International Journal of Advances in Applied Sciences (IJAAS)
Vol. 7, No. 2, June 2018, pp. 177~190
ISSN: 2252-8814, DOI: 10.11591/ijaas.v7.i2.pp177-190

On the High Dimentional Information Processing in Quaternionic Domain and its Applications

Sushil Kumar, Bipin Kumar Tripathi
Department of Computer Science and Engineering, Harcourt Butler Technical University, Kanpur, India

Article Info

Article history:

Received Des 28, 2017
Revised Apr 12, 2018
Accepted May 11, 2018

Keyword:

3D imaging
3D motion
Quaternion
Quaternionic domain neural network

ABSTRACT

There are various high dimensional engineering and scientific applications in communication, control, robotics, computer vision, biometrics, etc.; where researchers are facing problem to design an intelligent and robust neural system which can process higher dimensional information efficiently. The conventional real-valued neural networks are tried to solve the problem associated with high dimensional parameters, but the required network structure possesses high complexity and are very time consuming and weak to noise. These networks are also not able to learn magnitude and phase values simultaneously in space. The quaternion is the number, which possesses the magnitude in all four directions and phase information is embedded within it. This paper presents a well generalized learning machine with a quaternionic domain neural network that can finely process magnitude and phase information of high dimension data without any hassle. The learning and generalization capability of the proposed learning machine is presented through a wide spectrum of simulations which demonstrate the significance of the work.

Corresponding Author:

Sushil Kumar,
Department of Computer Science and Engineering,
Harcourt Butler Technical University, Kanpur, India
Email: sushil0402k5@gmail.com

1. INTRODUCTION

The high dimensional information processing through neural network is emerging as a fascinating but challenging field of research in the second generation neurocomputing. The recent researches in high dimensional neural networks have established their superiority [1], [2], [3], [4] over real-valued or first generation neural networks. Although, real-valued neural networks (RVNN) have been used to process high dimensional data, but the network needs to employ too many neurons resulting huge structure and slow learning. The RVNN can also not process phase information during learning and generalization of mapping on the plane [2], [5], [6]. The complex-valued neural networks (CVNN) can promptly process two dimensional information with phase as a single number, which leads to a drastic reduction in the complexity of the network along with better performance. But, neural network of three dimensional information still needs an exhaustive investigation. The applications with three dimensional information are popular in computer vision, robotics, biometrics, bioinformatics etc. The few researchers attempted machine learning with three dimensional information considering it as a vector [7], [8]. The corresponding learning algorithms have restrictions on weight matrix and a vector does not provide freedom like a complex number, as in CVNN [8]. Thus, it is very demanding to have neural network, which may promptly process different high dimensional parameters as numbers and can be simply incorporated in various applications of intelligent machine design, like CVNN [9], [2]-[3]. In the enhancement of higher order number systems the complex numbers (2D), quaternions (4D), octaves (8D), sedenions (16D) were developed by mathematicians in the past but there is no number system in three dimensions [10]. The researches [1-3, 6]also elaborate that the

CVNN has outperformed over RVNN even for real-valued problems, therefore we propose to exploit quaternions in neural network to process three dimensional problems.

The neurocomputing with high dimensional number systems will definitely overcome from learning and generalization of huge conventional neural network and lead to lower complexity. The quaternion is one of the hypercomplex number introduced by Iris mathematician Hamilton [11] which has been extensively employed in the field of quantum mathematics, physics, computer graphics, signal processing and control [12-13, 18-17]. This number system has recently popped up in neural network through quaternionic neurons, as complex or real -valued neurons, to develop efficient machine learning in higher dimensions. Few attempts have been made in this direction, the orthogonal decision boundary of single quaternionic neuron has been utilized to solve 4-bit parity problem in [14]; quaternionic MLPs proposed in [15] has the problem of existence of singularities; quaternion-valued algorithms are proposed for adaptive filtering [18]. [17]; a basic work for quaternionic-valued neural network with sigmoidal activation function is presented in [18, 19]. In this paper, we present not only simple, straightforward, but potential machine learning algorithm for sufficient general structure of the quaternionic domain neural network (QDNN) but also demonstrate the evaluation over the wide spectrum of applications, like function approximation, motion interpretation and recognition in space. The parameters in QDNN, like synaptic weights, biases, inputs-outputs signals and internal potentials are quaternions and represented as quaternion matrix, in multilayer neural network. Although, Hamilton proposed quaternionic numbers ($\boldsymbol{q} = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}$) for 4D number system [11], but it can also bring into play any 3D information in the space after equating its real part zero. The presented learning algorithm based on the error backpropagation for QDNN can efficiently solve any typical class of problems in 3D and 4D. The analytic [1, 8] or split type [1], [5], [7] activation functions have been chosen for complex-valued neuron which have their own issues concerning boundedness and analyticity. Therefore, selection of suitable activation function for neuron dealing with quaternion is one of the important concerns. The split type function may not be appropriate when analyticity is concerned, similarly the analytic function is not suitable when the singularity arises. The presented QDNN prefer boundednes over analyticity and use "split-type" activation function. The QDNN outperform with lesser number of neurons and faster learning where conventional real-valued neural network (RVNN) lacks. The quaternionic-valued neural network (QDNN) has an ability to learn and generalize 3D motion of objects and recognition of the point cloud object, but RVNN cannot, because QDNN has ability to capture and maintain phase information of each point during the learning and generalization.

This paper investigates the general structure of QDNN with learning algorithm through simulation on various benchmark problems of different sphere of influence. The sections and sub-sections of the paper are organized as follows: The section 2, presents a complete machine learning framework with pseudo code of learning in quaternionic domain. Section 3 evaluates the learning and generalization capability through function approximations, linear transformations and 3D face recognition. Section 4 presents the final conclusion and future scope of the work.

2. MACHINE LEARNING IN QUATERNIONIC DOMAIN

A quaternionic number system is the straightforward extension of real and complex number system, where four components are incorporated in single number; the first component acts as real and other three as imaginary with unit vectors ($\boldsymbol{i}$, $\boldsymbol{j}$, $\boldsymbol{k}$). These imaginary components overlie on the axes in three dimensional space [11, 12]. A quaternionic variable ($\boldsymbol{q} = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}$) consists of a real component (q_0) and three imaginary components (q_1, q_2, q_3). Its bases ($\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$) are orthogonal special vectors. Thus, they follow the properties as $\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = -1$ and cross product properties as $\boldsymbol{i} \times \boldsymbol{j} = -(\boldsymbol{j} \times \boldsymbol{i}) = \boldsymbol{k}$, $\boldsymbol{j} \times \boldsymbol{k} = -(\boldsymbol{k} \times \boldsymbol{j}) = \boldsymbol{i}$, $\boldsymbol{k} \times \boldsymbol{i} = -(\boldsymbol{i} \times \boldsymbol{k}) = \boldsymbol{j}$. In a prominent representation, a quaternion ($\boldsymbol{q}$) can be expressed in the form of a matrix (quaternionic matrix):

$$\boldsymbol{q} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \\ -q_1 & q_0 & -q_3 & q_2 \\ -q_2 & q_3 & q_0 & -q_1 \\ -q_3 & -q_2 & q_1 & q_0 \end{bmatrix}. \quad (1)$$

The bold type letter denotes quaternionic variable or quaternionic matrix. The conjugate of quaternionic variable ($\boldsymbol{q}^* = q_0 - q_1\boldsymbol{i} - q_2\boldsymbol{j} - q_3\boldsymbol{k}$) is similar to complex conjugate and the conjugate of quaternionic matrix denotes the transpose of the quaternionic matrix, defined as:

$$\boldsymbol{q}^{*} = \boldsymbol{q}^{T} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \\ -q_1 & q_0 & -q_3 & q_2 \\ -q_2 & q_3 & q_0 & -q_1 \\ -q_3 & -q_2 & q_1 & q_0 \end{bmatrix}^{T} = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{bmatrix}. \quad (2)$$

The machine learning optimization technique incorporates the basic operations of quaternion algebra [11, 12]. The addition and subtraction of two quaternionic matrices $\boldsymbol{q}$ and $\boldsymbol{r}$ cab be obtained simply as matrix operations. The multiplication of two quaternionic matrices $\boldsymbol{q}$ and $\boldsymbol{r}$ does not follow the commutative property ($\boldsymbol{qr} \neq \boldsymbol{rq}$). The inner product of two quaternionic matrices $\boldsymbol{q}$ and $\boldsymbol{r}$ is expressed by:

$$\boldsymbol{q} \odot \boldsymbol{r} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \\ -q_1 & q_0 & -q_3 & q_2 \\ -q_2 & q_3 & q_0 & -q_1 \\ -q_3 & -q_2 & q_1 & q_0 \end{bmatrix} \odot \begin{bmatrix} r_0 & r_1 & r_2 & r_3 \\ -r_1 & r_0 & -r_3 & r_2 \\ -r_2 & r_3 & r_0 & -r_1 \\ -r_3 & -r_2 & r_1 & r_0 \end{bmatrix} = \begin{bmatrix} q_0 r_0 & q_1 r_1 & q_2 r_2 & q_3 r_3 \\ q_1 r_1 & q_0 r_0 & q_3 r_3 & q_2 r_2 \\ q_2 r_2 & q_3 r_3 & q_0 r_0 & q_1 r_1 \\ q_3 r_3 & q_2 r_2 & q_1 r_1 & q_0 r_0 \end{bmatrix}. \quad (3)$$

The norm of quaternionic matrix $\boldsymbol{q}$ is expressed as:

$$\|\boldsymbol{q}\| = \frac{1}{2}\sqrt{\sum diag(\boldsymbol{q}\boldsymbol{q}^{T})} = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \quad (4)$$

2.1. Learning in Quaternionic Domain Neural Networks

Let a three layer $(L - M - N)$ QDNN possesses L inputs; M and N quaternionic neurons in hidden and output layers respectively. All inputs, outputs, weights and biases signals are considered as quaternionic matrices, as represented in Eq. (1). The derivation of optimization technique incorporates the basic operations of quaternion algebra which present the compact and the generalized derivation of the backpropagation algorithm (QDBP) of three-layer network. The bold letters denote the quternionic matrix or matrix containing quaternionic matrices as elements.

2.1.1. Forward Pass

Let us consider $I_l^r, I_l^x, I_l^y, I_l^z$ be the 4D quaternionic input of $l^{th}(l = 1 \dots L)$ neuron in the input layer of the network. The quaternionic input can be expressed as a quaternionic matrix ($\boldsymbol{I}_l$):

$$\boldsymbol{I}_l = \begin{bmatrix} I_l^r & I_l^x & I_l^y & I_l^z \\ -I_l^x & I_l^r & -I_l^z & I_l^y \\ -I_l^y & I_l^z & I_l^r & -I_l^x \\ -I_l^z & -I_l^y & I_l^x & I_l^r \end{bmatrix}. \quad (5)$$

The matrix of inputs ($\boldsymbol{I}$) at the input layer of the network is defined by: $\boldsymbol{I} = [\boldsymbol{I}_1\ \boldsymbol{I}_2\ \boldsymbol{I}_3 \cdots\ \boldsymbol{I}_L]^T$ (6)

The initialization of synaptic connection weights $\boldsymbol{w}_{ml}$ and $\boldsymbol{s}_{nm}$ are defined for l^{th} input to $m^{th}(m = 1 \dots M)$ hidden neuron pair and for m^{th} hidden to $n^{th}(n = 1 \dots N)$ output neuron pair of network respectively. These weights are presented in quaternionic matrices containing a real and other three imaginary components as follows:

$$\boldsymbol{w}_{ml} = \begin{bmatrix} w_{ml}^r & w_{ml}^x & w_{ml}^y & w_{ml}^z \\ -w_{ml}^x & w_{ml}^r & -w_{ml}^z & w_{ml}^y \\ -w_{ml}^y & w_{ml}^z & w_{ml}^r & -w_{ml}^x \\ -w_{ml}^z & -w_{ml}^y & w_{ml}^x & w_{ml}^r \end{bmatrix}. \quad (7)$$

$$\boldsymbol{s}_{nm} = \begin{bmatrix} s_{nm}^r & s_{nm}^x & s_{nm}^y & s_{nm}^z \\ -s_{nm}^x & s_{nm}^r & -s_{nm}^z & s_{nm}^y \\ -s_{nm}^y & s_{nm}^z & s_{nm}^r & -s_{nm}^x \\ -s_{nm}^z & -s_{nm}^y & s_{nm}^x & s_{nm}^r \end{bmatrix}. \quad (8)$$

Similarly, the initialization of biases $\boldsymbol{\alpha}_m$ and $\boldsymbol{\beta}_n$ are defined for m^{th} hidden and n^{th} output neuron of network:

$$\boldsymbol{\alpha}_m = \begin{bmatrix} \alpha_m^r & \alpha_m^x & \alpha_m^y & \alpha_m^z \\ -\alpha_m^x & \alpha_m^r & -\alpha_m^z & \alpha_m^y \\ -\alpha_m^y & \alpha_m^z & \alpha_m^r & -\alpha_m^x \\ -\alpha_m^z & -\alpha_m^y & \alpha_m^x & \alpha_m^r \end{bmatrix}. \tag{9}$$

$$\boldsymbol{\beta}_n = \begin{bmatrix} \beta_n^r & \beta_n^x & \beta_n^y & \beta_n^z \\ -\beta_n^x & \beta_n^r & -\beta_n^z & \beta_n^y \\ -\beta_n^y & \beta_n^z & \beta_n^r & -\beta_n^x \\ -\beta_n^z & -\beta_n^y & \beta_n^x & \beta_n^r \end{bmatrix} \tag{10}$$

The internal potential matrix $\boldsymbol{U}$, for neurons (1 .. M) at hidden layer of the network is defined as:

$$\boldsymbol{U} = \boldsymbol{WI} + \boldsymbol{\alpha}. \tag{11}$$

$$\begin{bmatrix} \boldsymbol{U}_1 \\ \boldsymbol{U}_2 \\ \boldsymbol{U}_3 \\ \vdots \\ \boldsymbol{U}_M \end{bmatrix} = \begin{bmatrix} \boldsymbol{w}_{11}\, \boldsymbol{w}_{12}\, \boldsymbol{w}_{13} \ldots \boldsymbol{w}_{1L} \\ \boldsymbol{w}_{21}\, \boldsymbol{w}_{22}\, \boldsymbol{w}_{23} \ldots \boldsymbol{w}_{2L} \\ \boldsymbol{w}_{31}\, \boldsymbol{w}_{32}\, \boldsymbol{w}_{33} \ldots \boldsymbol{w}_{3L} \\ \vdots \quad \vdots \quad \vdots \quad \vdots \\ \boldsymbol{w}_{M1} \boldsymbol{w}_{M2} \boldsymbol{w}_{M3} \cdots \boldsymbol{w}_{ML} \end{bmatrix} \begin{bmatrix} \boldsymbol{I}_1 \\ \boldsymbol{I}_2 \\ \boldsymbol{I}_3 \\ \vdots \\ \boldsymbol{I}_L \end{bmatrix} + \begin{bmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \\ \boldsymbol{\alpha}_3 \\ \vdots \\ \boldsymbol{\alpha}_M \end{bmatrix}. \tag{12}$$

where, elements of weight matrix $\boldsymbol{W}$ contains corresponding weights between input to hidden neurons and elements of bias matrix $\boldsymbol{\alpha}$ contains biases of hidden neurons. Let f be an activation function and f' be its derivative. The output matrix ($\boldsymbol{O}$) is obtained by split-type activation function over internal potential matrix ($\boldsymbol{U}$) at hidden layer:

$$O=f(U). \tag{13}$$

$$[\boldsymbol{O}_1\, \boldsymbol{O}_2 \ \ldots \ \boldsymbol{O}_m \cdots \boldsymbol{O}_M]^T = [f(\boldsymbol{U}_1)\, f(\boldsymbol{U}_2) \ \ldots \ f(\boldsymbol{U}_m) \cdots f(\boldsymbol{U}_M)]^T. \tag{14}$$

where,

$$\boldsymbol{O}_m = f(\boldsymbol{U}_m) = \begin{bmatrix} f(U_m^r) & f(U_m^x) & f(U_m^y) & f(U_m^z) \\ f(-U_m^x) & f(U_m^r) & f(-U_m^z) & f(U_m^y) \\ f(-U_m^y) & f(U_m^z) & f(U_m^r) & f(-U_m^x) \\ f(-U_m^z) & f(-U_m^y) & f(U_m^x) & f(U_m^r) \end{bmatrix}. \tag{15}$$

The internal potential matrix $\boldsymbol{V}$ at output layer of the network is defined as:

$$\boldsymbol{V} = \boldsymbol{SO} + \boldsymbol{\beta}. \tag{16}$$

$$\begin{bmatrix} \boldsymbol{V}_1 \\ \boldsymbol{V}_2 \\ \boldsymbol{V}_3 \\ \vdots \\ \boldsymbol{V}_N \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_{11}\, \boldsymbol{s}_{12}\, \boldsymbol{s}_{13} \ldots \boldsymbol{s}_{1M} \\ \boldsymbol{s}_{21}\, \boldsymbol{s}_{22}\, \boldsymbol{s}_{23} \ldots \boldsymbol{s}_{2M} \\ \boldsymbol{s}_{31}\, \boldsymbol{s}_{32}\, \boldsymbol{s}_{33} \ldots \boldsymbol{s}_{3M} \\ \vdots \quad \vdots \quad \vdots \quad \vdots \\ \boldsymbol{s}_{N1} \boldsymbol{s}_{N2} \boldsymbol{s}_{N3} \cdots \boldsymbol{s}_{NM} \end{bmatrix} \begin{bmatrix} \boldsymbol{O}_1 \\ \boldsymbol{O}_2 \\ \boldsymbol{O}_3 \\ \vdots \\ \boldsymbol{O}_M \end{bmatrix} + \begin{bmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \boldsymbol{\beta}_3 \\ \vdots \\ \boldsymbol{\beta}_N \end{bmatrix}. \tag{17}$$

where, elements of weight matrix $\boldsymbol{S}$ possesses strength of synaptic connections between hidden and output neurons and column vector $\boldsymbol{\beta}$ possesses all quaternionic biases of respective output neurons. The output matrix ($\boldsymbol{Y}$) is obtained by applying split-type activation function over internal potential matrix ($\boldsymbol{V}$) at the output layer:

$$\boldsymbol{Y} = f(\boldsymbol{V}). \tag{18}$$

$$[\boldsymbol{Y}_1\, \boldsymbol{Y}_2\, \boldsymbol{Y}_3 \cdots \boldsymbol{Y}_N]^T = [f(\boldsymbol{V}_1)\, f(\boldsymbol{V}_2)\, f(\boldsymbol{V}_3) \cdots f(\boldsymbol{V}_N)]^T. \tag{19}$$

where,

$$\boldsymbol{Y}_n = f(\boldsymbol{V}_n) = \begin{bmatrix} f(V_n^r) & f(V_n^x) & f(V_n^y) & f(V_n^z) \\ f(-V_n^x) & f(V_n^r) & f(-V_n^z) & f(V_n^y) \\ f(-V_n^y) & f(V_n^z) & f(V_n^r) & f(-V_n^x) \\ f(-V_n^z) & f(-V_n^y) & f(V_n^x) & f(V_n^r) \end{bmatrix}. \tag{20}$$

2.1.2. Backward Pass

In order to develop a QDNN based learning machine, we present the derivation of the error backpropagation learning algorithm in quaternion domain (QDBP) through minimization of average mean square error (E) of the network:

$$E = \frac{1}{8N}\sum_{n=1}^{4N} diag\big(diagonalmatrix(\boldsymbol{e})diagonalmatrix(\boldsymbol{e}^*)\big)$$

$$= \frac{1}{8N}\sum_{n=1}^{4N} diag\left(diagonalmatrix\left(\begin{bmatrix}\boldsymbol{e}_1\\ \boldsymbol{e}_2\\ \boldsymbol{e}_3\\ \vdots\\ \boldsymbol{e}_N\end{bmatrix}\right) diagonalmatrix\left(\begin{bmatrix}\boldsymbol{e}_1^*\\ \boldsymbol{e}_2^*\\ \boldsymbol{e}_3^*\\ \vdots\\ \boldsymbol{e}_N^*\end{bmatrix}\right)\right)$$

$$= \frac{1}{8N}\sum_{n=1}^{4N} diag\left(\begin{bmatrix}\boldsymbol{e}_1 & & & 0\\ & \boldsymbol{e}_2 & & \\ & & \boldsymbol{e}_3 & \\ & & & \ddots\\ 0 & & & \boldsymbol{e}_N\end{bmatrix}\begin{bmatrix}\boldsymbol{e}_1^* & & & 0\\ & \boldsymbol{e}_2^* & & \\ & & \boldsymbol{e}_3^* & \\ & & & \ddots\\ 0 & & & \boldsymbol{e}_N^*\end{bmatrix}\right). \tag{21}$$

where, $*$ denotes quaternionic conjugate (as defined in Eq. (2)) and the output error matrix ($\boldsymbol{e}$) presents the difference between actual ($\boldsymbol{Y}$) and desired ($\boldsymbol{Y^D}$) output at output layer, defined as:

$$\boldsymbol{e} = \boldsymbol{Y} - \boldsymbol{Y^D}. \tag{22}$$

$$\begin{bmatrix}\boldsymbol{e}_1\\ \boldsymbol{e}_2\\ \boldsymbol{e}_3\\ \vdots\\ \boldsymbol{e}_N\end{bmatrix} = \begin{bmatrix}\boldsymbol{Y}_1\\ \boldsymbol{Y}_2\\ \boldsymbol{Y}_3\\ \vdots\\ \boldsymbol{Y}_N\end{bmatrix} - \begin{bmatrix}\boldsymbol{Y}_1^D\\ \boldsymbol{Y}_2^D\\ \boldsymbol{Y}_3^D\\ \vdots\\ \boldsymbol{Y}_N^D\end{bmatrix} = \begin{bmatrix}\boldsymbol{Y}_1 - \boldsymbol{Y}_1^D\\ \boldsymbol{Y}_2 - \boldsymbol{Y}_2^D\\ \boldsymbol{Y}_3 - \boldsymbol{Y}_3^D\\ \vdots\\ \boldsymbol{Y}_N - \boldsymbol{Y}_N^D\end{bmatrix}. \tag{23}$$

The update equations of weight and bias matrices are obtained by employing a gradient decent optimization approach on MSE, mean square error (E). The weight update matrix ($\Delta\boldsymbol{S}$) between hidden-output layers and bias update matrix ($\Delta\boldsymbol{\beta}$) at the output layer of the network are presented as follows:

$$\Delta\boldsymbol{\beta} = \begin{bmatrix}\Delta\boldsymbol{\beta}_1\\ \Delta\boldsymbol{\beta}_2\\ \Delta\boldsymbol{\beta}_3\\ \vdots\\ \Delta\boldsymbol{\beta}_N\end{bmatrix} = \frac{\eta}{N}\begin{bmatrix}\boldsymbol{e}_1 \odot f'(\boldsymbol{V}_1)\\ \boldsymbol{e}_2 \odot f'(\boldsymbol{V}_2)\\ \boldsymbol{e}_3 \odot f'(\boldsymbol{V}_3)\\ \vdots\\ \boldsymbol{e}_N \odot f'(\boldsymbol{V}_N)\end{bmatrix}. \tag{24}$$

$$\Delta\boldsymbol{S} = \begin{bmatrix}\Delta\boldsymbol{s}_{11}\Delta\boldsymbol{s}_{12}\Delta\boldsymbol{s}_{13}\cdots\Delta\boldsymbol{s}_{1M}\\ \Delta\boldsymbol{s}_{21}\Delta\boldsymbol{s}_{22}\Delta\boldsymbol{s}_{23}\cdots\Delta\boldsymbol{s}_{2M}\\ \Delta\boldsymbol{s}_{31}\Delta\boldsymbol{s}_{32}\Delta\boldsymbol{s}_{33}\cdots\Delta\boldsymbol{s}_{3M}\\ \vdots\quad\vdots\quad\vdots\qquad\vdots\\ \Delta\boldsymbol{s}_{N1}\Delta\boldsymbol{s}_{N2}\Delta\boldsymbol{s}_{N3}\cdots\Delta\boldsymbol{s}_{NM}\end{bmatrix} = \frac{\eta}{N}\begin{bmatrix}\boldsymbol{e}_1 \odot f'(\boldsymbol{V}_1)\\ \boldsymbol{e}_2 \odot f'(\boldsymbol{V}_2)\\ \boldsymbol{e}_3 \odot f'(\boldsymbol{V}_3)\\ \vdots\\ \boldsymbol{e}_N \odot f'(\boldsymbol{V}_N)\end{bmatrix}\begin{bmatrix}\boldsymbol{O}_1^*\\ \boldsymbol{O}_2^*\\ \boldsymbol{O}_3^*\\ \vdots\\ \boldsymbol{O}_N^*\end{bmatrix}^T. \tag{25}$$

where, $\eta \in \mathbb{R}^+$ denotes a learning rate and $\odot$ denotes element-wise multiplication of two quaternionic matrices (as defined in Eq. (3)). Similarly, weight update matrix ($\Delta \boldsymbol{W}$) between input-hidden layers and bias update matrix ($\Delta \boldsymbol{\alpha}$) at hidden layer of the network are presented as follows:

$$\Delta \boldsymbol{\alpha} = \begin{bmatrix} \Delta \boldsymbol{\alpha}_1 \\ \Delta \boldsymbol{\alpha}_2 \\ \Delta \boldsymbol{\alpha}_3 \\ \vdots \\ \Delta \boldsymbol{\alpha}_M \end{bmatrix} = \frac{\eta}{N} \left(\begin{bmatrix} \Delta \boldsymbol{s}_{11} \Delta \boldsymbol{s}_{12} \Delta \boldsymbol{s}_{13} \cdots \Delta \boldsymbol{s}_{1M} \\ \Delta \boldsymbol{s}_{21} \Delta \boldsymbol{s}_{22} \Delta \boldsymbol{s}_{23} \cdots \Delta \boldsymbol{s}_{2M} \\ \Delta \boldsymbol{s}_{31} \Delta \boldsymbol{s}_{32} \Delta \boldsymbol{s}_{33} \cdots \Delta \boldsymbol{s}_{3M} \\ \vdots \quad \vdots \quad \vdots \quad \vdots \\ \Delta \boldsymbol{s}_{N1} \Delta \boldsymbol{s}_{N2} \Delta \boldsymbol{s}_{N3} \cdots \Delta \boldsymbol{s}_{NM} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{e}_1 \odot f'(\boldsymbol{V}_1) \\ \boldsymbol{e}_2 \odot f'(\boldsymbol{V}_2) \\ \boldsymbol{e}_3 \odot f'(\boldsymbol{V}_3) \\ \vdots \\ \boldsymbol{e}_N \odot f'(\boldsymbol{V}_N) \end{bmatrix} \right) \odot \begin{bmatrix} f'(\boldsymbol{U}_1) \\ f'(\boldsymbol{U}_2) \\ f'(\boldsymbol{U}_3) \\ \vdots \\ f'(\boldsymbol{U}_M) \end{bmatrix}. \tag{26}$$

$$\begin{aligned} \Delta \boldsymbol{W} &= \begin{bmatrix} \Delta \boldsymbol{w}_{11} \Delta \boldsymbol{w}_{12} \Delta \boldsymbol{w}_{13} \cdots \Delta \boldsymbol{w}_{1L} \\ \Delta \boldsymbol{w}_{21} \Delta \boldsymbol{w}_{22} \Delta \boldsymbol{w}_{23} \cdots \Delta \boldsymbol{w}_{2L} \\ \Delta \boldsymbol{w}_{31} \Delta \boldsymbol{w}_{32} \Delta \boldsymbol{w}_{33} \cdots \Delta \boldsymbol{w}_{3L} \\ \vdots \quad \vdots \quad \vdots \quad \vdots \\ \Delta \boldsymbol{w}_{M1} \Delta \boldsymbol{w}_{M2} \Delta \boldsymbol{w}_{M3} \cdots \Delta \boldsymbol{w}_{ML} \end{bmatrix} \\ &= \frac{\eta}{N} \left(\left(\begin{bmatrix} \Delta \boldsymbol{s}_{11} \Delta \boldsymbol{s}_{12} \Delta \boldsymbol{s}_{13} \cdots \Delta \boldsymbol{s}_{1M} \\ \Delta \boldsymbol{s}_{21} \Delta \boldsymbol{s}_{22} \Delta \boldsymbol{s}_{23} \cdots \Delta \boldsymbol{s}_{2M} \\ \Delta \boldsymbol{s}_{31} \Delta \boldsymbol{s}_{32} \Delta \boldsymbol{s}_{33} \cdots \Delta \boldsymbol{s}_{3M} \\ \vdots \quad \vdots \quad \vdots \quad \vdots \\ \Delta \boldsymbol{s}_{N1} \Delta \boldsymbol{s}_{N2} \Delta \boldsymbol{s}_{N3} \cdots \Delta \boldsymbol{s}_{NM} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{e}_1 \odot f'(\boldsymbol{V}_1) \\ \boldsymbol{e}_2 \odot f'(\boldsymbol{V}_2) \\ \boldsymbol{e}_3 \odot f'(\boldsymbol{V}_3) \\ \vdots \\ \boldsymbol{e}_N \odot f'(\boldsymbol{V}_N) \end{bmatrix} \right) \odot \begin{bmatrix} f'(\boldsymbol{U}_1) \\ f'(\boldsymbol{U}_2) \\ f'(\boldsymbol{U}_3) \\ \vdots \\ f'(\boldsymbol{U}_M) \end{bmatrix} \right) \begin{bmatrix} \boldsymbol{I}_1^* \\ \boldsymbol{I}_2^* \\ \boldsymbol{I}_3^* \\ \vdots \\ \boldsymbol{I}_L^* \end{bmatrix}^T. \end{aligned} \tag{27}$$

2.2. Learning algorithm in quaternionic domain

For the sake of simplicity and better understanding, we further present an algorithm QDNN_TRAIN(.) for training of quaternionic domain neural network (QDNN), which is elaborated by procedures QDNN_INIT(.), QDNN_FORWARD(.) and QDNN_BACKWARD(.). The learning and generalization ability of a three-layered neural structure is obtained through optimization of mean square error. The procedure QDNN_INIT(.) randomly initializes the weight and bias matrices in considered network. It calls the RANDOM_QM(*a*, *b*) procedure which randomly generates the quaternionic matrix of each interconnection weight and bias of neuron in the range from *a* to *b*. The QDNN_FORWARD(.) procedure is intended to implement forward pass of QDNN, hence generate internal potentials ($\boldsymbol{U}, \boldsymbol{V}$) and hence outputs ($\boldsymbol{O}, \boldsymbol{Y}$) matrices at respective layers. The ACTIVATION_FUNCTION(.) limits the output of corresponding neuron of the network. For updates weight and bias matrices, QDNN_BACKWARD(.) is developed for the backward pass of QDNN. All required procedures are presented in pseudo code are as follows:

procedure QDNN_TRAIN($\boldsymbol{I}, \boldsymbol{Y^D}, \eta, \epsilon$)
begin
    QDNN_INIT(L, M, N);
    while $E_T > \epsilon$ **do**
        for $i \leftarrow 1$ **until** $S = length\ (\boldsymbol{I})$ **do**
            $\boldsymbol{U}, \boldsymbol{O}, \boldsymbol{V}, \boldsymbol{Y} \leftarrow$ QDNN_FORWARD($\boldsymbol{W}, \boldsymbol{\alpha}, \boldsymbol{S}, \boldsymbol{\beta}, \boldsymbol{I}$);
            $\boldsymbol{e} \leftarrow \boldsymbol{Y} - \boldsymbol{Y^D}$;
            $E_i \leftarrow \frac{1}{8N} \sum_{n=1}^{4N} diag(diagonalmatrix(\boldsymbol{e}) diagonalmatrix(\boldsymbol{e}^*))$;
            QDNN_BACKWARD($\boldsymbol{W}, \boldsymbol{\alpha}, \boldsymbol{U}, \boldsymbol{O}, \boldsymbol{S}, \boldsymbol{\beta}, \boldsymbol{V}, \boldsymbol{Y}, \eta, \boldsymbol{e}$)
        $E_T \leftarrow \frac{1}{S} \sum_{i=1}^{S} E_i$;
end

procedure QDNN_INIT(L, M, N)
begin
    for $m \leftarrow 1$ **until** M **do**
        for $l \leftarrow 1$ **until** L **do**
            $\boldsymbol{w}_{ml} \leftarrow$ RANDOM_QM(a, b);
        $\boldsymbol{\alpha}_m \leftarrow$ RANDOM_QM(a, b);
    for $n \leftarrow 1$ **until** N **do**
        for $m \leftarrow 1$ **until** M **do**
            $\boldsymbol{s}_{nm} \leftarrow$ RANDOM_QM(a, b);
        $\boldsymbol{\beta}_n \leftarrow$ RANDOM_QM(a, b);
end

procedure QDNN_FORWARD$(\boldsymbol{W}, \boldsymbol{\alpha}, \boldsymbol{S}, \boldsymbol{\beta}, \boldsymbol{I})$
begin
    $\boldsymbol{U} \leftarrow \boldsymbol{W}\boldsymbol{I} + \boldsymbol{\alpha}$;
    $\boldsymbol{O} \leftarrow$ ACTIVATION_FUNCTION$(\boldsymbol{U})$;
    $\boldsymbol{V} \leftarrow \boldsymbol{S}\boldsymbol{O} + \boldsymbol{\beta}$;
    $\boldsymbol{Y} \leftarrow$ ACTIVATION_FUNCTION$(\boldsymbol{V})$;
end

procedure QDNN_BACKWARD$(\boldsymbol{W}, \boldsymbol{\alpha}, \boldsymbol{U}, \boldsymbol{O}, \boldsymbol{S}, \boldsymbol{\beta}, \boldsymbol{V}, \boldsymbol{Y}, \eta, \boldsymbol{e})$
begin
    $\Delta\boldsymbol{\beta} \leftarrow (\eta/N)\boldsymbol{e} \odot$ DER_ACTIVATION$(\boldsymbol{V})$;
    $\Delta\boldsymbol{S} \leftarrow (\eta/N)(\boldsymbol{e} \odot$ DER_ACTIVATION$(\boldsymbol{V}))\boldsymbol{O}^{*T}$;

    $\Delta\boldsymbol{\alpha} \leftarrow (\eta/N)(\boldsymbol{S}^T(\boldsymbol{e} \odot$ DER_ACTIVATION$(\boldsymbol{V}))) \odot$ DER_ACTIVATION$(\boldsymbol{U})$;
    $\Delta\boldsymbol{W} \leftarrow (\eta/N)((\boldsymbol{S}^T(\boldsymbol{e} \odot$ DER_ACTIVATION$(\boldsymbol{V}))) \odot$ DER_ACTIVATION$(\boldsymbol{U}))\boldsymbol{I}^{*T}$;

    $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \Delta\boldsymbol{\beta}$;
    $\boldsymbol{S} \leftarrow \boldsymbol{S} + \Delta\boldsymbol{S}$;
    $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + \Delta\boldsymbol{\alpha}$;
    $\boldsymbol{W} \leftarrow \boldsymbol{W} + \Delta\boldsymbol{W}$;
end

procedure RANDOM_QM(a, b)
begin
    $q_0 \leftarrow [a + (b - a)]$RAND$(1)$;
    $q_1 \leftarrow [a + (b - a)]$RAND$(1)$;
    $q_2 \leftarrow [a + (b - a)]$RAND$(1)$;
    $q_3 \leftarrow [a + (b - a)]$RAND$(1)$;

$$\boldsymbol{q} \leftarrow \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \\ -q_1 & q_0 & -q_3 & q_2 \\ -q_2 & q_3 & q_0 & -q_1 \\ -q_3 & -q_2 & q_1 & q_0 \end{bmatrix};$$

end

procedure ACTIVATION_FUNCTION$(\boldsymbol{q})$
begin
    $\boldsymbol{Q} = f(\boldsymbol{q})$;
end

3. PERFORMANCE EVALUATION OF LEARNING MACHINE THROUGH BENCHMARK PROBLEMS

In this section, we evaluate the effectiveness of learning machine through a wide spectrum of benchmark problems: function approximations, linear transformations, and 3D face recognition. The components of all quaternionic weights and biases are randomly initialized in the range -1 to 1. The quaternionic variable $\boldsymbol{q}_0 = 1 + \boldsymbol{i} + \boldsymbol{j} + \boldsymbol{k}$ is assumed as bias input and the hyperbolic tangent function is used

as activation function. A comparative performance between first generation 'real-valued neural network' and second generation 'quaternionic-valued neural network' with respective algorithms real-valued backpropagation (RVBP) and quaternionic-domain backpropagation (QDBP) is thoroughly evaluated for function approximations by statistical parameters like error variance, correlation, and AIC [20]. Another class of benchmark problems, the learning of linear transformations (rotation, scaling, and translation and their combinations), is promising one as training is performed through a few sets of point lying on the line and trained network is able to generalize over complicated 3D geometric structures. In last subsection, two primary experiments are presented for 3D face recognition; surely it will be stepping stone for prospective researchers to extend this novel technique over a large data set. In last two experiments, each point is represented by a quaternion which contains intended components along with phase information embedded within a number, therefore RVNN is not able to perform such experiments.

3.1. Function Approximations

3.1.1. The Lorenz System

The dynamics of the Lorenz system [21] is presented by the system of three differential equations which shows the chaotic behavior depending on its parameter values.

$$\begin{aligned} dx/dt &= \sigma(y-x) \\ dy/dt &= x(\rho-z)-y \\ dz/dt &= xy-\beta z \end{aligned} \tag{28}$$

where, the symbols σ, ρ and β are parameters of the Lorenz's system. On the basis of its parameters ($\sigma = 15$, $\rho = 28$ and $\beta = 8/3$), this system (Eq. (28)) generates 6537 terms of the time series with initial condition ($x = 0.7$, $y = 0.1$, $z = 0.1$) using fourth order Runge-Kutta method. Each term can be considered in the form of quaternionic input as $0 + x\boldsymbol{i} + y\boldsymbol{j} + z\boldsymbol{k}$. Further, the normalization is performed in the range from -0.8 to 0.8. The first 500 terms of the time series have been used for training and rest for testing of three-layered RVNN (3-11-3) and QDNN networks (1-3-1) separately. Experiments demonstrate that the second network requires a lesser number of training cycles to achieve the desired MSE, as presented in Table 1. Figure. 1 shows the testing results of the networks trained by QDBP for prediction of time series of Lorenz system. Table 1 demonstrates the significant outperformance of QDNN in terms of network topology, training cycles, testing MSE, error variance, correlation and AIC.

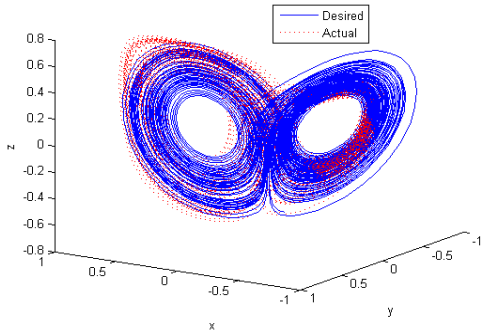


Figure. 1. 3D plot of the Lorenz system tested by the QDNN network trained through QDBP

Table 1. Comparison of training and testing performance for Lorenz system

Neuron Type	Real-valued	Quaternionic-valued
Algorithm	RVBP	QDBP
Network Topology	3-11-3	1-3-1
MSE Training	0.0015	0.0006
Average Epoch	15000	9000
MSE Testing	0.0042	0.0012
Error Variance	0.0026	0.0009
Correlation	0.87327	0.9323
AIC	-6.3329	-7.4503

3.1.2. The Chua's Circuit

Chua's circuit is the simplest autonomous electronic circuit containing registers, capacitors and inductors that exhibit the chaotic behavior under specific parametric conditions [22]. This circuit satisfies the chaotic criterion which contains one or more non-linear elements, one or more active registers and three or more energy storage devices. It uses the one chua's diode as non-linear element, one locally active register and two capacitors and one inductor as energy storage devices. The dynamics of Chua's circuit are governed by three state equations as

$$\frac{dx}{dt} = \alpha[y - x - h(x)]$$
$$\frac{dy}{dt} = x - y + z \quad (29)$$
$$\frac{dz}{dt} = -\beta y - \gamma z$$

where, $h(x)$ presents the electrical response of non-linear register defined as

$$h(x) = m_1 x + \frac{1}{2}(m_0 - m_1)(|x+1| - |x-1|)$$

and α, β, γ, m_0 and m_1 are the constant parameters. The symbols x, y and z are voltages across two capacitors and an inductor respectively, and their combinations show the chaotic attractor in three dimensions. The double scrolled chaotic attractor [22] is obtained with the parameters $\alpha = 15.6$, $\beta = 28$, $\gamma = 0$, $m_0 = -1.143$ and $m_1 = -0.714$. The chaotic time series has been obtained from the simulation of the system (Eq. 29) with time step 0.1 Sec and initial voltages x=0.1, $y = 0.1$ and $z = 0.1$ by using fourth order Runge-Kutta method. The normalization of input-output imaginary quaternions is done in -0.8 to 0.8 (real part is zero and imaginary parts (x, y, z) present corresponding voltages). A time series containing 500 terms obtained from simulated system has been used to train RVNN and QDNN. The training results of both networks, in Table 2, demonstrate that QDNN trained by the QDBP algorithm requires a significantly smaller number of average epochs to achieve the threshold training error than RVBP. The next 500 terms of that time series have been tested through networks trained by both algorithms. Figure. 2 shows the 3D patterns of desired and actual data for chaotic behavior of Chua's circuit. The testing results shown in Table 2 in terms of error, variance, correlation, and AIC again infer the superiority of QDNN over real-valued neural network.

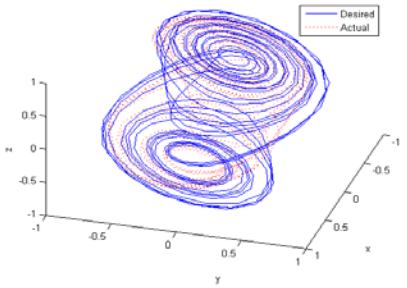


Figure 2. Testing result of QVNN network trained by QDBP for Chua's circuit

Table 2. Comparison of training and testing performance for Chua's circuit

Neuron Type	Real-valued	Quaternionic-valued
Algorithm	RVBP	QDBP
Network Topology	3-12-3	1-3-1
MSE Training	0.0012	0.0008
Average Epoch	10000	7000
MSE Testing	0.0025	0.0017
Error Variance	0.0020	0.0008
Correlation	0.9734	0.9874
AIC	-6.5332	-7.0101

3.2. Linear Transformations

In order to evaluate the performance of QDNN, we have considered a three layer neural structure (2-M-2). This section presents the learning of linear transformations (rotation, scaling, and translation and their combinations) by QDNN through a few sets of points on the line and generalization over complicated 3D objects. Each quaternionic variable $\boldsymbol{q_i} = 0 + x_i\boldsymbol{i} + y_i\boldsymbol{j} + z_i\boldsymbol{k}$ undergoes a transformation function (T) and correspondingly yields a transformed quaternionic variable $\boldsymbol{q_i'} = 0 + x_i'\boldsymbol{i} + y_i'\boldsymbol{j} + z_i'\boldsymbol{k}$ represented in the quaternionic matrix as follows:

$$\boldsymbol{q_i'} = T(\boldsymbol{q_i}) = \boldsymbol{a}\boldsymbol{q_i} + \boldsymbol{b} \quad (i = 1, 2, 3, \ldots n_p)$$

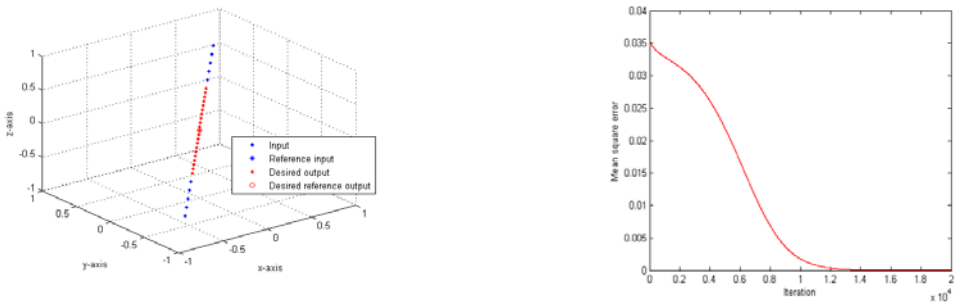
$$\begin{bmatrix} 0 & x_i' & y_i' & z_i' \\ -x_i' & 0 & -z_i' & y_i' \\ -y_i' & z_i' & 0 & -x_i' \\ -z_i' & -y_i' & x_i' & 0 \end{bmatrix} = \begin{bmatrix} 0 & a_x & a_y & a_z \\ -a_x & 0 & -a_z & a_y \\ -a_y & a_z & 0 & -a_x \\ -a_z & -a_y & a_x & 0 \end{bmatrix} \begin{bmatrix} 0 & x_i & y_i & z_i \\ x_i & 0 & -z_i & y_i \\ -y_i & z_i & 0 & -x_i \\ -z_i & -y_i & x_i & 0 \end{bmatrix} + \begin{bmatrix} 0 & b_x & b_y & b_z \\ -b_x & 0 & -b_z & b_y \\ -b_y & b_z & 0 & -b_x \\ -b_z & -b_y & b_x & 0 \end{bmatrix}$$

where n_p denotes the number of points that lies on the surface of 3D objects and $\boldsymbol{a}$ and $\boldsymbol{b}$ are quaternions such that norm of $\boldsymbol{a}$ $i.e.$ $\|\boldsymbol{a}\| = \sqrt{0^2 + a_1^2 + a_2^2 + a_3^2}$ denotes the scaling factor. Argument of $\boldsymbol{a}$ yields rotation in $\boldsymbol{q}$ while $\boldsymbol{b}$ performs translation of 3D object in the distance ($\|\boldsymbol{b}\|$). The combinations of transformations facilitate the viewing of 3D objects from different orientations, interpretation of their motion, etc.

For training on a three layered 2-6-2 QDNN, all experiments consider a straight line in space containing few input data points (21 points) on line and a reference point ($mid\ point$). The set of point (x, y, z) lying on line goes to the first input and a second input passes the reference point (x_r, y_r, z_r). The incorporation of the reference point provides more information to learning a system which yields better accuracy. Similarly, the first and second output neurons of output layer result the transformed point (x', y', z') on line and transformed reference point (x'_r, y'_r, z'_r) respectively. The learning of the transformation is achieved by learning the algorithm presented in section 2.2 with a suitable learning rate. The trained QDNN is able to generalize over huge number of points cloud data of complicated geometrical structure like sphere, cylinder, torus and this ability of the network presents the 3D motion interpretation of objects. It is worthwhile to mention here that learning of phase information is not possible by RVNN hence such transformation is not possible through RVNN; therefore this section only presents the result obtained by QDNN.

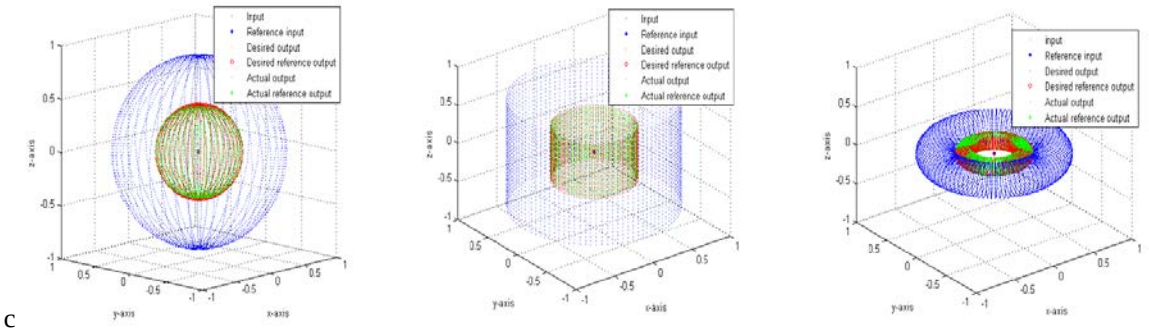
3.2.1. Similarity Transformation

The learning of QDNN (2-6-2 model) is performed for similarity transformation, through input-output mapping for scaling factor ½ over the line containing 21 points, referenced in (0,0,0), as shown in Figure. 3(a). Convergence of mean square error (Figure. 3(b)) shows the smart learning capability of the proposed network. The training of QDNN with 0.00005 learning rate converges to MSE = 1.005567e-05 after 20000 iterations. The trained network is able to generalize over many complicated standard geometric structures like sphere (4141 data points), cylinder (2929 data points), and torus (10201 data points) which is presented in Figure. 4(a), 4(b), and 4(c) respectively.



(a) (b)

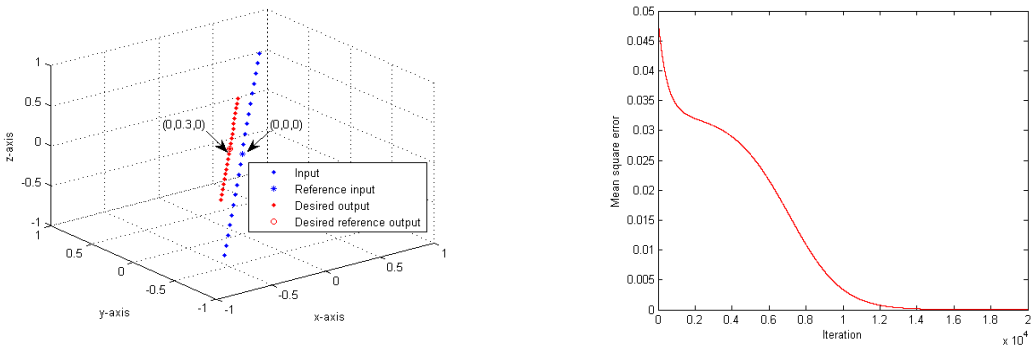
Figure 3. (a) Training input-output mapping for scaling with scaling factor ½;
(b) Convergence of mean square error



c

(a) (b) (c)

Figure. 4. Testing results from similarity transformation over (a) sphere, (b) cylinder, and (c) torus.

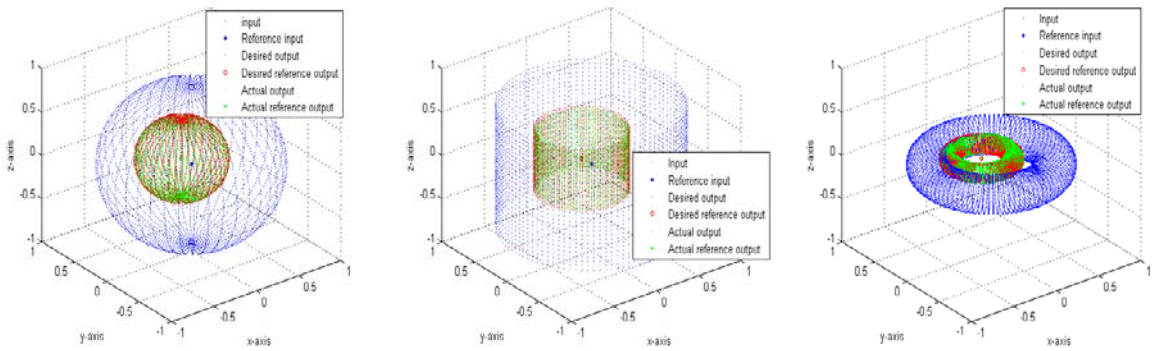


(a) (b)

Figure 5. (a) Training patterns: input-output mapping shows transformation with scaling factor 1/2, followed by translation with 0.3 units in positive y-direction (b) Convergence of mean square error

3.2.2. Scaling and translation

The learning of 2-6-2 QDNN is performed in combination of scaling (scaling factor 1/2) and translation (0.3 unit in positive y-direction), through input-output mapping over the line (21 data points) and referenced in (0,0,0), as shown in Figure. 5(a). The convergence of QDNN in Figure. 5(b), with learning rate 0.00005, up to 2.58514e-05 mean square error shows the smart learning capability of the proposed learning machine after 20000 iterations. The trained network is able to generalize well over many complicated standard geometric structures like sphere (4141 data points), cylinder (2929 data points), and torus (10201 data points) as shown in Figure. 6(a), 6(b), and 6(c) respectively.

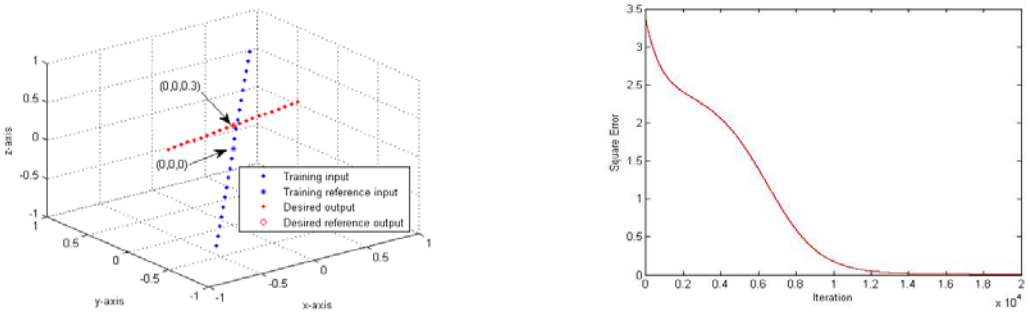


(a) (b) (c)

Figure 6. Testing results from similarity transformation through (a) sphere, (b) cylinder, and (c) torus

3.2.3. Scaling, translation and rotation

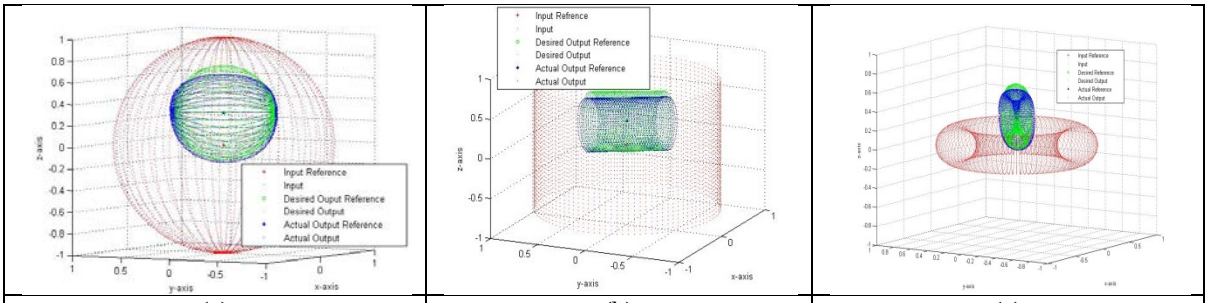
The learning of QDNN for general linear transformation (scaling factor 1/2, counterclockwise rotation about the x-axis by $\pi/2$ radian, and translation by (0,0,0.3)) is performed for, through input-output mapping over straight line and reference (0,0,0), as shown in Figure. 7(a). The 2-6-2 QDNN model is used for training of these transformations through 21 data points in a straight line. Convergence of mean square error 1.0e-04 after 20000 iterations is achieved with the 0.00005 learning rate, as shown in Figure. 7(b). The trained network is also able to generalize over many complicated standard geometric structures like sphere (4141 data points), cylinder (4141 data points), and torus (10201 data points) as shown in Figure. 8(a), 8(b), and 8(c) respectively.



(a) (b)

Figure 7. (a) Training mapping patterns through straight line (scaling factor 1/2, counterclockwise rotated about the x-axis by $\pi/2$ radian, and translated by (0,0,0.3)); (b) Square error during training of straight line pattern

All transformation experiments promise the intelligent behavior of QDNN for motion interpretation of 3D objects. Further, this novel experiment provides a direction to generalize the motion for intelligent system design for a variety of operations.



(a) (b) (c)

Figure 8. Generalization of a linear transformation (scaling factor 1/2, counterclockwise rotated about the x-axis by $\pi/2$ radian, and translated by (0, 0, 0.3) over (a) sphere, (b) cylinder, and (c) torus

3.3. 3D face recognition

This section presents a basic experiment, though with a small data set but its implication is wide for the applicability of proposed learning machine for 3D recognition. Our method has a great deal to perform successful recognition in variable head position, orientation, and facial expressions. Two experiments are conducted here to learn and classify point cloud data of 3D faces using proposed quaternionic domain backpropagation algorithm. A simple structure of (1-2-1) QDNN with single input-output performs experiments using only two quaternionic neurons at hidden layer.

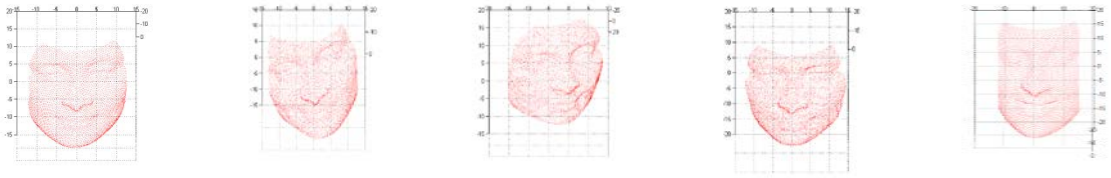


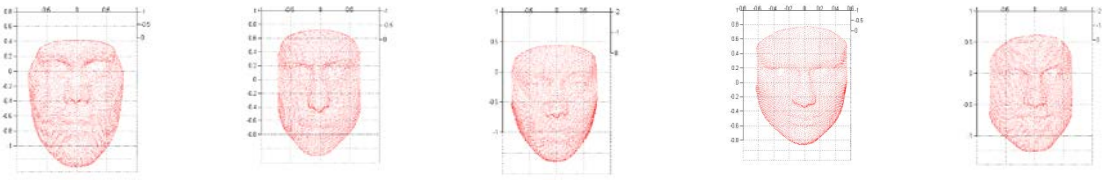
Figure. 9. Five 3D faces of same person with different orientation and poses.

The first experiment is performed on a dataset containing 05 faces of the same person (4654 points cloud data) with different orientation and poses; the learning of QDNN is made with one face (Figure. 9(a)) and testing over all faces. Table 3 presents the testing MSE (mean square error) of all five faces which are comparable, hence demonstrate that they are faces of same person irrespective of variations in face orientation and poses. It infers straightforward learning and generalization ability of a simple QDNN which is not possible by RVNN.

Table 3. Comparison of testing MSE of faces of same person with different orientation (MSE Training=0.0001)

S. No.	Face (Figure)	Test error
1.	9(a)	2.4842e-04
2.	9(b)	3.5431e-03
3.	9(c)	5.1153e-03
4.	9(d)	4.5212e-04
5.	9(e)	3.9148e-04

Similarly, the second experiment is performed on a dataset containing 05 faces of different people (6397 points cloud data); the learning of QDNN is made with one face (Figure. 10(a)) and testing over all faces. Table 4 presents the testing MSE of each face obtained from trained network, which shows that the MSE of other four faces are much higher in comparison to the face (Figure. 10(a)) used in training. This demonstrates that the simple QDNN correctly classifies the faces of same or different person. It again reveals the learning and generalization capability of a proposed learning machine where real-valued neural network lacks.



(a) (b) (c) (d) (e)

Figure. 10. Five 3D faces of different persons

Table 4. Comparison of testing MSE of faces of different person (MSE Training = 0.0001)

S. No.	Face (Figure)	Test error
1.	10(a)	1.8214e-04
2.	10(b)	8.1344e-01
3.	10(c)	3.5709e-00
4.	10(d)	6.2814e-02
5.	10(e)	3.1738e-01

3. CONCLUSION

In this paper, we present an efficient and generalized learning machine for high dimensional problems and evaluate it with variety of problems of different areas. The proposed neural network with learning algorithm in quaternionic domain directly process three or four dimension data without the hassle of

its different components and phase information among them. The quaternion is the number which possesses the magnitude of intended components and phase information of each component is embedded in it. Thus, quaternionic domain neural network (QDNN) leads to simple network structure, efficient learning and better performance; whereas conventional real-valued neural network (RVNN) deals with individual components hence need huge topology, slow learning and poor performance. Apart from that RVNN does not work for problems where it is required to learn and generalize phase information like object recognition and motion or transformation of objects in space. It is worth to mention here again that proposed machine learns the composition of transformations through input-output mapping over a line containing a small set of points and generalize this motion over complex geometrical structure such as sphere, cylinder, and torus. Although, the problem presented for recognition in 3D imaging is small and basic but it is very encouraging for prospective researcher due to network simplicity, faster convergence and the result.

REFERENCES

[1] Tripathi BK (2016) On the complex domain deep machine learning for face recognition. Applied Intelligence, Springer, ISSN: 0924-669X, DOI 10.1007/s10489-017-0902-7.
[2] Nitta T (1997) An extension of the back-propagation algorithm to complex numbers. Neural Networks 10(8):1391–1415
[3] Muezzinoglu MK, Guzelis C, Zurada JM (2003) A new design method for complex-valued multistate Hopfield associative memory. IEEE Transaction Neural Networks 14(4):891–899
[4] Nitta T (1992) 3D vector version of the back-propagation algorithm. Int Joint Conf on Neural Networks 2:511–516
[5] Hirose A (2006) Complex-valued neural networks. Springer-Verlag, New Yark
[6] Tripathi BK, Kalra PK (2011) Complex generalized-mean neuron model and its applications. Applied Soft Computing, Elsevier Science 11(01):768–777
[7] Tripathi BK, Kalra PK (2011) On the learning machine in three dimensional mapping. Neural Computing and Applications 20:105–111
[8] Tripathi BK, Kalra PK (2011) On efficient learning machine with root power mean neuron in complex domain. IEEE Transaction on Neural Networks 22(05):727–738
[9] Tripathi BK (2014) High dimensional neurocomputing : growth, appraisal and applications. Springer, London
[10] Hamilton WR (1853) Lectures on quaternions. Hodges and Smith: Dublin, Ireland
[11] Kuipers JB (1998) Quaternions and rotation sequences: a primer with applications to orbits, aerospace and virtual Reality. Princeton University Press: Princeton, NJ, USA
[12] Hoggar SG (1992) Mathematics for computer graphics. Cambridge University Press: Cambridge, MA, USA
[13] Ujang BC, Took CC, Mandic DP (2011) Quaternion-valued nonlinear adaptive filtering. IEEE Transaction on Neural Networks 22(8):1193–1206
[14] Wang M, Took CC, Mandic DP (2011) A class of fast quaternion valued variable stepsize stochastic gradient learning algorithms for vector sensor processes. IJCNN : 2783–2786
[15] Nitta T (2004) A solution to the 4-bit parity problem with a single quaternary neuron. Neural Inf Process Lett Rev 5(2):33–39
[16] Isokawa T1, Nishimura H, Matsui N (2012) Quaternionic multilayer perceptron with local analyticity. Information 3:756–770 doi:10.3390/info3040756
[17] Isokawa T, Kusakabe T, Matsui N, Peper F (2003) Quaternion neural network and its application. LNAI 2774: 318–324
[18] Foggel DB (1991) An information criterion for optimal neural network selection. IEEE Trans. Neural Netw 2(5):490-497
[19] Lorenz EN (1963) Deterministic nonperiodic flow. Journal of the Atmospheric Sciences 20(2):130-141
[20] Chua LO, Matsumoto T, Komuro M (1985) The Double Scroll. IEEE Transactions on Circuits and Systems CAS-32(8):798-818