Exploring Quaternion-Valued Neural Networks

Shezad Hassan

August 21, 2022

1 Introduction

With the increasing popularity of artificial neural networks in modern society, many variations of said networks have arisen and proven useful in certain sectors. One example of this is the complex-valued neural network which is used in the fields of computer vision and and signal processing which inherently deals with complex numbers instead of real numbers. A natural extension of the complex valued neural network would be the quaternion-valued neural network which could prove useful when dealing with four dimensional values. I aim to explore the quaternion-valued neural network and build one from scratch using Python. To do so, I will begin by explaining the basics of neural networks and quaternions in sections 2 and 3 before combining the two concepts in section 4. I will then give a walk through of how I built a quaternion-valued neural network from scratch and discuss the corresponding results. The code associated with this paper can be found on my github (https://github.com/SHEZADHASSAN22/Quaternion-Neural-Network).

2 Neural Network Basics

Artificial neural networks are essentially functions made up of units called neurons. These neurons are arranged in layers and each layer is connected by units called weights. Each neuron is also given a number called a bias. To run this function, values are set for the input layer of neurons and these are propagated forward through each layer using certain formulas involving the weights, biases and an activation function. Adjusting the weights and biases using the back-propagation algorithm is where the 'learning' occurs as it changes its parameters so as to reduce the cost of the output given a set of training data.

This structure together with the back-propagation algorithm has proved to be effective in making predictions and approximating functions. The back-propagation algorithm is also used in a variety of other types of neural networks such as convolutional neural networks and recurrent neural networks. However, these networks all use real values in their implementations which begs the question, are there any advantages of using higher valued number systems such as complex values or quaternions instead? Complex-valued neural networks have proven to be useful especially when dealing with non-linear data. Similarly, a quaternion-valued neural network may also prove to have advantages especially when dealing with multidimensional data.

3 Quaternions

Just as complex numbers are used to describe two dimensional space, the quaternion was designed to describe three dimensional space. It is made up of four components which are the real component and three 'imaginary' components, commonly expressed as

$$q = a + bi + cj + dk : a, b, c, d \in R$$

Quaternion arithmetic is governed by the following rule:

$$i^2 = j^2 = k^2 = ijk = -1$$

It is important to note that quaternions do not abide by the law of commutativity. For example, doing some basic arithmetic using the equation above, we can derive the following:

$$k = ij \neq ji = -k$$

Multiplication of two quaternions are demonstrated below:

$$
\begin{aligned}
q_1 q_2 &= (a_1 + b_i + c_j + d_k)(a_2 + b_2 i + c_2 j + d_2 k) \\
&= a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2 + \\
&= (a_1 b_2 + b_1 a_2 + c_1 d_2 - d_1 c_2) i + \\
&= (a_1 c_2 - b_1 d_2 + c_1 a_2 + d_1 b_2) j + \\
&= (a_1 d_2 + b_1 c_2 - c_1 b_2 + d_1 a_2) k
\end{aligned}
$$

4 Quaternion-Valued Neural Network

A quaternion-valued neural network is similar to a regular artificial neural network except that all the real values are replaced with quaternions. This includes everything from the weights and biases to the neuron activations. Additionally, since quaternion arithmetic differs from real arithmetic, adjustments have to be made to the math involved in entire process. Firstly, the way that the activation function, such as the sigmoid function, is applied to a neuron changes. The activation function is applied to each individual component of the quaternion. For example, given that an inactivated neuron has the value $z = w + xi + yj + zk$, the activation function α will be applied as follows:

$$\alpha(z) = \alpha(w + xi + yj + zj) = \alpha(w) + \alpha(x)i + \alpha(y)j + \alpha(z)j$$

Secondly, the formulas involved in the back-propagation algorithm will need to change to accommodate the quaternion values. Several papers have suggested various versions of the back-propagation algorithm [2, 5, 7], and we will use one of these in the next section of the paper.

5 Building a Quaternion-Valued Neural Network

To build a quaternion-valued neural network using Python, I first checked to see if any of the popular machine learning frameworks, such as Tensorflow and PyTorch, supported such a network but they did not. This meant that I needed to build one from scratch using Numpy. To do this, I first built a super simple network containing only three neurons as represented below:

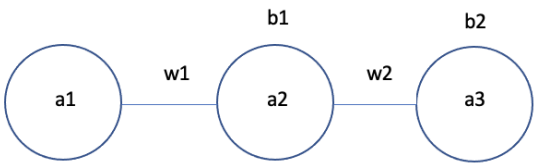

Figure 1: Neural network with three layers and one neuron in each layer.

where a_1 is the input neuron, a_2 is the hidden layer neuron and a_3 is the output neuron. The weights and biases are denoted by w_1, w_2, b_1 and b_2 and they are all randomly initialised between $0+0i+0j+0k$ and $1+i+j+k$. The purpose of this network is to simply take a single randomised input and output a fixed value $q_o = 0.5 + 0.5i + 0.5j + 0.5k$. I thus wanted the network to learn the appropriate weights and biases to achieve this.

For the most part, the logic and actual code is similar to that of a real-valued artificial neural network. The only difference is that the real values are replaced by quaternions. However, complications arise when it comes to working out the back-propagation formulas as mentioned earlier. To get around this, I used the back-propagation formulas shared by Tohru Nitta [2] which worked decently well in this super simple neural network. Running the stochastic gradient step ten thousand times would consistently output the desired quaternion value with an extremely small error. I then played around with the input and desired output with the same network and achieved similar results.

However, this is a very simple and not very useful network that only works with one piece of data. I then expanded network to accommodate more neurons and layers, whilst keeping the math essentially the same. I attempted to train this model using self generated data to learn simple functions such as $f(x) = x$ which is commonly done with artificial neural networks using the universality theorem. Here, the network started to fail as it could not consistently simulate the desired function. To fix this, I tried to vary factors such as the number of training epochs and

the range of values of the randomly initialised weights and biases but this did not work. As the back-propagation formulas seemed to work in the super simple network mentioned earlier, I was hesitant to conclude that that was the reason the network was failing. However, I believe that the problem likely lies in the choice of formulas used which perhaps may not be suited for this specific use case.

6 Conclusion

To continue this research, I would need to learn more about the math involved in the back-propagation step with regards to the quaternion version and perhaps try and write my own formulas. Currently due to the time constraints of the project, I was unable to dive deep into the higher level mathematics involved and had to rely on the formulas of others found online. I would also try training my network to perform different tasks using different sets of data.

7 Bibliography

1. Clark, T. H. (2020, July 11). Why neural nets can approximate any function. Medium. Retrieved August 8, 2022, from https://towardsdatascience.com/why-neural-nets-can-approximate-any-function-a878768502f0

2. Nitta, Tohru (1995). A Quaternary Version of the Back-propagation Algorithm, Proceedings of IEEE International Conference on Neural Networks, ICNN'95-Perth, Nov 27 - Dec 1, Vol.5 pp.2753-2756.

3. Bassey Joshua, Xiangfang Li, Lijun Qian(2021). A survey of Complex-Valued Neural Networks. Center of Excellence in Research and Education for Big Military Data Intelligence.

4. Georgiou George M., Koutsougeras Cris(1992). IEEE Transactions on circuits and systems-II:Analog and digital signal processing, Vol 39, No.5, pp.330-334

5. Aaron B. Greenblatt (2017). Introducing Quaternion Multi-Valued Neural Networks with Numerical Examples.

6. Chase J. Gaudet and Anthony S. Maida(2018). Deep Quaternion Networks, School of Computing Informatics University of Lousiana at Lafayette.

7. P.Arena, S.Baglio, L.Fortuna, M.G.Xibilia. Chaotic Time Series prediction Via Quaternionic Multilayer Perceptrons, Dipartimento Ellettrico Elettronico e Sistemistico Universita' di Catania(Italian).

8. Zhu, A. (2022, April 10). Simulate any functions with a neural network. Medium. Retrieved August 8, 2022, from https://towardsdatascience.com/simulate-any-functions-with-neural-network-8e52f2083e3d

9. Dramsch J. S., Luthje M., Christensen A. N. (2019, Nov). Complex-valued neural networks for machine learning on non-stationary physical data.

10. Jalab H. A., Ibrahim R. W. (2011, March). New activation functions for complex-valued neural network

11. Nielsen, Michael (2019). Neural networks and deem learning.