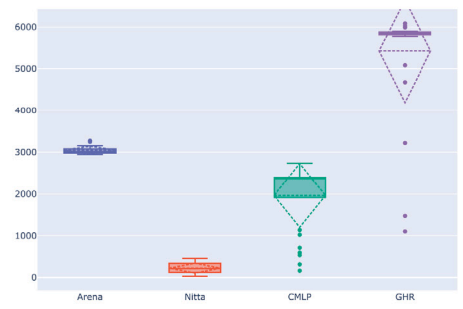
improving directions or diverged from good minima and triggered the early stopping conditions outlined in Section 3.

5. Conclusions

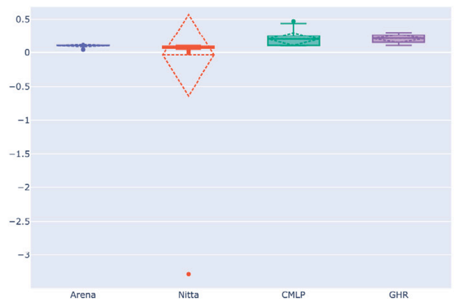
While each of the four backpropagation algorithms provide a viable means of training quaternion neural networks, the subtle differences between each algorithm result in statistically significant differences in final network results. This study employed a robust design of experiments methodology across a variety of test functions to compare the algorithms. The results show that the CMLP algorithm outperforms the other three methods in terms of test set performance while maintaining adequate runtime performance. In general, the full calculus backpropagation rules developed using the $\mathbb{GHR}$ method did not result in a significant improvement in terms of accuracy or runtime over the CMLP algorithm.



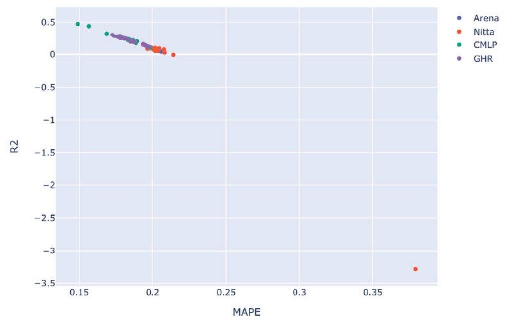
(a) Schwefel MAPE Results (lower is better)



(b) Schwefel Runtime Results (lower is better)

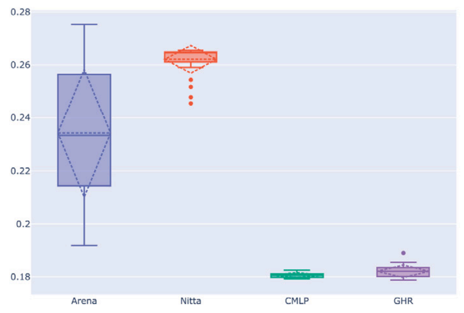


(c) Schwefel R^2 Results (higher is better)

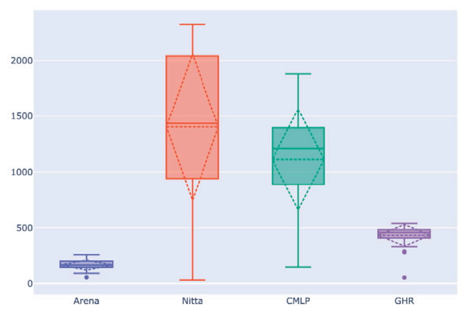


(d) Schwefel MAPE vs R^2

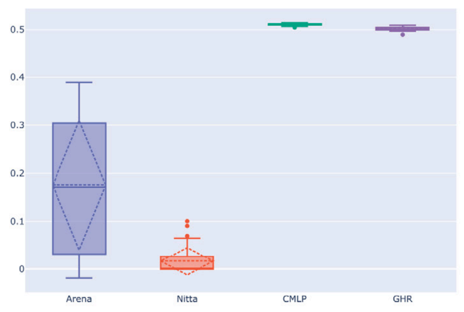
Fig. 6. Function approximation results for the Schwefel function.



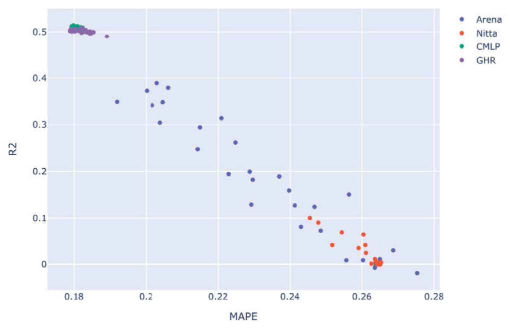
(a) Rastrigin MAPE Results (lower is better)



(b) Rastrigin Runtime Results (lower is better)

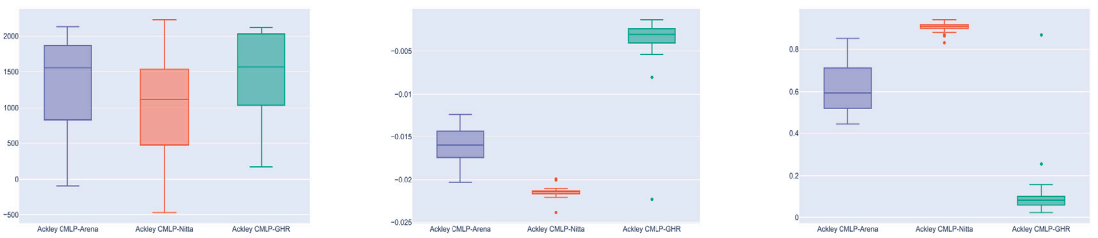


(c) Rastrigin R^2 Results (higher is better)



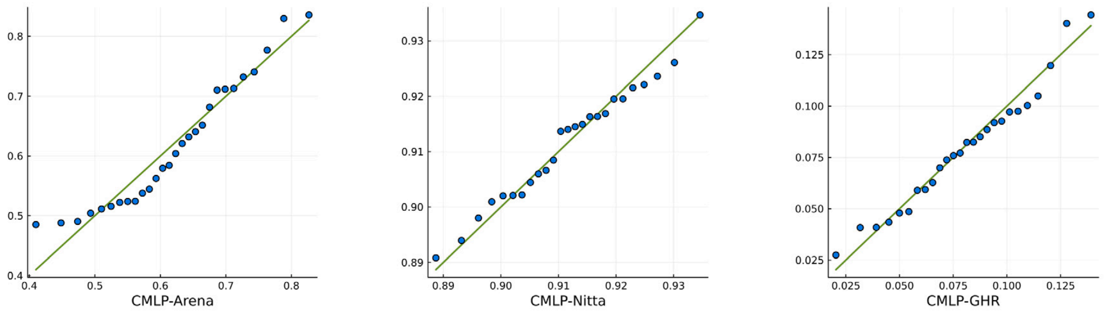
(d) Rastrigin MAPE vs R^2

Fig. 7. Function approximation results for the Rastrigin function.



(a) Ackley Runtime Paired Differences (b) Ackley MAPE Paired Differences (c) Ackley R^2 Paired Differences

Fig. 8. Boxplots of the paired differences for each metric.



(a) Ackley CMLP - Arena R^2 (b) Ackley CMLP - Nitta R^2 (c) Ackley CMLP - GHR R^2

Fig. 9. Ackley R^2 QQ plots.

While the Nitta and Arena methods are the most often cited quaternion backpropagation papers with 96 and 65 citations, respectively, the CMLP update rules provide the strongest theoretical support and the highest level of generality. In Buchholz and Sommer (2008), the authors demonstrate the universal approximation capabilities of CMLP networks and show that the CMLP backpropagation algorithm applies to neural networks in any non-degenerate Clifford algebra. Hence, the CMLP backpropagation algorithm reduces exactly to ordinary backpropagation in $\mathbb{R}$ and extends to neural networks in $\mathbb{C}$, $\mathbb{H}$, $\mathbb{H} \oplus \mathbb{H}$, and a host of other algebras in higher dimensions.

Due to the CMLP's strong theoretical foundation, statistically significant improvement in trained network accuracy, and comparable runtime results the authors recommend the CMLP architecture be used as the basis for any future HNN regression tasks. The algorithm is easy to implement for algebra specific architectures such as quaternion-valued neural networks and also provides a simple basis for creating a generic HNN framework using the language of geometric or Clifford algebra that is extendable to hypercomplex neural networks of any dimension. Such a framework would open HNN research to the full range of modern geometric algebra research, which includes a broad range of applications to robotics, color image processing, and multispectral image processing (Bayro-Corrochano, 2021; Hitzer et al., 2013).

However, there are several limitations of this study that must be considered. First, this study focused solely on the performance of the four backpropagation algorithms applied to regression problems. Hence, the authors can make no claims regarding the generalizability of the results to machine learning classification problems. A necessary next step is the examination of the various quaternion backpropagation methods to basic classification tasks such as pattern recognition or image classification. Second, the study only considered four distinct regression problems, which may not be representative of all possible regression tasks. While the superlative performance of the CMLP algorithm strongly indicates an overall trend, future work could consider a wider range of regression problems to ensure the robustness of the results.

Finally, there is a current gap in the HNN literature regarding the interpretability of results. As noted in Section 1, HNNs have demonstrated superlative performance over real-valued neural networks in a host of problem domains. However, very little remains known as to why HNNs are able to achieve this improved performance. There is a current need for experimental and theoretical results explaining the mechanics of optimization in hypercomplex domains.

All code and computer design files for this experiment can be found in the supplemental electronic material for this paper and HNN researchers are invited to use the code base as a starting point or inspiration for future developments in HNN research and applications.

CRediT authorship contribution statement

Jeremiah Bill: Conceptualization, Methodology, Software, Formal analysis, Writing – original draft, Visualization. **Bruce A. Cox:** Conceptualization, Methodology, Writing – review & editing, Supervision. **Lance Champagne:** Conceptualization, Methodology, Validation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request

Appendix A. Arena and Nitta QMLP algorithms

A.1. Arena QMLP

The construction and forward pass rules for the Arena QMLP are identical to the Clifford Multilayer Perceptron listed in Section 2.3. However, the backpropagation rules are slightly different.

Error Backpropagation (Arena QMLP):
For each layer $l = 1, \dots, M$ and each neuron $n = 1, \dots, N_l$:

$$\bar{\delta}_n^{(l)} = \begin{cases} (\bar{y}_n - \bar{x}_n^{(M)}), & l = M \\ \sum_{h=1}^{N_{l+1}} \bar{w}_{hn}^{(l+1)} \otimes \left(\bar{\delta}_h^{(l+1)} \odot \dot{\bar{\sigma}}\left(s_n^{(l+1)}\right)\right), & l \neq M \end{cases} \tag{A.1}$$

Finally, the weight and bias update rules are given by:

$$\bar{w}_{nm}^{(l)} = \bar{w}_{nm}^{(l)} + \eta \cdot \bar{\delta}_n^{(l)} \otimes \bar{x}_m^{*(l-1)}, \tag{A.2}$$

$$\bar{\theta}_n^{(l)} = \bar{\theta}_n^{(l)} + \eta \cdot \bar{\delta}_n^{(l)}, \tag{A.3}$$

where η is the learning rate and $\bar{x}_m^{*(l-1)}$ is the quaternion conjugate of $\bar{x}_m^{(l-1)}$, i.e., the conjugate of the quaternion input into the lth layer of the network.

These rules can also be represented in matrix–vector notation using quaternionic vectors and matrices, represented in bold, as follows:

$$\bar{\boldsymbol{\delta}}^{(l)} = \begin{cases} \bar{\mathbf{e}} = (\bar{\mathbf{y}} - \bar{\mathbf{x}}^{(M)}), & l = M \\ \left(\bar{\mathbf{W}}^{(l+1)}\right)' \times \left(\bar{\boldsymbol{\delta}}^{(l+1)} \odot \dot{\bar{\sigma}}\left(\mathbf{s}^{(l+1)}\right)\right), & l \neq M \end{cases} \tag{A.4}$$

Using the conjugate (Hermitian) transpose operation $(\cdot)^H$, the weight update is then given by:

$$\bar{\mathbf{W}}^{(l)} = \bar{\mathbf{W}}^{(l)} + \eta \cdot \bar{\boldsymbol{\delta}}^{(l)} \times \left(\bar{\mathbf{x}}^{(l-1)}\right)^H, \tag{A.5}$$

$$\bar{\theta}^{(l)} = \bar{\theta}^{(l)} + \eta \cdot \bar{\delta}^{(l)}, \tag{A.6}$$

Arena et al. (1994)'s update rules have been implemented in a variety of neural network structures and have been shown to improve on real-valued neural networks with an equivalent number of trainable parameters in tasks such as chaotic time series prediction (Arena et al., 1995), color image classification (Yin et al., 2019), and many others.

While Arena et al. (1994) presented the first QMLP model, Nitta (1995) independently and concurrently proposed a QMLP model using the same split activation and loss function construct as Arena et al. (1994). Nitta (1995)'s QMLP model was identical in structure to Arena et al. (1994)'s but utilized a slightly different weight update rule in the proposed backpropagation algorithm that leveraged the quaternion conjugate of the network weights. Nitta (1995)'s proposed backpropagation algorithm is presented below and the full derivation of the algorithm can be found in Nitta (1995) and Parcollet et al. (2019), with a succinct presentation of the algorithm in Parcollet et al. (2020):

Error Backpropagation (Nitta QMLP):
For each layer $l = 1, \dots, M$ and each neuron $n = 1, \dots, N_l$:

$$\bar{\delta}_n^{(l)} = \begin{cases} (\bar{y}_n - \bar{x}_n^{(M)}), & l = M \\ \sum_{h=1}^{N_{l+1}} \bar{w}_{hn}^{*(l+1)} \otimes \left(\bar{\delta}_h^{(l+1)} \odot \dot{\bar{\sigma}}\left(s_n^{(l+1)}\right)\right), & l \neq M \end{cases} \tag{A.7}$$

Finally, the weight and bias update rules are given by:

$$\bar{w}_{nm}^{(l)} = \bar{w}_{nm}^{(l)} + \eta \cdot \bar{\delta}_n^{(l)} \otimes \bar{s}_m^{*(l-1)}, \tag{A.8}$$

$$\bar{\theta}_n^{(l)} = \bar{\theta}_n^{(l)} + \eta \cdot \bar{\delta}_n^{(l)}. \tag{A.9}$$

In matrix–vector notation, the differences between the Arena et al. (1994) and Nitta (1995) rules become more apparent:

$$\bar{\boldsymbol{\delta}}^{(l)} = \begin{cases} \bar{\mathbf{e}} = (\bar{\mathbf{y}} - \bar{\mathbf{x}}^{(M)}), & l = M \\ \left(\bar{\mathbf{W}}^{(l+1)}\right)^H \times \left(\bar{\boldsymbol{\delta}}^{(l+1)} \odot \dot{\bar{\sigma}}\left(\mathbf{s}^{(l+1)}\right)\right), & l \neq M \end{cases} \tag{A.10}$$

with weight updates given by:

$$\bar{\mathbf{W}}^{(l)} = \bar{\mathbf{W}}^{(l)} + \eta \cdot \bar{\boldsymbol{\delta}}^{(l)} \times \left(\bar{\mathbf{s}}^{(l-1)}\right)^H, \tag{A.11}$$

$$\bar{\theta}^{(l)} = \bar{\theta}^{(l)} + \eta \cdot \bar{\delta}^{(l)}. \tag{A.12}$$

In particular, there are two key differences to note between Arena et al. (1994)'s backpropagation algorithm versus Nitta (1995)'s backpropagation algorithm: the first is the use of the quaternion conjugate of the weights in the calculation of $\delta^{(l)}$ for each layer, as was previously discussed. In addition, the two weight update rules themselves are slightly different, as can be seen in Eqs. (A.2) and (A.8). Under the Arena rules, the weights of a given layer are updated utilizing the quaternion conjugate of the input into that layer. Under the Nitta rules, the weights of a given layer are updated utilizing the quaternion conjugate of the weighted sums from the previous network layer (i.e., the output of the previous network layer prior to passing through that layer's activation function).

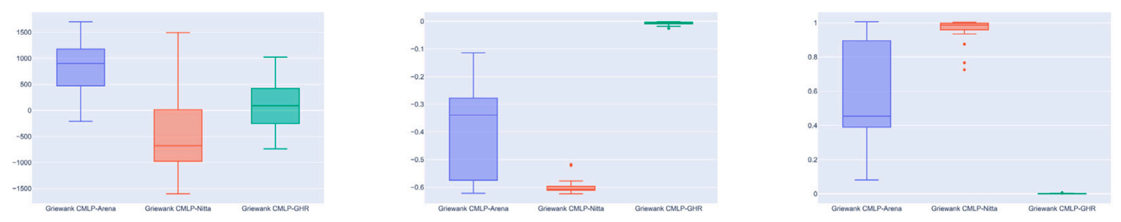
These two slight changes result in different performance between the two algorithms. Most modern works tend to favor Nitta (1995)'s derivation, but both algorithms have been cited consistently over the years. While researchers can generally show performance improvements between equivalent real-valued and quaternion-valued neural networks, the discrepancy between these two quaternion learning algorithms could provide an opportunity for even further optimization. This work seeks to elucidate the true differences between both backpropagation methods listed above, as well as a backpropagation method developed for Clifford Algebras and the novel $\mathbb{GHR}$-calculus backpropagation algorithm, which utilizes a true quaternion gradient in contrast to the pseudo-gradient.

Appendix B. Paired difference plots

See Figs. B.10–B.12.

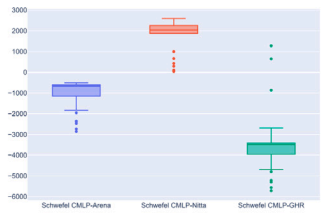
Appendix C. QQ plots

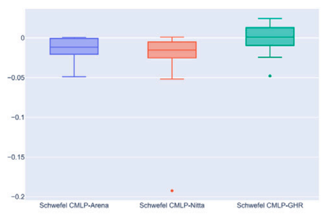
See Figs. C.13–C.24.



(a) Griewank Runtime Paired Differences (b) Griewank MAPE Paired Differences (c) Griewank R^2 Paired Differences

Fig. B.10. Griewank paired differences for each metric.


(a) Schwefel Runtime Paired Differences


(b) Schwefel MAPE Paired Differences

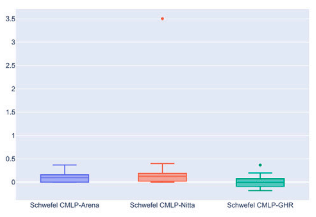
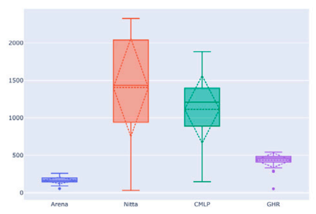
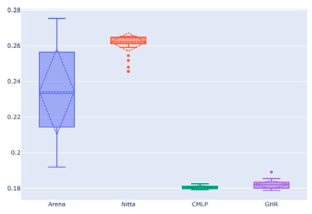

(c) Schwefel R^2 Paired Differences

Fig. B.11. Schwefel paired differences for each metric.


(a) Rastrigin Runtime Paired Differences


(b) Rastrigin MAPE Paired Differences

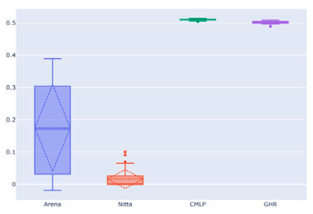
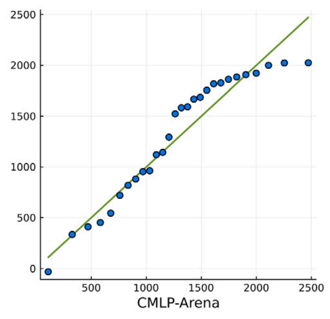
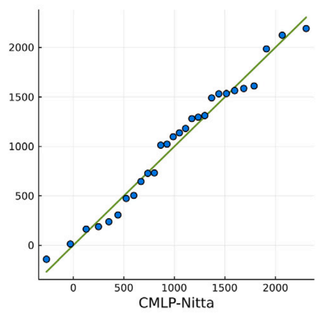

(c) Rastrigin R^2 Paired Differences

Fig. B.12. Rastrigin paired differences for each metric.


(a) Ackley CMLP - Arena Runtime


(b) Ackley CMLP - Nitta Runtime

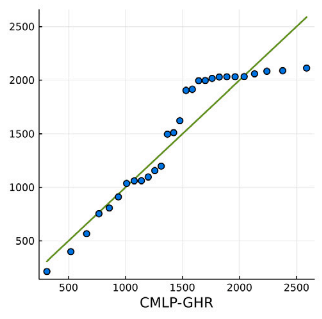
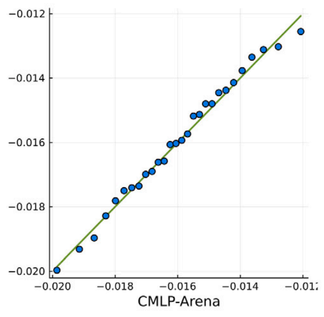
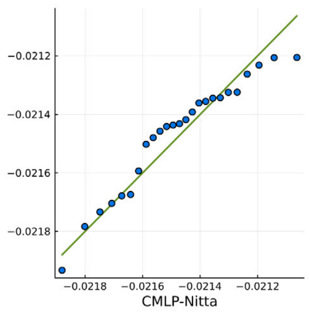

(c) Ackley CMLP - GHR Runtime

Fig. C.13. Ackley Runtime QQ plots.


(a) Ackley CMLP - Arena MAPE


(b) Ackley CMLP - Nitta MAPE

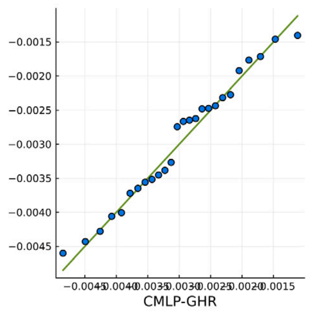
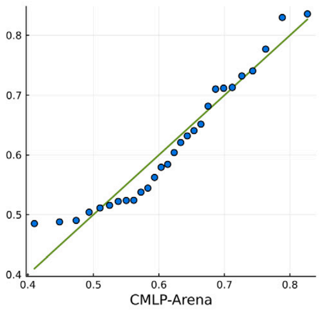
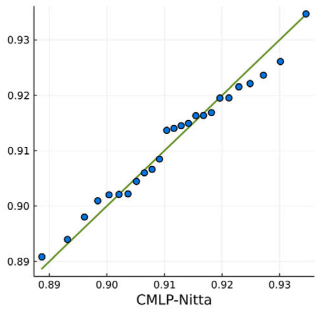

(c) Ackley CMLP - GHR MAPE

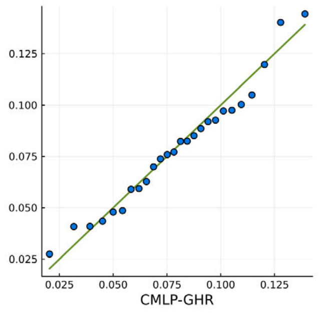
Fig. C.14. Ackley MAPE QQ plots.



(a) Ackley CMLP - Arena R^2

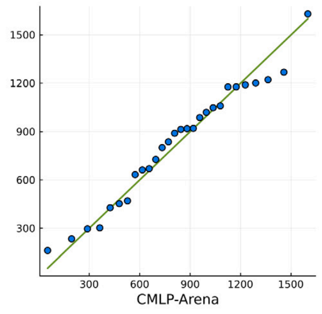


(b) Ackley CMLP - Nitta R^2

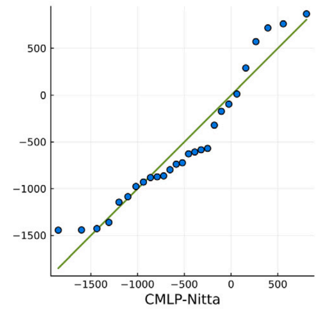


(c) Ackley CMLP - GHR R^2

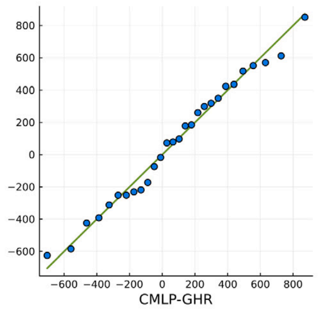
Fig. C.15. Ackley R^2 QQ plots.



(a) Griewank CMLP - Arena Runtime

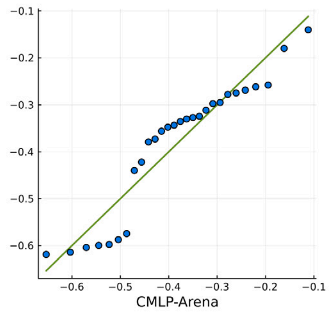


(b) Griewank CMLP - Nitta Runtime

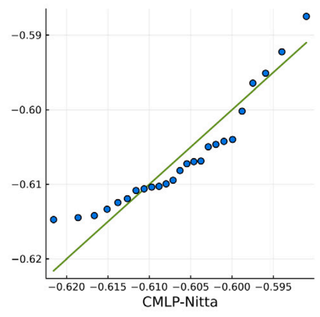


(c) Griewank CMLP - GHR Runtime

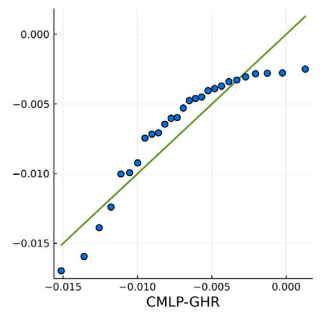
Fig. C.16. Griewank runtime QQ plots.



(a) Griewank CMLP - Arena MAPE

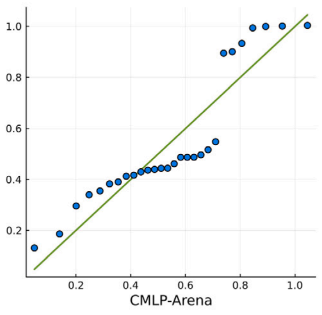


(b) Griewank CMLP - Nitta MAPE

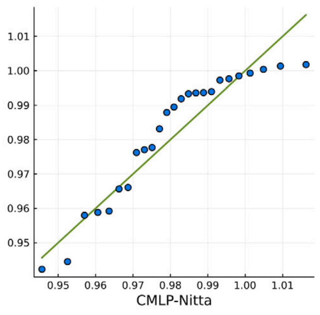


(c) Griewank CMLP - GHR MAPE

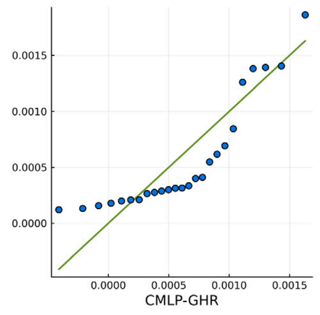
Fig. C.17. Griewank MAPE QQ plots.



(a) Griewank CMLP - Arena R^2

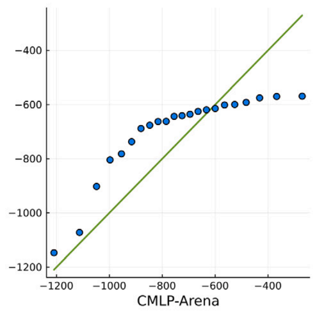


(b) Griewank CMLP - Nitta R^2

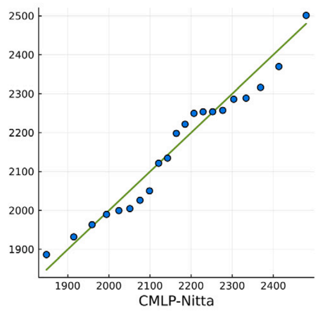


(c) Griewank CMLP - GHR R^2

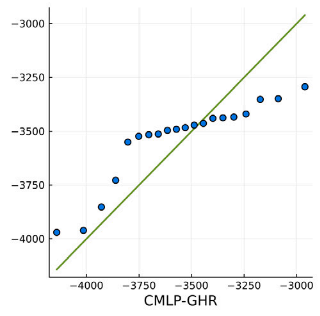
Fig. C.18. Griewank R^2 QQ plots.



(a) Schwefel CMLP - Arena Runtime

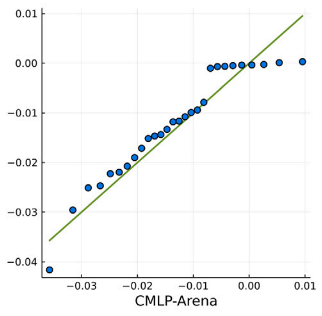


(b) Schwefel CMLP - Nitta Runtime

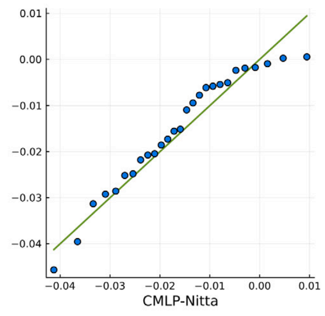


(c) Schwefel CMLP - GHR Runtime

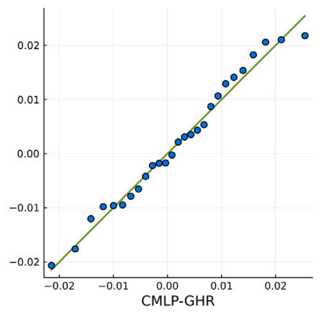
Fig. C.19. Schwefel runtime QQ plots.



(a) Schwefel CMLP - Arena MAPE

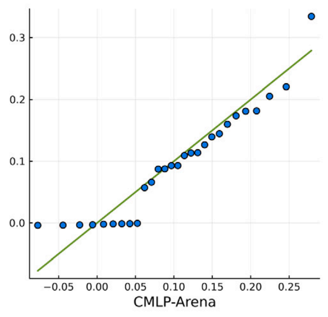


(b) Schwefel CMLP - Nitta MAPE

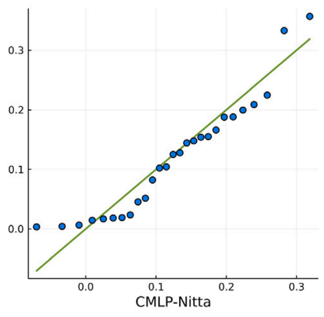


(c) Schwefel CMLP - GHR MAPE

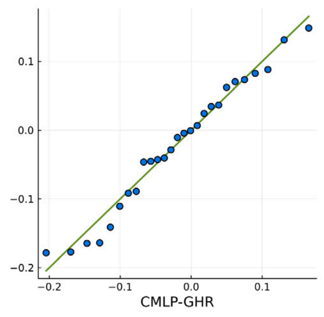
Fig. C.20. Schwefel MAPE QQ plots.



(a) Schwefel CMLP - Arena R^2

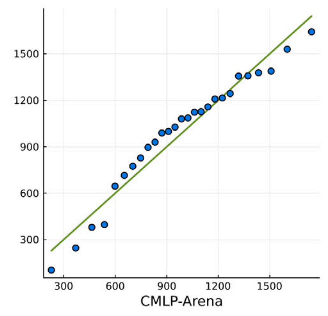


(b) Schwefel CMLP - Nitta R^2

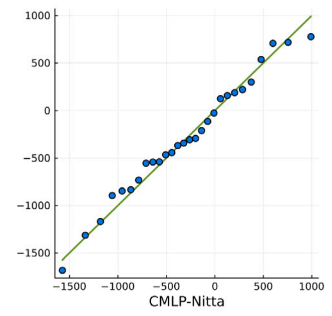


(c) Schwefel CMLP - GHR R^2

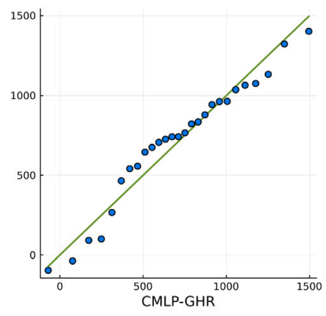
Fig. C.21. Schwefel R^2 QQ plots.



(a) Rastrigin CMLP - Arena Runtime

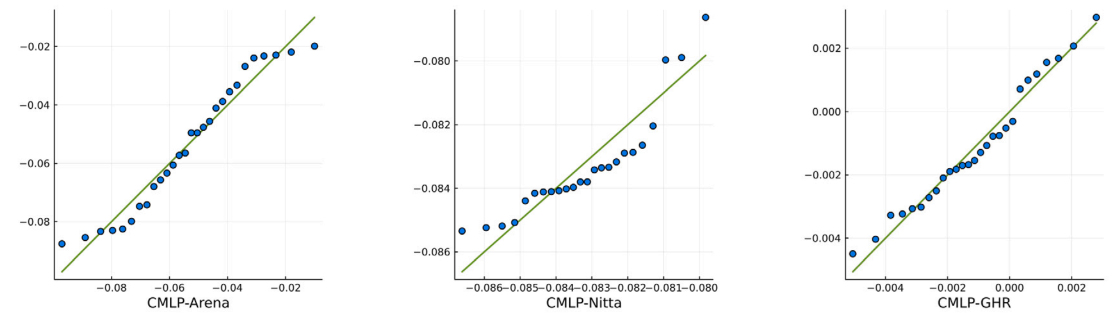


(b) Rastrigin CMLP - Nitta Runtime



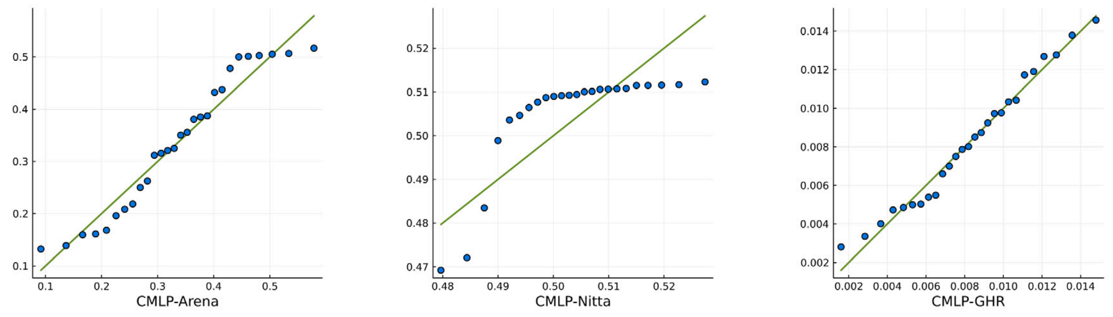
(c) Rastrigin CMLP - GHR Runtime

Fig. C.22. Rastrigin runtime QQ plots.



(a) Rastrigin CMLP - Arena MAPE (b) Rastrigin CMLP - Nitta MAPE (c) Rastrigin CMLP - GHR MAPE

Fig. C.23. Rastrigin MAPE QQ plots.



(a) Rastrigin CMLP - Arena R^2 (b) Rastrigin CMLP - Nitta R^2 (c) Rastrigin CMLP - GHR R^2

Fig. C.24. Rastrigin R^2 QQ plots.

Appendix D. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.eswa.2023.120448.

References

Ackley, D. (2012). *A connectionist machine for genetic hillclimbing. vol. 28*. Springer Science & Business Media.

Arena, P., Baglio, S., Fortuna, L., & Xibilia, M. (1995). Chaotic time series prediction via quaternionic multilayer perceptrons. In *1995 IEEE international conference on systems, man and cybernetics. intelligent systems for the 21st century, vol. 2* (pp. 1790–1794). Vancouver, BC, Canada: IEEE, http://dx.doi.org/10.1109/ICSMC.1995.538035.

Arena, P., Fortuna, L., Muscato, G., & Xibilia, M. G. (1997). Multilayer perceptrons to approximate quaternion valued functions. *Neural Networks*, *10*(2), 335–342.

Arena, P., Fortuna, L., Occhipinti, L., & Xibilia, M. (1994). Neural networks for quaternion-valued function approximation. In *Proceedings of IEEE International Symposium on Circuits and Systems, vol. 6* (pp. 307–310). http://dx.doi.org/10.1109/ISCAS.1994.409587.

Arena, P., Fortuna, L., Re, R., & Xibilia, M. (1993). On the capability of neural networks with complex neurons in complex valued functions approximation. In *1993 IEEE international symposium on circuits and systems* (pp. 2168–2171). http://dx.doi.org/10.1109/ISCAS.1993.394188.

Bacanin, N., Zivkovic, M., Al-Turjman, F., Venkatachalam, K., Trojovskỳ, P., Strumberger, I., et al. (2022). Hybridized sine cosine algorithm with convolutional neural networks dropout regularization application. *Scientific Reports*, *12*(1), 1–20.

Bayro-Corrochano, E. (2021). A survey on quaternion algebra and geometric algebra applications in engineering and computer science 1995–2020. *IEEE Access*, 1. http://dx.doi.org/10.1109/ACCESS.2021.3097756, Conference Name: IEEE Access.

Benvenuto, N., & Piazza, F. (1992). On the complex backpropagation algorithm. *IEEE Transactions on Signal Processing*, *40*(4), 967–969. http://dx.doi.org/10.1109/78.127967, Conference Name: IEEE Transactions on Signal Processing.

Besard, T., Foket, C., & De Sutter, B. (2018). Effective extensible programming: Unleashing Julia on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, http://dx.doi.org/10.1109/TPDS.2018.2872064, arXiv:1712.03112.

Bezanson, J., Karpinski, S., Shah, V. B., & Edelman, A. (2012). Julia: A fast dynamic language for technical computing. arXiv preprint arXiv:1209.5145.

Bill, J., Champagne, L., Cox, B., & Bihl, T. (2021). Meta-heuristic optimization methods for quaternion-valued neural networks. *Mathematics*, *9*(9), 938. http://dx.doi.org/10.3390/math9090938, URL https://www.mdpi.com/2227-7390/9/9/938, Number: 9 Publisher: Multidisciplinary Digital Publishing Institute.

Breuils, S., Tachibana, K., & Hitzer, E. (2022). New applications of Clifford's geometric algebra. *Advances in Applied Clifford Algebras*, *32*(2), 17. http://dx.doi.org/10.1007/s00006-021-01196-7.

Buchholz, S., & Sommer, G. (2008). On Clifford neurons and Clifford multi-layer perceptrons. *Neural Networks*, *21*(7), 925–935. http://dx.doi.org/10.1016/j.neunet.2008.03.004.

Cao, W., Li, S., & Zhong, J. (2022). QMEDNet: A quaternion-based multi-order differential encoder–decoder model for 3D human motion prediction. *Neural Networks*, *154*, 141–151.

Chappell, J. M., Iqbal, A., Hartnett, J. G., & Abbott, D. (2016). The vector algebra war: A historical perspective. *IEEE Access*, *4*, 1997–2004. http://dx.doi.org/10.1109/ACCESS.2016.2538262.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, *2*(4), 303–314.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*(7).

Ell, T. A., & Sangwine, S. J. (2007). Quaternion involutions and anti-involutions. *Computers & Mathematics with Applications*, *53*(1), 137–143. http://dx.doi.org/10.1016/j.camwa.2006.10.029.

Flamant, J., Miron, S., & Brie, D. (2021). A general framework for constrained convex quaternion optimization. arXiv:2102.02763 [eess, math], URL http://arxiv.org/abs/2102.02763.

Griewank, A. O. (1981). Generalized descent for global optimization. *Journal of Optimization Theory and Applications*, *34*(1), 11–39.

Hamilton, W. R. (1844). LXXVIII. On quaternions; or on a new system of imaginaries in algebra: To the editors of the philosophical magazine and journal. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *25*(169), 489–495.

Hirose, A. (1992). Dynamics of fully complex-valued neural networks. *Electronics Letters*, *28*(16), 1492–1494.

Hitzer, E., Nitta, T., & Kuroe, Y. (2013). Applications of Clifford's geometric algebra. *Advances in Applied Clifford Algebras*, *23*(2), 377–404.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, *4*(2), 251–257. http://dx.doi.org/10.1016/0893-6080(91)90009-T.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366.

Huang, X., & Gai, S. (2020). Banknote classification based on convolutional neural network in quaternion wavelet domain. *IEEE Access*, *8*, 162141–162148. http://dx.doi.org/10.1109/ACCESS.2020.3021181.

Innes, M. (2018). Flux: Elegant machine learning with Julia. *Journal of Open Source Software*, http://dx.doi.org/10.21105/joss.00602.

Innes, M., Saba, E., Fischer, K., Gandhi, D., Rudilosso, M. C., Joy, N. M., et al. (2018). Fashionable modelling with flux. CoRR abs/1811.01457, URL https://arxiv.org/abs/1811.01457, arXiv:1811.01457.

Johnson, R. T., Montgomery, D. C., & Jones, B. A. (2011). An expository paper on optimal design. *Quality Engineering*, *23*(3), 287–301.

Johnson, M. E., Moore, L. M., & Ylvisaker, D. (1990). Minimax and maximin distance designs. *Journal of Statistical Planning and Inference*, *26*(2), 131–148.

Kuipers, J. B. (1999). *Quaternions and rotation sequences: A primer with applications to orbits, aerospace, and virtual reality*. Princeton University Press.

Kusamichi, H., Isokawa, T., Matsui, N., Ogawa, Y., & Maeda, K. (2004). A new scheme for color night vision by quaternion neural network. In *Proceedings of the 2nd international conference on autonomous robots and agents, vol. 1315*. Citeseer.

Lekivetz, R., & Jones, B. (2015). Fast flexible space-filling designs for nonrectangular regions. *Quality and Reliability Engineering International*, *31*(5), 829–837.

Liu, Y., Zheng, Y., Lu, J., Cao, J., & Rutkowski, L. (2020). Constrained quaternion-variable convex optimization: A quaternion-valued recurrent neural network approach. *IEEE Transactions on Neural Networks and Learning Systems*, *31*(3), 1022–1035. http://dx.doi.org/10.1109/TNNLS.2019.2916597, Conference Name: IEEE Transactions on Neural Networks and Learning Systems.

Malakar, S., Ghosh, M., Bhowmik, S., Sarkar, R., & Nasipuri, M. (2020). A GA based hierarchical feature selection approach for handwritten word recognition. *Neural Computing and Applications*, *32*, 2533–2552.

McKay, M. D., Beckman, R. J., & Conover, W. J. (1979). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code.. *Technometrics*, *21*(2), 239–245.

Montgomery, D. C. (2017). *Design and analysis of experiments*. John wiley & sons.

Myers, R. H., Montgomery, D. C., & Anderson-Cook, C. M. (2016). *Response surface methodology: Process and product optimization using designed experiments*. John Wiley & Sons.

Nelson, E. (1961). A proof of Liouville's theorem. *Proceedings of the Americal Mathematical Society*, *12*(6), 995. http://dx.doi.org/10.1090/S0002-9939-1961-0259149-4.

Nitta, T. (1995). A quaternary version of the back-propagation algorithm. In *Proceedings of ICNN'95-international conference on neural networks, vol. 5* (pp. 2753–2756). IEEE.

Parcollet, T., Morchid, M., Bousquet, P.-M., Dufour, R., Linarès, G., & De Mori, R. (2016). Quaternion neural networks for spoken language understanding. In *2016 IEEE spoken language technology workshop* (pp. 362–368). IEEE.

Parcollet, T., Morchid, M., & Linarès, G. (2018). Quaternion convolutional neural networks for heterogeneous image processing. arXiv:1811.02656 [cs, stat], URL http://arxiv.org/abs/1811.02656.

Parcollet, T., Morchid, M., & Linarès, G. (2020). A survey of quaternion neural networks. *Artificial Intelligence Review*, *53*(4), 2957–2982. http://dx.doi.org/10.1007/s10462-019-09752-1.

Parcollet, T., Morchid, M., Linares, G., & De Mori, R. (2018). Quaternion convolutional neural networks for theme identification of telephone conversations. In *2018 IEEE spoken language technology workshop* (pp. 685–691). Athens, Greece: IEEE, http://dx.doi.org/10.1109/SLT.2018.8639676.

Parcollet, T., Ravanelli, M., Morchid, M., Linarès, G., Trabelsi, C., De Mori, R., et al. (2019). Quaternion recurrent neural networks. In *ICLR 2019*. Nouvelle Orléans, United States.

Popa, C. A. (2016). Octonion-valued neural networks. In *International conference on artificial neural networks* (pp. 435–443). Springer.

Porteous, I. R., et al. (1995). *Clifford algebras and the classical groups, no. 50*. Cambridge University Press.

Prechelt, L. (1998). Early stopping-but when? In *Neural networks: Tricks of the trade* (pp. 55–69). Springer.

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks*, *12*(1), 145–151.

Rastrigin, L. A. (1974). Systems of extremal control. *Nauka*.

Saoud, L. S., Al-Marzouqi, H., & Deriche, M. (2021). Wind speed forecasting using the stationary wavelet transform and quaternion adaptive-gradient methods. *IEEE Access*, *9*, 127356–127367. http://dx.doi.org/10.1109/ACCESS.2021.3111667.

SAS Institute Inc. , Cary, N. C. (1989–2021). JMP pro (version 15).

Sawilowsky, S. S., & Blair, R. C. (1992). A more realistic look at the robustness and type II error properties of the t test to departures from population normality. *Psychological Bulletin*, *111*(2), 352.

Schwefel, H. P. P. (1993). *Evolution and optimum seeking: The sixth generation*. John Wiley & Sons, Inc..

Shahadat, N., & Maida, A. S. (2021). Adding quaternion representations to attention networks for classification. arXiv:2110.01185 [cs], URL http://arxiv.org/abs/2110.01185.

Shen, W., Zhang, B., Huang, S., Wei, Z., & Zhang, Q. (2020). 3D-rotation-equivariant quaternion neural networks. In *Computer vision–ECCV 2020: 16th European conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XX 16* (pp. 531–547). Springer.

Shewry, M. C., & Wynn, H. P. (1987). Maximum entropy sampling. *Journal of Applied Statistics*, *14*(2), 165–170.

Sommer, G. (2013). *Geometric computing with Clifford algebras: Theoretical foundations and applications in computer vision and robotics*. Springer Science & Business Media.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(1), 1929–1958.

Surjanovic, S., & Bingham, D. (2013). Virtual library of simulation experiments: Test functions and datasets. Retrieved January 9, 2022, from http://www.sfu.ca/~ssurjano.

Takahashi, K. (2018). Remarks on control of robot manipulator using quaternion neural network. In *2018 Asia-Pacific signal and information processing association annual summit and conference* (pp. 560–565). IEEE.

Tay, Y., Zhang, A., Tuan, L. A., Rao, J., Zhang, S., Wang, S., et al. (2019). Lightweight and efficient neural natural language processing with quaternion networks. arXiv preprint arXiv:1906.04393.

Urquhart, M., Ljungskog, E., & Sebben, S. (2020). Surrogate-based optimisation using adaptively scaled radial basis functions. *Applied Soft Computing*, *88*, http://dx.doi.org/10.1016/j.asoc.2019.106050.

Wehage, R. A. (1984). Quaternions and Euler parameters — A brief exposition. In E. J. Haug (Ed.), *NATO ASI series*, *Computer aided analysis and optimization of mechanical system dynamics* (pp. 147–180). Berlin, Heidelberg: Springer.

Xia, Z., Liu, Y., Kou, K. I., & Wang, J. (2022). Clifford-valued distributed optimization based on recurrent neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 1–12. http://dx.doi.org/10.1109/TNNLS.2021.3139865, Conference Name: IEEE Transactions on Neural Networks and Learning Systems.

Xu, D., Gao, H., & Mandic, D. P. (2016). A new proof of the generalized Hamiltonian–Real calculus. *Royal Society Open Science*, *3*(9), Article 160211. http://dx.doi.org/10.1098/rsos.160211.

Xu, D., Jahanchahi, C., Took, C. C., & Mandic, D. P. (2015). Enabling quaternion derivatives: The generalized HR calculus. *Royal Society Open Science*, *2*(8), Article 150255. http://dx.doi.org/10.1098/rsos.150255.

Xu, D., Xia, Y., & Mandic, D. P. (2016). Optimization in quaternion dynamic systems: Gradient, Hessian, and learning algorithms. *IEEE Transactions on Neural Networks and Learning Systems*, *27*(2), 249–261. http://dx.doi.org/10.1109/TNNLS.2015.2440473.

Xu, D., Zhang, H., & Mandic, D. P. (2015). Convergence analysis of an augmented algorithm for fully complex-valued neural networks. *Neural Networks*, *69*, 44–50. http://dx.doi.org/10.1016/j.neunet.2015.05.003.

Yin, Q., Wang, J., Luo, X., Zhai, J., Jha, S. K., & Shi, Y. Q. (2019). Quaternion convolutional neural network for color image classification and forensics. *IEEE Access*, *7*, 20293–20301.

Zeiler, M. D. (2012). Adadelta: An adaptive learning rate method. arXiv preprint arXiv:1212.5701.