3-2021

Meta-Heuristic Optimization Methods for Quaternion-Valued Neural Networks

Jeremiah P. Bill

Meta-Heuristic Optimization Methods for Quaternion-Valued Neural Networks

THESIS

Jeremiah P. Bill, Capt, USAF

AFIT-ENS-MS-21-M-143

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENS-MS-21-M-143

META-HEURISTIC OPTIMIZATION METHODS FOR QUATERNION-VALUED NEURAL NETWORKS

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science of Operations Research

Jeremiah P. Bill, BA in Mathematics

Capt, USAF

March 25, 2021

AFIT-ENS-MS-21-M-143

META-HEURISTIC OPTIMIZATION METHODS FOR QUATERNION-VALUED NEURAL NETWORKS

THESIS

Jeremiah P. Bill, BA in Mathematics
Capt, USAF

Committee Membership:

Dr. Lance E. Champagne
Advisor

Dr. Bruce Cox
Member

Dr. Trevor Bihl
Member

Capt Phillip R. Jenkins, PhD
Reader

AFIT-ENS-MS-21-M-143

Abstract

In recent years, real-valued neural networks have demonstrated promising, and often striking, results across a broad range of domains. This has driven a surge of applications utilizing high-dimensional datasets. While many techniques exist to alleviate issues of high-dimensionality, they all induce a cost in terms of network size or computational runtime. This work examines the use of quaternions, a form of hypercomplex numbers, in neural networks. The constructed networks demonstrate the ability of quaternions to encode high-dimensional data in an efficient neural network structure, showing that hypercomplex neural networks reduce the number of total trainable parameters compared to their real-valued equivalents. Finally, this work introduces a novel training algorithm using a meta-heuristic approach that bypasses the need for analytic quaternion loss or activation functions. This algorithm allows for a broader range of activation functions over current quaternion networks and presents a proof-of-concept for future work.

To my family (and my dog) for putting up with my Zoom meetings, Teams classes, and constant presence in the midst of a work-from-home pandemic. This one's for you.

Acknowledgements

I would like to thank my advisor, Dr. Champagne, for giving me free rein to explore this fascinating topic. Additionally, I would to thank Dr. Hill and Dr. Jenkins for their invaluable inputs, and to my many classmates, friends, and family that have supported me along the way.

Jeremiah P. Bill

Table of Contents

Page

Abstract . . . iv

Dedication . . . v

Acknowledgements . . . vi

List of Figures . . . ix

List of Tables . . . x

I. Introduction . . . 1

1.1 Motivation and Background . . . 2
1.2 Organization of the Thesis . . . 3

II. Literature Review . . . 4

2.1 Neural Networks & Multi-Layer Perceptrons . . . 4
2.1.1 The Backpropagation Algorithm . . . 6
2.1.2 Shortfalls . . . 8
2.2 The Quaternions . . . 9
2.2.1 Quaternion Algebra . . . 10
2.2.2 Quaternion Conjugates, Norms, and Distance . . . 11
2.2.3 Quaternionic Matrices . . . 11
2.3 Quaternion-valued Neural Networks (QNNs) . . . 12
2.3.1 A Note on Quaternion Calculus and Quaternionic Analysis . . . 13
2.3.2 Quaternion Neural Networks . . . 15
2.4 Metaheuristic Optimization Techniques . . . 17

III. Solution Methodology . . . 20

3.1 Test Functions . . . 20
3.1.1 The Ackley Function . . . 20
3.1.2 The Lorenz Attractor Chaotic System . . . 21
3.2 MLP Network Topologies . . . 24
3.2.1 Function Approximation . . . 24
3.2.2 Chaotic Time Series Prediction . . . 25
3.3 Quaternion Genetic Algorithm . . . 29
3.4 Evaluation & Analysis Strategy . . . 31

Page

IV. Results and Analysis 32

4.1 Function Approximation Results 32
4.2 Time Series Prediction Results 34
4.3 Discussion 37

V. Conclusions and Recommendations 40

5.1 Conclusions 40
5.2 Recommendations 40
Bibliography 42

List of Figures

Figure		Page
1	Original Rosenblatt Perceptron diagram [38]	4
2	Overview of a TLU from [13]	5
3	Representation of a basic MLP [17]	6
4	3D Ackley Function	22
5	Lorenz Attractor	23
6	Impact of Initial Conditions on Lorenz System	27
7	Genetic Algorithm/Genetic Programming Process	29
8	Training Set Mean Absolute Error for Each Network	33
9	10-Step Ahead Predicted Coordinate Values	35
10	10-Step Ahead Path Predictions	36
11	50-Step Ahead Predicted Coordinate Values	37
12	50-Step Ahead Path Predictions	37
13	10-Step Predictions	38
14	50-Step Predictions	38
15	Unscaled QMAE Training Error	38

List of Tables

Table		Page
1	Neural Network Topologies for Ackley Function Approximation	25
2	Neural Network Topology for Chaotic Prediction	27
3	Neural Network Comparison Results	33
4	Lorenz Prediction Results	36

META-HEURISTIC OPTIMIZATION METHODS FOR QUATERNION-VALUED NEURAL NETWORKS

I. Introduction

Over the last several decades, the explosive growth in artificial intelligence and machine learning (AI/ML) research has driven a need for more efficient data representations and machine learning training methods. As machine learning applications have expanded into new and exciting domains, the scale of data processed through enterprise systems, e.g. Department of Defense (DoD) and industry, has grown to an almost incomprehensible level. While computational resources have grown commensurately with this increase in data-driven systems, inefficiencies in current neural network architectures continue to hamper progress on difficult optimization problems.

This work examines the use of hypercomplex numbers in neural networks, with a particular emphasis on the use of quaternions in neural network architectures. This thesis demonstrates that quaternion data representations can reduce the total number of trainable neural network parameters by a factor of four, resulting in improvements in both computer memory allocations and computational runtime. Additionally, this work presents a novel, gradient-free, quaternion genetic algorithm that enables the use of several loss and activation functions previously unavailable due to differentiability requirements. This chapter provides a brief review of the DoD's Artificial Intelligence Strategy, highlighting key application areas for hypercomplex neural networks. This chapter concludes with an outline of the overall thesis organization.

1.1 Motivation and Background

Nearly all modern machine learning applications involve high dimensional datasets and complex search spaces. Such datasets are often plagued by the "curse of dimensionality" [21, p. 242] which can stymy many of the current multivariate analysis techniques. In addition, high dimensional datasets often contain hidden interdependencies between data elements that can be distorted in surprising ways when using blackbox optimization techniques such as neural networks. While increasing computational resources and improving the quality of the training data can generally mitigate many of these issues, this is not always a practical solution, especially in resource constrained environments.

The 2018 Department of Defense Artificial Intelligence Strategy [45] communicates the department's overall strategy for maintaining technological and operational advantages in the AI/ML domain. In particular, the strategy highlights several focus areas for artificial intelligence development, many of which involve multidimensional data. Two such examples are:

1. *Improving situational awareness and decision-making*: specifically, AI applied to perception tasks such as imagery analysis. Image data is naturally multidimensional and often requires special processing to maintain spatial hierarchies between RGB pixel intensity values, for example.

2. *Increasing safety of operating equipment*: military equipment necessarily operates in 3-dimensional space, recording positional information using 3D coordinates over time. Safety systems and autonomous vehicles can reduce human error and improve operator safety using efficient data representations for positional information.

This research proposes, develops, and investigates a quaternion neural network for

solving multidimensional regression and prediction problems. In addition, this work introduces a novel quaternion genetic algorithm (GA)-based approach for training the neural network. Initial experiments with the proposed network structure demonstrate substantial performance improvements over equivalent real-valued networks in terms of both accuracy and computational resources. Additionally the quaternion genetic algorithm removes the need for analytic loss or activation functions in the network, expanding the aperture on available functions for use in the quaternion domain.

1.2 Organization of the Thesis

The remainder of this thesis is organized as follows. Chapter II provides a review of neural networks, the quaternion number system, quaternion neural networks, and metaheuristic optimization techniques. Chapter III describes the methodology used to develop a quaternion neural network and a novel quaternion genetic training algorithm. Chapter IV presents the network results, comparing the quaternion genetic algorithm performance to two analogous real-valued networks. Additionally, a multidimensional input/multidimensional output network is presented for predicting the Lorenz attractor chaotic dynamical system. Finally, Chapter V provides conclusions, recommendations, and proposals for future work.

II. Literature Review

This work evaluates various methods for training quaternion-valued neural networks (QNNs). In particular, this work implements a meta-heuristic optimization technique to train a QNN on both real-valued and quaternion-valued function approximation problems. Neural networks have received a tremendous amount of attention in recent years, with a large body of research emerging from both academia and industry. Meta-heuristic optimization techniques are also well-studied and have been applied to real-, complex-, and quaternion-valued problems. This section provides a brief review of the quaternion algebra as well as related work regarding neural networks and meta-heuristic optimization.

2.1 Neural Networks & Multi-Layer Perceptrons

Statistical learning processes have received increasing attention in recent years with the proliferation of large datasets, ever-increasing computing power, and simplified data exploration tools. In general, statistical learning is a set of mathematical and statistical tools for understanding patterns in data [21]. Artificial neural networks (ANNs) fall under the umbrella of statistical learning tools and were originally developed as a mathematical structure that mimics the human brain.

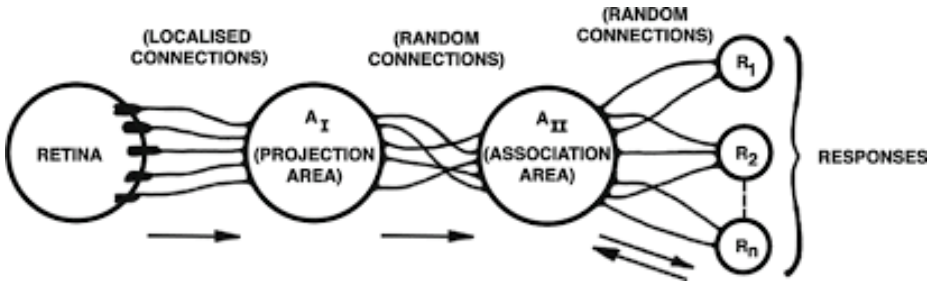

Figure 1. Original Rosenblatt Perceptron diagram [38]

The idea for a mathematical representation of the human brain was first proposed

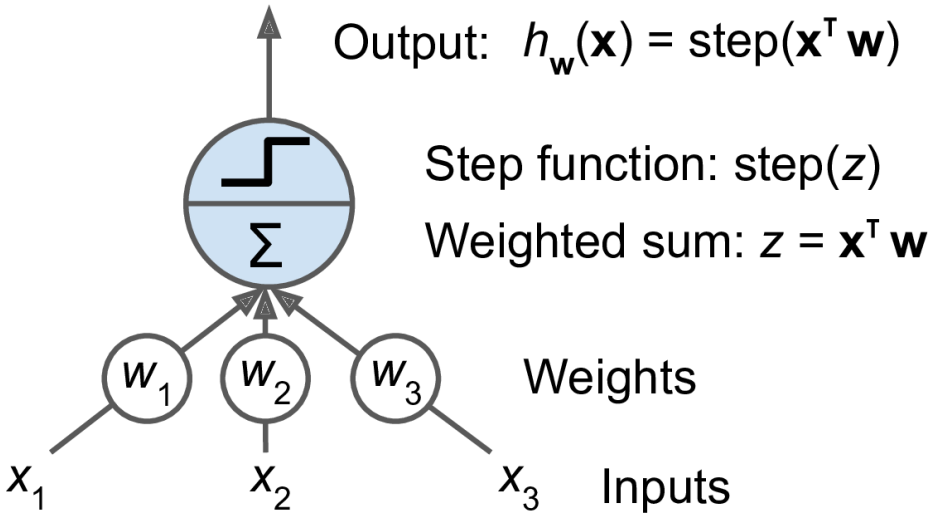

Figure 2. Overview of a TLU from [13]

by Warren McCulloch and Walter Pitts in 1943 [27]. McCulloch and Pitts envisioned a network structure where each node in the network was governed by propositional logic rules. In 1957, Frank Rosenblatt proposed a slightly different neural structure called the *Perceptron* [38]. A perceptron is composed of several *threshold logic units* (TLUs), each of which takes a weighted sum of input values and uses the resulting sum as the input to a non-linear activation function. While each TLU computes a linear combination of the inputs based on the network weights, the use of a non-linear activation function allows the perceptron to estimate a number of non-linear functions by adjusting the weights of each input. The original perceptron diagram is shown in Figure 1, whereas a modern representation of an individual TLU is shown in Figure 2.

While the perceptron has proven to be a very capable statistical learning tool, perceptrons are incapable of solving some trivial non-linear problems, such as replicating the Exclusive Or (XOR) function [28]. However, stacking multiple layers of perceptrons together so that the output of one perceptron forms the input to a subsequent perceptron allows for the estimation of a vast set of linear and non-linear problems. In fact, two contemporaries, Cybenko [9] and Hornik et al. [16] both inde-

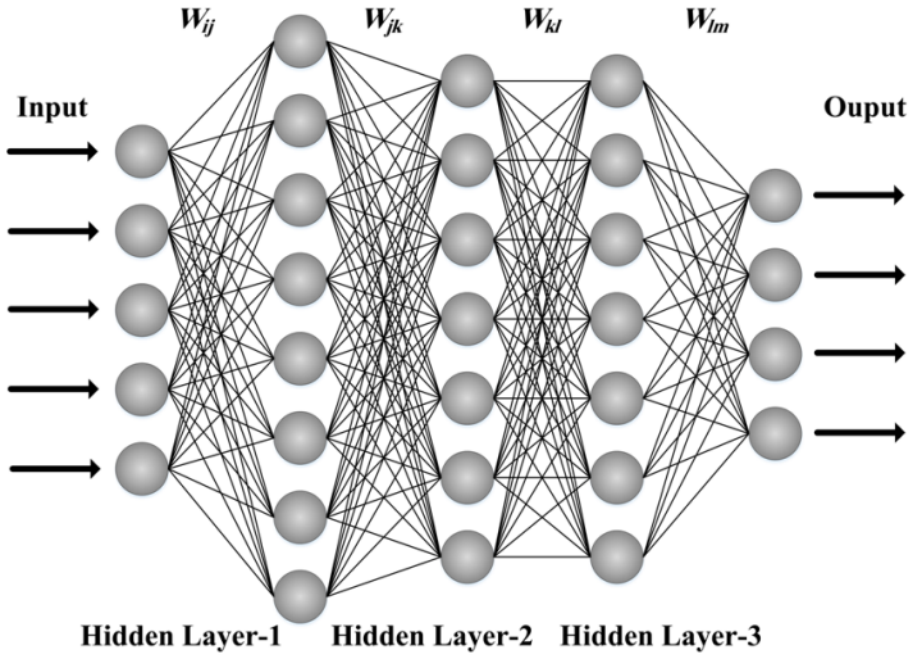

Figure 3. Representation of a basic MLP [17]

pendently showed that a network with three layers and sigmoidal activation functions at each layer is able to approximate any nonlinear function to an arbitrary degree of accuracy. This network structure is called the Multilayer Perceptron (MLP) and it forms the most basic deep neural network (DNN). This result (called the Universal Approximation Theorem) has provided the theoretical justification that has driven neural network research to the present day. A representation of an MLP is shown in Figure 3, and [13] provides an overview of MLPs and other common neural network structures.

2.1.1 The Backpropagation Algorithm

Although artificial neural network architectures have existed since the mid-twentieth century, researchers found them to be computationally expensive to use and impractical for most applications. As a result, neural network research was largely stagnant until 1986, when Rumelhart et al. [39] introduced the backpropagation algorithm for

training a neural network. The algorithm developed by Rumelhart et al. extended several key ideas that Werbos [46] presented in his unpublished doctoral dissertation. The basic steps of the backpropagation algorithm are to:

1. Process all of the training data through the neural network, calculating the loss at the final output layer of the network. This is referred to as a forward pass through the network.

2. Use the chain rule to calculate the gradient of the loss function at each layer of the network. This determines an improving direction in which to "move" the network. The partial derivatives at each layer calculated by the chain rule indicate the amount of error that each individual weight contributes to the total error in the network. This is referred to as the backpropagation step, since the loss function gradients are calculated layer-by-layer through the network, starting at the output layer and working backwards.

3. Perform a gradient descent step to update each weight in the network using the error gradients.

The backpropagation algorithm has proven to be a straightforward, easy to understand, and easy to implement algorithm that has enabled efficient implementations of neural networks across a wide-range of problem sets. In fact, backpropagation has proven so successful that modern deep learning practitioners have developed specialized neural network structures to solve specific problems and overcome shortfalls of the MLP. Examples of custom architectures include convolutional neural networks (CNNs) for processing image data, recurrent neural networks (RNNs) for processing sequence data, and generative adversarial networks (GANs) which have been used in recent years to create deep fakes and very convincing counterfeit data [22].

2.1.2 Shortfalls

Artificial neural networks have achieved state-of-the-art results in a truly breath-taking array of problem domains. However, ANNs are not without their shortfalls. First, ANNs often require a vast amount of training data. Because of this, training an ANN requires a large amount of computer resources, in terms of both RAM and processing time. Additionally, the backpropagation algorithm requires a significant amount of low-level computational power in order to perform the matrix multiplications for each forward and backward pass. While GPUs have proven to be particularly well-suited for this task, many of the current large-scale ANN research applications require prohibitive amounts of computer memory and GPU hours.

Finally, ANNs (and MLPs in particular) can struggle to maintain any sort of spatial relationships that are present within the training data. A simple example of this is seen in color image processing. In general, each of the three color channels of an RGB image are processed separately in an MLP since the 3-dimensional matrix representation of the image must first be flattened into a vector for the network forward pass step. This results in the loss of the spatial relationship between the red, green, and blue pixel intensities at each pixel. Yin et al. [49] highlight the fact that this spatial hierarchy can be maintained when using higher-dimensional number systems such as quaternions as opposed to real numbers and is a significant motivation for this paper. Matsui et al. [26] demonstrated similar experimental results on a 3-dimensional affine transformation problem, showing that quaternion-valued deep neural networks were able to recover the spatial relationships between 3-dimensional coordinates. Section 2.2 provides a brief summary of hypercomplex number systems, along with a review of their use and success in advanced neural network applications.

2.2 The Quaternions

The quaternion numbers (denoted by $\mathbb{H}$) are a four-dimensional extension of the complex numbers. Complex numbers have the form $x + iy$, consisting of a real part x and an imaginary part y, and can be thought of as an isomorphism of $\mathbb{R}^2$. That is, the complex numbers contain two copies of the real number line, allowing a single complex number to encode twice as much information as a single real number. Complex numbers are particularly useful for describing motion in 2-dimensional space, since there is a very succinct analogue between complex multiplication and rotations in the plane [7].

Quaternions are referred to as hypercomplex numbers. Each quaternion $\mathbf{q}$ consists of a real part and three imaginary parts, so that the quaternions form an isomorphism with $\mathbb{R}^4$ with basis elements 1, $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$:

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

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

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

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

$$\mathbf{jk} = -\mathbf{kj} = \mathbf{i} \tag{3}$$

$$\mathbf{ki} = -\mathbf{ik} = \mathbf{j} \tag{4}$$

$$\mathbf{ij} = -\mathbf{ji} = \mathbf{k}. \tag{5}$$

These rules clearly demonstrate that quaternion multiplication is non-commutative.

However, since the multiplication of any two basis elements is plus or minus another basis element, the quaternions under these rules form a non-abelian group, denoted Q_8. The group Q_8, along with the operations of addition and multiplication form a division algebra, which is an algebraic structure similar to a field where multiplication is non-commutative.

The 4-dimensional structure of each quaternion number indicates that quaternions are capable of encoding four copies of the real number line into a single quaternion number, analogous to the two copies of $\mathbb{R}$ encoded in the complex numbers. Quaternions were discovered by the Irish mathematician Sir William Rowan Hamilton in 1843 [15], hence why the set of quaternions is referred to as $\mathbb{H}$ and the quaternion notion of multiplication, described below, is referred to as the *Hamilton Product*.

2.2.1 Quaternion Algebra

The quaternions form a division algebra, meaning that the set of quaternions along with the operations of addition and multiplication follow 8 of the 9 field axioms (all but commutativity). Quaternion addition is defined using the element-wise addition operation. For two quaternions $\mathbf{q}_1, \mathbf{q}_2 \in \mathbb{H}$, where

$$\mathbf{q}_1 = r_1 + x_1\mathbf{i} + y_1\mathbf{j} + z_1\mathbf{k}$$

and

$$\mathbf{q}_2 = r_2 + x_2\mathbf{i} + y_2\mathbf{j} + z_2\mathbf{k}.$$

The sum $\mathbf{q}_1 + \mathbf{q}_2$ is defined as,

$$\mathbf{q}_1 + \mathbf{q}_2 := (r_1 + r_2) + (x_1 + x_2)\mathbf{i} + (y_1 + y_2)\mathbf{j} + (z_1 + z_2)\mathbf{k}. \tag{6}$$

Quaternion multiplication, referred to as the Hamilton Product, can easily be derived using the basis multiplication rules in equations (3) - (5) and the distributive property. In reduced form, the Hamilton product of two quaternions $\mathbf{q}_1$ and $\mathbf{q}_2$ is defined as:

$$\begin{aligned}\mathbf{q}_1 * \mathbf{q}_2 :=&(r_1 r_2 - x_1 x_2 - y_1 y_2 - z_1 z_2) \\ &+(r_1 x_2 + x_1 r_2 + y_1 z_2 - z_1 y_2)\mathbf{i} \\ &+(r_1 y_2 - x_1 z_2 + y_1 r_2 + z_1 x_2)\mathbf{j} \\ &+(r_1 z_2 + x_1 y_2 - y_1 x_2 + z_1 r_2)\mathbf{k}.\end{aligned} \tag{7}$$

2.2.2 Quaternion Conjugates, Norms, and Distance

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

$$||\mathbf{q}|| := \sqrt{\mathbf{q}\mathbf{q}^*} = \sqrt{r^2 + x^2 + y^2 + z^2}. \tag{8}$$

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

$$d(\mathbf{q}, \mathbf{p}) := ||\mathbf{q} - \mathbf{p}||. \tag{9}$$

2.2.3 Quaternionic Matrices

Since the set of quaternions $\mathbb{H}$ form a division algebra under addition and the Hamilton product, they also form a non-commutative ring under the same operations. Hence, quaternionic matrix operations can be defined as for matrices over an arbitrary ring. Given any two quaternionic matrices $A, B \in \mathbb{H}^{M \times N}$, the sum $A + B$ is defined

element-wise

$$(A + B)_{ij} := A_{ij} + B_{ij}. \tag{10}$$

Similarly, for any quaternionic matrix $A \in \mathbb{H}^{M\times N}$ and $B \in \mathbb{H}^{N\times P}$, the product $AB \in \mathbb{H}^{M\times P}$ is defined as

$$(AB)(m, p) := \sum_{n=1}^{N} A(m, n)B(n, p), \quad \forall m = 1, \ldots, M, p = 1, \ldots, P. \tag{11}$$

As with matrix multiplication over an arbitrary ring, quaternionic matrix multiplication is non-commutative. Additionally, great care must be taken to ensure the proper execution of the Hamilton product when multiplying each row of A with each column of B, since the Hamilton product itself is non-commutative.

2.3 Quaternion-valued Neural Networks (QNNs)

Many practical applications of machine learning techniques involve data that are multidimensional. With the mathematical machinery described in Section 2.2, the quaternions provide a succinct and efficient way of representing multidimensional data. Additionally, when applied to neural network architectures, quaternions have been shown to preserve spatial hierarchies and interrelated data components that are often separated and distorted in real-valued MLP architectures. This section provides a brief review of QNN research, starting with a brief note on some of the issues in QNN construction stemming from quaternionic analysis and quaternion calculus. Then, the development of QNNs is traced chronologically from early works to the state of the art.

2.3.1 A Note on Quaternion Calculus and Quaternionic Analysis

Quaternionic analysis is a relatively unexplored field. As such, there are very few analytic functions of a quaternion variable. To account for this, quaternion networks generally utilize "split" activation functions, where a real-valued activation function is applied to each quaternion coefficient. For example, the split quaternion sigmoid function [5] for a quaternion $\mathbf{q} = r + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$ is given by

$$\sigma(\mathbf{q}) = \sigma(r) + \sigma(x)\mathbf{i} + \sigma(y)\mathbf{j} + \sigma(z)\mathbf{k}, \tag{12}$$

where $\sigma(\cdot)$ is the real-valued sigmoid function. Similar definitions hold for any real-valued activation function, and many QNNs utilize these split activation functions even when quaternionic functions are available such as the quaternion-valued hyperbolic tangent function. Research has shown that true quaternionic activation functions can improve performance over split activation functions [43], but they require special considerations since their analyticity can only be defined over a localized domain, and the composition of two locally analytic quaternion functions is generally not locally analytic [48], providing limited utility in deep neural networks. Additionally, many complex and quaternion-valued elementary transcendental functions, including the hyperbolic tangent, are unbounded and contain singularities [24] that make neural network training difficult.

These issues, along with the non-commutativity of quaternions, also affect the gradient descent algorithm employed in many quaternion networks. Generally speaking, the non-commutativity of quaternions precludes the development of a general product rule and a quaternion chain rule to compute quaternion derivatives and partial derivatives. Thus, quaternion networks must employ split loss functions and the partial derivatives used in the backpropagation algorithm are calculated using a

similar "split" definition. The split partial derivative used in training a Quaternion Multilayer Perceptron (QMLP) network, first defined by [5], is given by

$$\frac{\partial E}{\partial W^l} = \frac{\partial E}{\partial W_r^l} + \frac{\partial E}{\partial W_x^l}\mathbf{i} + \frac{\partial E}{\partial W_y^l}\mathbf{j} + \frac{\partial E}{\partial W_z^l}\mathbf{k}, \tag{13}$$

where E is the loss function and W^l is the weight matrix at layer l. Some researchers refer to this as a "channelwise" [43] or vectorized implementation.

In 2015, Xu et al. [47] developed the generalized Hamilton-Real ($G\mathbb{HR}$) calculus, which presents a novel product and chain rule using left- and right- directional derivatives to account for the non-commutativity of the quaternions. Following this, [48] developed several quaternion-valued equivalents for some of the most popular neural network learning algorithms, including a Quaternion Gradient Descent algorithm, a Quaternion Gauss-Newton algorithm, and a Quaternion Levenberg-Marquardt algorithm. However, the authors demonstrate the effectiveness of each algorithm on a very small toy problem, and as of this writing, the $G\mathbb{HR}$ calculus and the associated learning algorithms have yet to be applied to any real-world machine learning dataset with a deep quaternion network.

Calculus-based learning algorithms aside, the lack of QNN implementations with locally analytic, quaternion-valued loss or activation functions is a substantial obstacle to current QNN research. While QNNs with split activation functions have shown some empirical improvements over their real-valued equivalents, the mathematical justification for using such functions has yet to be proven in the quaternion domain (although the universal approximation theorem has been proven in the complex domain [6]). Furthermore, the split backpropagation algorithm requires a substantial amount of computing resources over the relatively straightforward and simple real-valued backpropagation algorithm, effectively negating any computational runtime gains that QNNs should have over real-valued networks due to the smaller number of

nodes, weights and biases.

This work attempts to address this issue directly, using a genetic algorithm to train a quaternion-valued neural network with fully quaternion activation functions at each layer of the network. The genetic algorithm circumvents the need for the convoluted calculus rules that one must use in QNNs due to the non-commutativity of quaternions and the locally analytic nature of the activation functions. While not yet proven in the quaternion domain, this approach has a strong theoretical basis that is supported in both the complex- and real-valued domains ([24], [9], and [16]).

2.3.2 Quaternion Neural Networks

The QMLP was first introduced by Arena et. al. [5] in 1994, as noted in Section 2.3.1. The initial QMLP used split sigmoid activation functions and a version of the Mean Square Error (MSE) loss function E, formed by substituting quaternions into the real-valued MSE equation. For a network with $l = 1, \ldots, M$ layers and $1 < n < N_l$ nodes per layer, the output of each node n in each layer l is computed as

$$y_n^l = \sigma(S_n^l), \tag{14}$$

where σ is any split sigmoidal activation function and S_n^l is the linear combination of network weights, biases, and the output of the $l-1$ layer computed as in a normal MLP

$$S_n^l = \sum_{m=0}^{N_{l-1}} w_{nm}^l * y_m^{l-1} + b_n^l. \tag{15}$$

For each S_n^l, the weights, biases, and y-values are all quaternions. Thus, $*$ represents the Hamilton Product. The loss function E is given by

$$E = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{t}_n - \mathbf{y}_n^{(M)})^2, \tag{16}$$

where $\mathbf{t}$ represents the target (truth) data and $\mathbf{y}^{(M)}$ represents the neural network output at the Mth layer.

The authors also introduced a simple learning algorithm using the split or "channelwise" partial derivatives discussed in Section 2.3.1, where the gradient Δ_n^l at the output layer is simply the output error of the network $(\mathbf{t}_n - \mathbf{y}_n^{(M)})$ and the error at each prior layer l is calculated using the formula

$$\Delta_n^l = \sum_{n=1}^{N^{l+1}} w_{hn}^{*l+1} * (\Delta_n^{l+1} \cdot \sigma'(S_n^{l+1})), \tag{17}$$

where w_{hn}^{*l+1} represents the quaternion conjugate of the weight connecting node h in the lth layer to node n in the $l+1$st layer. Additionally, $(\cdot)$ represents the componentwise product, not the Hamilton product between the gradient at the $l+1$st layer and the channelwise partial derivative of $\sigma(\cdot)$. Using this gradient rule, the biases at each layer are updated according to the normal backpropagation process

$$b_n^l = b_n^l + \epsilon\Delta_n^l, \tag{18}$$

where ϵ is the learning rate. Note, however, that the weights are updated using the rule

$$w_{nm}^l = w_{nm}^l + \epsilon\Delta_n^l * S_m^{*l-1}, \tag{19}$$

where S_m^{*l-1} represents the conjugate of the input to the lth layer S_m^{l-1}.

Although the quaternion backpropagation algorithm bears similarities to the real-valued backpropagation algorithm, it is unique in several ways. The first is the use of split derivatives in the weight and bias update step. Although the use of split derivatives may seem like a trick to bypass a true quaternion derivative definition, it builds on [6], which proved that split activation functions and derivatives in the complex domain could universally approximate complex-valued functions. While unproven in the quaternion domain, Arena et. al. demonstrated the effectiveness of this network on a small function approximation problem, where a quaternion network was used to approximate a quaternion-valued function. Additionally, the weight update and the gradients leverage the quaternion conjugate, which improves training performance.

Since the introduction of the QMLP and its associated training algorithm, researchers have used QMLPs for a variety of tasks. In particular, QMLPs have been used as autoencoders [19], for color image processing [14], text processing [35], and polarized signal processing [8]. Another natural application of quaternions is in robotic control [11], since quaternions can compactly represent 3-dimensional rotation and motion through space. Parcollet et al. [36] note that in every scenario, QMLPs always outperform real-valued MLPs when processing 3- or 4-dimensional signals. These simple networks have driven further research in more advanced network architectures such as convolutional neural networks and recurrent neural networks, both of which have shown promise in the quaternion domain for advanced image processing [12], speech recognition [37], and other tasks.

2.4 Metaheuristic Optimization Techniques

While the backpropagation algorithm discussed in Section 2.1.1 has dominated nearly all neural network research since it was first introduced, recent work has

shown that heuristic search methods can also effectively train neural networks at a scale comparable to gradient descent and backpropagation. Metaheuristic optimization encompasses a broad range of optimization techniques that do not provide guarantees of algorithmic closure or convergence, but have shown empirically to perform well in a variety of complex optimization tasks. In contrast to gradient-based methods such as the backpropagation algorithm, many metaheuristics do not require any gradient information. Instead, metaheuristic algorithms utilize rules to generate search patterns [31]. Metaheuristic algorithms are often inspired by natural processes that are reflected in the names of various algorithms such as Evolutionary Search, Ant Colony Search, Whale Optimization, and the Bat Algorithm (among many others).

Perhaps the most famous application of a metaheuristic approach in training neural networks is the NeuroEvolution through Augmenting Topologies (NEAT) [41] process, which uses a genetic algorithm to simultaneously train and grow neural networks through the evolutionary process. NEAT has proven to be a very effective neural network training tool, and subsequent variants of NEAT have successfully evolved neural networks with millions of weight and bias parameters [40]. More recently, researchers with Uber's OpenAI Labs have shown that even basic Genetic Algorithms can compete with backpropagation in training large networks with up to four million parameters [42]. Several other metaheuristic implementations have shown promise in training neural networks and optimizing the hyperparameters of neural networks. See [30] for a full review of metaheuristic optimization in neural network design.

Metaheuristic optimization methods have also been applied to a limited number of search problems in the quaternion domain. A quaternion variant of the Firefly Algorithm [10] demonstrated comparable performance to the real-valued Firefly Algorithm in optimizing nonlinear test functions. In addition, [34] introduced a quaternion-based

Harmony Search algorithm, demonstrating the algorithm's performance on a similar range of nonlinear test functions. The hypothesis of both approaches is that the search space in the hypercomplex domain is smoother than the search space in $\mathbb{R}$. While not proven, [32] summarizes the approach. Additionally, Khuat et. al. [23] introduced a quaternion genetic algorithm with multi-parent crossover that was used to optimize a similar set of nonlinear test functions. Finally, [33] used the Harmony Search algorithm introduced in [34] to fine-tune the hyperparameters of a neural network. However, as of this writing, quaternion metaheuristic search methods have yet to be applied to more complex tasks, such as optimizing a large number of weights and biases in a quaternion neural network.

Given the difficulties in defining globally analytic quaternion loss functions, activation functions, and quaternion partial derivatives, metaheuristic optimization provides an ideal method of training quaternion neural networks. Chapter III outlines a novel quaternion genetic algorithm for training the weights and biases of quaternion neural networks. The algorithm does not require gradient information and makes no assumptions on the analyticity of the activation functions of the network at each layer, allowing for a broader range of quaternion activation functions than have been available in prior works.

III. Solution Methodology

This chapter describes the test methodology employed in comparing the performance of real-valued MLPs to quaternion-valued MLPs in several multidimensional function approximation tasks. First, Section 3.1 describes the test functions selected for use in the study. Section 3.2 outlines the structure of the neural networks, including an overview of the nodes, layers, and total trainable parameters of each network. Section 3.3 details the genetic algorithm used to train the real- and quaternion-valued networks. Finally, Section 3.4 concludes the chapter with a description of the evaluation strategy and key comparison metrics.

3.1 Test Functions

Demonstrating the ability of a neural network to approximate an arbitrary nonlinear function is a crucial step in the development of any ANN structure. Cybenko's Universal Approximation Theorem [9], discussed in Section 2.1, provides the theoretical underpinning for all modern ANN research and has legitimized many of the ANN applications to date. While still unproven for the quaternion domain, this research demonstrates that quaternion neural networks with elementary transcendental activation functions and a genetic training algorithm can effectively approximate arbitrary nonlinear functions, using the Ackley function and the Lorenz Attractor chaotic system as test cases.

3.1.1 The Ackley Function

The Ackley function is a non-convex test function that is often used to test global optimization algorithms. It was first introduced by David Ackley [1] and has since been included in a standard library of optimization test functions. In three dimen-

sions, the function is characterized by an elevated eggcrate-like surface, with a global minimum in the center of the function that sinks down to zero. The Ackley function is a good test case for quaternion networks since it can easily be defined in any number of dimensions. A vector representation of the function is given in Equation (20), where a, b, and c are constants and n represents the dimensionality of the vector $\mathbf{x}$. Additionally, a three dimensional plot of the Ackley function is shown in Figure 4.

$$f(\mathbf{x}) = -a \exp\left(-b\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(c \cdot x_i)\right) + a + \exp(1) \qquad (20)$$

This research uses a 4-dimensional Ackley function, with the a, b, and c coefficient values set to 20.0, -0.2, and 2π, respectively. The function's x, y, and z values are generated over the range $[-5, 5]$, using a meshgrid with a spacing of 0.5 between each point. With a three-dimensional input, this results in 9,261 data-points. The coordinate values are then translated from $\mathbb{R}$ into $\mathbb{H}$ by taking the coordinates of each point and casting them into the three imaginary parts of a quaternion. For example, the point $(-5, -5, -5) \Rightarrow \mathbf{q}_1 = 0r - 5\mathbf{i} - 5\mathbf{j} - 5\mathbf{k}$.

Finally, the data is split into a training set and a test set. The purpose of this split is to ensure that the neural networks are producing functions with good generalization capabilities. The data points are randomly shuffled and 80% of the data points are retained as training data while 20% of the data points are split into the test set.

3.1.2 The Lorenz Attractor Chaotic System

The Lorenz Attractor is a deterministic system of differential equations first presented by Edward Lorenz [25]. The Lorenz Attractor is a chaotic system, meaning that while it is deterministic, the system never cycles and never reaches a steady state. Additionally, the system is very sensitive to initial conditions. When represented as a

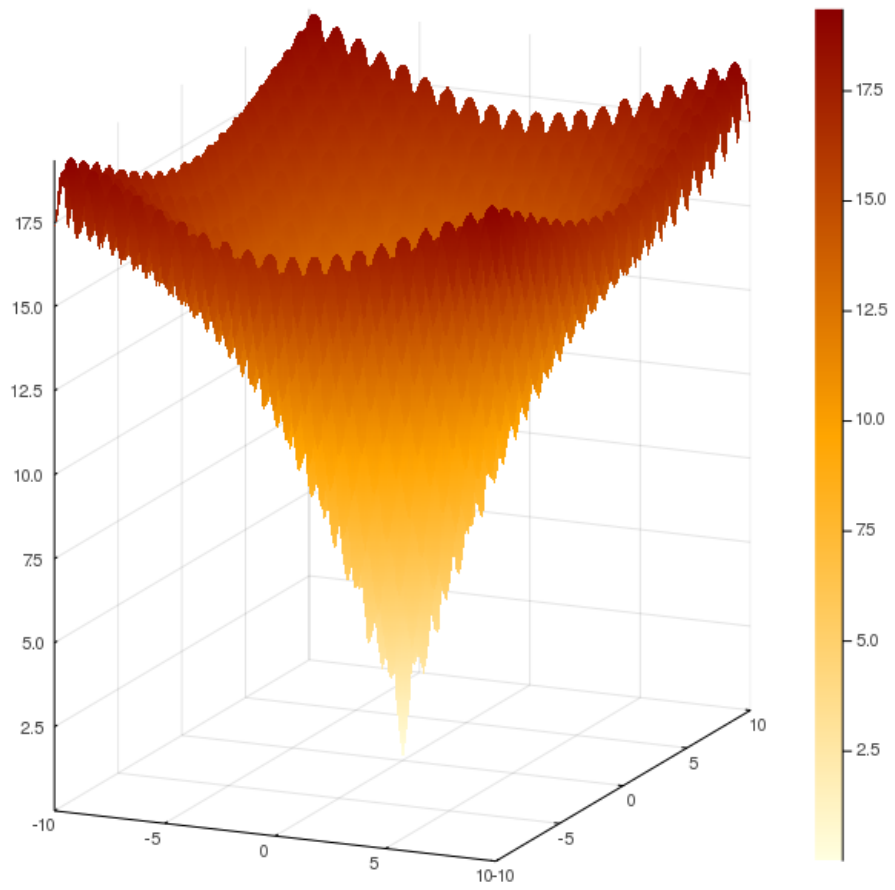

Figure 4. 3D Ackley Function

set of 3-dimensional coordinates, the Lorenz Attractor produces a mesmerizing graph often referred to as the Lorenz butterfly. A static representation of this is shown in Figure 5.

The Lorenz Attractor is governed by the following system of differential equations:

$$\frac{dx}{dt} = \sigma(y - x) \tag{21}$$

$$\frac{dy}{dt} = \rho x - y - xz \tag{22}$$

$$\frac{dz}{dt} = xy - \beta z \tag{23}$$

where σ, ρ, and β are constants. For this experiment (and in Figure 5), $\sigma = 10$, $\rho = 28$,

and $\beta = \frac{8}{3}$. Quaternions are naturally well-suited to predicting chaotic time series, including the Lorenz Attractor, since the problem involves both a multidimensional input and a multidimensional output. Split quaternion neural networks have proven quite successful at chaotic time series prediction based on small training datasets ([4], [3], [2], and [44]).

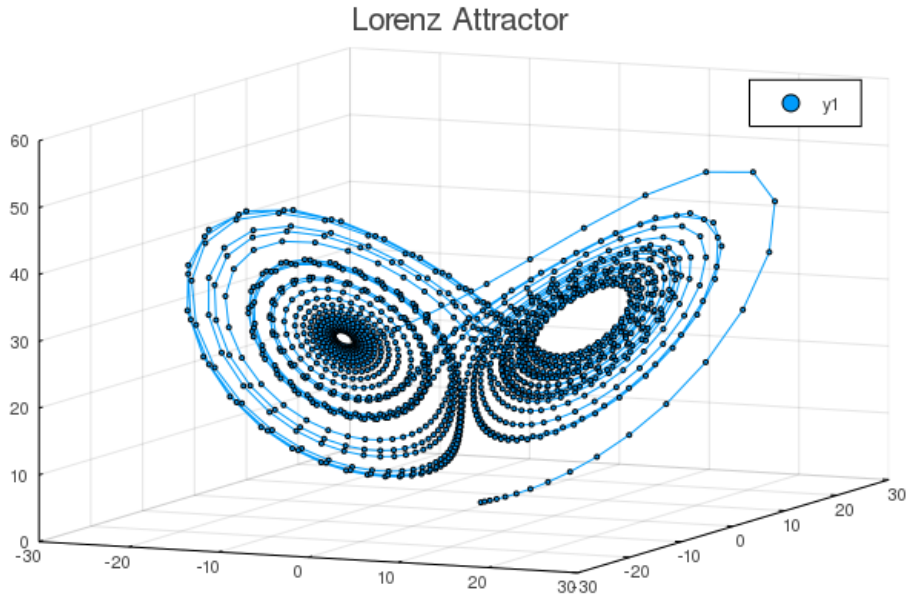

Figure 5. Lorenz Attractor

The data for the Lorenz Attractor was again split 80%/20% between training and test datasets. Additionally, both the inputs and the outputs were cast into the quaternion domain. This allowed for a direct output error calculation using the quaternion distance metric defined in Section 2.2.2. The full details of the loss function, the activation functions, and the nodes and layers of the networks used in both experiments are discussed in Section 3.2.

3.2 MLP Network Topologies

3.2.1 Function Approximation

The function approximation experiment focused on the relative performance of real-valued network architectures to quaternion networks with pure quaternion activation functions. The comparison experiment operated on three distinct network architecture and training algorithm combinations. The first is the Quaternion Multilayer Perceptron trained with a genetic algorithm (from here on referred to as QMLP+GA). This network consists of an input layer, two hidden layers, and an output layer.

Between each layer of the network, a "normalization" step was added, where the output of each layer is individually normalized. Since the training data-points were encoded into quaternion values, the input and output layer require a single node each. The two hidden layers of the network contain 3 nodes each, resulting in a total of 22 trainable weights and biases for the network. The pure-quaternion hyperbolic tangent ($tanh$) function was selected as the nonlinear activation function for the input layer and both hidden layers. The $tanh$ function in the quaternion domain is defined as

$$tanh(\mathbf{q}) = \frac{e^{2\mathbf{q}} - 1}{e^{2\mathbf{q}} + 1}, \qquad \mathbf{q} \in \mathbb{H}. \tag{24}$$

To determine the loss at the output layer, the final output is first mapped from $\mathbb{H}$ into $\mathbb{R}$ using the norm defined in Section 2.2.2. This mapping allows for the use of any real-valued loss function, and the Mean Absolute Error (MAE) loss function was selected due to its simplicity. The Mean Absolute Error is given by

$$\frac{1}{N}\sum_{i=1}^{N} |\hat{y} - y|, \tag{25}$$

where N is the number of data-points, $\hat{y}$ is the predicted value, and y is the truth or

target value.

To provide a baseline comparison for the QMLP+GA network, an equivalent real-valued network is constructed and trained using the same genetic algorithm as the QMLP+GA. Finally, an identical MLP is constructed and trained using the gradient descent (GD) algorithm. These two variants are referred to as the MLP+GA network and the MLP+GD network, respectively. The layers, nodes per layer, and total parameters of each of the three networks are summarized in Table 1. The real-valued hyperbolic tangent was used as the activation function on the input layer and both hidden layers, with a Mean Absolute Error loss function. However, since the hyperbolic tangent is globally analytic in $\mathbb{R}$, the normalization layers from the QMLP were removed. The learning rate η for the gradient descent algorithm was set to $\eta = 0.03$. The real-valued MLPs contained a total of 136 trainable weight and bias parameters, a six-fold increase over the QMLP.

Table 1. Neural Network Topologies for Ackley Function Approximation

Network	Input	Hidden 1	Hidden 2	Output	Parameters
QMLP+GA	1	3	3	1	22
MLP+GA	3	9	9	3	136
MLP+GD	3	9	9	3	136

3.2.2 Chaotic Time Series Prediction

Chaotic time series prediction of the Lorenz attractor requires multidimensional input data as well as multidimensional output data. It is a notoriously difficult problem, especially considering the system's sensitivity to initial conditions. In contrast with the function approximation experiment, the time series prediction experiment focused on the ability of quaternion networks to learn complex multidimensional nonlinearities. To that end, the time series prediction experiment centered on optimizing a set of quaternion network hyperparameters and did not consider any equivalent

real-valued networks.

To test the predictive capabilities of a simple QMLP+GA network, a set of 500 time series inputs were generated using a fixed-timestep 4th-order Runge-Kutta Ordinary Differential Equation (ODE) solver. The first 400 time series formed the training dataset, while the last 100 were held out for the test set. The starting point for each time series was randomly generated using a uniform $U[-10.0, 10.0]$ distribution for the x- and y-coordinates and a uniform $U[0.0, 10.0]$ distribution for the z-coordinates. Initial tests focused on relatively short time series inputs. Each series was generated over a range of 20 timesteps, and the first 10 values of each series formed the input training data, while the last 10 values formed the target values for training. Subsequent tests involved larger prediction windows, and both the 10-step and 100-step results are discussed in Chapter IV.

Figure 6 illustrates the sensitivity of the Lorenz system to initial starting conditions. Several initial starting points were generated using the distributions defined above for the x-, y-, and z-coordinates. Each system was then solved for 500 timesteps, starting at the initial position in 3-space. While each curve exhibits the characteristic "butterfly" shape, the individual coordinates of each series at each time step are drastically different.

Initial experiments showed that simple, smaller networks performed better with the genetic algorithm then larger networks. A 4-layer network was constructed for the time series prediction experiment. The structure of the network closely resembles an autoencoder network, where large input layers are scaled down throughout the network before being scaled back up for the output layer. This structure proved successful over several rounds of experimentation in predicting the 10-step ahead x, y, and z coordinates for the test set data. As a final experiment, a QMLP was created to predict the Lorenz coordinates 50 steps ahead based on in input time series

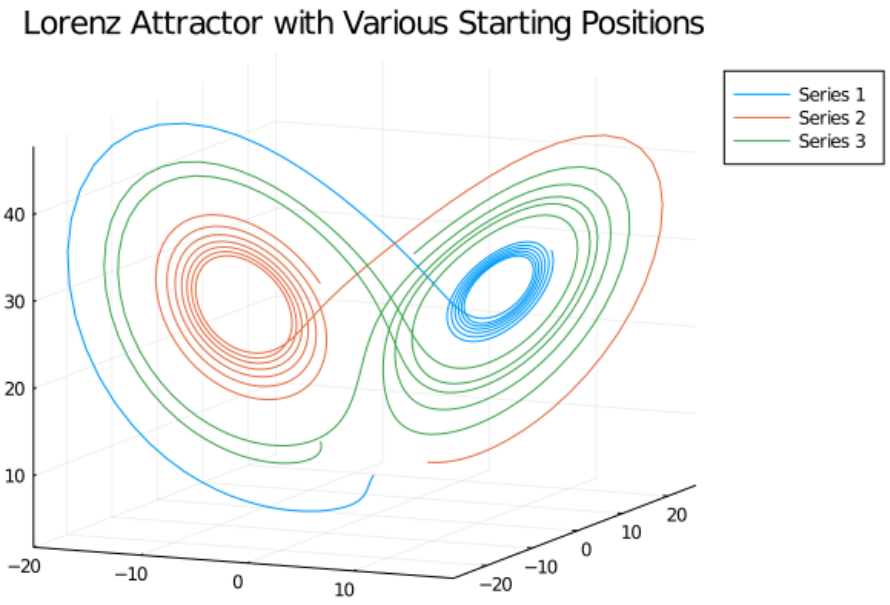

Figure 6. Impact of Initial Conditions on Lorenz System

of 25 steps. The layers, nodes per layer, and total parameters of each network are summarized in Table 2.

Table 2. Neural Network Topology for Chaotic Prediction

Network	Input	Hidden 1	Hidden 2	Output	Parameters
QMLP+GA	10	3	3	10	85
QMLP+GA	25	5	10	50	740

Before processing, the training and test datasets were cast into the quaternion domain using a vectorized approach. For an input vector τ_i, the corresponding quaternion input vector was constructed using the following approach:

$$\tau_i = \begin{bmatrix} \vec{x}_1 \\ \vec{x}_2 \\ \vdots \\ \vec{x}_{10} \end{bmatrix} \quad \Longrightarrow \quad \tau_{q_i} = \begin{bmatrix} 0.0 + x_1\mathbf{i} + y_1\mathbf{j} + z_1\mathbf{k} \\ 0.0 + x_2\mathbf{i} + y_2\mathbf{j} + z_2\mathbf{k} \\ \vdots \\ 0.0 + x_{10}\mathbf{i} + y_{10}\mathbf{j} + z_{10}\mathbf{k} \end{bmatrix} \tag{26}$$

Additionally, the target values were cast into quaternions. At each iteration, a quater-

nionic form of the Mean Absolute Error measured the fitness of each solution. Only the imaginary components of each input and target vector contained coordinate information, so this experiment introduced a $QMAE_{imag}$ calculation, defined in Equation (27) below.

$$\begin{aligned} QMAE_{imag} : &= \frac{1}{N}\sum_{i=1}^{N} ||\hat{y}_{\mathbf{q_i}} - y_{\mathbf{q_i}}||_{imag} \\ &= \frac{1}{N}\sum_{i=1}^{N} ||(\hat{x}_i\mathbf{i} + \hat{y}_i\mathbf{j} + \hat{z}_i\mathbf{k}) - (x_i\mathbf{i} + y_i\mathbf{j} + z_i\mathbf{k})|| \\ &= \frac{1}{N}\sum_{i=1}^{N} \left(\sqrt{(\hat{x}_i - x_i)^2 + (\hat{y}_i - y_i)^2 + (\hat{z}_i - z_i)^2}\right)^2. \end{aligned} \tag{27}$$

Since this experiment did not consider any real-valued networks, several quaternion activation functions were utilized during testing that are not available in the real-domain. In particular, [20] notes that quaternionic functions with local analytic conditions are isomorphic to analytic complex functions. Additionally, [24] demonstrate that hyperbolic and inverse hyperbolic trigonometric functions are universal approximators in the complex domain. This experiment explored the use of several quaternionic elementary transcendental functions and found the inverse hyperbolic tangent, defined in [29], to provide the best performance:

$$arctanh(p) := \frac{ln(1+p) - ln(1-p)}{2}. \tag{28}$$

While the Lorenz prediction QMLP+GA networks required a slightly different network structure than the Ackley function approximation networks, both networks employed an identical genetic algorithm in the training phase. This approach eliminated the need for differentiability of both the loss function and the activation functions of the network. Additionally, it eliminated the need for a quaternion partial derivative calculation, which is a notoriously difficult problem. Section 3.3 describes

the details of the algorithm, while Chapter IV discusses the results and performance of the algorithm in both experiments.

3.3 Quaternion Genetic Algorithm

This section describes the quaternion genetic algorithm that was developed to train the QMLP-GA. A simple change of the underlying data type from quaternions to real-valued inputs, weights, and biases enabled the training of the MLP-GA with an identical algorithm. This research took a similar approach to Uber's OpenAI Labs genetic algorithm training process [42], opting for a very basic algorithm with minimal enhancements to demonstrate the proof-of-concept. Based on the success of this approach in Uber's experiments as well as in the quaternion domain presented here, a more advanced algorithm incorporating any of the many algorithmic improvements would likely improve on the baseline results discussed in Chapter IV.

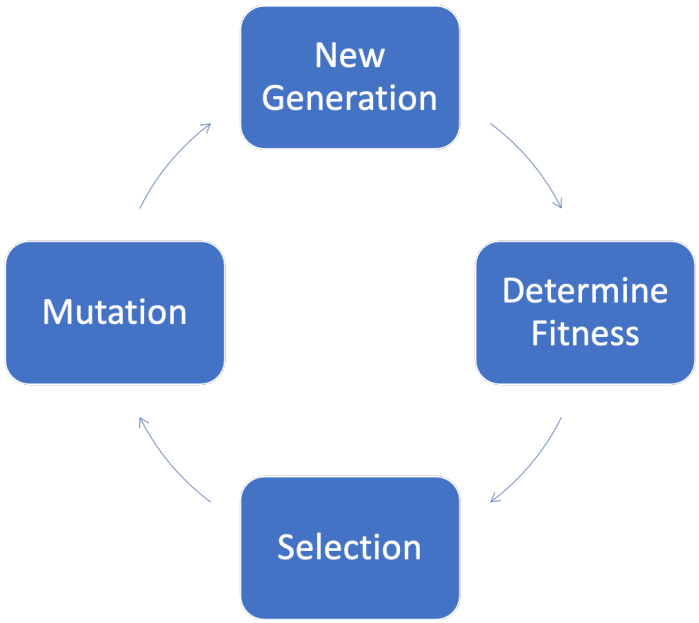

Figure 7. Genetic Algorithm/Genetic Programming Process

A general diagram of the genetic algorithm process flow is shown in Figure 7. A genetic algorithm is a population-based search method, operating on a population

of solutions to iteratively find better and better solutions. In this case, an individual neural network, defined by its weights and biases, represents a single solution. To initialize the algorithm, a population of $N = 20$ distinct neural networks were instantiated, with all weights and biases randomly generated following a Uniform distribution over $[-1, 1]$.

After instantiation, the algorithm measures the fitness of each solution. For each neural network, the entire training dataset is processed through the network, capturing the total MAE for each network. The networks are then rank-ordered based on the lowest MAE value.

Algorithm 1 Quaternion Genetic Algorithm

Instantiate $\mathcal{P}_m$ parent networks, $m \in N = \{1, \ldots, 20\}$, input mutation function ψ.
for $i \in N$ **do**
    Evaluate population fitness F_i
end for
for $g = 1$ to G generations **do**
    Sort population $\leftarrow F_i$
    Select best parents $\mathcal{P}_n^{g-1}$, $n = 1, \ldots, 5$
    for $j = n + 1$ to N **do**
        Generate $k =$ **UniformInt**$(1, n)$
        $\mathcal{P}_j^g = \psi(\mathcal{P}_k^{g-1})$.
    end for
end for
Return final population $\mathcal{P}_m^G$ for $m \in N$.

In the selection step, the n best solutions are retained as the "parents" for the next generation of the algorithm. In this research, $n = 5$ networks were retained as the parent generation in each iteration of the algorithm. While many advanced selection techniques exist, this work employed a simple rank selection, which selected the five best networks from each generation.

Finally, to generate a new population of solutions, the genetic algorithm performs a random mutation step, where a parent solution is randomly selected from the $n = 5$

best parent solutions. Then, the algorithm creates a "child" solution by mutating roughly half of the weights and biases of the parent solution with random noise. In this case, the generating distribution for the random noise was the standard Normal distribution, $\mathcal{N}(0, 1)$. This process repeats for $N - n = 20 - 5 = 15$ times to create the new generation of solutions.

This process is commonly referred to as a genetic program, where generations are created solely through the mutation process. Often, genetic algorithms will include an additional crossover step prior to mutation, where new child solutions are created using a selection of features from separate parent solutions. Crossover was omitted from this algorithm, since mutation alone provided good baseline performance, reiterating the fact that the most simple genetic algorithms are competitive to the popular backpropagation algorithm. A summary of the algorithm is shown in Algorithm 1.

3.4 Evaluation & Analysis Strategy

Each of the networks described in Section 3.2 processed the training data from the Ackley function and the Lorenz Attractor system. At each training epoch, the algorithms either recorded the MAE of the overall system in the case of the gradient descent network, or the MAE or (QMAE) of the best solution for the genetic algorithm networks. Additionally, several computational metrics were recorded including memory allocations and computational runtime. Finally, each of the trained models processed the test data, recording the test set percentage error for each instance. Chapter IV contains a discussion of network performance in each problem instance for each network in regards to these metrics.

IV. Results and Analysis

This chapter presents the results and analysis of the function approximation and chaotic time series prediction experiments described in Chapter III. This chapter is organized as follows. First, Section 4.1 describes the principal results from the function approximation experiment, noting that the results demonstrate the viability of both quaternion-valued networks and the quaternion genetic training algorithm. Then, Section 4.2 summarizes the results of the chaotic time series prediction experiment, describing the initial results and their significance.

All computations presented here were performed on a desktop workstation running Windows 10 Enterprise with 64 GB of RAM and dual Intel Xeon Silver 4108 CPUs. Each CPU contained 8 physical cores running at 1.80 GHz. Coding was performed in Julia 1.5.3 using the Quaternion.jl package and Flux.jl [18] for the MLP+GD network.

4.1 Function Approximation Results

The focus of the function approximation test was twofold. First, the function approximation task served as a proof-of-concept for the QMLP-GA. While quaternion neural networks and metaheuristic neural network training algorithms both exist separately in the literature, this work demonstrates the first use of metaheuristics to effectively train quaternion neural networks. Second, this experiment demonstrated some of the computational benefits that quaternions provide.

In keeping with these two goals, the three neural networks employed default parameters and very basic training algorithm implementations. No attempt was made to tune the hyperparameters of any of the models; instead, the results speak for themselves. The training set error for each of the three networks versus epoch is shown in Figure 8.

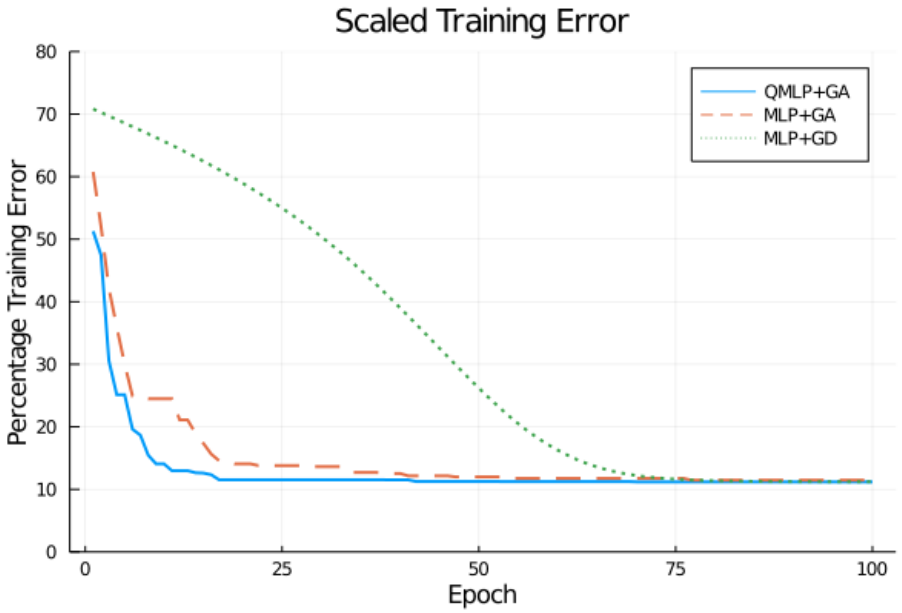

Figure 8. Training Set Mean Absolute Error for Each Network

The QMLP+GA initialized using the random uniform weight initialization scheme described in Chapter III had the lowest initial prediction error, at roughly 50% in the first epoch. In contrast, the MLP+GA started with nearly 60% initial error, while the MLP+GD was above 70%. The genetic algorithm improved rapidly, showing significantly faster initial algorithmic improvement versus the gradient descent algorithm. Both GA-trained networks showed rapid improvements over the first 25 training epochs, while the MLP+GD network searched for nearly 75 epochs before catching up to the GA-trained networks. The MLP+GD eventually caught up to the other two networks, but the prediction error remained slightly higher for the gradient descent network throughout the entire training process.

Table 3. Neural Network Comparison Results

Network	Runtime (sec)	Memory (GB)	Parameters	Test Error
QMLP+GA	**17.421**	10.238	**22**	**11.01**%
MLP+GA	18.069	**9.497**	136	11.15%
MLP+GD	955.040	778.027	136	11.23%

Table 3 shows the test set performance for each of the three networks across several

measures of merit. In each column, the best results are highlighted in bold text. The quaternion network had the fastest overall runtime, resulting in the lowest test set error with the fewest number of trainable parameters. The real-valued MLP had similar performance and required less overall system memory throughout the runtime of the algorithm, but required nearly six times the number of trainable parameters. Finally, the gradient descent-trained MLP had the worst performance in every category. While the test set error was comparable to the other two networks, the MLP+GD took more than 50 times as long to run with over 70 times as much memory allocated to store the gradient and error information for the backpropagation process.

These results, while cursory, clearly demonstrate the viability of quaternion networks trained with genetic algorithms. The quaternion network showed the fastest overall improvement, lowest final error, and lowest computational cost (in terms of runtime) when compared to two comparable networks. Additionally, the two GA-trained networks outperformed the gradient descent network across all measures of merit. These results validate the use of genetic algorithms in neural network training and show that quaternion networks can easily outperform equivalent real-valued networks involving multidimensional input data.

4.2 Time Series Prediction Results

Whereas the function approximation results demonstrate a viable proof-of-concept for quaternion neural networks, the chaotic time series prediction task illustrates the power of QNNs in the difficult task of predicting noisy systems. Additionally, chaotic time series prediction provides a natural multidimensional input + multidimensional output test that is almost tailor made for quaternion networks. In each of the figures displayed in this section, the orange graph represents the true chaotic time series, while the blue graph represents the predicted values. The final prediction results

presented in Figure 9 are far from current state-of-the-art results using deep recurrent neural networks (RNNs) or Long-Short Term Memory (LSTM) networks, yet they illustrate the ability of simple QNNs to learn complex nonlinearities over time.

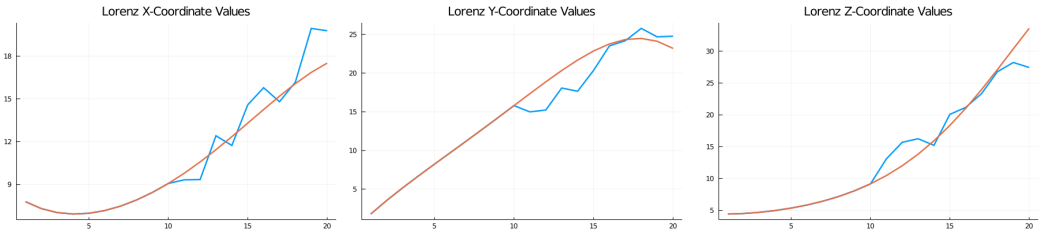

Figure 9. 10-Step Ahead Predicted Coordinate Values

This experiment utilized two distinct QNN network topologies. The first network predicted the Lorenz attractor for 10 timesteps in the future based on an input time series of 10 timesteps. The second network predicted the Lorenz attractor for 50 timesteps in the future based on an input of 25 observations. The structure of each network is listed in Table 2, while the results for both networks are listed in Table 4. The test error percentage listed in Table 4 was measured using the Mean Absolute Percentage Error (MAPE) for time series forecasting, defined in Equation (29), where e_t is the unscaled prediction error for observation t and y_t is the target value at t.

$$MAPE = mean\left(\left|100\frac{e_t}{y_t}\right|\right) \tag{29}$$

Early tests indicated that smaller networks performed better with the genetic algorithm. The final two networks contained comparatively few nodes in each layer and were structured as autoencoder networks, which perform a type of downsampling and subsequent upsampling as information passes through the network. Each network was trained for 50,000 epochs, which equated to roughly 28 minutes for the 10-step prediction network and around 4 hours for the 50-step prediction network.

The test set error listed in Table 4 indicates that on average, individual predicted values were off by about 11%. The actual versus predicted x-, y-, and z-coordinates

Table 4. Lorenz Prediction Results

Prediction Steps	Runtime (sec)	Memory (GB)	Params	Test Error
10	1668.565	947.304	85	10.89%
50	14769.069	2.815 (TB)	740	9.59%

for one of the test set time series are shown in Figure 9, while two 3-dimensional path predictions are shown in Figure 10. While the test error is relatively high, the QMLP+GA performs remarkably well on future predictions, especially in the long sweeping sections of the Lorenz attractor curves. The errors understandably grow and compound in the two "wings" of the curve, where the graph circles closely around each pole of the attractor.

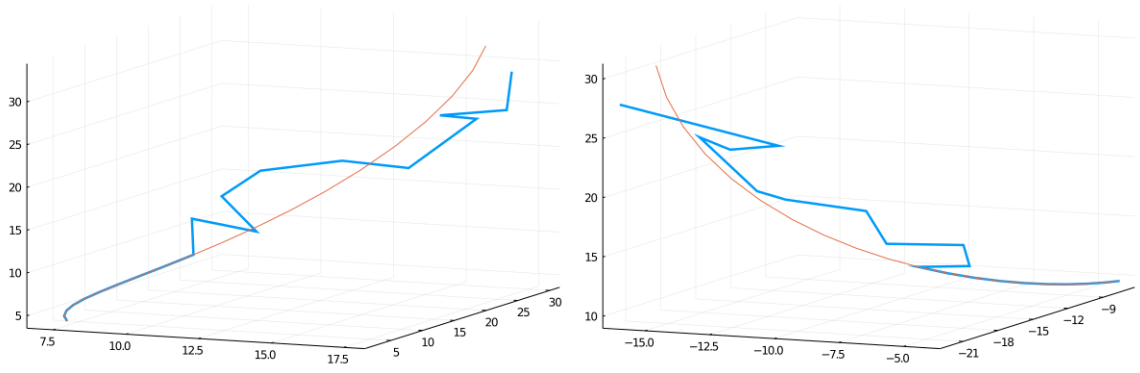

Figure 10. 10-Step Ahead Path Predictions

The final experiment tested the ability of the QMLP to predict long sequences based on a relatively short input. The network was trained over 50,000 epochs to predict 50 observations based on an input sequence of length 25. Table 4 summarizes several measures of merit for the network, while the x-, y-, and z-coordinate results for a representative test set sequence are shown in Figure 11.

In each coordinate direction, there is some clear noise at each prediction step, but the network accurately predicts the general motion of each variable. The motion of each prediction path is even more evident in the 3-dimensional plots shown in Figure 12, which shows two path predictions for two series from the test set data.

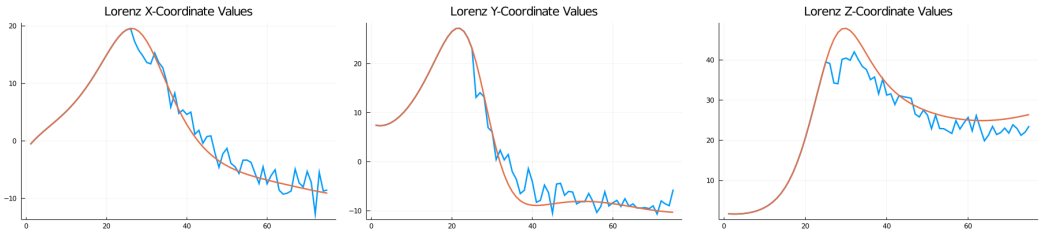

Figure 11. 50-Step Ahead Predicted Coordinate Values

As with the 10-step prediction model, the 50-step model makes the best predictions along the long sweeping arcs of the system, with errors compounding near the two "wings" of the attractor.

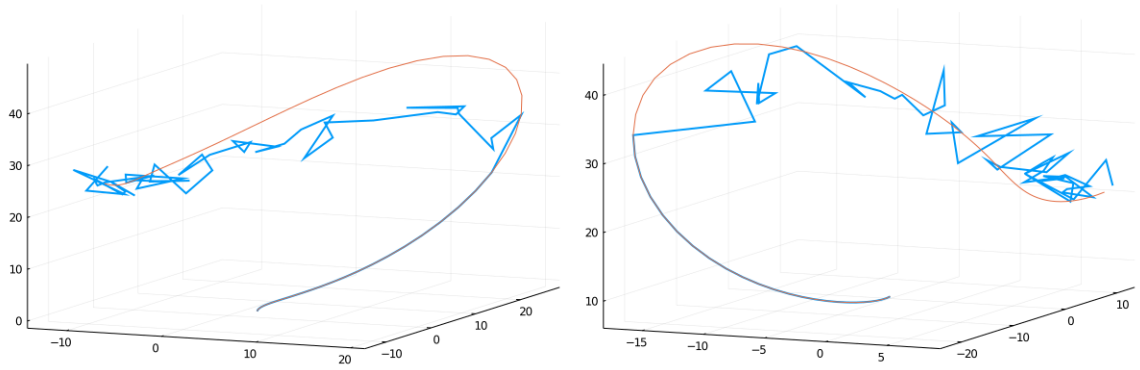

Figure 12. 50-Step Ahead Path Predictions

Finally, the unscaled training error plots for both networks are shown in Figure 15. The genetic algorithm showed similar performance in both time series prediction tasks as it did in the function approximation task, with dramatic initial improvements and slow but consistent improvements as the iterations progressed.

4.3 Discussion

In the first experiment, the three networks all utilized a random uniform weight initialization scheme. However, the quaternion network had between 10-20% lower initial prediction error than the real-valued networks. This is likely due to the fact that the quaternion network employed six times fewer weight and bias parameters

than the real-valued networks. The quaternion network maintained the lowest training set error across the entire 100-epoch training period, resulting in the best test set performance. The larger networks constructed in the second experiment demonstrated similar training characteristics and test set performance. Surprisingly, the 50-step prediction experiment resulted in lower test set prediction error than the 10-step prediction network, likely due to the scale of each predicted value.

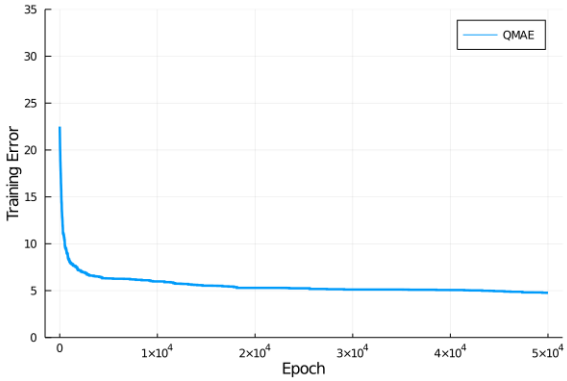

Figure 13. 10-Step Predictions

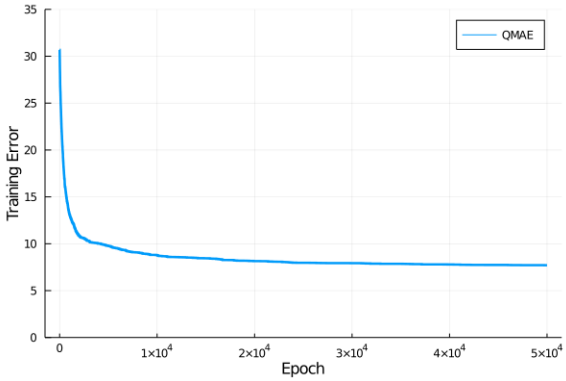

Figure 14. 50-Step Predictions

Figure 15. Unscaled QMAE Training Error

The genetic algorithm removes the need for expensive gradient calculations, resulting in better memory performance and more than 50x faster runtime in the first experiment versus the real-valued gradient descent algorithm. Given the difficulty of calculating quaternion gradients, the improvement over a quaternion gradient descent algorithm would likely be even greater. However, a genetic programming approach does come with some drawbacks. In the naive approach presented here, the algorithm would sometimes stall for several iterations while searching for an improving solution. There are many existing techniques designed to mitigate this stalling, but the literature on genetic algorithms is much less developed compared to comparable work on gradient descent optimization.

Finally, the genetic algorithm opened the aperture on viable activation functions and loss functions for use with quaternion networks. This is perhaps the most sig-

nificant contribution of this research. The results from [20] indicate that any locally analytic complex-valued activation function can be extended and used in the quaternion domain, but this work presents the first successful implementation of inverse hyperbolic trigonometric functions in quaternion networks. The success of the inverse hyperbolic tangent function in the chaotic time series prediction task demonstrate the value of using gradient free optimization methods in the quaternion domain. Chapter V presents the final conclusions of this work, along with several recommendations for future work.

V. Conclusions and Recommendations

5.1 Conclusions

In summary, this research presented a novel approach to training quaternion neural networks using a simple genetic algorithm. This approach provides two crucial improvements over current processes. First, the genetic algorithm allows for the use of a wide range of quaternion-valued loss and activation functions, including non-analytic functions and elementary transcendental functions. Current quaternion networks are restricted to either a small set of pure quaternion functions such as the hyperbolic tangent, or split activation functions which provide suboptimal performance. Second, the genetic algorithm approach removes the need for quaternion calculus rules, since the GA-based approach does not require any gradient information. This provides a significant improvement in terms of computational cost and overall network complexity. The preliminary results presented here demonstrate the viability of this process in two multidimensional learning tasks, with the potential to improve on current neural network structures in terms of computational runtime, storage requirements, and network simplicity.

5.2 Recommendations

The quaternions and quaternion neural networks are relatively unexplored compared to real analysis and real-valued neural networks. While certain applications in image processing and other domains have driven research in the quaternions and QNNs, there is still room for significant improvement in both the theoretical and practical aspects of quaternions. Going forward, the following lines of research will be crucial for continued innovation in the quaternion domain.

First, a solid foundation of quaternionic analysis is crucial to theoretically sound

QNN research. While a handful of researchers have published works on quaternionic analysis, the corpus is quite thin. Research in novel quaternion activation functions, quaternion differentiability, quaternion analytic conditions, and novel quaternion training algorithms could significantly enhance both the current understanding of quaternion optimization as well as quaternion implementations of common machine learning models. Additionally, the quaternion Universal Approximation Theorem for either split or pure quaternion activation functions is an outstanding problem that is vital for establishing the legitimacy of quaternion networks from a theoretical point of view. Proving either variant of the Universal Approximation Theorem would be a substantial contribution to the field.

Finally, this research simply provided a proof-of-concept for GA-trained quaternion neural networks. The two examples presented were limited in scope and future work should build on these results to demonstrate the viability of GA-trained networks in large-scale optimization problems. In particular, quaternions are particularly well-suited to the fields of image processing and robotic control, both of which have a plethora of neural network-related application opportunities.

Bibliography

1. Ackley, D. [2012], *A connectionist machine for genetic hillclimbing*, Vol. 28, Springer Science & Business Media.

2. Arena, P., Baglio, S., Fortuna, L. and 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, IEEE, pp. 1790–1794.

3. ARENA, P., CAPONETTO, R., FORTUNA, L., MUSCATO, G. and XIBILIA, M. G. [1996], 'Quaternionic multilayer perceptrons for chaotic time series prediction', *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* **79**(10), 1682–1688.

4. Arena, P., Fortuna, L., Muscato, G. and Xibilia, M. G. [1998], *Neural Networks in Multidimensional Domains: fundamentals and new trends in modelling and control*, Vol. 234, Springer.

5. Arena, P., Fortuna, L., Occhipinti, L. and Xibilia, M. G. [1994], Neural networks for quaternion-valued function approximation, *in* 'Proceedings of IEEE International Symposium on Circuits and Systems-ISCAS'94', Vol. 6, IEEE, pp. 307–310.

6. Arena, P., Fortuna, L., Re, R. and Xibilia, M. G. [1993], On the capability of neural networks with complex neurons in complex valued functions approximation, *in* '1993 IEEE International Symposium on Circuits and Systems', IEEE, pp. 2168–2171.

7. Brown, J. W., Churchill, R. V. et al. [2009], *Complex variables and applications*, Boston: McGraw-Hill Higher Education,.

8. Buchholz, S. and Le Bihan, N. [2006], Optimal separation of polarized signals by quaternionic neural networks, *in* '2006 14th European Signal Processing Conference', IEEE, pp. 1–5.

9. Cybenko, G. [1989], 'Approximation by superpositions of a sigmoidal function', *Mathematics of control, signals and systems* **2**(4), 303–314.

10. Fister, I., Yang, X.-S., Brest, J. and Fister Jr, I. [2013], 'Modified firefly algorithm using quaternion representation', *Expert Systems with Applications* **40**(18), 7220–7230.

11. Fortuna, L., Muscato, G. and Xibilia, M. G. [2001], 'A comparison between hmlp and hrbf for attitude control', *IEEE transactions on neural networks* **12**(2), 318–328.

12. Gaudet, C. J. and Maida, A. S. [2018], Deep quaternion networks, *in* '2018 International Joint Conference on Neural Networks (IJCNN)', IEEE, pp. 1–8.

13. Géron, A. [2019], *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, O'Reilly Media.

14. Greenblatt, A., Mosquera-Lopez, C. and Agaian, S. [2013], Quaternion neural networks applied to prostate cancer gleason grading, *in* '2013 IEEE International Conference on Systems, Man, and Cybernetics', IEEE, pp. 1144–1149.

15. 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.

16. Hornik, K., Stinchcombe, M. and White, H. [1989], 'Multilayer feedforward networks are universal approximators', *Neural networks* **2**(5), 359–366.

17. Hosseini, S. H. and Samanipour, M. [2015], 'Prediction of final concentrate grade using artificial neural networks from gol-e-gohar iron ore plant', *American Journal of Mining and Metallurgy* **3**(3), 58–62.

18. Innes, M. [2018], 'Flux: Elegant machine learning with julia', *Journal of Open Source Software* .

19. Isokawa, T., Kusakabe, T., Matsui, N. and Peper, F. [2003], Quaternion neural network and its application, *in* 'International conference on knowledge-based and intelligent information and engineering systems', Springer, pp. 318–324.

20. Isokawa, T., Nishimura, H. and Matsui, N. [2012], 'Quaternionic multilayer perceptron with local analyticity', *Information* **3**(4), 756–770.
URL: *https://www.mdpi.com/2078-2489/3/4/756*

21. James, G., Witten, D., Hastie, T. and Tibshirani, R. [2013], *An introduction to statistical learning*, Vol. 112, Springer.

22. Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J. and Aila, T. [2020], Analyzing and improving the image quality of StyleGAN, *in* 'Proc. CVPR'.

23. Khuat, T. T. and Le, M. H. [2017], 'A genetic algorithm with multi-parent crossover using quaternion representation for numerical function optimization', *Applied Intelligence* **46**(4), 810–826.

24. Kim, T. and Adalı, T. [2003], 'Approximation by fully complex multilayer perceptrons', *Neural computation* **15**(7), 1641–1666.

25. Lorenz, E. N. [1963], 'Deterministic nonperiodic flow', *Journal of atmospheric sciences* **20**(2), 130–141.

26. Matsui, N., Isokawa, T., Kusamichi, H., Peper, F. and Nishimura, H. [2004], 'Quaternion neural network with geometrical operators', *Journal of Intelligent & Fuzzy Systems* **15**(3, 4), 149–164.

27. McCulloch, W. S. and Pitts, W. [1943], 'A logical calculus of the ideas immanent in nervous activity', *The bulletin of mathematical biophysics* **5**(4), 115–133.

28. Minsky, M. and Papert, S. A. [2017], *Perceptrons: An introduction to computational geometry*.

29. Morais, J. P., Georgiev, S. and Sprößig, W. [2014], *Real quaternionic calculus handbook*, Springer.

30. Ojha, V. K., Abraham, A. and Snášel, V. [2017], 'Metaheuristic design of feedforward neural networks: A review of two decades of research', *Engineering Applications of Artificial Intelligence* **60**, 97–116.

31. Oliva, D., Abd Elaziz, M. and Hinojosa, S. [2019], *Metaheuristic algorithms for image segmentation: theory and applications*, Springer.

32. Papa, J. P., de Rosa, G. H. and Yang, X.-S. [2018], On the hypercomplex-based search spaces for optimization purposes, *in* 'Nature-Inspired Algorithms and Applied Optimization', Springer, pp. 119–147.

33. Papa, J. P., Rosa, G. H., Pereira, D. R. and Yang, X.-S. [2017], 'Quaternion-based deep belief networks fine-tuning', *Applied Soft Computing* **60**, 328–335.

34. Papa, J., Pereira, D., Baldassin, A. and Yang, X.-S. [2016], On the harmony search using quaternions, *in* 'IAPR Workshop on Artificial Neural Networks in Pattern Recognition', Springer, pp. 126–137.

35. Parcollet, T., Morchid, M., Bousquet, P.-M., Dufour, R., Linarès, G. and De Mori, R. [2016], Quaternion neural networks for spoken language understanding, *in* '2016 IEEE Spoken Language Technology Workshop (SLT)', IEEE, pp. 362–368.

36. Parcollet, T., Morchid, M. and Linares, G. [2020], 'A survey of quaternion neural networks', *Artificial Intelligence Review* **53**(4), 2957–2982.

37. Parcollet, T., Ravanelli, M., Morchid, M., Linarès, G., Trabelsi, C., De Mori, R. and Bengio, Y. [2018], 'Quaternion recurrent neural networks', *arXiv preprint arXiv:1806.04418* .

38. Rosenblatt, F. [1958], 'The perceptron: a probabilistic model for information storage and organization in the brain.', *Psychological review* **65**(6), 386.

39. Rumelhart, D. E., Hinton, G. E. and Williams, R. J. [1985], Learning internal representations by error propagation, Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

40. Stanley, K. O., D'Ambrosio, D. B. and Gauci, J. [2009], 'A hypercube-based encoding for evolving large-scale neural networks', *Artificial life* **15**(2), 185–212.

41. Stanley, K. O. and Miikkulainen, R. [2002], 'Evolving neural networks through augmenting topologies', *Evolutionary Computation* **10**(2), 99–127.
URL: *https://doi.org/10.1162/106365602320169811*

42. Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O. and Clune, J. [2017], 'Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning', *arXiv preprint arXiv:1712.06567* .

43. Ujang, B. C., Jahanchahi, C., Took, C. C. and Mandic, D. [2010], 'Quaternion valued neural networks and nonlinear adaptive filters'.

44. Ujang, B. C., Took, C. C. and Mandic, D. P. [2010], 'Split quaternion nonlinear adaptive filtering', *Neural Networks* **23**(3), 426–434.

45. US Department of Defense, U. [2019], 'Summary of the 2018 department of defense artificial intelligence strategy: Harnessing ai to advance our security and prosperity'.

46. Werbos, P. [1974], 'Beyond regression:" new tools for prediction and analysis in the behavioral sciences', *PhD dissertation, Harvard University* .

47. Xu, D., Jahanchahi, C., Took, C. C. and Mandic, D. P. [2015], 'Enabling quaternion derivatives: the generalized hr calculus', *Royal Society open science* **2**(8), 150255.

48. Xu, D., Zhang, L. and Zhang, H. [2017], 'Learning algorithms in quaternion neural networks using ghr calculus', *Neural Network World* **27**(3), 271.

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

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
25–03–2021	Master's Thesis	August 2019 — March 2021

4. TITLE AND SUBTITLE

Meta-Heuristic Optimization Methods for Quaternion-Valued Neural Networks

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Bill, Jeremiah P., Capt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENS-MS-21-M-143

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
Dr. Trevor Bihl, Senior Electronics Engineer
2241 Avionics Cir
Wright-Patterson AFB, OH 45433
trevor.bihl.2@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RYAR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution Statement A. Approved for Public Release; Distribution Unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

In recent years, real-valued neural networks have demonstrated promising, and often striking, results across a broad range of domains. This has driven a surge of applications utilizing high-dimensional datasets. While many techniques exist to alleviate issues of high-dimensionality, they all induce a cost in terms of network size or computational runtime. This work examines the use of quaternions, a form of hypercomplex numbers, in neural networks. The constructed networks demonstrate the ability of quaternions to encode high-dimensional data in an efficient neural network structure, showing that hypercomplex neural networks reduce the number of total trainable parameters compared to their real-valued equivalents. Finally, this work introduces a novel training algorithm using a meta-heuristic approach that bypasses the need for analytic quaternion loss or activation functions. This algorithm allows for a broader range of activation functions over current quaternion networks and presents a proof-of-concept for future work.

15. SUBJECT TERMS

machine learning, evolutionary computation, quaternion metaheuristics

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Lance E. Champagne, AFIT/ENS
U	U	U	UU	57	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, 4646; lance.champagne@afit.edu