Clifford Networks
An Introduction

J.K.Pearson*
D.L.Bisset †
Electronic Engineering Laboratories
The University
Canterbury
Kent CT2 7NT
U.K.

February 23, 1994

Contents

*email: jkp@ukc.ac.uk,
†email dlb@ukc.ac.uk

1 Introduction

Neural networks have traditionally adopted a simple view of the relationship between data and the space in which it exists. This stems from the fact that pattern recognition as generally being formulated in terms of structureless images (that is in Neural networks little or no assumptions are made as to pattern structure, for example very few neural network based pattern recognition schemes take advantage of the fact that most images are two dimensional). This is a very restricted view of the relationship between data and space. In many problems the data can be said to be multidimensional, such as for example a colour image, or a moving object. There is no fundamental reason why the mathematics of neural networks and therefore their practice, cannot be extended to cover multidimensional data.

In order to do this it is important to tighten up the formal descriptions of existing networks so that they can be manipulated more easily. The first section of this report presents a formal description of back error propagation. Subsequent sections expand this description using Clifford algebras, to the multidimensional case.

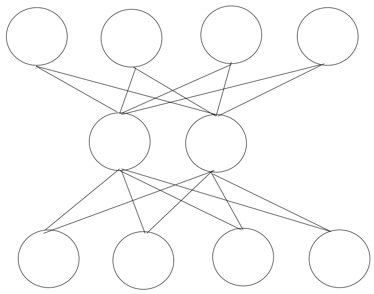
Feed-forward networks are built up from layers of neuronal like processing units, with the output units of one layer connected to the next layer, where the whole network has one input layer and one output layer, see figure 1. Each neuron has many weighted inputs and a single output, each neuron computes a function of the form:

$$o_j = f_j\left(\sum_i \omega_{ij} o_i + \theta_j\right) \tag{1}$$

Where o_j is the output of neuron j, ω_{ij} is the weight on the connection from neuron i to neuron j , θ_j is a threshold value internal to the neuron and f_j is some possible non-linear activation function associated with the neuron.

A feed-forward network with n inputs and m outputs can be seen to be computing a function $\phi_{\Omega,\Theta} : \mathbf{R}^n \to \mathbf{R}^m$, where Ω and Θ represent the weight and threshold values for all the nodes in the network. Feed-forward networks have been used in many applications such as pattern recognition, non-linear control theory and time predication, but all these tasks can be seen as some form of function approximation. For instance in pattern recognition if we are given some pattern set $P \subset \mathbf{R}^n$ what is required is for a network

Figure 1: A feed-forward Network



to approximate as closely as possible a function $\psi : \mathbf{R}^n \to \mathbf{R}$ which gives $\psi(p) = 1$ for all $p \in P$ (i.e. classify each member of the pattern set as valid) and $\psi(x) = 0$ for each non pattern.

Mathematically some care has to be taken with the notion of approximation. What is required is some function E which given two functions ϕ,ψ gives:

$$E(\phi, \psi) = 0 \text{ if and only if } \phi = \psi \tag{2}$$

and $E(\phi, \psi)$ be close to zero if ϕ is 'close' to ψ. Formalizing these requirements leads to the notion of a metric space, see [3] for a good introduction.

Various choices of metrics are available, first the metric of uniform convergence:

$$E(\phi, \psi) = \sup\{|\phi(x) - \psi(x)| \; x \in X\} \tag{3}$$

Where X is a subset on which both ϕ and ψ are defined, more formally $X \subseteq \mathrm{dom}(\phi) \cap \mathrm{dom}(\psi)$. To guarantee this value is defined extra conditions either have to be imposed on X or on ϕ and ψ, if X is stipulated to be compact (i.e. X is finite) or extra conditions must be imposed on the behavior of the functions ϕ and ψ as both tend to infinity. Uniform convergence metrics are useful where networks are required to perform equally well over the whole of the input space.

Second there are the L_p metrics which are much easier to deal with mathematically and are defined as:

$$\rho_{p,\mu}(\phi, \psi) = \left(\int (\phi - \psi)^p d\mu \right)^{\frac{1}{p}} \tag{4}$$

again extra conditions have to stipulated to make the metric always well defined similar to the conditions for the metric of uniform convergence see [9] for details.

Much of the work in neural networks can be condensed to the following question. Given a function ψ which we wish to approximate and a feed-forward work $\phi_{\Omega,\Theta}$ what is the best weight and threshold set which will approximate the function ψ with accuracy ϵ? Thankfully a method of optimization by gradient descent exist, which is often referred to as back error propagation in the literature [14]. Give a metric E the partial derivatives:

$$\frac{\partial E}{\partial \omega_{ij}} \text{ and } \frac{\partial E}{\partial \theta_j} \tag{5}$$

are calculated for all the weight and threshold values (typically involving repeated application of the chain rule, where the error is propagated back from the output layer to the input layer), this optimization process can be seen as a dynamic system:

$$\frac{\partial E}{\partial t} = \frac{\partial E}{\partial \omega_{ij}} \frac{\partial \omega_{ij}}{\partial t} \tag{6}$$

and

$$\frac{\partial E}{\partial t} = \frac{\partial E}{\partial \theta_j} \frac{\partial \theta_j}{\partial t} \tag{7}$$

Then setting:

$$\frac{\partial \omega_{ij}}{\partial t} = -\eta \frac{\partial E}{\partial \omega_{ij}} \tag{8}$$

and,

$$\frac{\partial \theta_j}{\partial t} = -\eta \frac{\partial E}{\partial \theta_j} \tag{9}$$

Should result in E reducing in time.

The above has given a brief review of the current state of neural networks but expressed in a more formal mathematical form than is usual. The main body of this report is concerned with extending this formalism through the use of Clifford algebras.

Clifford algebras are multi-dimensional extensions of the real and complex numbers which arise from quadratic form theory and tensor algebras, but to understand the present work it is only required to think of them as multi-dimensional complex like algebras with extra imaginary units.

Clifford networks generalize both real and complex valued network [4] by replacing all activation weight and threshold values by Clifford values. Corresponding error metrics can be written down which subsequently make it possible to derive back error propagation equations.

The rest of this report presents an introduction to Clifford algebras the derivation of a back error propagation equivalent for Clifford networks, a discussion of activation functions and some experimental results.

2 Clifford algebras an introduction.

This chapter is provided to serve as an introduction to Clifford algebras. The reader with knowledge of Clifford algebras can safely skip this chapter, where possible the nottation has been kept consistent with [6].

A Clifford algebra is constructed over some vector space. In what follows only Clifford algebras over real vector spaces will be treated but without difficulty complex and abstract vector spaces could be considered.

The approach used here is essentially a constructive approach, with one simple representation of each algebra.

A $p+q$ dimensional real vector space will be denoted as $\mathbf{R}^{p+q}$ (the reason for the notation $p+q$ will become clear later on). A Clifford algebra $\mathbf{R}_{p,q}$ is two things, a 2^{p+q} vectors space constructed from $\mathbf{R}^{p+q}$ and a set of algebraic rules defining multiplication and addition of vectors.

A $\mathbf{R}^{p+q}$ will have a basis of the form:

$$e_1, e_2, e_3, \ldots e_{p+q}$$

From this construct a $2^{n=p+q}$ dimensional vector space with basis elements:

$$\{e_A = e_{h_1 \ldots h_r} | A = (h_1, \ldots, h_r) \in \mathcal{P}(\mathbb{N}), 1 \leq h_1 < \ldots < h_r \leq n\}.$$

For example the vector space over $\mathbf{R}^{1,1}$ would have the basis,

$$e_\emptyset,\ e_{(1)},\ e_{(2)}\ e_{(1,2)}$$

For notational convenience when no confusion can arise, $e_{(h_1, \ldots h_r)}$ will be denoted as $e_{h_1 h_2 \ldots h_r}$ and $e_\emptyset = e_0$.

An element of the Clifford algebra can be written as a formal sum,

$$x = \sum_A x_A e_A \tag{10}$$

With each $x_A \in \mathbf{R}$. In what follows a summation with a capital letter near the begin of the alphabet, denotes a sum over the basis elements of a Clifford algebra.

Thus an element in $\mathbf{R}_{1,1}$ can be written as,

$$x = x_0 + x_1 e_1 + x_2 e_2 + x_{12} e_{12} \tag{11}$$

It is useful to drop the e_0 when writing out formal sums, because e_0 acts as the unit of the algebra.

Addition of two elements of the algebra is defined as for vectors:

$$x + y = \sum_A (x_A + y_A) e_A \tag{12}$$

Multiplication is slightly more complicated. It is done formally element by element as in expanding brackets subject to the following algebraic rules,

$$e_i^2 = 1 \quad , \quad i = 1, \ldots, p \tag{13}$$

$$e_i^2 = -1 \quad , \quad i = p+1, \ldots, p+q \tag{14}$$

$$e_i e_j = -e_j e_i \quad , \quad i \neq j. \tag{15}$$

With for $1, \leq h_1, < \ldots h_r \leq n$, $e_{h_1} \cdot e_{h_2} \cdots e_{h_r} = e_{h_1 \ldots h_r}$. This can be expressed more compactly in the following way,

$$e_A e_B = (-1)^{\#((A \cap B) \setminus P)} (-1)^{p(A,B)} e_{A \Delta B}, \tag{16}$$

where P stands for the set $1, \ldots p$, and $\#X$ represents the number of elements in X,

$$p(A, B) = \sum_{j \in B} p'(A, j), \quad p'(A, j) = \#\{i \in A | i > j\},$$

and the sets A, B and $A \Delta B$(the set difference of A and B) are ordered in the prescribed way.

Thus for example if in $R_{1,1}$

$$\begin{aligned}
x &= 3 + 4e_1 + e_2 \\
y &= e_2 + 2e_{12} \\
xy &= (3 + 4e_1 + e_2)(e_2 + 2e_{12}) \\
&= 3e_2 + 6e_{12} + 4e_1 e_2 + 8e_1 e_{12} + e_2^2 + 2e_2 e_{12}
\end{aligned}$$

By using the reduction rules $e_1 e_{12} = e_1 e_1 e_2 = e_1^2 e_2 = e_2$ and $e_2 e_{12} = -e_2 e_{21} = -e_2^2 e_1 = e_1$. So

$$xy = -1 + 2e_1 + 11e_2 + 10e_{12} \tag{17}$$

In general a Clifford algebra is associative but non-commutative.

2.1 Some familiar Clifford algebras

This section is to show that some Clifford algebras are certain familiar algebras in disguise. The real numbers $\mathbf{R}$ are trivially isomorphic to the Clifford algebras $\mathbf{R}_{0,0}$.

2.1.1 Complex numbers

The complex numbers are isomorphic to the Clifford algebra $\mathbf{R}_{0,1}$, to see this an element of $\mathbf{R}_{0,1}$ is written as the formal sum,

$$x = x_0 + x_1 e_1 \tag{18}$$

with the multiplication rule $e_1^2 = -1$. This is simply the complex numbers with $e_1 = i$.

2.1.2 The Quaternions.

The Quaternions algebra is generated from the basis elements $1, i, j, k$ with the relations,

$$\begin{aligned} i^2 = j^2 + k^2 &= -1, \\ ij = k &= -ji, \\ jk - i &= -kj, \\ ki = j &= -ik, \end{aligned}$$

The quaternions can be seen as isomorphic to $\mathbf{R}_{0,2}$ with the following isomorphisms,

$$\begin{aligned} e_0 &\longleftrightarrow 1 \\ e_1 &\longleftrightarrow i \\ e_2 &\longleftrightarrow j \\ e_{12} &\longleftrightarrow k \end{aligned}$$

Some perhaps less familiar examples include the Dirac algebra $\mathbf{R}_{4,1}$ and the Pauli algebra $\mathbf{R}_{3,3}$.

2.1.3 The algebra ${}^2\mathbf{R}$

The algebra ${}^2R$ is defined over ordered pairs (x_1, x_2) addition and multiplication is defined as:

$$\begin{aligned}(x_1, x_2) + (y_1, y_2) &= (x_1 + y_1, x_2 + y_2)\\ (x_1, x_2) * (y_1, y_2) &= (x_1 * y_1, x_2 * y_2)\end{aligned}$$

This algebra is isomorphic to the algebra $\mathbf{R}_{1,0}$ this can be defined by defining the isomorphisms ϕ by:

$$\phi(\alpha + \beta e_1) \mapsto (\alpha + \beta, \alpha - \beta)$$

with,

$$\phi^{-1}(a, b) \mapsto \frac{a+b}{2} + \frac{a-b}{2}e_1$$

2.1.4 2 by 2 matrices $R(2)$

$R_{1,1}$ is isomorphic to the set of 2 by 2 matrices, this can seen by the following identification,

$$e_1 \longleftrightarrow \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad e_2 \longleftrightarrow \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$
$$e_{12} \longleftrightarrow \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad e_0 \longleftrightarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

This is a basis for $\mathbf{R}(2)$ and defines the isomorphism of algebras.
This can be generalized to show that $\mathbf{R}_{n,n} \cong R(2^n)$.

2.2 Some relationships between Clifford algebras.

The general idea of a Clifford network is to be able to process multi-dimensional signals, with fewer Clifford nodes instead of many real valued nodes. For some problems the choice of which Clifford algebra to use might be obvious, but in some cases such an obvious choice might not present itself. The engineer then has to experiment with different Clifford algebras. This section is a short guide to the relationships between algebras. Knowing the relationships between each algebra saves the engineer from needless duplication.

Figure 2: A table of Clifford algebras up to dimension 256

$\mathbf{R}_{p,q}$	$p=0$	1	2	3	4	5	6	7	8
$q=0$	$\mathbf{R}$	$\mathbf{C}$	$\mathbf{H}$	${}^2\mathbf{H}$	$\mathbf{H}(2)$	$\mathbf{C}(4)$	$\mathbf{R}(8)$	${}^2\mathbf{R}(8)$	$\mathbf{R}(16)$
1	${}^2\mathbf{R}$	$\mathbf{R}(2)$	$\mathbf{C}(2)$	$\mathbf{H}(2)$	${}^2\mathbf{H}(2)$	$\mathbf{H}(4)$	$\mathbf{C}(8)$	$\mathbf{R}(16)$	${}^2\mathbf{R}(16)$
2	$\mathbf{R}(2)$	${}^2\mathbf{R}(2)$	$\mathbf{R}(4)$	$\mathbf{C}(4)$	$\mathbf{H}(4)$	${}^2\mathbf{H}(4)$	$\mathbf{H}(8)$	$\mathbf{C}(16)$	$\mathbf{R}(32)$
3	$\mathbf{C}(2)$	$\mathbf{R}(4)$	${}^2\mathbf{R}(4)$	$\mathbf{R}(8)$	$\mathbf{C}(8)$	$\mathbf{H}(8)$	${}^2\mathbf{H}(8)$	$\mathbf{H}(16)$	$\mathbf{C}(32)$
4	$\mathbf{H}(2)$	$\mathbf{C}(4)$	$\mathbf{R}(8)$	${}^2\mathbf{R}(8)$	$\mathbf{R}(16)$	$\mathbf{C}(16)$	$\mathbf{H}(16)$	${}^2\mathbf{H}(16)$	$\mathbf{H}(32)$
5	${}^2\mathbf{H}(2)$	$\mathbf{H}(4)$	$\mathbf{C}(8)$	$\mathbf{R}(16)$	${}^2\mathbf{R}(16)$	$\mathbf{R}(32)$	$\mathbf{C}(32)$	$\mathbf{H}(32)$	${}^2\mathbf{H}(32)$
6	$\mathbf{H}(4)$	${}^2\mathbf{H}(4)$	$\mathbf{H}(8)$	$\mathbf{C}(16)$	$\mathbf{R}(32)$	${}^2\mathbf{R}(32)$	$\mathbf{R}(64)$	$\mathbf{C}(64)$	$\mathbf{H}(64)$
7	$\mathbf{C}(8)$	$\mathbf{H}(8)$	${}^2\mathbf{H}(8)$	$\mathbf{H}(16)$	$\mathbf{C}(32)$	$\mathbf{R}(64)$	${}^2\mathbf{R}(64)$	$\mathbf{R}(128)$	$\mathbf{C}(128)$
8	$\mathbf{R}(16)$	$\mathbf{C}(16)$	$\mathbf{H}(16)$	${}^2\mathbf{H}(16)$	$\mathbf{H}(32)$	$\mathbf{C}(64)$	$\mathbf{R}(128)$	${}^2\mathbf{R}(128)$	$\mathbf{R}(256)$

The most comprehensive reference on the properties of Clifford algebras is perhaps [6], all proofs that are omitted in this section can be found in Chapter 13 of [6].

So far the following relationships have been demonstrated,

$$\begin{aligned}
\mathbf{R}_{0,0} &\cong \mathbf{R} \\
\mathbf{R}_{0,1} &\cong \mathbf{C} \\
\mathbf{R}_{0,2} &\cong \mathbf{R} \\
\mathbf{R}_{n,n} &\cong \mathbf{R}(2^n) \\
\mathbf{R}_{1,1} &\cong {}^2\mathbf{R}
\end{aligned}$$

In fact all Clifford algebras defined over the real numbers can be constructed in some way from $\mathbf{R}$,$\mathbf{C}$ or $\mathbf{H}$. Table 2.2 extends this information. The reader interested in the explicit construction of the table, should again consult [6] or [8].

The most useful fact (again for a proof see [6]) is perhaps,

$$\mathbf{R}_{p+1,q} \cong \mathbf{R}_{q+1,p} \tag{19}$$

.

To complete the table to arbitrary algebras it is enough to know that,

$$\mathbf{R}_{p,q+8} \cong \mathbf{R}_{p,q} \otimes \mathbf{R}(16) \cong \mathbf{R}_{p,q}(16). \tag{20}$$

3 The Back Propagation algorithm.

This section deals with back error propagation (BEP) in clifford algebras. The case for real networks was first presented in [14], The complex case has been treated by many people, see for instance [12] [5] [11] [1]. The account here of Clifford back propagation is a generalization of Georgiou and Koutsougeras work [4]. The Clifford case was first presented by the authors in [7]. Because error metrics are dealt with differently from most accounts, the case for a single real valued neuron is presented, extensions to multi-layer and Clifford valued networks are then more easy to perform.

3.1 The Single Real Valued Neuron.

Consider a single real valued neuron with n-inputs and one output. The transfer function $\Psi : \mathbf{R}^n \to \mathbf{R}$ is defined by:

$$\Psi = f(x \cdot \omega) \tag{21}$$

With x and ω vectors in $\mathbf{R}^n$ being the input and weight vectors respectively.

BEP is an iterative algorithm which attempts to minimize some error measure by gradient descent. The minimization takes place in the weight space, thus the partial derivative,

$$\frac{\partial E}{\partial \omega_i} \tag{22}$$

where ω_i represents the i'th part of the vector ω.

Define the error metric E by,

$$E = \frac{1}{2} \int_{x \in X} |\Psi - \Phi|^2 \tag{23}$$

X is some compact subset of $\mathbf{R}^n$ which ensures the integral converges and Φ is the target function.

Then,

$$\frac{\partial E}{\partial \omega_i} = \int_{x \in X} \frac{\partial}{\partial \omega_i} |\Psi - \Phi|^2 \tag{24}$$

Define,

$$net = x \cdot w = \sum_i x_i w_i \tag{25}$$

and,

$$\lambda = |\Psi - \Phi|^2 \tag{26}$$

Thus,

$$\frac{\partial E}{\partial \omega_i} = \int_{x \in X} \frac{\partial \lambda}{\partial \omega_i}$$

Then using the chain rule,

$$\frac{\partial \lambda}{\partial w_i} = \frac{\partial \lambda}{\partial net} \frac{\partial net}{\partial w_i} \tag{27}$$

Since $net = \sum x_i \omega_i$

$$\frac{\partial net}{\partial \omega_i} = x_i \tag{28}$$

Then again using the chain rule,

$$\frac{\partial \lambda}{\partial net} = \frac{\partial \lambda}{\partial \Psi} \frac{\partial \Psi}{\partial net} \tag{29}$$

$$\frac{\partial \Psi}{\partial net} = f'(net), \tag{30}$$

$$\frac{\partial \lambda}{\partial \Psi} = \frac{\partial}{\partial \Psi} |\Psi - \Phi|^2 \tag{31}$$

Rewriting $|\Psi - \Phi|^2$ as $(\Psi - \Phi)^2$,

$$\frac{\partial \lambda}{\partial \Psi} = (\Psi - \Phi) \tag{32}$$

So bringing all this together,

$$\frac{\partial E}{\partial \omega_i} = \int_{x \in X} ((\Psi - \Phi) f'(net) x_i) \tag{33}$$

This equations obviously needs some sort of approximation if it is to be implemented on a computer. If ω_i^t (the value of ω_i at time t is updated as,

$$\omega_i^{t+1} = \omega_i^t + \eta \frac{\partial E}{\partial \omega_i} \tag{34}$$

for a sufficiently small value of η. Also the integral

$$\int_{x \in X} (\Psi - \Phi)$$

needs to be approximated, but the method of approximation depends on the form of the function Φ.

3.2 BEP for a multi-layer feed-forward network.

Consider a multi-layer network with a transfer function,

$$\Psi : \mathbf{R}^n \rightarrow \mathbf{R}^m$$

Each node will be assigned a unique number j and ω_{ij} represents the weight on the connection from node i to node j.

Again it is required to find,

$$\frac{\partial E}{\partial \omega_{ij}} = \frac{1}{2}\int_{x \in X} \frac{\partial}{\partial \omega_{ij}} |\Psi - \Phi|^2$$

Define $\lambda = |\Psi - \Phi|^2$ as before.

The first steps are the same as for the previous case,

$$\frac{\partial \lambda}{\partial \omega_{ij}} = \frac{\partial \lambda}{\partial net_j}\frac{\partial net_j}{\partial \omega_{ij}} \tag{35}$$

$$\frac{\partial net_j}{\partial \omega_{ij}} = o_i \tag{36}$$

Where o_i represents the output of the i'th neuron.

$$\frac{\partial \lambda}{\partial net_j} = \frac{\partial \lambda}{\partial o_j}\frac{\partial o_j}{\partial net_j} \tag{37}$$

again,

$$\frac{\partial o_j}{\partial net_j} = f'(net_j) \tag{38}$$

If o_j is an output neuron then,

$$\frac{\partial \lambda}{\partial o_j} = (\Psi_j - \Phi_j) \tag{39}$$

where Ψ_j represents the j'th component of Ψ.

When the unit is hidden a further application of the chain rule is required,

$$\begin{aligned}\frac{\partial \lambda}{\partial o_j} &= \sum_k \frac{\partial \lambda}{\partial net_k}\frac{\partial net_k}{\partial o_j} \\ &= \sum_k \frac{\partial \lambda}{\partial net_k}\omega_{jk} \end{aligned} \tag{40}$$

Where k is summing over all the nodes in the layer above.

So define for hidden units j,

$$\delta_j = f'(net_j) \sum_k \delta_k \omega_{jk} \tag{41}$$

For output units j,

$$\delta_j = (\Psi_j - \Phi_j) f'(net_j) \tag{42}$$

Thus,

$$\frac{\partial E}{\partial \omega_{ij}} = \int_{x \in X} \delta_j o_i \tag{43}$$

3.3 Clifford Back error Propagation

Some care has to be taken when choosing a norm for a Clifford algebra. In what follows the norm $|\cdot|$ will be used, where,

$$|x| = \left(\sum_A [x]_A^2 \right)^{\frac{1}{2}} \tag{44}$$

Recalling $[x]_A$ represents the A'th part of the Clifford number x.

A feed-forward Clifford network with n inputs and m outputs will have a transfer function,

$$\Psi : (\mathbf{R}_{p,q})^n \to (\mathbf{R}_{p,q})^m \tag{45}$$

Where $(\mathbf{R}_{p,q})^n$ is the n-dimensional left module over the Clifford algebra $\mathbf{R}_{p,q}$[1].

Again some sort of error metric has to be defined. The basic form is the same,

$$E = \frac{1}{2} \int_{x \in X} \|\Psi - \Phi\|^2 \tag{46}$$

where X is some compact subset of the Clifford module $(\mathbf{R}_{p,q})^n$ with the product topology derived from the norm 44.

[1]If the reader is not familiar with the concept of a module, it is enough to view these Clifford modules as weaker forms of n dimensional vectors with Clifford valued scalars instead of real values scalars

It is convenient from the point of view of the derivation of the BEP equations to define $\|\cdot\|$ as,

$$\|x\|^2 = \sum_{i=1}^{k} |(x)_i|^2 \tag{47}$$

Where (x_i) is a Clifford number representing the i'the part of x in the m dimensional Clifford module over $\mathbf{R}_{p,q}$.

Assume that each node in the network has the same Clifford valued activation function $f : \mathbf{R}_{p,q} \to \mathbf{R}_{p,q}$.

The output o_j of the j'th neuron can be written as,

$$o_j = f(net_j) = \sum_A u_A^j e_A \tag{48}$$

With u_A^j a function from $\mathbf{R}_{p,q}$ to $\mathbf{R}$ and

$$net_j = \sum_l \omega_{lj} o_k \tag{49}$$

Where k sums over all the inputs to neuron j.

It is important to notice since $\mathbf{R}_{p,q}$ is in general non-commutative the order of multiplication in the above equation is important.

In the real case E depends on the number of weights in the network. In the Clifford case E depends not only on all the weights but on the components of each of the weights. Again define $\lambda = \|\Psi - \Phi\|^2$. Then:

$$\frac{\partial E}{\partial [\omega_{ij}]_A} = \frac{1}{2} \int_{x \in X} \frac{\partial \lambda}{\partial [\omega_{ij}]_A} \tag{50}$$

Using the chain rule,

$$\frac{\partial \lambda}{\partial [\omega_{ij}]_A} = \sum_B \left(\frac{\partial \lambda}{\partial u_B^j} \left(\sum_C \frac{\partial u_B^j}{\partial [net_j]_C} \frac{\partial [net_j]_C}{\partial [\omega_{ij}]_A} \right) \right) \tag{51}$$

The partial derivative

$$\frac{\partial [net_j]_C}{\partial [\omega_{ij}]_A}$$

needs a bit of care. Using equation 49:

$$\frac{\partial[net_j]_C}{\partial[\omega_{ij}]_A} = \sum_l \frac{\partial[\omega_{lj}x_l]_C}{\partial[\omega_{ij}]_A} = \frac{\partial[\omega_{ij}o_i]_C}{\partial[\omega_{ij}]_A}$$

Then using the fact that

$$\omega_{kj}o_k = \sum_{D,E}[\omega_{kj}]_D[o_k]_E(-1)^\kappa e_{D\Delta E}$$

with κ defined as in 16, then,

$$\frac{\partial[\omega_{kj}o_k]_C}{\partial[\omega_{kj}]_A} = (-1)^\kappa[o_k]_E \tag{52}$$

where E is the basis element satisfying the equation:

$$(-1)^k e_A e_E = e_C$$

For example in the algebra $\mathbf{R}_{2,0}$ the table of derivatives would look like,

$\frac{\partial[x]_B}{\partial[\omega_{jl}]_A}$	$B=0$	1	2	12
$A=0$	$[x_{jl}]_0$	$[x_{jl}]_1$	$[x_{jl}]_2$	$[x_{jl}]_{12}$
1	$[x_{jl}]_1$	$[x_{jl}]_0$	$[x_{jl}]_{12}$	$[x_{jl}]_2$
2	$[x_{jl}]_2$	$-[x_{jl}]_{12}$	$[x_{jl}]_0$	$-[x_{jl}]_1$
12	$-[x_{jl}]_{12}$	$[x_{jl}]_2$	$-[x_{jl}]_1$	$[x_{jl}]_0$

(53)

The error derivative is now quite easy to calculate, if j is an output neuron then,

$$\frac{\partial\lambda}{\partial u_A^j} = \frac{\partial}{\partial u_A^j}\|\Psi - \Phi\|$$

$$\frac{\partial}{\partial u_A^j}|o_j - \Phi_j^2|^2 = 2[o_j - \Phi_j]_A$$

If j is not an output unit then the chain rule has to be used again.

$$\frac{\partial\lambda}{\partial u_A^j} = \sum_k \frac{\partial\lambda}{\partial u_A^k}\left(\sum_{B,C}\frac{\partial u_B^k}{\partial[net_k]^C}\frac{\partial[net_k]^C}{\partial u_A^j}\right) \tag{54}$$

With k running over the neurons that receive input from neuron j.

The term

$$\frac{\partial[net_k]_C}{\partial[u_j]_A}$$

is calculated in a similar manner to 52,

$$\frac{\partial[net_l]_C}{\partial u_A^j} = (-1)^{\kappa'}[\omega_{jk}]_D \tag{55}$$

Where κ' and D satisfy the same conditions as before.

The derivatives:

$$\frac{\partial u_B^k}{\partial[x_k]_C}$$

play the same rôle as $f'(net_j)$ does in the real valued case and depends on the activation function used, this will be discussed in the next chapter.

Bringing this all together we have,

$$\frac{\partial E}{\partial[\omega_{ij}]_A} = \frac{1}{2}\int_{x\in X}\sum_B \lambda_j^B \left(\sum_C \frac{\partial u_B^j}{\partial[net_j]_C}\kappa[o_k]_E\right) \tag{56}$$

With

$$\lambda_j^B = \frac{\partial\|\Psi - \Phi\|^2}{\partial u_B^j} = 2[o_j - \Phi_j]_B \tag{57}$$

If j is an output neuron.

With κ as the sign of the product $e_D e_E$, and E is the basis element satisfying the equation:

$$\kappa e_A e_E = e_C$$

If j is not an output unit then the chain rule has to be used again.

$$\lambda_j^B = \sum_k \lambda_k^B \left(\sum_{B,C} \frac{\partial u_B^k}{\partial[net_k]^C}\kappa'[\omega_{jk}]_D\right) \tag{58}$$

With k running over the neurons that receive input from neuron j and κ' and D satisfying similar conditions to κ and E.

4 Choice of activation function

So far nothing has been said about the choice of activation functions used in networks. The most widely used class of activation functions are the so called semi-linear functions. A semi-linear function $f(x)$ is a continuous function that satisfies the following condition:

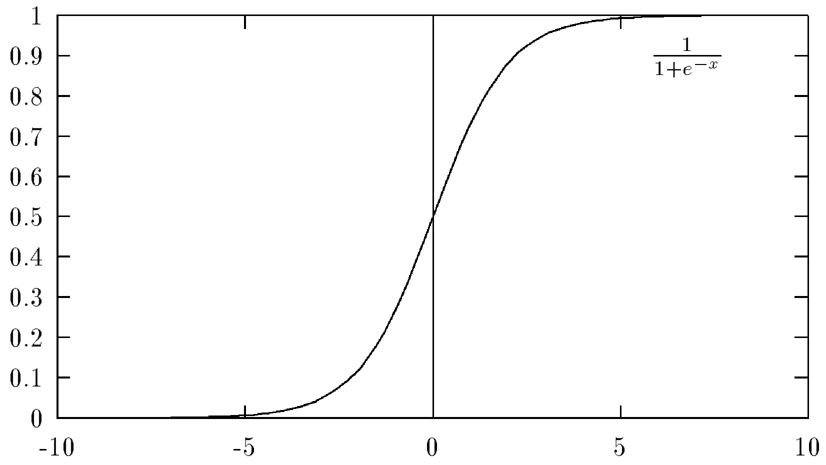
$$\lim_{x \to \infty} f(x) = 1 \quad \lim_{x \to -\infty} f(x) = -1 \tag{59}$$

Typically the second condition is replaced by the condition:

$$\lim_{x \to -\infty} f(x) = 0$$

Typical examples include the sigmoid function:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{60}$$



and the tanh function:

$$f(x) = \tanh(x) \tag{61}$$

The reason for this choice of functions comes from two observations. First linear functions are not satisfactory since any multi-layer network composed of linear functions is equivalent to a single layer network, and further many decision are not linearly separable. The second reason, is that traditionally the first neural networks(see for instance McCulloch and Pitts [15]) had an all or nothing response, typically neurons had step valued activation function:

$$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{62}$$

These functions do indeed provide multi-layer networks with more power than a single layer network. Indeed it was shown by Minsky and Papert [13] that single layer networks with step functions can not solve some non-trivial decision problems. But because the step functions are non-linear and are integer valued this turns training into an integer programming problem, which is known to be NP-hard (although there might exist individual cases solvable in polynomial time). Further it limits the networks to functions defined on binary data.

Semi-linear functions provide a compromise between linear functions and step functions. From the observation that $f(\alpha x)$ for large values of α acts like a step functions or more correctly the function:

$$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ \frac{1}{2} & \text{if } x = 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{63}$$

and for values of α close to zero, $f(\alpha x)$ acts like a linear function. Indeed Sontag [2] has shown that networks composed of sigmoid functions are better at binary decision problems than networks with only step functions, for exactly the reasons stated above(i.e. sigmoids can approximate either linear or step valued activation functions).

Further it has been shown by many authors see for instance Hornick [10] than a network with one hidden layer with sigmoid activation functions is able to approximate any continuous function defined on a compact subset, providing a sufficient number of neurons is present.

4.1 Complex activation functions.

It might be naively assumed that the rich field of complex analysis is going to provide a suitable class of activation functions. Indeed there exists a complex extension of the sigmoid function:

$$f(z) = \frac{1}{1 + e^{-z}}$$

where e^{-z} is the complex exponential function. This function is analytic(in the sense of complex analysis) but it is not bounded. Indeed any function that is complex analytic and bounded is necessary constant by Liouville's theorem(see any standard text on complex analysis for a proof).

The most important characteristic of a complex activation function is that it should be bounded and nonlinear in it's components and the partial derivatives should exist and be continuous and the partial derivatives must be such that learning always takes place in the presence of non-zero error. For a fuller discussion see [4].

In [4] a simple complex activation is proposed:

$$f(z) = \frac{z}{c + \frac{1}{r}|z|} \tag{64}$$

This activation function has been used successfully in some simple applications, for instance the complex encoder decoder problem.

4.2 The Clifford case

The simple activation function proposed above extends to the Clifford case, the partial derivatives are easy to work out(using the notation of the previous chapter):

$$\frac{\partial u_A}{\partial [x]_B} = \begin{cases} -\frac{r[x]_A[x]_B}{(c+\frac{1}{r}|x|)^2 r|x|} & \text{if } |x| > 0 \\ 0 & \text{if } |x| = 0 \end{cases} \tag{65}$$

if $A \neq B$ and if $A = B$ then,

$$\frac{\partial u_A}{\partial [x]_B} = \begin{cases} \frac{r(|x|^2 - [x]_A^2 + cr|x|)}{|x|(cr+|x|)^2} & \text{if } |x| \neq 0 \\ \frac{1}{c} & \text{if } |x| = 0. \end{cases} \tag{66}$$

The norm being the Clifford norm defined in the previous chapter.

5 Experimental results

The encoder-decoder problem is often used to test new techniques in back error propagation. While it is not a formal benchmark it is useful to get a feel of how new algorithms can perform. Essentially for a network to solve the encoder-decoder problem a training set is presented to the network which forces the hidden units to encode the training set in some way. For instance in the real case with a 3-2-3 network if the network is trained on the set of vectors $(1,0,0)$, $(0,1,0)$, $(0,0,1)$ then then two hidden units will learn a binary coding of the input signals.

The results presented in this section show how a Clifford network is able to solve this encoder-decoder problem for multidimensional patterns.

Results are given for different algebras and different network configurations. In each case the system successfully encoders and decodes the input patterns providing a well separated response. It is also interesting to note that the epoch count is approximately the same as would be expected for a similar problem on a conventional BEP network (wall clock time is obviously increased due to the extra complexity in the individual arithmetic operations). This preliminary test shows that Clifford networks are able to learn simple tasks and confirms the convergent operation of the learning algorithm.

Figure 3: Table of outputs from a trained 3-2-3 encoder-decoder $\mathbf{R}_{0,2}$.

Pattern 0	(1.00000,0.00000,0.00000,0.00000) (0.00000,0.00000,0.00000,0.00000) (0.00000,0.00000,0.00000,0.00000)
Output	(0.82741,0.00180,-0.00034,0.00075) (-0.00896,-0.01346,0.03117,-0.00653) (-0.02300,0.00890,-0.00371,-0.00493)
Pattern 1	(0.00000,0.00000,0.00000,0.00000) (1.00000,0.00000,0.00000,0.00000) (0.00000,0.00000,0.00000,0.00000)
Output	(0.01103,0.02384,-0.00875,-0.00106) (0.82393,-0.00412,-0.00112,-0.00050) (-0.02084,0.00501,-0.00617,0.00553)
Pattern 2	(0.00000,0.00000,0.00000,0.00000) (0.00000,0.00000,0.00000,0.00000) (0.00000,0.00000,0.00000,-1.00000)
Output	(0.01646,0.03726,-0.02079,-0.00340) (-0.01691,-0.02265,0.03517,-0.00731) (-0.00301,0.00194,-0.00061,-0.82819)

Figure 4: Table of input-output for a trained 4-2-4 encoder-decoder over the algebra $\mathbf{R}_{0,2}$.

Pattern 0	(1.00000,0.00000,0.00000,0.00000) (0.00000,0.00000,0.00000,0.00000) (0.00000,0.00000,0.00000,0.00000) (0.00000,0.0 0000,0.00000,0.00000)
Output	(0.89258,-0.00044,0.00001,0.00025) (0.00020,-0.00002,-0.00002,-0.00004) (0.00018,-0.00004,0.00012,-0.00015) (-0 .00000,-0.00000,-0.00000,0.00000)
Pattern 1	(0.00000,0.00000,0.00000,0.00000) (1.00000,0.00000,0.00000,0.00000) (0.00000,0.00000,0.00000,0.00000) (0.00000,0.0 0000,0.00000,0.00000)
Output	(0.00020,0.00006,-0.00006,-0.00005) (0.89238,0.00023,0.00064,0.00041) (0.00018,0.00001,-0.00019,-0.00002) (-0.0 0000,-0.00000,0.00000,0.00000)
Pattern 2	(0.00000,0.00000,0.00000,0.00000) (0.00000,0.00000,0.00000,0.00000) (1.00000,0.00000,0.00000,0.00000) (0.00000,0.0 0000,0.00000,0.00000)
Output	(0.00019,0.00007,0.00005,0.00001) (0.00019,-0.00006,-0.00005,-0.00002) (0.89278,0.00014,0.00065,0.00152) (0.000 00,0.00000,0.00000,-0.00000)

Figure 5: Table of input-ouput relationships for a trained 4-2-4 encoder-decoder the algebra $\mathbf{R}_{1,1}$.

Pattern 0	(1.00000,0.00000,0.00000,0.00000) (0.00000,0.00000,0.00000,0.00000) (0.00000,0.00000,0.00000,0.00000) (0.00000,0.0 0000,0.00000,0.00000)
Output	(0.79505,0.00213,-0.03843,0.00260) (0.60009,0.00017,-0.07628,0.00026) (-0.51896,-0.00026,0.17408,-0.00081) (0.0 0000,0.00000,-0.00000,-0.00000)
Pattern 1	(0.00000,0.00000,0.00000,0.00000) (1.00000,0.00000,0.00000,0.00000) (0.00000,0.00000,0.00000,0.00000) (0.00000,0.0 0000,0.00000,0.00000)
Output	(0.49496,0.00049,0.28122,-0.00068) (-0.46598,-0.05589,0.07024,0.00565) (0.78560,0.00071,-0.15706,0.00044) (0.00 000,0.00000,-0.00000,0.00000)
Pattern 2	(0.00000,0.00000,0.00000,0.00000) (0.00000,0.00000,0.00000,0.00000) (1.00000,0.00000,0.00000,0.00000) (0.00000,0.0 0000,0.00000,0.00000)
Output	(-0.42091,0.00011,-0.31815,0.00012) (0.78305,-0.00003,0.13566,-0.00022) (0.57565,0.00087,0.00391,0.00224) (-0.0 0000,-0.00000,0.00000,-0.00000)

Figure 6: Table of input and output of a 3-2-3 encoder over the algebra $\mathbf{R}_{0,3}$ with graph of rms error.

Pattern 0	(1.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00) (0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00) (0.00,0.00,0.00,0.00,0.00,0.00,0 .00,0.00)
Output	(0.81,-0.00,-0.00,-0.00,0.00,-0.00,0.00,0.00) (0.00,0.00,0.00,-0.00,-0.00,0.00,0.00,0.00) (-0.00,0.00,-0.00,-0.00, -0.00,-0.00,0.00,0.00)
Pattern 1	(0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00) (1.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00) (0.00,0.00,0.00,0.00,0.00,0.00,0 .00,0.00)
Output	(0.00,0.00,0.00,0.00,-0.00,0.00,0.00,0.00) (0.81,0.00,-0.00,0.00,0.00,-0.00,-0.00,-0.00) (-0.00,-0.00,0.00,-0.00,0 .00,0.00,0.00,0.00)
Pattern 2	(0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00) (0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00) (0.00,0.00,0.00,0.00,0.00,0.00,0 .00,1.00)
Output	(0.00,0.00,-0.00,0.00,0.00,0.00,-0.00,-0.00) (0.00,-0.00,-0.00,0.00,0.00,0.00,-0.00,-0.00) (0.00,-0.00,-0.00,0.00, -0.00,0.00,-0.00,0.81)

Figure 7: Graph of RMS error for a 3-2-3 encoder-decoder problem over the algebra $\mathbf{R}_{0,2}$.

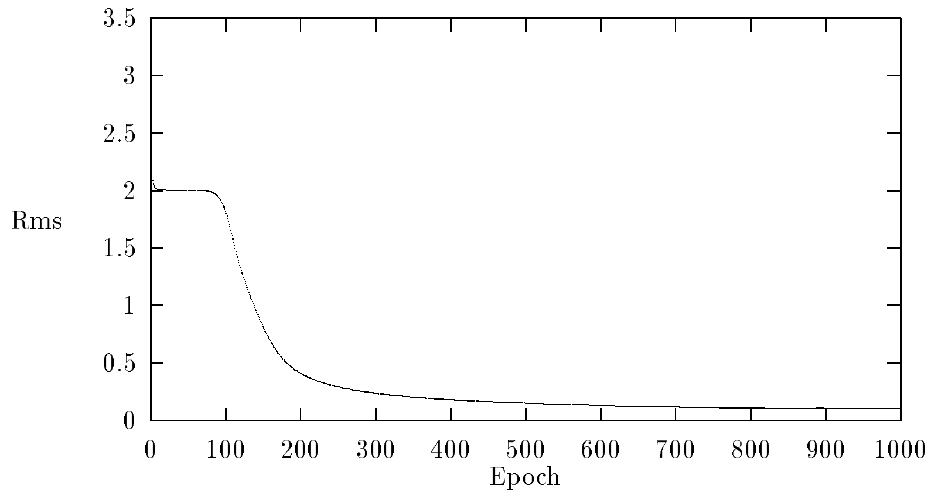


Figure 8: Graph of RMS error during training the the 4-2-4 encoder-decoder over the algebra $\mathbf{R}_{0,2}$.

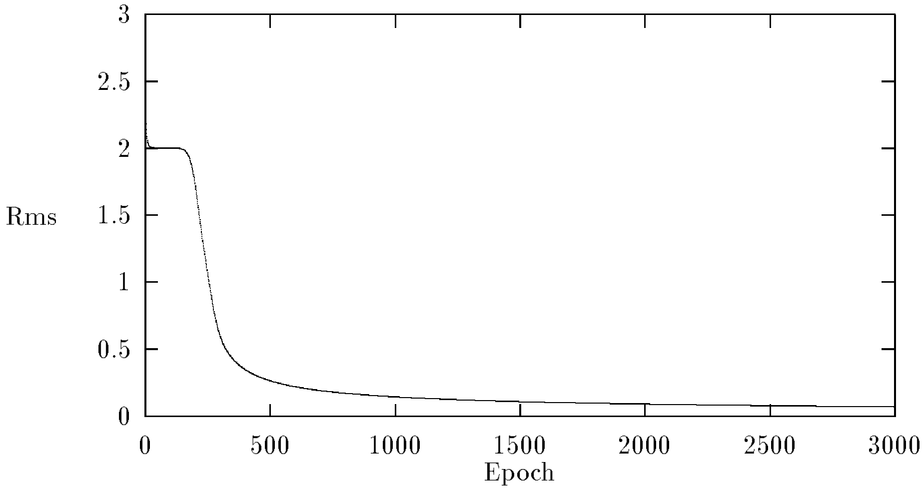


Figure 9: Graph of RMS during learning for a 4-2-4 encoder-decoder problem over the algebra $\mathbf{R}_{1,1}$.

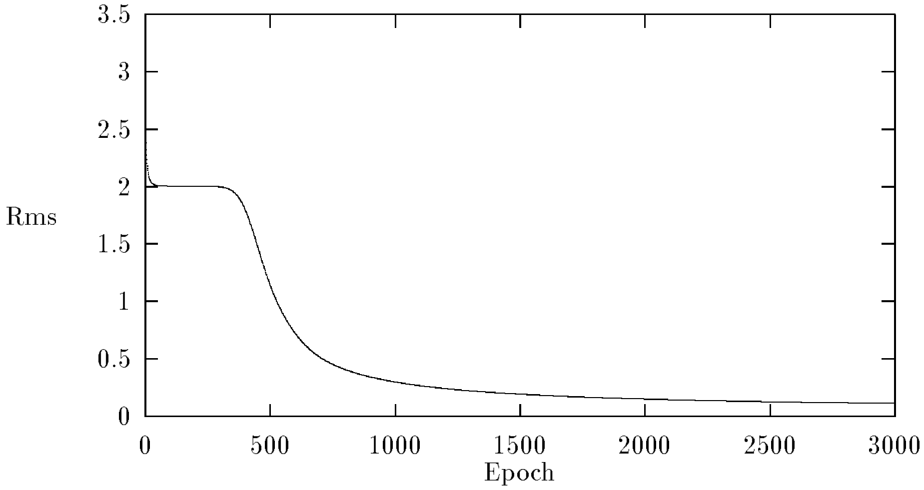
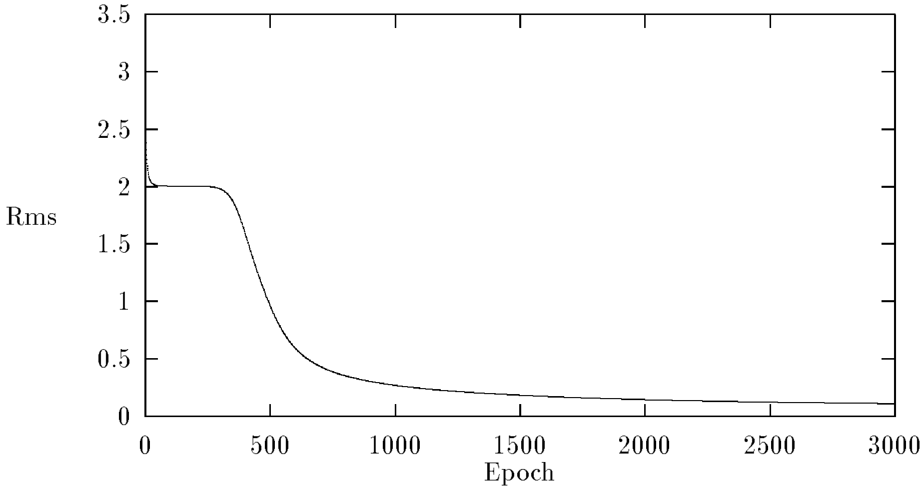


Figure 10: Graph of RMS error during training for a encoder-decoder problem over the algebra $\mathbf{R}_{0,2}$.



6 Conclusion

This report has formalized the multi-layer preceptron and the back error propagation algorithm. It has extended these formalisms by incorporating Clifford algebras to form Clifford networks. A new learning algorithm has been derived, based on gradient descent. The results for the multi-dimensional encoder decoder problems demonstrate that the proposed network is capable of learning simple problems.

7 Future work

This report is concerned very much with work in progress. Many issues are as yet unresolved, on the theoretical side an adequate characterization of the class of functions computable by Clifford nets needs to be found. On the practical side almost nothing has yet been done, the question perhaps many readers will be asking is, 'What are Clifford nets good for?'. A Clifford algebra is a natural extension of a vector algebra that has a true vector multiplication, indeed if associative multiplication is required then necessarily a Clifford algebra is produced. Thus Clifford networks provide, a unified way of treating multi-dimensional signals. It is not yet clear theoretically what advantages Clifford networks provide, but is hoped that because Clifford networks treat vector valued signals as single entities, Clifford networks will produce better internal representations of multi-dimensional problems. Such multi-dimensional problems include colour pattern recognition, where colour signals are treated as 3-dimensional red, green and blue signals which are embedded in a suitable Clifford algebra. Other problems could include motion problems in 3-dimensions, these have quite natural Clifford algebras and hence could be treated with Clifford networks.

Work is under way to use Clifford networks as demodulators for Q^2PSK modulation schemes (a 4-dimensional generalization of QPSK) initial result seem very promising, results from this are to be reported in a forth coming paper.

Acknowledgements

The authors wish to thank the S.E.R.C. and British Telecom for financial support. Also many conversations with Roy Chisholm and Alan Common of the University of Kent Clifford algebra group has been most helpful.

References

[1] N. Benvenuto and F. Piazza. On the complex backpropagation algorithm. *IEEE Transactions on Signal Processing*, 40(4):967–969, April 1992.

[2] E.D.Sontag. Feedforwards nets for interpolation and classication. *Jounral of Computer and System Sciences*, 4:299–322, 1992.

[3] E.T.Copson. *Metric Spaces.* Cambridge University Press, first paper back edition edition, 1988.

[4] George M. Georgiou and Cris Koutsougeras. Complex domain backpropagation. *IEEE Transactions on Circuits and Systems*, pages 330–334, 1992.

[5] J. Henseler and P. J. Braspenning. Training complex multi-layer neural networks. Technical Report CS 90-02, University of Limburg, Department of Computer Science, P.O. Box 616, 6200 MD Maastricht, The Netherlands, 1990.

[6] I.R.Porteous. *Topological Geometry.* Cambridge University Press, 1981.

[7] J.K.Pearson and D.L.Bisset. Back Propagation in a Clifford Algebra. In *ICANN Brighton*, September 1992.

[8] H. Blaine Lawson Jr. and Marie-Louise Michelsohn. *Spin Geometry.* Princeton University Press, Princeton, New Jersey, 1989.

[9] K.Hornick. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4:251–257, 1991.

[10] Stinchcombe K.Hornick and White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[11] H. Leung and S. Haykin. The complex backpropagation algorithm. *IEEE Transactions on Signal Processing*, 39:2101–2104, September 1991.

[12] Gordon R. Little, Steven C. Gustafson, and Robert A. Senn. Generalization of the backpropagation neural network learning algorithms to permit complex weights. *Applied Optics*, 29(11):1591–1592, April 1990.

[13] Marvin L.Minsky and Seymour A.Papert. *Perceptrons Expanded edition.* MIT Press, Cambridge M.A., 1988.

[14] D. E. Rumelhart and J. L. McClelland. Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume I. M.I.T. Press, Cambridge, MA, 1986.

[15] W.S.McCulloch and Walter Pitts. A logical calculus of the idea immanent in neural nets. *Bulletin of Mathematical Biophysics*, pages 115–137, 1943.