INSTITUT FÜR INFORMATIK

UND PRAKTISCHE MATHEMATIK

A Theory of Neural Computation with Clifford Algebras

Sven Buchholz

Bericht Nr. 0504
Mai 2005

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

KIEL

Institut für Informatik und Praktische Mathematik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

A Theory of Neural Computation with Clifford Algebras

Sven Buchholz

Bericht Nr. 0504
Mai 2005

e-mail: sbh@ks.informatik.uni-kiel.de

Dieser Bericht gibt den Inhalt der Dissertation wieder, die der Verfasser im
November 2004 bei der Technischen Fakultät der
Christian–Albrechts–Universität zu Kiel eingereicht hat.
Datum der Disputation: 16. März 2005.

1. Gutachter	Prof. Dr. Gerald Sommer (Kiel)
2. Gutachter	Prof. Dr. Reinhold Schneider (Kiel)
3. Gutachter	Prof. Dr. Thomas Martinetz (Lübeck)

Datum der mündlichen Prüfung: 16.03.2005

Abstract

The present thesis introduces Clifford Algebra as a framework for neural computation. Clifford Algebra subsumes, for example, the reals, complex numbers and quaternions. Neural computation with Clifford algebras is model–based. This principle is established by constructing Clifford algebras from quadratic spaces. Then the subspace grading inherent to any Clifford algebra is introduced, which allows the representation of different geometric entities like points, lines, and so on. The above features of Clifford algebras are then taken as motivation for introducing the Basic Clifford Neuron (BCN), which is solely based on the geometric product of the underlying Clifford algebra. Using BCNs the Linear Associator is generalized to the Clifford associator. As a second type of Clifford neuron the Spinor Clifford Neuron (SCN) is presented. The propagation function of a SCN is an orthogonal transformation. Examples of how Clifford neurons can be used advantageously are given, including the linear computation of Möbius transformations by a SCN. A systematic basis for Clifford neural computation is provided by the important notions of isomorphic Clifford neurons and isomorphic representations. After the neuron level is established, the discussion continues with (Spinor) Clifford Multilayer Perceptrons. The treatment is divided into two parts according to the type of activation function used. First, (Spinor) Clifford Multilayer Perceptrons with real–valued activation functions ((S)CMLPs) are studied. A generic Backpropagation algorithm for CMLPs is derived. Also, universal approximation theorems for (S)CMLPs are presented. The efficency of (S)CMLPs is shown in a couple of simulations. Finally, the class of Clifford Multilayer Perceptrons with Clifford–valued activation functions is studied.

Acknowledgments

The making of this thesis would not have been possible without the support of a lot of people. It is my great pleasure to thank them here.

First of all I thank my supervisor Professor Gerald Sommer for his confidence in my work and for the encouraging discussions during all the years. From the very first ideas to this final version of the text he has been a constant source of advise and inspiration.

I also thank the co–referees Professor Thomas Martinetz and Professor Reinhold Schneider for their interest in my work and their helpful comments.

Furthermore, I thank all members of the Cognitive Systems Group, Kiel for their support and help. My very special thanks go to Vladimir Banarer, Dirk Kukulenz, Francoise Maillard, Christian Perwass and Nils Siebel.

Last, but not least, I thank my parents Claus and Jutta, to whom I dedicate this thesis.

Contents

Chapter 1

Introduction

The three stages of intelligent action according to [22] are conversion of the stimulus into an internal representation, manipulation of that representation by a cognitive system to produce a new one, and conversion of that new representation into a response. This clearly maps well onto (feed–forward) neural networks. However, such networks rather process unstructured data than structured representations. Many problems arise from that lack of structure, most important the integration of prior knowledge. This thesis introduces Clifford Algebra as a framework for the design of neural architectures processing representations advantageously.

1.1 Motivation

Thinking about mind, consciousness, and thinking itself is the root of all philosophy. Therefore philosophy was the first discipline challenging the question

What is intelligence?

In the first half of the 20th century other disciplines started their own challenge of particular versions of the above question. Each one driven by its own special origins, methods and hopes.

A psychological approach was undertaken in 1938 by Skinner. In [81] he showed how the environment could be used to train an animal's behavior. A refinement of that principle, reinforcement learning, is widely used today for mobile robots and multi–agent systems [43].

Already in 1933 Thorndike presumed [85] that learning accounts in the brain by the change of connectivity patterns among neurons. For this postulated principle he had coined the term *connectionism*. Some years later Hebb reported in [35] biological evidence for connectionism. From brain slice experiments he inferred the following rule: If two neurons on either side of a synapse (i.e. connection) are activated simultaneously, then the strength of that synapse is selectively increased. This can be seen as the offspring of unsupervised learning, which is the general theory of learning without (external) feedback (from a teacher).

Meanwhile the seeds of a new era were sown. Many mathematicians were challenged by Hilbert's Entscheidungsproblem

What are the intuitively computable functions?

The work of Church [17] and Kleene [46] cumulated in 1936 in what is now famous as the Church thesis: *The computable functions are the general recursive function*. In the same year Turing proposed in his famous paper [86] a hypothetical device capable of computing every general recursive function.

Inspired by both neurophysiology and Turing's work McCulloch and Pitts published in 1943 a highly influential paper [58]. Their idea was to view biological neurons as sort of logical gates. Thus way biological neurons were "turned into" processing units for the first time. This was the birth of *neural computation* — a biologically inspired paradigm for computation.

Well, there was another computational model which also emerged in that period of time. That is of course the computer itself. When the computer era started in the 1950s neural computation was one of the first research fields participating from its benefits. Computers allowed for simulation of neural models, for which [28] is a very early example.

The next milestone in neural computation was set in 1958 when Rosenblatt [72] proposed a neural architecture that he called *Perceptron*. The Perceptron was intended as a model for human perception and recognition. Later in [73] he introduced as modification an error correction procedure for it. Learning by error correction is termed *supervised learning*. Perceptrons created much excitement and interest in neural computation. Neural networks like Perceptrons seemed to deliver what Turing once defined to be the creative challenge of *artificial intelligence* (AI) [21]

What we want is a machine that can learn from experience.

That interest was abruptly stopped at the end of the 1960s. Whether this was caused by the 1969 book *Perceptrons* [59] by Minsky and Papert has been a controversial question ever since. The book contained many examples for the limited power of single Perceptrons, among them the famous exclusive–or (XOR) problem. For Perceptrons with several layers an efficient learning procedure was still not known at that time. It was not until the mid 1980s that neural networks trained by supervised learning entered the stage again. But this time it meant a real revolution.

That new chapter in the history of neural computation is attributed with the names of Rumelhart and McClelland [77, 57]. One particular contribution by these authors [76] introduced an error correction procedure for multi–layer neural networks. Since then the procedure is known as *Backpropagation* and the associated network as *Multilayer Perceptron* (MLP). The MLP soon turned out to be very powerful for almost any type of applications. Theoretically, it was proven to be able to learn any reasonable function [23]. Around that time neural networks were widely recognized as leading directly towards real artificial intelligence. Or, as stated in [34]

The neural network revolution has happened. We are living in the aftermath.

That statement remains true. However, many of the enthusiasm originally directed to neural networks seems to be gone today. Sure, like everything else, science has its modes. But there are better optimization techniques than Backpropagation. Learning from examples has theoretical bounds — one being the so-called *bias/variance dilemma* [31]. New players have entered the scene - like Support Vector Machines [88] or approximative algorithms. So, has neural computation lost itself in too many technical details? Of course, neural networks are well established and things are still away from a crisis. Nevertheless some important roots seem fallen into oblivion. Those being the cognitive ones, i.e. representational aspects.

Cognitive science studies the processes and representations underlying intelligent action. One particular question arising is

How can it be that a representation means something for the cognitive systems itself?

We believe that this question , although casted very philosophically, is of high relevance for neural computation. The integration of prior knowledge is the widely accepted "solution" to the bias/variance dilemma mentioned above. However,

this requires nothing less than to solve the representation problem — how to encode knowledge. If one wants to come up with a fairly general solution one has to tackle the previous question.

From the system design perspective this calls for an appropriate mathematical framework. Here we propose Clifford algebra which allows the processing of geometric entities like points, lines and so on. It is a very efficient language for solving many tasks connected to the design of intelligent systems [24, 74, 82]. To establish the theory of Clifford neural computation from the outlined motivation as a powerful model–based approach is the main goal of this thesis. We will start with the design of Clifford neurons for which weight association is interpretable as a geometric transformation. Then it is demonstrated how different operation modes of such neurons can be selected by different data representations. From the neuron level we then proceed to Clifford Multilayer Perceptrons.

1.2 Related Work

Technically speaking, a Clifford algebra is a generalization of complex numbers and quaternions. Neural networks in such domains are not new. The history of complex neural networks already started in 1990 with a paper by Clarke [18]. Soon this was followed by Leung and Haykin [54] presenting the complex Backpropagation algorithm. The most influential paper was published by Georgiou and Koutsougeras [32] in 1992. Therein the topic of suitable activation functions for Complex Multilayer Perceptrons was discussed for the first time. In particular, Georgiou and Koutsougeras proved a list of requirements that complex–valued activation functions have to fulfill in order to be applicable. Unfortunately, the complex version of the standard sigmoidal activation function mostly used in the real MLP was excluded by those requirements. Another rather trivial complex–valued activation function ($\frac{z}{1+|z|}$) was proposed, and, also the use of real–valued activation functions for Complex MLPs was suggested in this paper. In 1995 Arena et al. [2] proved the universal approximation property of a Complex MLP with real sigmoidal activation function. They also revealed drawbacks of the only known complex activation function proposed in [32]. Complex MLPs with real–valued activation became the standard notion of complex neural networks, and complex neural computation remained unattractive for most researchers.

Meanwhile Clifford neural networks had entered the stage. Pearson [62] introduced Clifford MLPs utilizing a Clifford version of the complex activation function

from [32]. The same function claimed to be useless by Arena et al. a year later. In 1997 Arena et al. introduced the Quaternionic MLP, a MLP formulated in terms of quaternions using real–valued activation functions again. The same year saw a paper of Nitta [61] on Complex MLPs which did not add anything new to the case. A new attempt to vitalize complex–valued activation functions was started recently by Kim and Adali [44].

Most of the above mentioned literature will be reviewed in this thesis. Clifford MLPs with both real–valued and Clifford–valued activation functions will be studied for algebras not considered before in the literature. Moreover, new propagation functions for Clifford MLPs will be presented. However, all this is not the main goal of this thesis, it results from it. As outlined in the previous section, this thesis tries to establish Clifford neural computation as a generic model–based approach to design neural networks capable of processing different geometric entities. In particular, data is not viewed as ,say, complex numbers, but as points in the plane. Consequently, a Complex MLP is viewed as transforming such data in some certain geometric way. That way we will have a new, different and unified look at those networks. In that sense, also work like [16, 29] can be seen as roughly related.

Closely related is the work of our colleague V. Banarer [5, 4, 64]. However, his work focuses on classification and practical applications.

1.3 Structure of the Thesis

After this introduction the thesis starts with an outline of Clifford algebra in chapter two. The material is presented in a self–contained way. Special emphasis is given to the geometric interpretation of the algebraical framework. In particular, the Clifford group is studied which acts as geometric transformation group on different geometric entities. The insights gained are then directly used for the design and motivation of Clifford neurons in the two subsequent chapters.

Chapter three introduces neurons based on one single geometric product, which are the atoms of all Clifford neural computation. Many illustrations for the model–based nature of Clifford neurons are worked out. A complete overview over the two–dimensional case in terms of algorithms and dynamics is given. The fundamental topic of isomorphic Clifford neurons and isomorphic representations is also covered in detail. Finally, a linear architecture utilizing a line representation for Euclidean transformations of the plane is presented.

The fourth chapter is devoted to Clifford neurons based on two geometric products that perform orthogonal transformations on arbitrary geometric entities by mimicking the operation of the Clifford group. Efficient learning algorithms for such neurons are derived. As a representative of that class of Clifford neurons the Quaternionic Spinor Neuron is studied in detail. Again, a discussion of isomorphic issues is provided. In the last section of the chapter an architecture linearizing the computation of Möbius transformations is introduced. For this architecture a conformal embedding of the data is utilized.

Based on the methodical and algorithmical foundations of chapter three and four the thesis proceeds with the study of Clifford Multilayer Perceptrons. The focus thereby is set to the topic of function approximation. According to the type of activation functions used the material is divided into two separate chapters.

Clifford Multilayer Perceptrons with real–valued activation functions are studied in chapter five. The chapter begins with reviewing the literature on the subject including the complex and quaternion case. The architecture is then generalized to arbitrary Clifford algebras and also extended to networks based on the new neurons developed in chapter four. Universal approximation is proved for all new derived networks with underlying Clifford algebras up to dimension four. The chapter concludes with an extensive section of experiments comparing the performance of the architectures known from the literature with the new developed ones.

Chapter six deals with Multilayer Perceptrons with Clifford–valued activation functions. In contrast to the architectures of chapter five real analysis is no longer sufficient for the mathematical treatment of such networks. The case of complex–valued activation functions is examined first using the available literature. Then the theory of hyperbolic–activation functions is developed. Analysis in higher dimensional Clifford algebras is still an ongoing field of mathematical research rather than an established theory. Therefore, the topic of general Clifford–valued activation function is only outlined.

Each of the chapters three to six cover one particular Clifford neural architecture. Therefore all of them are provided with an individual summary. The thesis concludes with chapter seven, which reviews the proposed methods and obtained results upon the whole. The benefits of the chosen approach, but also open problems and directions for further work are discussed.

Part of the work in this thesis has been presented in the following publications [11, 12, 13, 14, 15].

Chapter 2

Clifford Algebra

Clifford algebras are named after the British mathematician William K. Clifford. In the 1960s David Hestenes started to extend Clifford Algebra with geometric concepts. His broader mathematical system is nowadays known as Geometric Algebra, a term originally coined by Clifford himself.

This introductory chapter on Clifford Algebra has the following structure. Although the definition of an algebra is pretty much common knowledge, that will be exactly our entrance into the world of Clifford Algebra. This is simply due to the fact that there is no better way to understand what a (Clifford) algebra is all about. A vector space is endowed with an additional structure by introducing a product on it.

After reviewing some basic facts about algebras and rings we proceed by looking at complex numbers and quaternions as algebras. That way the generalization to Clifford algebras is prepared. The distinguished role of complex numbers and quaternions is well pointed out by recalling a famous theorem of Frobenius.

Then the main definition and theorems of Clifford algebras will be presented following mostly the books by Porteus [67] and Lounesto [56]. By doing so Clifford algebras will be constructed from quadratic spaces, and, hence, will have a metric structure right from the beginning[1]. The two–dimensional Clifford algebras will be studied in greater detail including the only degenerate algebra concerned in this thesis. In the last section of the chapter the Clifford group is introduced which

[1]There is and will always be an ongoing discussion in the Clifford (Geometric) Algebra community, if this is a good idea and how important metric aspects are for the very first foundation of Clifford (Geometric) Algebra [40]. For our approach, since being heavily based on the idea of transformation groups, they are mandatory.

will provide a first geometrical interpretation for many of the Clifford neural architectures developed later then.

The mathematical material is presented in a self–contained way. Since we believe that the theory of Clifford neural computation is in large parts an algebraic theory, algebraic aspects are the focus of this introduction. For a more geometrical introduction to Clifford (Geometric) Algebra we refer to the original work of Hestenes [39, 40].

2.1 Preliminaries

To begin at the beginning, let us start with the definition of a real algebra.

Definition 2.1 (Real Algebra) *A real algebra is a real linear space* $(\mathcal{A}, +, \cdot)$ *endowed with a bilinear product*

$$\otimes : \mathcal{A} \times \mathcal{A} \to \mathcal{A}, (a, b) \mapsto a \otimes b \,.$$

Hence, a real algebra is a pair $((\mathcal{A}, +, \cdot), \otimes)$.

Since we only consider real algebras throughout this thesis, we shall often speak loosely of algebras hereafter. Also, when there is no danger of confusion, we will just write ab instead of $a \otimes b$ in order to shorten expressions.

An algebra may, or may not, have the following additional properties.

Definition 2.2 (Types of Algebras) *An algebra* $((\mathcal{A}, +, \cdot), \otimes)$ *is called*

(i) *associative, if for all* $a, b, c \in \mathcal{A} : (a \otimes b) \otimes c = a \otimes (b \otimes c)$,

(ii) *commutative, if for all* $a, b \in \mathcal{A} : a \otimes b = b \otimes a$,

(iii) *an algebra with identity, if there exists* $1 \in \mathcal{A}$ *such that for all* $a \in \mathcal{A}$: $1 \otimes a = a \otimes 1 = 1$.

Note that all of the properties listed above are independent of each other.

The real numbers considered as algebra $((\mathbb{R}, +, \cdot), \cdot)$, for example, do comprise all the attributes of Definition 2.2.

The bilinearity of the product of an algebra has two important consequences. The first one will be used frequently in this chapter.

Proposition 2.3 *For any algebra $((\mathcal{A}, +, \cdot), \otimes)$, the product $\otimes$ is already uniquely determined given only the products for an arbitrary basis of $\mathcal{A}$.*

The second one relates algebras to another well known algebraic concept.

Proposition 2.4 *Any algebra $((\mathcal{A}, +, \cdot), \otimes)$ is distributive by definition, or, equivalently, $(\mathcal{A}, +, \otimes)$ is always a ring.*

Thus, known results from ring theory are also applicable to algebras.

Proposition 2.5 *Two finite dimensional algebras $\mathcal{A}$ and $\mathcal{B}$ are isomorphic, written as $\mathcal{A} \cong \mathcal{B}$, if they are isomorphic as rings, that is if there exists a bijective mapping $\phi : \mathcal{A} \to \mathcal{B}$ such that, for all $a, b \in \mathcal{A}$*

(i) $\phi(a + b) = \phi(a) + \phi(b)$,

(ii) $\phi(a \otimes b) = \phi(a) \otimes \phi(b)$.

Also, a tensor product for algebras can be easily established.

Definition 2.6 (Tensor Product) *Let $\mathcal{A}$ be a finite dimensional real associative algebra with identity. If there exist subalgebras $\mathcal{B}$ and $\mathcal{C}$ of $\mathcal{A}$ such that*

(i) *for any $b \in \mathcal{B}, c \in \mathcal{C}, bc = cb$,*

(ii) *$\mathcal{A}$ is generated as an algebra by $\mathcal{B}$ and $\mathcal{C}$,*

(iii) *dim $\mathcal{A}$ = dim $\mathcal{B}$ dim $\mathcal{C}$*

then $\mathcal{A}$ is said to be the tensor product of $\mathcal{B}$ and $\mathcal{C}$, written as $\mathcal{B} \otimes \mathcal{C}$.

Two special types of rings are introduced in the next definition.

Definition 2.7 (Field) *Let $(\mathcal{R}, +, \otimes)$ be a ring. If $(\mathcal{R} \setminus \{0\}, \otimes)$ is a (non–)commutative group, then $(\mathcal{R}, +, \otimes)$ is called a (skew) field.*

Let us now study how complex numbers and quaternions fit into the algebraic concepts developed so far. Nowadays complex numbers are mostly introduced in the following way.

Definition 2.8 (The Field of Complex Numbers) *Consider the set of all ordered pairs of real numbers*

$$\mathbb{R}^2 = \{z = (a, b) \mid a, b \in \mathbb{R}\}$$

together with addition and multiplication defined for all $z_1 = (a_1, b_1), z_2 = (a_2, b_2) \in \mathbb{R}^2$ *as*

$$z_1 + z_2 = (a_1 + a_2, b_1 + b_2) \tag{2.1}$$

$$z_1 \otimes z_2 = (a_1a_2 - b_1b_2, a_1b_2 + a_2b_1)\,. \tag{2.2}$$

Then $\mathbb{C} := (\mathbb{R}^2, +, \otimes)$ *is called the field of complex numbers.*

The above modern definition is free of any myth regarding the nature of complex numbers. In particular, the imaginary unit i is obtained by setting $i := (0, 1)$. The law $i^2 = -1$ then is just a direct consequence of (2.2). Furthermore, the usual notion of a complex number $z = a + ib$ is easily obtained from the identity

$$z = (a, b) = (a, 0) + (0, 1) \otimes (b, 0)\,. \tag{2.3}$$

It is also easy to check that $\mathbb{C}$ is indeed a field. Obviously, the multiplication of complex numbers is both associative and commutative. Moreover, for all complex numbers $z = (a, b) \in \mathbb{C} \setminus (0, 0)$ the following holds

$$(a, b) \otimes (a/(a^2 + b^2), b/(a^2 + b^2)) = (1, 0)\,. \tag{2.4}$$

The complex numbers $\mathbb{C}$, although mostly viewed as a field, comprise yet another algebraic structure. $\mathbb{C}$ contains infinitely many subfields isomorphic to $\mathbb{R}$. Choosing one also defines a real linear structure on $\mathbb{C}$. The obvious choice for that distinguished copy of $\mathbb{R}$ in $\mathbb{C}$ is given by the map

$$\alpha : \mathbb{R} \to \mathbb{C}, a \mapsto (a, 0)\,. \tag{2.5}$$

For any $\lambda \in \mathbb{R}$ and any $z = (a, b) \in \mathbb{C}$ we then get

$$\alpha(\lambda) \otimes (a, b) = (\lambda, 0) \otimes (a, b) = (\lambda a, \lambda b)\,, \tag{2.6}$$

which turns $\mathbb{C}$ also into a real algebra. More precisely, $\mathbb{C}$ thereby becomes a real associative and commutative algebra of dimension 2 with $(1, 0)$ as identity element.

The geometric view of complex numbers as points in the complex plane actually depends only on that real linear structure [56]. Hamilton, the famous Irish mathematician, was well aware of that fact. He therefore used (2.6) to motivate the multiplication rule (2.2) in his construction of complex numbers [27]. We now proceed to study his most famous invention — the quaternions [33].

Definition 2.9 (The Algebra of Quaternions) *Consider the linear space* $(\mathbb{R}^4, +, \cdot)$ *with standard basis* $\{1 := (1_{\mathbb{R}}, 0, 0, 0), i := (0, 1_{\mathbb{R}}, 0, 0), j := (0, 0, 1_{\mathbb{R}}, 0), k := (0, 0, 0, 1_{\mathbb{R}})\}$ *and define a multiplication* $\otimes$ *on it according to the following table*

$\otimes$	1	i	j	k
1	1	i	j	k
i	i	-1	k	-j
j	j	-k	-1	i
k	k	j	-i	-1

.

Then $\mathbb{H} := ((\mathbb{R}^4, +, \cdot), \otimes)$ *is a real associative algebra of dimension 4, which is called the algebra of quaternions. Obviously,* 1 *is the identity element of* $\mathbb{H}$*.*

In honor of Hamilton, quaternions are sometimes also called Hamiltonian numbers in the literature. Any quaternion $q \in \mathbb{H}$ can be written in the form

$$q = q_0 + iq_1 + jq_2 + kq_3 \tag{2.7}$$

with $q, q_1, q_2, q_3 \in \mathbb{R}$. Analogously as in $\mathbb{C}$, the basis vectors $\{i, j, k\}$ are often named imaginary units. In particular, the following relations hold among them

$$jk = -kj = i \quad ki = -ik = j \quad ij = -ji = k\,, \tag{2.8}$$

which shows that multiplication in $\mathbb{H}$ is not commutative. Moreover, it can be concluded from (2.8) that there exists no other real linear structure in $\mathbb{H}$ than the one already introduced in Definition 2.9.

A quaternion q is often split into its so–called scalar part q_0, and its so–called vector part $\vec{q} = iq_1 + jq_2 + kq_3$. A quaternion whose scalar part is zero is called a pure quaternion.

For the multiplication of two general quaternions we obtain

$$\begin{aligned}(q_0 + iq_1 + jq_2 + kq_3) \otimes (r_0 + ir_1 + jr_2 + kr_3) &= (q_0r_0 - q_1r_1 - q_2r_2 - q_3r_3) \\ &+ i(q_0r_1 + q_1r_0 + q_2r_3 - q_3r_2) \\ &+ j(q_0r_2 + q_2r_0 - q_1r_3 + q_3r_1) \\ &+ k(q_0r_3 + q_3r_0 + q_1r_2 - q_2r_1)\,.\end{aligned} \tag{2.9}$$

The multiplication of quaternions is associative, which can be checked directly. However, we just refer to [27] for a proof. The multiplicative inverse of any non–zero quaternion $q = q_0 + iq_1 + jq_2 + kq_3$ is given by

$$q^{-1} = (q_0 - iq_1 - jq_2 - kq_3)/(q_0^2 + q_1^2 + q_2^2 + q_3^2)\,, \tag{2.10}$$

which is much easier to verify. Thus, $(\mathbb{H}, +, \otimes)$ is a skew field.

In both $\mathbb{C}$ and $\mathbb{H}$ a division is defined. If this holds for a general algebra one speaks of a division algebra.

Definition 2.10 (Division Algebra) *An algebra $\mathcal{A}$ is called a division algebra if, for all $a \in \mathcal{A} \setminus \{0\}$, both of the following two equations*

$$ax = b \tag{2.11a}$$
$$ya = b \tag{2.11b}$$

are uniquely solvable for all $b \in \mathcal{A}$.

Any division algebra is free of zero divisors.

Definition 2.11 (Zero Divisor) *Let $\mathcal{A}$ be an algebra. An element $a \in \mathcal{A}$ is called zero divisor, if there exists $b \in \mathcal{A}\setminus\{0\}$ such that*

$$ab = 0 \;\; or \;\; ba = 0. \tag{2.12}$$

According to a famous theorem of Frobenius, finite dimensional division algebras are quite exceptional.

Theorem 2.12 (Frobenius)
Any finite dimensional real associative division algebra is isomorphic to either $\mathbb{R}$, $\mathbb{C}$, or $\mathbb{H}$.

Even if associativity is dropped only one further algebra, the so-called octonions [27], would be obtained. The following result holds for any finite dimensional algebra.

Proposition 2.13 *Let $\mathcal{A}$ be a finite–dimensional algebra. Then $\mathcal{A}$ is a division algebra, if and only if, $\mathcal{A}$ is free of zero divisors.*

As a consequence, any Clifford algebra contains zero divisors in general.

2.2 Main Definitions and Theorems

There are many geometric concepts that are not covered by the structure of a linear space alone. These are all metric concepts such as , for example, distance and angle. Equipping a linear space with such an additional metric structure, however, leads naturally to a new specific algebra. This algebra, being itself a linear space, then has to be an embedding of larger dimension of the original linear space. Also, it has to comprise, somehow, the metric structure within its multiplication.

What we outlined above is exactly the notion behind Clifford algebras. We shall now turn it, step by step, into a formal concept. The first step is to introduce some metric structure, from which the algebras are then constructed.

Definition 2.14 (Quadratic Form) *Let X be a real linear space endowed with a scalar product, i.e. with a symmetric bilinear form,*

$$F : X \times X \to \mathbb{R}, (a, b) \mapsto a \cdot b\,.$$

Then the map

$$Q : X \to \mathbb{R}, a \mapsto a \cdot a$$

is called the quadratic form of F. Furthermore, the pair (X, Q) is called a real quadratic space.

Note that Q is uniquely determined by F, and vice versa, in virtue of

$$F(a, b) = \frac{1}{2} \cdot (Q(a + b) - Q(a) - Q(b))\,. \tag{2.13}$$

Thus, one can arbitrarily switch between Q and F. Any finite dimensional real quadratic space does possess a distinguished basis.

Proposition 2.1 *Let (X, Q) be an n–dimensional real quadratic space. Then there exists a basis $\{e_1, ..., e_n\}$ of (X, Q) and uniquely determined $p, q, r \in \{0, \ldots, n\}$ such that, for all $i, j \in \{1, \ldots, n\}$, the following two conditions are fulfilled*

(i) $$Q(e_i) = \begin{cases} 1, & i \leq p \\ -1, & p + 1 \leq i \leq p + q \\ 0, & p + q + 1 \leq i \leq p + q + r = n, \end{cases}$$

(ii) $Q(e_i + e_j) - Q(e_i) - Q(e_j) = 0\,.$

A basis with the above properties is called an orthonormal basis of (X, Q), and the triple (p, q, r) is called the signature of (X, Q).

Quadratic spaces are further distinguished by the value of r.

Definition 2.15 (Degenerate Space) *Let (X, Q) be a finite dimensional real quadratic space. Then (X, Q) is said to be a degenerate space if*

$$\{a \in X \mid Q(a) = 0\} \neq \emptyset\,, \tag{2.14}$$

and to be a non–degenerate space otherwise.

Clifford algebras inherit their metric properties from quadratic spaces.

Main Definitions and Theorems

The most general definition of a Clifford algebra is as follows.

Definition 2.16 (Clifford Algebra [67]) *Let (X, Q) be an arbitrary finite dimensional real quadratic space and let $\mathcal{A}$ be a real associative algebra with identity. Furthermore, let $\alpha : \mathbb{R} \to \mathcal{A}$ and $\nu : X \to \mathcal{A}$ be linear injections such that*

(i) *$\mathcal{A}$ is generated as an algebra by its distinct subspaces*
$\{\nu(v) \mid v \in X\}$ *and* $\{\alpha(a) \mid a \in \mathbb{R}\}$,

(ii) $\forall v \in X : (\nu(v))^2 = \alpha(Q(v))$.

Then $\mathcal{A}$ is said to be a Clifford algebra for (X, Q). The elements of a Clifford algebra are called multivectors. The product of a Clifford algebra is named geometric product. The signature of the quadratic space is also the signature of the algebra.

The mappings α and ν embed the reals (likewise as in (2.5)) and the quadratic space, respectively. Usually, one simply identifies $\mathbb{R}$ and Q with their corresponding copies in $\mathcal{A}$. We shall do the same from now on. The name multivector for the elements of a Clifford algebra will be explained soon. As indicated by the name geometric product, every Clifford algebra models a certain geometry [38, 39, 40]. For example, the algebras of signature $(n, 0)$ model Euclidean spaces.

Since any Clifford algebra is a real associative algebra by definition, the following important theorem holds.

Theorem 2.17 *Any Clifford algebra is isomorphic to some matrix algebra.*

From condition (ii) of Definition 2.16 further results follow.

Proposition 2.18 *Let (X, Q) be an n–dimensional real quadratic space with an orthonormal basis $\{e_i \mid 1 \leq i < n\}$. Furthermore, let $\mathcal{A}$ be a real associative algebra with identity containing $\mathbb{R}$ and X as distinct linear subspaces. Then $x^2 = Q(x)$, for all $x \in X$, if and only if*

$$e_i^2 = Q(e_i) \qquad \forall i \in \{1, \ldots, n\} \tag{2.15}$$

$$e_i e_j + e_j e_i = 0 \qquad \forall i \neq j \in \{1, \ldots, n\}\,. \tag{2.16}$$

Equation (2.16) allows directly to draw the following conclusion.

Proposition 2.19 *Any commutative Clifford algebra is of dimension ≤ 2.*

The steps necessary to derive the following statement from Proposition 2.18 can be found in [67].

Proposition 2.20 *Let $\mathcal{A}$ be a Clifford algebra for an n–dimensional quadratic space X. Then $\dim \mathcal{A} \leq 2^n$.*

The special role of Clifford algebras of dimension 2^n is highlighted by the next definition.

Definition 2.21 (Universal Clifford Algebra) *A Clifford algebra of dimension 2^n is called universal.*

Let us next introduce those spaces deriving from the standard scalar products.

Definition 2.22 (Standard Quadratic Space) *For any $p, q, r \in \{0, \ldots, n\}$ define the following scalar product*

$$F : \mathbb{R}^{p+q+r} \times \mathbb{R}^{p+q+r} \to \mathbb{R}, (a, b) \mapsto \sum_{i=1}^{p} a_i b_i - \sum_{p+1}^{q} a_i b_i\,.$$

Then the corresponding standard quadratic space $(\mathbb{R}^{p+q+r}, Q)$ is denoted by $\mathbb{R}^{p,q,r}$.

Note that any real quadratic space is isomorphic to some space $\mathbb{R}^{p,q,r}$. In that sense the next theorem gives all real universal Clifford algebras.

Theorem 2.23 *For any quadratic space $\mathbb{R}^{p,q,r}$ there exists a unique universal Clifford algebra. This algebra shall be denoted by $\mathcal{C}_{p,q,r}$, and its geometric product by $\otimes_{p,q,r}$.*

A proof of Theorem 2.23 can be found in many sources (see e.g. [40, 56, 67]). The basic idea is to proof first, that, for each n, the full matrix algebra $\mathbb{R}(2^n)$ of all $2^n \times 2^n$ matrices with real entries is a unique universal Clifford algebra for the spaces $\mathbb{R}^{n,n,0}$. Then this result is extended to quadratic spaces of arbitrary signature by forming tensor products with appropriate matrices.

This indicates that both matrix isomorphisms and tensor products play a crucial role in the theory of Clifford algebras. We shall return to both concepts later on.

All what follows deals with Clifford algebras $\mathcal{C}_{p,q,r}$. Since the algebras $\mathcal{C}_{p,q,r}$ are universal by definition, we will drop that attribute from now on. Also, if $r = 0$ the shorter notations $\mathcal{C}_{p,q}$ and $\otimes_{p,q}$ will be used.

Subspace Grading

Our next goal is to construct a basis of $\mathcal{C}_{p,q,r}$. For that purpose let us introduce the following notation. Let

$$\mathcal{I} := \{\{i_1, \ldots, i_s\} \in \mathcal{P}(\{1, \ldots, n\}) \mid 1 \leq i_1 \leq \ldots \leq i_s \leq n\} \tag{2.17}$$

denote the set of all naturally ordered subsets of the power set $\mathcal{P}(\{1, \cdots, n\})$. Furthermore, let $(e_1, \cdots, e_n)$ be an orthogonal basis of $\mathbb{R}^{p,q,r}$. Now define, for all $I \in \mathcal{I}$, e_I to be the product

$$e_I := e_{i_1} \ldots e_{i_s} . \tag{2.18}$$

In particular, let $e_\emptyset$ be identified with 1.

In case of $n = 3$, for example, we get the following set of vectors from (2.18)

$$\{1, e_{\{1\}}, e_{\{2\}}, e_{\{3\}}, e_{\{1,2\}}, e_{\{1,3\}}, e_{\{2,3\}}, e_{\{1,2,3\}}\} .$$

It is convenient to write just e_{12} and so on, if there is no danger of confusion. By construction, the generated set of vectors by the above method is always of dimension 2^n. Since all of the vectors are also linearly independent of each other, we have already achieved our goal[2].

Proposition 2.24 *Let $(e_1, \cdots, e_n)$ be an orthogonal basis of $\mathbb{R}^{p,q,r}$. Then,*

$$\{e_I \mid I \in \mathcal{I}\}, \tag{2.19}$$

[2]Note that our construction of a basis slightly differs from the common one involving the outer product. However, it greatly simplifies many notations.

with $\mathcal{I}$ defined according to (2.17)*, is a basis of the Clifford algebra $\mathcal{C}_{p,q,r}$. In particular, any multivector in $\mathcal{C}_{p,q,r}$ can be written as*

$$x = \sum_{I \in \mathcal{I}} x_I e_I \,, \tag{2.20}$$

whereby all $x_I \in \mathbb{R}$.

From this follows that any Clifford algebra has an inherent subspace grading.

Definition 2.25 (K–Vector Part) *Let $(e_1, \cdots, e_n)$ be an orthogonal basis of $\mathbb{R}^{p,q,r}$. For every $k \in \{0, \cdots, n\}$ the set*

$$\{e_I \mid e_I \in \mathcal{I}, |I| = k\} \tag{2.21}$$

spans a linear subspace of $\mathcal{C}_{p,q,r}$. This linear subspace, denoted by $\mathcal{C}^k_{p,q,r}$, is called the k–vector part of $\mathcal{C}_{p,q,r}$.

From the above definition we get

$$\mathcal{C}_{p,q,r} = \sum_{k=0}^{n} \mathcal{C}^k_{p,q,r} \,. \tag{2.22}$$

We also obtain, in particular, $\mathcal{C}^0_{p,q,r} = \mathbb{R}$ and $\mathcal{C}^1_{p,q,r} = \mathbb{R}^{p,q,r}$. Any subspace $\mathcal{C}^k_{p,q,r}$ is of dimension $\binom{n}{k}$. Thus, a Clifford algebra can also be seen as an algebra of subspaces. All subspaces of even dimension form a subalgebra of $\mathcal{C}_{p,q,r}$.

Proposition 2.26 *The subspace sum given by*

$$\mathcal{C}^+_{p,q,r} := \sum_{\substack{k=0, \\ k \text{ even}}}^{n} \mathcal{C}^k_{p,q,r} \tag{2.23}$$

is a subalgebra of $\mathcal{C}_{p,q,r}$ of dimension 2^{n-1}, which is called the even subalgebra of $\mathcal{C}_{p,q,r}$.

The grading induced by (2.23) is a $\mathbb{Z}_2$ grading, whereas the whole subspace grading (2.22) is sometimes referred to as $\mathbb{Z}_n$ grading [56]. The subspace grading also induces the following operator.

Definition 2.27 (Grade Operator) *Define the following grade operator*

$$< \cdot >_k \colon \mathcal{C}_{p,q,r} \to \mathcal{C}_{p,q,r}, x \mapsto \sum_{\substack{I \in \mathcal{I}, \\ |I| = k}} x_I e_I \,,$$

for any $k \in \{0, \cdots, n\}$.

Using the grade operator any multivector can be written as

$$x = \sum_{k=0}^{n} < x >_k \ . \tag{2.24}$$

A multivector fulfilling $x=< x >_k$ is called a homogeneous multivector of grade k, or simply a k–vector. Common names for the lowest grade k–vectors are scalar, vector, bivector, and trivector (in ascending order). Thus, the name multivector for the elements of a Clifford algebra is pretty much true.

The grade operator allows a very intuitive definition of the most fundamental automorphisms for non–degenerate Clifford algebras $\mathcal{C}_{p,q}$, for which the next definition will serve as preparation.

Definition 2.28 (Involution) *An algebra automorphism f is said to be an involution if $f^2 = \mathrm{id}$.*

All automorphisms covered by the following definition are involutions.

Definition 2.29 (Main Involutions) *For any Clifford algebra $\mathcal{C}_{p,q}$ define three distinguished maps as follows*

(i) *Inversion* $\quad \tilde{}: \mathcal{C}_{p,q} \to \mathcal{C}_{p,q}, x \mapsto \sum_{k=0}^{n} (-1)^k < x >_k$,

(ii) *Reversion* $\quad \hat{}: \mathcal{C}_{p,q} \to \mathcal{C}_{p,q}, x \mapsto \sum_{k=0}^{n} (-1)^{\frac{k(k-1)}{2}} < x >_k$,

(iii) *Conjugation* $\quad \bar{}: \mathcal{C}_{p,q} \to \mathcal{C}_{p,q}, x \mapsto \sum_{k=0}^{n} (-1)^{\frac{k(k+1)}{2}} < x >_k \ .$

2.3 Isomorphisms

In Definition 2.5 the concept of isomorphic algebras was introduced. From Theorem 2.17 we know that any Clifford algebra is isomorphic to some matrix algebra. It is quite easy to check that the only one–dimensional Clifford algebra $\mathcal{C}_{0,0,0}$ is isomorphic to the reals considered as algebra $((\mathbb{R}, +, \cdot), \cdot)$. Actually, both algebraic objects are identical. In particular, the geometric product of $\mathcal{C}_{0,0,0}$ is the ordinary multiplication of $\mathbb{R}$.

There are three possible signatures for an one–dimensional quadratic space. The multiplication tables of the corresponding commutative (Proposition 2.19) two–dimensional Clifford algebras are given in table 2.1. The multiplication tables follow directly from Definition 2.16.

$\otimes_{1,0,0}$	1	e_1
1	1	e_1
e_1	e_1	1

$\otimes_{0,1,0}$	1	e_1
1	1	e_1
e_1	e_1	-1

$\otimes_{0,0,1}$	1	e_1
1	1	e_1
e_1	e_1	0

Table 2.1: Multiplication tables for the Clifford algebras $\mathcal{C}_{1,0,0}$ (left), $\mathcal{C}_{0,1,0}$ (middle) and $\mathcal{C}_{0,0,1}$ (right).

Obviously, the algebra $\mathcal{C}_{0,1,0}$ is isomorphic to the complex numbers $\mathbb{C}$ (identify e_1 with the complex imaginary unit i). The algebras $\mathcal{C}_{1,0,0}$ and $\mathcal{C}_{0,0,1}$ are isomorphic to the so–called hyperbolic numbers and dual numbers, respectively (see [93] for a detailed treatment of both number systems). Theorem 2.12 implies the existence of zero divisors for both algebras.

The geometric product of $\mathcal{C}_{1,0,0}$ reads

$$(x_0 + x_1e_1) \otimes_{1,0,0} (y_0 + y_1e_1) = (x_0y_0 + x_1y_1) + (x_0y_1 + x_0y_1)e_1\,, \tag{2.25}$$

which yields

$$\{x_0 + x_1e_1 \mid |x_0| = |x_1|\} \tag{2.26}$$

as the set of zero divisors.

The degenerate algebra $\mathcal{C}_{0,0,1}$ possess the geometric product

$$(x_0 + x_1e_1) \otimes_{0,0,1} (y_0 + y_1e_1) = (x_0y_0) + (x_0y_1 + x_1y_0)e_1\,, \tag{2.27}$$

which results in the following set of zero divisors

$$\{x_0 + x_1e_1 \mid\ x_1 = 0\}\,. \tag{2.28}$$

All two–dimensional Clifford algebras have very simple matrix representations.

Proposition 2.30 *The following isomorphism hold*

$$\mathcal{C}_{0,1,0} \cong \left\{ \begin{pmatrix} a & -b \\ b & a \end{pmatrix} \mid a, b \in \mathbb{R} \right\} \tag{2.29}$$

$$\mathcal{C}_{1,0,0} \cong \left\{ \begin{pmatrix} a & b \\ b & a \end{pmatrix} \mid a, b \in \mathbb{R} \right\} \tag{2.30}$$

$$\mathcal{C}_{0,0,1} \cong \left\{ \begin{pmatrix} a & 0 \\ b & a \end{pmatrix} \mid a, b \in \mathbb{R} \right\}. \tag{2.31}$$

The algebra $\mathcal{C}_{0,0,1}$ will be the only degenerate Clifford algebra considered in this work[3]. In the remainder of this section we proceed with the study of non-degenerate Clifford algebras, for which the following theorem is of fundamental importance.

Theorem 2.31 ([67]) *Any Clifford algebra $\mathcal{C}_{p,q}$ is isomorphic to a full matrix algebra over $\mathbb{K}$ or ${}^2\mathbb{K} := \mathbb{K} \oplus \mathbb{K}$, where $\mathbb{K} := \{\mathbb{R}, \mathbb{C}, \mathbb{H}\}$.*

An overview of the Clifford algebras $\mathcal{C}_{p,q}$ for $p + q \leq 8$ is given in table 2.2 below.

$p \backslash q$	0	1	2	3	4
0	$\mathbb{R}$	$\mathbb{C}$	$\mathbb{H}$	${}^2\mathbb{H}$	$\mathbb{H}(2)$
1	${}^2\mathbb{R}$	$\mathbb{R}(2)$	$\mathbb{C}(2)$	$\mathbb{H}(2)$	${}^2\mathbb{H}(2)$
2	$\mathbb{R}(2)$	${}^2\mathbb{R}(2)$	$\mathbb{R}(4)$	$\mathbb{C}(4)$	$\mathbb{H}(4)$
3	$\mathbb{C}(2)$	$\mathbb{R}(4)$	${}^2\mathbb{R}(4)$	$\mathbb{R}(8)$	$\mathbb{C}(8)$
4	$\mathbb{H}(2)$	$\mathbb{C}(4)$	$\mathbb{R}(8)$	${}^2\mathbb{R}(8)$	$\mathbb{R}(16)$

Table 2.2: The Clifford algebras $\mathcal{C}_{p,q}$ for $p + q \leq 8$. Redrawn from [67].

In particular, the hyperbolic numbers $\mathcal{C}_{1,0,0}$ are isomorphic to the double field ${}^2\mathbb{R}$, which is the algebra $((\mathbb{R} \oplus \mathbb{R}, +, \cdot), \otimes)$ with ordinary real componentwise multiplication $(a, b) \otimes (c, d) = (a, c) + (b, d)$. This isomorphism is more of algebraical importance, whereas the isomorphism (2.30) is more useful in practice.

Another result from table 2.2 is $\mathcal{C}_{0,2} \cong \mathbb{H}$.

Also, there are many isomorphic Clifford algebras. For example, $\mathcal{C}_{2,0} \cong \mathcal{C}_{1,1}$, which generalizes as follows.

Proposition 2.32 *The Clifford algebras $\mathcal{C}_{p+1,q}$ and $\mathcal{C}_{q+1,p}$ are isomorphic.*

Many other isomorphisms are utilizing tensor products[4]. The following one will be used frequently in this thesis.

Proposition 2.33 *The following relation holds for Clifford algebras*

$$\mathcal{C}_{p+1,q+1} \cong \mathcal{C}_{p,q} \otimes \mathcal{C}_{1,1} \,. \tag{2.32}$$

[3]Degenerate Clifford algebras are of little interest from the pure algebraical point of view. Any degenerate Clifford algebra can be embedded in a larger non-degenerate one [40].

[4]Among them is the famous periodicity theorem $\mathcal{C}_{p,q+8} \cong \mathcal{C}_{p,q} \otimes \mathbb{R}(16)$.

2.4 The Clifford Group

At the very beginning of this chapter we studied complex numbers and quaternions. That way we got a first algebraic intuition of Clifford algebras. A similar approach is taken again in this section. In the following, however, we are interested in getting geometrical insights about Clifford algebras.

Any complex number z can be represented in polar coordinates as

$$z = r(\cos\phi + i\sin\phi)\,, \tag{2.33}$$

with $r \geq 0$ and $0 \leq \phi < 2\pi$. If $z \neq 0$ then (2.33) is a unique representation. The multiplication of two complex numbers in polar coordinates reads

$$z_1 z_2 = r_1 r_2(\cos(\phi_1 + \phi 2) + i\sin(\phi_1 + \phi_2))\,. \tag{2.34}$$

Thus, a unit complex number ($r = \bar{z}z = 1$) represents an Euclidean rotation in the plane. This can also be concluded from applying the isomorphism (2.29) to (2.33), which yields the well known rotation matrix

$$\begin{pmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{pmatrix}. \tag{2.35}$$

The set of unit complex numbers $S^1 := \{z \in \mathbb{C} \mid \bar{z}z = 1\}$ forms the so-called circle group, and all of the above sums up to

$$S^1 \cong SO(2)\,. \tag{2.36}$$

The following definition introduces a very important notion.

Definition 2.34 (Action of a Group) *Let G be a group and M be a non–empty set. The map*

$$\star : G \times M \to M;\quad (a, x) \mapsto a \star x \tag{2.37}$$

is called the action of G on M, if $1_G \star x = x$ and $a \star (b \star x) = (a \star b) \star x$ for all $x \in M,\ a, b \in G$.

Technically, S^1 acts on $\mathbb{C}$ and $SO(2)$ acts on $\mathbb{R}^2$. Geometrically, however, these are identical actions of the same transformation group.

The following connection between the group $SO(3)$ of 3D Euclidean rotations and quaternions is well known too.

Proposition 2.35 *Let* q *be a unit quaternion,* r *be a pure quaternion. Then*

$$qrq^{-1} \tag{2.38}$$

describes a 3D Euclidean rotation of the point r*.*

Both aforementioned groups have a Clifford counterpart.

Definition 2.36 (Clifford Group) *The Clifford group* $\Gamma_{p,q}$ *of a Clifford algebra* $\mathcal{C}_{p,q}$ *is defined as*

$$\Gamma_{p,q} := \{s \in \mathcal{C}_{p,q} \mid \forall x \in \mathbb{R}^{p,q} : \; sx\hat{s}^{-1} \in \mathbb{R}^{p,q}\}\,. \tag{2.39}$$

This actually looks already quite similar to (2.38). Note that the elements of $\Gamma_{p,q}$ are linear transformations by definition. More precisely, they act on the underlying quadratic space $\mathbb{R}^{p,q}$ by virtue of

$$\Gamma_{p,q} \times \mathbb{R}^{p,q} \to \mathbb{R}^{p,q}, (s, x) \mapsto sx\hat{s}^{-1}\,. \tag{2.40}$$

Since consisting of linear transformations, any Clifford group has to be isomorphic to some classical group. In fact, the following theorem holds.

Theorem 2.37 ([67]) *Consider for all* $s \in \Gamma_{p,q}$ *the following map*

$$\omega_s : \mathbb{R}^{p,q} \to \mathbb{R}^{p,q}, x \mapsto sx\hat{s}^{-1}\,. \tag{2.41}$$

Then the map $\Omega : \Gamma_{p,q} \to O(p, q), s \mapsto \omega_s$ *is a group epimorphism.*

It is not hard to verify that ω_s is an orthogonal automorphism of $\mathbb{R}^{p,q}$. Since the kernel of Ω is $\mathbb{R} \setminus \{0\}$, it then follows that $\Gamma_{p,q}$ is indeed a multiple cover of the orthogonal group $O(p, q)$.

Normalizing the Clifford group $\Gamma_{p,q}$ yields

$$\mathrm{Pin}(p, q) = \{s \in \Gamma_{p,q} \mid s\tilde{s} = \pm 1\}\,. \tag{2.42}$$

The group $\mathrm{Pin}(p, q)$ is a two–fold covering of the orthogonal group $O(p, q)$. Further subgroups of $\mathrm{Pin}(p, q)$ are

$$\mathrm{Spin}(p, q) = \mathrm{Pin}(p, q) \cap \mathcal{C}^+_{p,q} \tag{2.43}$$

and (in case of $p \neq 0$ and $q \neq 0$)

$$\mathrm{Spin}^+(p, q) = \{s \in \mathrm{Spin}(p, q) \mid s\tilde{s} = 1\}\,. \tag{2.44}$$

Both groups are again two–fold covers of their classical counterparts. The whole situation can be summarized as

$$\begin{aligned} \mathrm{Pin}(p,q)\backslash\{\pm 1\} &\cong \mathrm{O}(p,q) && \text{(2.45a)} \\ \mathrm{Spin}(p,q)\backslash\{\pm 1\} &\cong \mathrm{SO}(p,q) && \text{(2.45b)} \\ \mathrm{Spin}^{+}(p,q)\backslash\{\pm 1\} &\cong \mathrm{SO}^{+}(p,q)\,. && \text{(2.45c)} \end{aligned}$$

Thereby $\mathrm{SO}^{+}(p,q)$ is formed by those elements which are connected with the identity. This does not carry over to the covering groups, i.e. $\mathrm{Spin}^{+}(p,q)$ does not have to be connected [56].

The elements of the group $\mathrm{Spin}(p,q)$ are named spinors. Spinors also operate on so–called paravectors.

Proposition 2.38 ([56]) *Define for any Clifford algebra $\mathcal{C}_{p,q}$ the space of paravectors as*

$$\lambda\mathbb{R}^{p,q} := \sum_{k=0}^{1} \mathcal{C}^{k}_{p,q}. \tag{2.46}$$

Then

$$\{s \in \mathcal{C}_{p,q} \mid \forall x \in \lambda\mathbb{R}^{p,q} :\ sx\hat{s}^{-1} \in \lambda\mathbb{R}^{p,q}\} \tag{2.47}$$

is a group isomorphic to $\mathrm{Spin}(p,q)$.

For complex numbers one has $\mathcal{C}_{0,1} = \lambda\mathbb{R}^{0,1}$ and (2.47) reduces to $sx\hat{s}^{-1} = xs\hat{s}^{-1} =: xs'$ in accordance to what was stated in the beginning of this section.

Chapter 3

Basic Clifford Neurons

Neurons are the atoms of neural computation. Out of those simple computational units all neural networks are build up. An illustration of a generic neuron is given in figure 3.1. The output computed by a generic neuron reads

$$y = g(f(w, x)) \,. \tag{3.1}$$

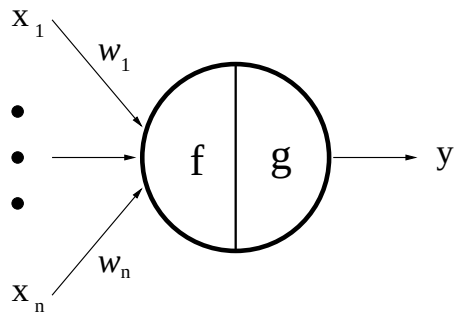

Figure 3.1: Model of a generic neuron.

The details of computation are as follows. In a first step the input to the neuron, $x := \{x_i\}$, is associated with the weights of the neuron, $w := \{w_i\}$, by invoking the so–called propagation function f. This can be thought as computing the activation potential from the pre–synaptic activities. Then from that result the so-called activation function g computes the output of the neuron. The weights, which mimic synaptic strength, constitute the adjustable internal parameters of the neuron. The process of adapting the weights is named learning. In accordance to the thesis introduction only supervised learning will be considered here.

Supervised learning is often characterized as learning with a "teacher". This means

that full feedback on the performance is provided throughout the learning process and learning itself is goal–driven. Technically, "the teacher" is realized in the following way. A training set $\{(x^p, d^p)\}_{p=1}^P$ consisting of correct input–output patterns is provided. Performance is measured by evaluating some given error function E. The most common error function is

$$E := \frac{1}{2P} \sum_{p=1}^{P} \| d^p - y^p \|_2 , \tag{3.2}$$

which measures the derivation between desired and actual output in the mean–squared sense. Learning is then formulated as finding the optimal weights that minimize E on the given training set. The actual goal of learning, however, is generalization. That is to perform well on unseen data. Generalization performance is therefore measured on a test set, which is disjoint from the training set. The accurate partition of data into the two sets is a complex problem both theoretically [89] and practically [92], especially when there is "not enough" data available.

Searching the weight space is done iteratively. This is a consequence of the neural computation paradigm itself. Which iterative technique is applicable depends on the properties of the error function E. If all partial derivatives $\frac{\partial E}{\partial w_i}$ exist error minimization can be done by performing gradient descent. In that case the weights are changed according to

$$w(t+1) = w(t) - \eta \frac{\partial E}{\partial w} . \tag{3.3}$$

In the rule above η is a real positive constant controlling the steepness of the update step. An update step is usually called an epoch. Using (3.3) requires a full scan through the training set. An alternative to this batch learning is to update the weights after each presented pattern p

$$w(t+1) = w(t) - \eta \frac{\partial E^p}{\partial w} , \tag{3.4}$$

whereby the true gradient (3.2) is replaced by

$$E^p := \frac{1}{2} \| d^p - y^p \|_2 . \tag{3.5}$$

E^p, which is only an approximation of E, is referred to as the stochastic gradient and therefore the weight update (3.4) is also named stochastic. Another common name for learning using (3.4) is online learning, which highlights the fact that no internal storage of training patterns is required. Each of the two learning regimes has its own advantages and disadvantages. Batch learning allows a deeper and

easier understanding of the dynamics of neurons and is therefore the appropriate choice for most theoretical studies.

From the biological point of view it is advisable to use an integrative propagation function. A therefore convenient choice would be to use the weighted sum of the input

$$f(w, x) = \sum_i w_i x_i \,, \tag{3.6}$$

that is the activation potential equals the scalar product of input and weights

$$f(w, x) = \langle w, x \rangle \,. \tag{3.7}$$

In fact, (3.6) is the most popular propagation function since the dawn of neural computation, however it is often used in the slightly different form

$$f(w, x) = \sum_i w_i x_i + \theta \,. \tag{3.8}$$

Obviously, setting $w := (w_i, \theta)$ and $x := (x_i, 1)$ yields (3.7) again. The special weight θ is called bias. Applying

$$\Theta(x) = \begin{cases} 1 & : \quad x > 0 \\ 0 & : \quad x \leq 0 \end{cases} \tag{3.9}$$

as activation function to (3.8) yields the famous perceptron of Rosenblatt. In that case θ works as a threshold. Besides (3.8) there are of course many other possible propagation functions. Another function of theoretical importance and some broader practical use is

$$f(w, x) = \prod_i x_i^{w_i} \,, \tag{3.10}$$

which aggregates the input signals in a multiplicative manner. A learning rule for neurons based on (3.10) is given in [52]. Mostly, such neurons are studied in boolean domains only. Summation as in (3.8) can be viewed as modelling linear correlation of the inputs. In pattern recognition so-called higher–order neurons

$$y = g(w_0 + \sum_i w_i x_i + \sum_{i,j} w_{i,j} x_i x_j + \sum_{i,j,k} w_{i,j,k} x_i x_j x_k + \ldots) \tag{3.11}$$

are known for a long time. Such neurons allow, to some extend, geometrically invariant learning of patterns. The drawback lies in the number of required weights, which for a k–th order neuron with n inputs equals

$$\sum_{i=0}^{k} \binom{n+i-1}{i} = \binom{n+k}{k} \,. \tag{3.12}$$

To overcome that exponential explosion of higher–order terms sigma–pi units were proposed, in which only a certain small number of higher–order terms is used. Unfortunately, that either requires to hard–wire appropriate terms in advance or to restrict the order of neurons to a small number, say 2 or 3, in advance. Due to such issues the practical use of higher–order neurons is limited, even in the field of pattern recognition.

If (3.8) is supplemented with the identity as activation function and real–valued domains are given a real linear neuron

$$y = \sum_i w_i x_i + \theta \tag{3.13}$$

is obtained. When used together with the error function (3.2) ordinary linear regression is computed. The real linear neuron (3.13) can be seen as the first example of a Clifford neuron. This is simply due to the isomorphism $\mathbb{R} \cong \mathcal{C}_{0,0,0}$ (section 2.2), which, in particular, also gives equivalence of real multiplication and geometric product in that case. Changing all underlying domains from real–valued to Clifford–valued gives the generalization of (3.13) to neurons in arbitrary Clifford algebras.

Definition 3.1 (Clifford Neuron) *A Clifford Neuron (CN) computes the following function from* $(\mathcal{C}_{p,q,r})^n$ *to* $\mathcal{C}_{p,q,r}$

$$y = \sum_{i=1}^{n} w_i \otimes_{p,q,r} x_i + \theta \,. \tag{3.14}$$

From the purely formal point of view Clifford neurons share with higher–order neurons (3.11) the capability to process polynomial functions of the input. Also, CNs could be formally interpreted as kind of a tensorial approach. However, these are rather technical issues of little interest caused by top–level algebraic coherence.

The aim of this thesis is to demonstrate the usefulness of Clifford algebra for neural computation due to its geometric nature. To actually start this challenge the special case of a "one–dimensional" Clifford Neuron shall be the first subject of detailed study.

Definition 3.2 (Basic Clifford Neuron) *A Basic Clifford Neuron (BCN) is derived from a Clifford Neuron in the case of* $n = 1$ *in* (3.14) *and therefore computes the following function from* $\mathcal{C}_{p,q,r}$ *to* $\mathcal{C}_{p,q,r}$

$$y = w \otimes_{p,q,r} x + \theta \,. \tag{3.15}$$

To denote a BCN of a particular algebra $\mathcal{C}_{p,q,r}$ *the abbreviation* $BCN_{p,q,r}$ *will be used.*

Clifford–valued error functions analog to (3.2) and (3.5) can be easily derived. However, the simplified error function

$$\begin{aligned} E_{p,q,r} &= \frac{1}{2} \parallel d - w \otimes_{p,q,r} x + \theta \parallel_2 \\ &= \frac{1}{2} \parallel d - y \parallel_2 \\ &= \frac{1}{2} \parallel \text{error} \parallel_2 \end{aligned} \tag{3.16}$$

is used in this chapter in order to reduce the notational complexity of update rules. For the bias θ of a BCN a generic update rule can be formulated.

Proposition 3.3 *For any $BCN_{p,q,r}$ the update rule for the bias θ reads*

$$\begin{aligned} \theta(t+1) &= \theta(t) + \Delta\theta \\ &= \theta(t) + \text{error}\,. \end{aligned} \tag{3.17}$$

Of course, Basic Clifford Neurons are only of interest for Clifford algebras of dimension greater than 1, in which case every quantity becomes a multi–dimensional object itself. The real–valued "counterpart" to which a BCN can be related is a feed–forward neural network consisting of one layer of linear neurons. It is convenient to define such a network with linear neurons as in (3.7), that is without an explicit bias weight. As mentioned before this is always possible without any loss of generality.

Definition 3.4 (Linear Associator) *A Linear Associator (LA) is a neural network consisting of n inputs and m output neurons computing the following function from $\mathbb{R}^n$ to $\mathbb{R}^m$*

$$y = Wx\,. \tag{3.18}$$

Thereby the weight matrix W is of size $m \times n$ and an entry w_{ij} represents the weight connecting the i–th input to the j–th output neuron.

The abbreviation $LA_{n,m}$ will be used to refer to a Linear Associator of a particular size. An illustration of a $LA_{3,2}$ is given below in figure 3.2. More material on the Linear Associator is provided in appendix A.1.

Any Basic Clifford Neuron can be viewed as a certain LA. This is a simple consequence of Theorem 2.17, which stated that any Clifford algebra is isomorphic to some matrix algebra. Thus, using a BCN means to apply a certain algebraic model to the data, namely that of the underlying Clifford algebra, instead of a general linear model when using a LA instead. In that sense a BCN is the most trivial example

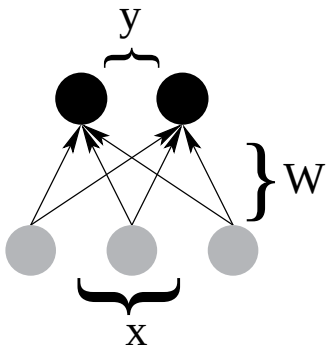

Figure 3.2: Linear Associator $LA_{3,2}$ with 3 inputs and 2 output neurons.

for the model–based nature of the Clifford algebra approach to neural computation. Moreover, this algebraic model becomes also a geometric model whenever a geometric interpretation for the general operation of the geometric product in use is at hand.

Linear algebra can always be used in Clifford algebra if needed. This advantage just carries over to Clifford neurons. In the next section the corresponding Basic Clifford Neurons of the two–dimensional Clifford algebras (2D Basic Clifford Neurons) are studied. This is the natural entry point for all Clifford neural computation. It allows to become familiar with the topic. To elucidate the model-based nature of the Clifford approach will be the main aim of the section. Also, the algorithmic basis for the later chapters on Clifford MLPs will be introduced there. Basic Clifford Neurons are also the subject of section 3.2. However, the point of view is slightly moved from neurons itself to inherent attributes of Clifford neural computation. In particular, issues of data representation and isomorphisms will be discussed. Then a section on Clifford Associators follows before the chapter will be concluded by a summary of the obtained results.

3.1 The 2D Basic Clifford Neurons

The Basic Clifford Neuron (BCN), as introduced in (3.15), is the most simple possible Clifford neuron. The propagation function of a BCN is a certain linear transformation. Hence the BCN can be viewed as a model–based architecture (compared to a generic Linear Associator). The Complex Basic Clifford Neuron, for example, computes a scaling–rotation (Euclidean) followed by a translation (section 2.4). Translation is a generic component common to all BCNs. Therefore the specific part of the propagation function of a BCN is fully determined by the geometric prod-

uct of the underlying Clifford algebra. In this section all three 2D Basic Clifford Neurons are studied from this point of view.

3.1.1 The Complex Basic Clifford Neuron

The geometric interpretation of the propagation function of the Complex Basic Clifford Neuron ($BCN_{0,1}$) was already reviewed at the beginning of this section. It is, technically, based on the isomorphism of the circle group, formed by all complex numbers of modulus one, and the group $SO(2)$ (2.36). Hence, the transformation group $SO(2)$ can be identified as the computational core of the Complex BCN.

From (3.3) the following update rule

$$\begin{aligned}
\Delta w_{0,1} = -\frac{\partial E_{0,1}}{\partial w} &= -\frac{\partial E_{01}}{\partial w_0} e_0 - \frac{\partial E_{01}}{\partial w_1} e_1 \\
&= -((-y_0 + w_0 x_0 - w_1 x_1) x_0 + (-y_1 + w_0 x_1 + w_1 x_0) x_1)\, e_0 \\
&\quad -((+y_0 - w_0 x_0 + w_1 x_1) x_1 + (-y_1 + w_0 x_1 + w_1 x_0) x_0)\, e_1 \\
&= (error_0 x_0 + error_1 x_1)\, e_0 + (-error_0 x_1 + error_1 x_0)\, e_1 \\
&= error \otimes_{0,1} \bar{x}\,. \qquad (3.19)
\end{aligned}$$

derives for the Complex Basic Clifford Neuron. The rule has a very simple form. The error made by the neuron multiplied with the conjugate of the input yields the amount of the update step.

For a CBCN with error function

$$E_{CBCN} = \frac{1}{2} \parallel d - w \otimes_{0,1} x \parallel_2 \qquad (3.20)$$

the following result holds.

Proposition 3.5 *The Hessian matrix of a Complex BCN with error function* (3.20) *is a real matrix of the form*

$$H_{CBCN} = \begin{pmatrix} a & 0 \\ 0 & a \end{pmatrix}. \qquad (3.21)$$

If the input is denoted by $\{x_0^p + x_1^p e_1\}_{p=1}^P$ *then*

$$a = \frac{1}{P} \left(\sum_{p=1}^{P} (x_0^p)^2 + \sum_{p=1}^{P} (x_1^p)^2 \right). \qquad (3.22)$$

This can be concluded from Proposition A.1, which states the general linear case. Since the matrix (3.21) has only one eigenvalue the following nice result is obtained.

Corollary 3.6 *Batch learning for a Complex BCN with error function* (3.20) *converges in only one step (to the global minimum of* (3.20)*) if the optimal learning rate* (A.4) *is used.*

Proposition 3.5 says that a Complex BCN with error function (3.20) applies an intrinsic one–dimensional view to the data. That is the orientation parameterized by the angle of rotation. In particular, this requires that the error function measures the Euclidean distance.

For demonstration a little experiment was performed. A random distribution of 100 points $\{(x_1^p, x_2^p)\}_{p=1}^{100}$ with zero mean, variance one and standard deviation one was created as input set. Then this set of points was rotated about an angle of 63 degrees and scaled by a factor of 4.5 yielding the output set of the experiment. A Complex BCN and a Linear Associator $LA_{2,2}$ were trained on that data using batch learning with optimal learning rates according to (A.4). The observed learning curves are reported in figure 3.3. Additionally, figure 3.4 shows the error surface of the Complex BCN.

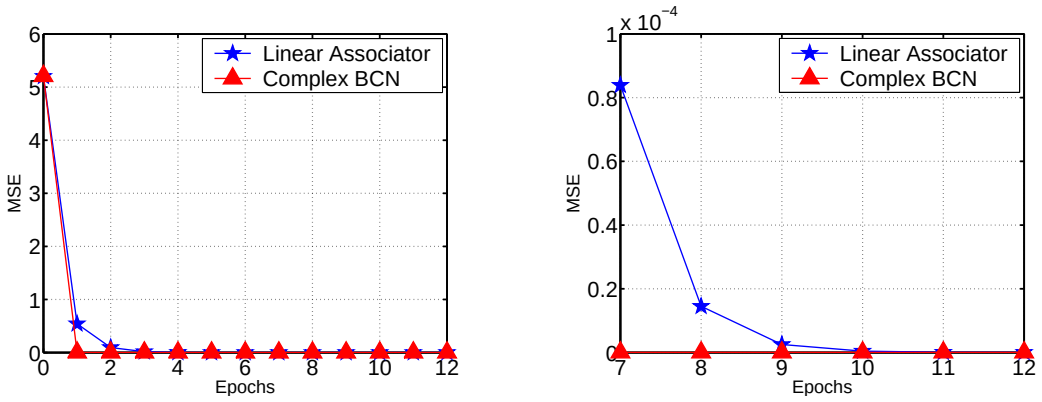

Figure 3.3: Training errors of the Complex BCN and the Linear Associator $LA_{2,2}$. Learning curves over all epochs (left) and epochs 7 to 12 (right).

As stated by Corollary 3.6 the Complex BCN did indeed converge in one epoch. In further steps of training Gaussian noise of different levels was added. Both architectures were then tested on the original noise–free data. The obtained results are reported below in figure 3.5. The Linear Associator has more degrees of freedom than the Complex BCN and no suitable model of the data. The performance of the

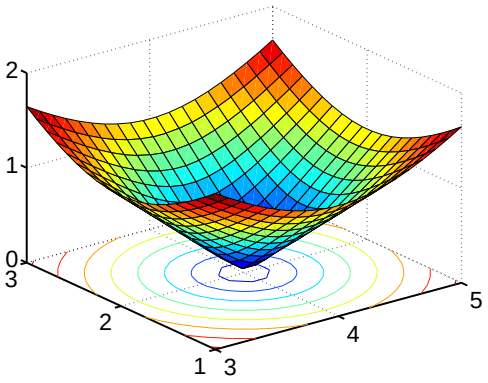

Figure 3.4: Error surface of the Complex BCN.

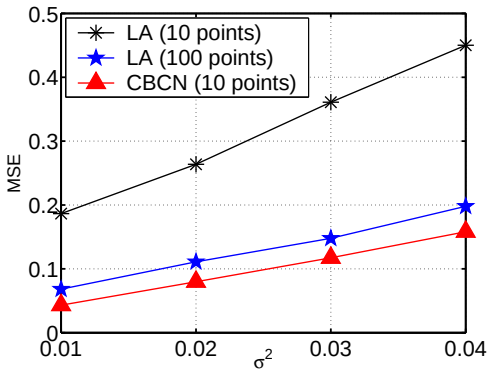

Figure 3.5: Comparison of the test performance of the Complex BCN and the Linear Associator $LA_{2,2}$ as the MSE (measured on the original noise–free data) versus the present Gaussian noise with variance σ^2 during training.

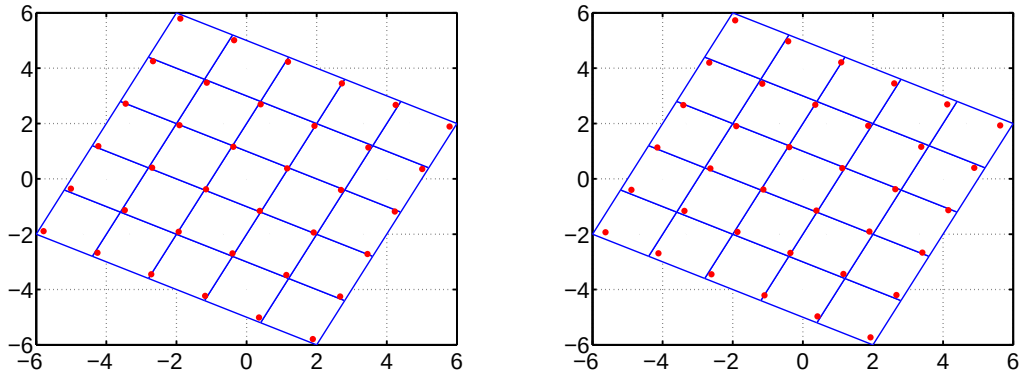

Figure 3.6: Learned transformations by the Complex BCN (left) and the Linear Associator $LA_{2,2}$ (right) from noisy data ($\sigma^2 = 0.4$). Points (learned output) should lie on the grid junctions (expected output).

$LA_{2,2}$ is clearly worse than that of the Complex BCN on a training set of only 10 points. Even if then trained on 100 points the $LA_{2,2}$ performance is still worse than that of the Complex BCN trained on only 10 points, which is also illustrated in figure 3.6.

3.1.2 The Hyperbolic Basic Clifford Neuron

The computational properties of the Complex BCN were shown to be induced by the group of unit complex numbers, or, equivalently, $SO(2)$. This way a geometric interpretation for the propagation function of a Complex BCN could be easily achieved.

The matrix representation of a unit hyperbolic number $x = x_0 + x_1 e_1$, which can be derived from (2.30), reads [37]

$$x = \begin{pmatrix} \epsilon \cosh(\phi) & \epsilon \sinh(\phi) \\ \epsilon \sinh(\phi) & \epsilon \cosh(\phi) \end{pmatrix} \tag{3.23}$$

for some $\phi \in \mathbb{R}$ and $\epsilon = \pm 1$. Thus there are actually two types of unit hyperbolic numbers. If $|x_0| > |x_1|$ then $\epsilon = 1$ in (3.23), and, if $|x_1| > |x_0|$ then $\epsilon = -1$ in (3.23)[1]. Unit hyperbolic numbers form a group which is isomorphic to $SO(1,1)$. Moreover, the numbers with $\epsilon = 1$ in (3.23) form a subgroup being isomorphic to $SO(1,1)^+$. Only the transformations belonging to $SO(1,1)^+$ are orientation preserving, and, are therefore called hyperbolic rotations. The elements of $SO(1,1) \backslash SO(1,1)^+$ are called anti–rotations. An illustration of the above is given in figure 3.7. Both type of transformations form the computational core of the Hyperbolic BCN ($BCN_{1,0}$).

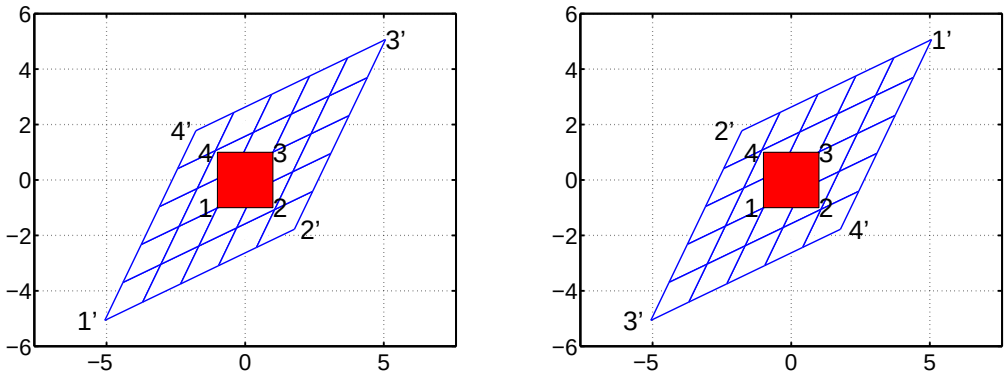

Figure 3.7: Illustration of hyperbolic numbers acting as transformations in the plane. Hyperbolic rotation about 30 degrees (left) and hyperbolic anti–rotation about 30 degrees (right) of the unit square.

[1] $|x_0| = |x_1|$ is not possible for a unit hyperbolic number.

The update rule for the Hyperbolic BCN is given by

$$\begin{aligned}\Delta w = -\frac{\partial E_{1,0}}{\partial w} &= -\frac{\partial E_{1,0}}{\partial w_0}e_0 - \frac{\partial E_{1,0}}{\partial w_1}e_1 \\ &= -((-y_0 + w_0x_0 + w_1x_1)x_0 + (-y_1 + w_0x_1 + w_1x_0)x_1)\, e_0 \\ &\quad -((-y_0 + w_0x_0 + w_1x_1)x_1 + (-y_1 + w_0x_1 + w_1x_0)x_0)\, e_1 \\ &= (error_0x_0 + error_1x_1)\, e_0 + (error_0x_1 + error_1x_0)\, e_1 \\ &= error \otimes_{1,0} x\,. \end{aligned} \tag{3.24}$$

The product of the error made by the neuron with the input yields the amount of the update step. Next, the dynamics of a Hyperbolic BCN with error function

$$E_{HBCN} = \frac{1}{2} \parallel d - w \otimes_{1,0} x \parallel_2 \tag{3.25}$$

are outlined.

Proposition 3.7 *The Hessian matrix of a Hyperbolic BCN with error function* (3.25) *is a real matrix of the form*

$$H_{HBCN} = \begin{pmatrix} a & b \\ b & a \end{pmatrix}. \tag{3.26}$$

If the input is denoted by $\{x_0^p + x_1^p e_1\}_{p=1}^P$ *then*

$$a = \frac{1}{P}\left(\sum_{p=1}^{P}(x_0^p)^2 + \sum_{p=1}^{P}(x_1^p)^2\right) \quad \textit{and} \quad b = \frac{2}{P}\sum_{p=1}^{P} x_0^p x_1^p\,. \tag{3.27}$$

The dynamics of a Hyperbolic BCN and a Linear Associator (on the same input set) are related as follows.

Proposition 3.8 *Assuming the same input set, batch learning, and the use of optimal learning rates, a* $LA_{2,2}$ *with standard error function* E_{LA} (A.1) *will not converge faster than a Hyperbolic BCN with error function* (3.25)*.*

This is a rather technical result from Proposition 3.7. Some more insights of the dynamics for a Hyperbolic BCN can be gained by a little experiment, for which a similar setup as in section 3.1.1 was chosen.

The input set was created from a random distribution of 100 points $\{(x_1^p, x_2^p)\}_{p=1}^{100}$ with zero mean, variance one and standard deviation one. From that the output set was created by applying the anti-rotation induced by the hyperbolic number

$2 + 4e_1$. The learning curves for the Hyperbolic BCN and the $LA_{2,2}$ are shown in figure 3.8.

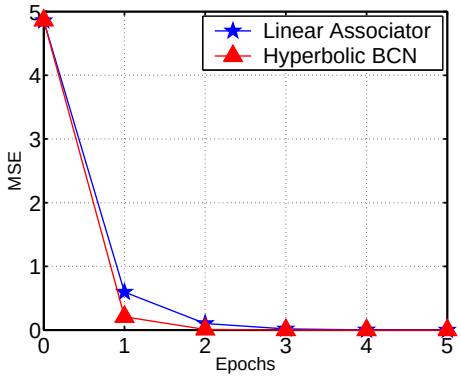

Figure 3.8: Training errors of the Hyperbolic BCN and the Linear Associator $LA_{2,2}$.

The Hyperbolic BCN converged faster than the Linear Associator. In figure 3.9 (left) the error surface of the Hyperbolic BCN is plotted. The contour lines are ellipses which are rotated about 45 degrees w.r.t. to the origin. Actually, this is a consequence of Proposition 3.7, i.e. it holds for any Hyperbolic BCN. That is the eigenvectors of the Hessian (3.26) are always predetermined. Unfortunately, this is not the case for the ratio of its eigenvalues. Therefore no generic coordinate transformation can be applied in advance. Since all of the above depends on the used error function a more "hyperbolic" error function seems tempting. Of course, the hyperbolic norm is the first candidate to look at. The hyperbolic norm, however, does not induce a metric on the whole $\mathbb{R}^2$, and, is therefore not applicable as can be seen from figure 3.9 (right).

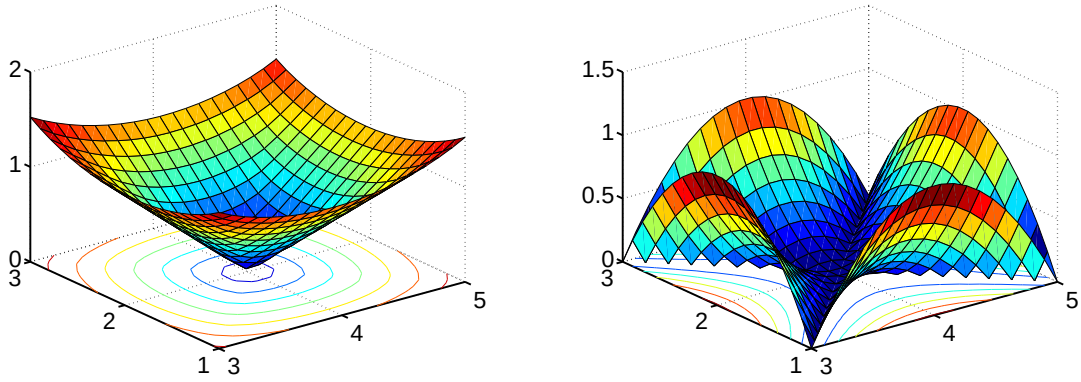

Figure 3.9: Error surface of the Hperbolic BCN as induced by the Euclidean norm (left) and the hyperbolic norm (right).

As outlined above, the geometric model (based on the group $SO(1,1)$) by which a Hyperbolic BCN looks at the data is not intrinsically one–dimensional. However, its presence can be clearly observed from the results presented in figure 3.10. The Hyperbolic BCN trained on 10 points outperformed the Linear Associator on noisy data. Even if the LA was trained with 100 points. This can also be seen from figure 3.11 which gives a visualization of the learned transformations for one particular noise level.

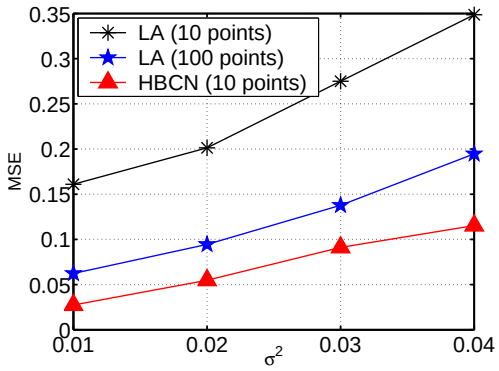

Figure 3.10: Comparison of the test performance of the Hyperbolic BCN and the Linear Associator $LA_{2,2}$ as the MSE (measured on the original noise–free data) versus the present Gaussian noise with variance σ^2 during training.

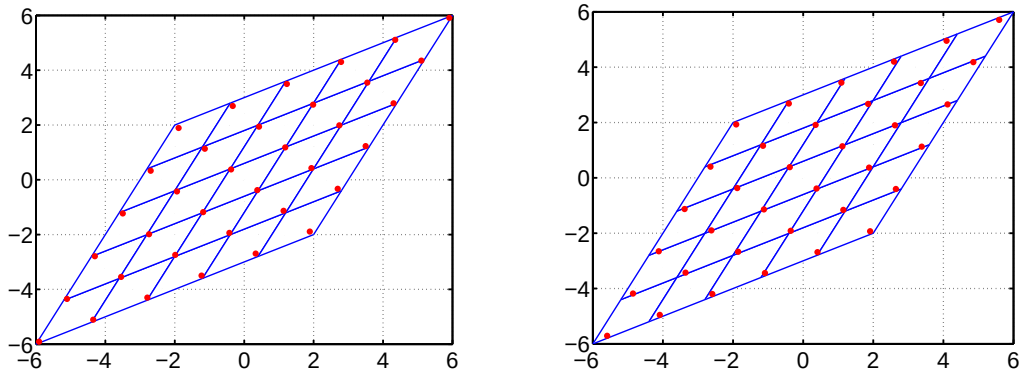

Figure 3.11: Learned transformations by the Hyperbolic BCN (left) and the Linear Associator $LA_{2,2}$ (right) from noisy data ($\sigma^2 = 0.4$). Points (learned output) should lie on the grid junctions (expected output).

3.1.3 The Dual Basic Clifford Neuron

The Dual Basic Clifford Neuron ($BCN_{0,0,1}$) remains as the last object for examination. In contrast to the other two 2D Basic Clifford Neurons it is based on a degenerate Clifford algebra. However, there is also a pretty easy interpretation of dual numbers as transformations acting in the plane. Every dual number induces a shear transformation having the y–axes as fixed–line (see figure 3.12 for an illustration). This also gives a geometric derivation for the set of zero divisors of $\mathcal{C}_{0,0,1}$ (2.28).

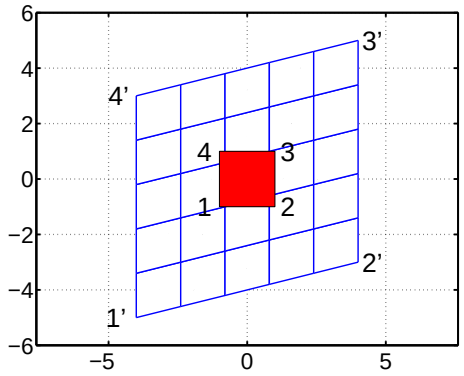

Figure 3.12: Illustration of dual numbers acting as transformations in the plane. The unit square transformed by the shearing transformation induced by the dual number $4 + 1e_1$.

The degenerate nature of $\mathcal{C}_{0,0,1}$ also effects the update rule for the Dual BCN which reads

$$\begin{aligned}
\Delta w = -\frac{\partial E_{0,0,1}}{\partial w} &= -\frac{\partial E_{0,0,1}}{\partial w_0} e_0 - \frac{\partial E_{0,0,1}}{\partial w_1} e_1 \\
&= -((-y_0 + w_0 x_0)x_0 + (-y_1 + w_0 x_1 + w_1 x_0)x_1)\, e_0 \\
&\quad -(\ 0 \qquad\qquad\quad + (-y_1 + w_0 x_1 + w_1 x_0)x_0)\, e_1 \\
&= (error_0 x_0 + error_1 x_1)\, e_0 + (0 + error_1 x_0)\, e_1 \,. \qquad (3.28)
\end{aligned}$$

In contrast to the previous neurons the update rule can not be formulated in terms of the underlying geometric product $\otimes_{0,0,1}$. Also, no quantitative statement independent of a particular data set can be made for the dynamics of a Dual BCN. The Hessian of a Dual BCN can not be narrowed down to a particular form. Therefore, the only thing that remains to check about a Dual BCN is the performance on noisy data. For that purpose a little experiment was performed again.

As in the previous experiments of this section the input set consisted of 100 2D–points drawn from a distribution having zero mean, variance one and standard

deviation one. For the output set those points were transformed by the shearing induced by the dual number $2 + 4e_1$. The error surface of the Dual BCN computed for the input set is shown in figure 3.13, which is of course a paraboloid.

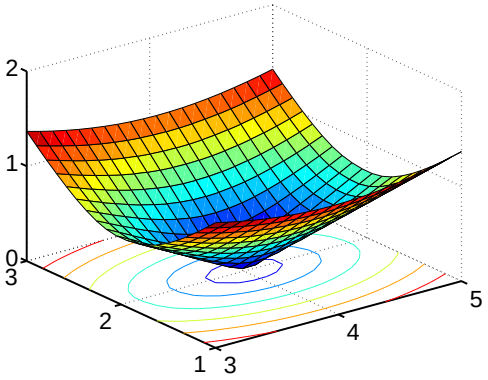

Figure 3.13: Error surface of the Dual BCN.

For the given data the Linear Associator did converge faster than the Dual BCN as can be seen from figure 3.14.

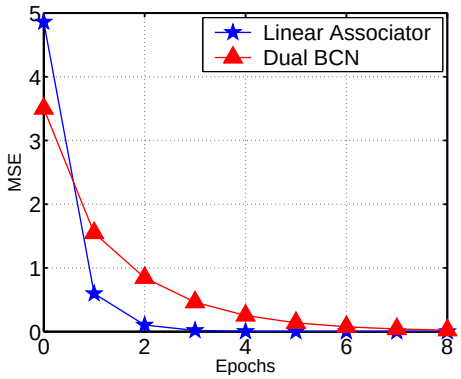

Figure 3.14: Training errors of the Dual BCN and the Linear Associator $LA_{2,2}$.

However, for training in the presence of noise similar numerical results as in the previous experiments were obtained. The Dual BCN, possessing the right model for the data, outperformed the Linear Associator, which, without such a model, followed much more the noise in the data. This becomes even more clear by looking at the visualizations of the learned transformations provided in figure 3.16.

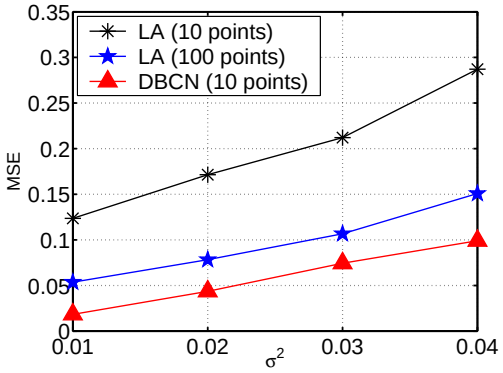

Figure 3.15: Comparison of the test performance of the Dual BCN and the Linear Associator $LA_{2,2}$ as the MSE (measured on the original noise–free data) versus the present Gaussian noise with variance σ^2 during training.

From all the introduced 2D Basic Clifford neurons the Dual BCN implies internally the strongest constraints to the fitted data. This makes it[2] obviously weaker than the other neurons in the following sense. Suppose we want to approximate a general linear mapping. This seems intuitively easier by a superposition of scaling–rotations, i.e. by an architecture based on Complex BCNs than by a superposition of shearing transformation, i.e. by an architecture based on Dual BCNs. For the case of Hyperbolic BCNs it is hard to come up with any such intuition.

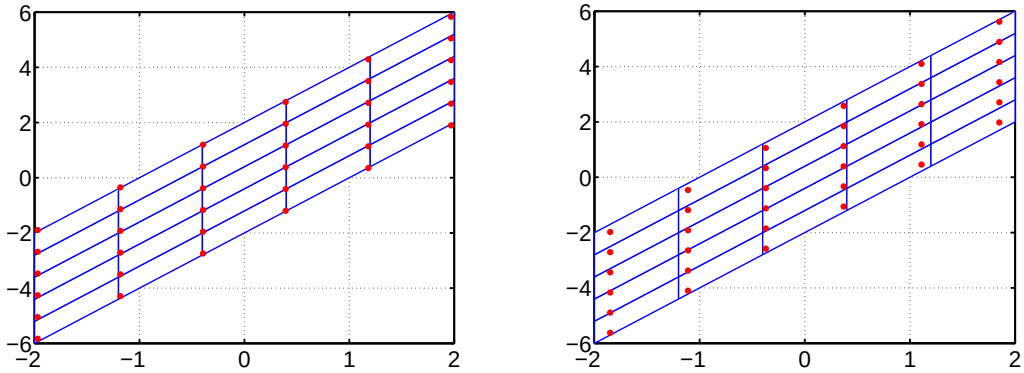

Figure 3.16: Learned transformations by the Dual BCN (left) and the Linear Associator $LA_{2,2}$ (right) from noisy data ($\sigma^2 = 0.4$). Points (learned output) should lie on the grid junctions (expected output).

[2]More precisely, architectures based on it.

3.2 Isomorphic BCNs and Isomorphic Representations

For every natural number $n > 1$ there exist Clifford algebras of dimension 2^n which are isomorphic to each other (see table 2.2). Basic Clifford Neurons (BCNs) are solely determined by the geometric product of the underlying algebra. Therefore it is quite natural to assume that isomorphic Clifford algebras also induce "isomorphic" BCNs. Proving this assumption and determining its precise meaning is the first goal of this section. The closely related topic of isomorphic representations is then discussed afterwards.

3.2.1 Isomorphic Basic Clifford Neurons

Isomorphic Clifford algebras first occur in dimension four, namely $\mathcal{C}_{1,1}$ and $\mathcal{C}_{2,0}$ (see table 2.2). Those two algebras will serve as an example throughout this section. Their multiplication tables are given below in table 3.1 and table 3.2, respectively.

$\otimes_{1,1}$	1	e_1	e_2	e_1e_2
1	1	e_1	e_2	e_1e_2
e_1	e_1	1	e_1e_2	e_2
e_2	e_2	$-e_1e_2$	-1	e_1
e_1e_2	e_1e_2	$-e_2$	$-e_1$	1

Table 3.1: Multiplication table of the Clifford algebra $\mathcal{C}_{1,1}$.

$\otimes_{2,0}$	1	e_1	e_2	e_1e_2
1	1	e_1	e_2	e_1e_2
e_1	e_1	1	e_1e_2	e_2
e_2	e_2	$-e_1e_2$	1	$-e_1$
e_1e_2	e_1e_2	$-e_2$	e_1	-1

Table 3.2: Multiplication table of the Clifford algebra $\mathcal{C}_{2,0}$.

The multiplication tables do only slightly differ. Therefore the two algebras are related by a rather simple isomorphism.

Proposition 3.9 *The map* $\phi : \mathcal{C}_{1,1} \to \mathcal{C}_{2,0}$

$$x_0 + x_1e_1 + x_2e_2 + x_{12}e_{12} \mapsto x_0 + x_{12}e_1 + x_1e_2 - x_2e_{12} \tag{3.29}$$

is an isomorphism.

Proof: Let $a := a_0e_0 + a_1e_1 + a_2e_2 + a_{12}e_{12}, b := b_0e_0 + b_1e_1 + b_2e_2 + b_{12}e_{12}$. Then yields

$$\begin{aligned}
\phi(a \otimes_{1,1} b) &= \phi\big(\; (a_0b_0 + a_1b_1 - a_2b_2 + a_{12}b_{12})\, e_0 \\
&\quad + (a_0b_1 + a_1b_0 - a_2b_{12} - a_{12}b_2)\, e_1 \\
&\quad + (a_0b_2 + a_1b_{12} + a_2b_0 - a_{12}b_1)\, e_2 \\
&\quad + (a_0b_{12} + a_1b_2 - a_2b_1 + a_{12}b_0)\, e_{12}\big) \\
&= \quad (a_0b_0 + a_1b_1 - a_2b_2 + a_{12}b_{12})\, e_0 \\
&\quad + (a_0b_{12} + a_1b_2 - a_2b_1 + a_{12}b_0)\, e_1 \\
&\quad + (a_0b_1 + a_1b_0 + a_2b_{12} - a_{12}b_2)\, e_2 \\
&\quad - (a_0b_2 + a_1b_{12} + a_2b_0 - a_{12}b_1)\, e_{12} \\
&= (a_0e_0 + a_{12}e_1 + a_1e_2 - a_2e_{12}) \otimes_{2,0} (b_0e_0 + b_{12}e_1 + b_1e_2 - b_2e_{12}) \\
&= \phi(a) \otimes_{2,0} \phi(b)
\end{aligned}$$

□

In order to be able to perform actual experiments the update rules for the corresponding Clifford neurons are derived next. The weight update for the $BCN_{1,1}$ reads

$$\begin{aligned}
\Delta w_{1,1} &= -\nabla E_{1,1} \\
&= (error \cdot (x_0, x_1, x_2, x_{12})^T)\, e_0 \\
&+ (error \cdot (x_1, x_0, x_{12}, x_2)^T)\, e_1 \\
&+ (error \cdot (-x_2, x_{12}, x_0, -x_1)^T)\, e_2 \\
&+ (error \cdot (x_{12}, -x_2, -x_1, x_0)^T)\, e_{12} \\
&= error \otimes_{1,1} (x_0e_0 + x_1e_1 - x_2e_2 + x_{12}e_{12})\,. \qquad (3.30)
\end{aligned}$$

It is possible do derive the update rule for the $BCN_{2,0}$ from (3.30) and the isomorphism (3.29). The direct approach, however, is much easier and yields the following rule

$$\begin{aligned}
\Delta w_{2,0} &= -\nabla E_{2,0} \\
&= (error \cdot (x_0, x_1, x_2, x_3)^T)\, e_0 \\
&+ (error \cdot (x_1, x_0, x_{12}, x_2)^T)\, e_1 \\
&+ (error \cdot (x_2, -x_{12}, x_0, -x_1)^T)\, e_2 \\
&+ (error \cdot (-x_{12}, x_2, -x_1, x_0)^T)\, e_{12} \\
&= error \otimes_{2,0} (x_0 e_0 + x_1 e_1 + x_2 e_2 - x_{12} e_{12})\,. \qquad (3.31)
\end{aligned}$$

To get an understanding of isomorphic Clifford neurons the following experiment was performed with the neurons $BCN_{1,1}$ and $BCN_{2,0}$. As input set for the $BCN_{1,1}$ four[3] 4–dimensional points were randomly drawn. Any input point (regarded as a multivector in $\mathcal{C}_{1,1}$) was multiplied from the left with $1e_0 + 2e_1 + 3e_2 + 4e_{12}$ via the geometric product $\otimes_{1,1}$. That way the $BCN_{1,1}$ output set was created. Then the isomorphism ϕ (3.29) was applied to both sets yielding the input and output data for the $BCN_{2,0}$. Both neurons were trained by batch learning with optimal learning rate. The optimal learning rate for the two neurons turned out to be identical. The initial multivector weights of the neurons were set according to table 3.3.

$w(0)$	e_0	e_1	e_2	e_{12}
$BCN_{1,1}$	+0.3462	+0.0252	-0.2974	+0.2721
$BCN_{2,0}$	+0.3381	-0.4804	+0.1813	-0.1205

Table 3.3: Initial multivector weights for the $BCN_{1,1}$ and the $BCN_{2,0}$.

The obtained learning curves (see figure 3.17) only differ at the very first epochs of training.

[3]That small number of points was chosen to exclude any kind of averaging effects.

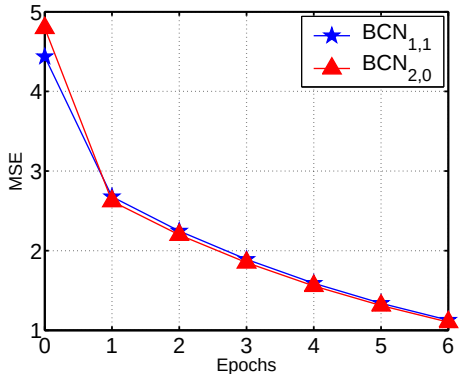

Figure 3.17: Training errors of the $BCN_{1,1}$ and the $BCN_{2,0}$.

Training was finished after 50 epochs and the neurons converged indeed to "isomorphic" solutions as reported in table 3.4. In addition, isomorphic initial multivector weights resulted in totally identical learning dynamics.

$w(50)$	e_0	e_1	e_2	e_{12}
$BCN_{1,1}$	+1.0000	+1.9999	+2.9994	+3.9999
$BCN_{2,0}$	+1.0000	+3.9994	+1.9999	-2.9994

Table 3.4: *Multivector weights for the $BCN_{1,1}$ and the $BCN_{2,0}$ after 50 epochs of training. Equivalence of the results follows from the isomorphism* (3.29).

A formal notion for identical dynamics of isomorphic Basic Clifford Neurons is presented in the next theorem. The result holds for both online and batch learning.

Theorem 3.10 *Let ϕ be an isomorphism from $\mathcal{C}_{p,q}$ to $\mathcal{C}_{p',q'}$. Let $BCN_{p,q}$ be a neuron trained on a training set $(x^p, d^p)_{p=1}^P$ with learning rate η and initial multivector weight $w_{p,q}^0$. Furthermore, let $w_{p,q}^t$ denote the multivector weight of $BCN_{p,q}$ after t training steps. Then the neuron $BCN_{p',q'}$ trained on the training set $(\phi(x^p), \phi(d^p))_{p=1}^P$ with learning rate η and initial multivector weight $\phi(w_{p',q'}^0)$ has the multivector weight $\phi(w_{p',q'}^t)$ after t steps of training.*

Besides being very powerful, the statement of Theorem 3.10 seems very intuitive at first sight. However, it is only valid because every isomorphism of two Clifford algebras is of particular simple nature (likewise (3.29)). This can be seen more clearly from the following important corollary.

Corollary 3.11 *Let the assumptions of Theorem 3.10 be valid. Then two neurons $BCN_{p,q}$ and $BCN_{p',q'}$ have the same optimal learning rate for batch learning.*

This corollary, taken into account Proposition A.2, puts some restrictions on the nature of the underlying algebra isomorphism. Therefore Theorem 3.10 results from the very insights of Clifford algebra.

All of the above motivates the following definition.

Definition 3.12 *The two neurons $BCN_{p,q}$ and $BCN_{p',q'}$ are called isomorphic if $\mathcal{C}_{p,q} \cong \mathcal{C}_{p',q'}$ holds.*

Anything that can be computed by a Basic Clifford Neuron can also be computed with equivalent dynamics (Theorem 3.10) by any other isomorphic BCN. In that sense, two isomorphic Basic Clifford Neurons are identical neural architectures. Then, so being all Clifford neural architectures that only differ by isomorphic neurons.

3.2.2 Isomorphic Representations

There are many fundamental differences between Clifford neural networks and standard real–valued ones. Many of them are related to how the data is seen by them, or, how the data can be represented to them. In what follows the technical basis for data representation inside the Clifford framework is outlined. Many powerful applications of this feature of Clifford neural computation will be given later in this thesis.

For a fully–connected real–valued neural network neither the order of the components of a presented input vector nor the order of the components of a presented output vector has a semantical meaning. Consider for example the Linear Associator $LA_{4,4}$ trained on a training set $\{(x_1^p, x_2^p, x_3^p, x_4^p), (d_1^p, d_2^p, d_3^p, d_4^p)\}_{p=1}^P$. Presenting the training set $\{(x_4^p, x_1^p, x_2^p, x_3^p), (d_4^p, d_1^p, d_2^p, d_3^p)\}_{p=1}^P$ is obviously the same task. Therefore both data representations can be viewed as "isomorphic" for a $LA_{4,4}$. Nothing new or different is gained by using the second one instead of the first one. This is very similar to the notion of isomorphic neurons we have discussed before.

However, the same data representations as above

$$\{(x_1e_0 + x_2e_1 + x_3e_2 + x_4e_{12})^p, (d_1e_0 + d_2e_1 + d_3e_2 + d_4e_{12})^p\}_{p=1}^P \tag{3.32}$$

and

$$\{(x_4e_0 + x_1e_1 + x_2e_2 + x_3e_{12})^p, (d_4e_0 + d_1e_1 + d_3e_2 + d_3e_{12})^p\}_{p=1}^{P} \tag{3.33}$$

are not isomorphic for the $BCN_{1,1}$. The order of components is relevant for Clifford neurons. Multivectors are, for Clifford neurons, tuples, and, a geometrical semantic is induced by the graded subspace structure of the underlying Clifford algebra. In order to proceed the next rather technical definition is needed.

Definition 3.13 (Isomorphic Representations for Basic Clifford Neurons) *Two representations* $f : \mathbb{R}^{2^{p+q+r}} \to \mathcal{C}_{p,q,r}$ *and* $g : \mathbb{R}^{2^{p+q+r}} \to \mathcal{C}_{p',q',r'}$ *are called (left–)isomorphic if for all* $x \in \mathbb{R}^{2^{p+q+r}}$ *and for all* $y \in \mathcal{C}_{p,q,r}$ *there exists some* $y' \in \mathcal{C}_{p',q',r'}$ *such that*

$$f(x) \otimes_{p,q,r} y = g(x) \otimes_{p',q',r'} y' . \tag{3.34}$$

Obviously the algebras have to be of same dimension for (3.34) to hold. Isomorphic representations define equivalence classes of data representations. For example, one might be interested in isomorphic representations to (3.32) for the $BCN_{1,1}$. This translates to $\mathcal{C}_{p,q,r} = \mathcal{C}_{p',q',r'} = \mathcal{C}_{1,1,0}$, $f = id$ and then asking for permutations g fulfilling (3.34).

Of course, the most interesting thing about isomorphic representations are the underlying data representations itself. The most important class of representations for Clifford neural computation arises from presenting lower–dimensional data to higher–dimensional Clifford neurons. This may be the normal case since every Clifford algebra is particularly of dimension 2^n. Moreover, representing data in such a way may often be intended, since it allows for many uses of the subspace grading of a Clifford algebra. Then it is much more precise to speak of lower–dimensional objects (having a semantical meaning, say vectors, for example) that are presented. The next chapter will introduce a new Clifford neuron solely designed for that purpose.

Knowledge of isomorphic representations for the above type of representations is therefore essential. However, in its general form the problem of determining all possible isomorphic presentations cannot be treated efficiently. This is simply due to the following combinatorial argument. It is a common fact that the number of combinations of n things taken k at a time is

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} . \tag{3.35}$$

There are $\binom{n}{k}^2$ possibilities for representing a k–dimensional data set to a Basic Clifford Neuron of dimension n. For example, 3–dimensional data can be presented to

a 8–dimensional BCN, say $BCN_{1,1,1}$, in 3136 ways. Plugging $k = 4$ and $n = 16$ into (3.35) yields 3312400 possibilities.

Let us consider the following two–dimensional data representations

$$\begin{aligned}
(1) &: \{(x_1e_0 + x_2e_1 + 0e_2 + 0e_{12})^p, (d_1e_0 + d_2e_1 + 0e_2 + 0e_{12})^p\} \\
(2) &: \{(x_1e_0 + 0e_1 + x_2e_2 + 0e_{12})^p, (d_1e_0 + 0e_1 + d_2e_2 + 0e_{12})^p\} \\
(3) &: \{(x_1e_0 + 0e_1 + 0e_2 + x_2e_{12})^p, (d_1e_0 + 0e_1 + 0e_2 + d_2e_{12})^p\} \\
(4) &: \{(0e_0 + x_1e_1 + x_2e_2 + 0e_{12})^p, (0e_0 + d_1e_1 + d_2e_2 + 0e_{12})^p\} \\
(5) &: \{(0e_0 + x_1e_1 + 0e_2 + x_2e_{12})^p, (0e_0 + d_1e_1 + 0e_2 + d_2e_{12})^p\} \\
(6) &: \{(0e_0 + 0e_1 + x_1e_2 + x_2e_{12})^p, (0e_0 + 0e_1 + d_1e_2 + d_2e_{12})^p\}.
\end{aligned}$$

for the $BCN_{1,1}$ as a walk–trough example.

Proposition 3.14 *The following relations hold*

$$BCN_{1,1}(1) \cong BCN_{1,1}(6) \tag{3.36}$$
$$BCN_{1,1}(2) \cong BCN_{1,1}(5) \tag{3.37}$$
$$BCN_{1,1}(3) \cong BCN_{1,1}(4)\,. \tag{3.38}$$

Proof: Statement (3.36) is verified by the following little computation

$$\begin{aligned}
& (x_1e_0 + x_2e_1 + 0e_2 + 0e_{12}) \otimes_{1,1} (ae_0 + be_1 + ce_2 + de_{12}) \\
= \; & (x_1a + x_2b)e_0 + (x_1b + x_2a)e_1 + (x_1c + x_2d)e_2 + (x_1d + x_2c)e_{12} \\
= \; & (0e_0 + 0e_1 + x_1e_2 + x_2e_{12}) \otimes_{1,1} (ce_0 - de_1 - ae_2 + be_{12})
\end{aligned}$$

The other results follow by similar simple computations. □

Note that

$$(x_1e_0 + x_2e_2) \otimes_{1,1} (ae_0 + be_2) = (x_1 - a)e_0 + (x_2 + b)e_2 \tag{3.39}$$

and

$$(x_1e_1 + x_2e_2) \otimes_{1,1} (ae_1 + be_2) = (x_1 + a)e_1 + (x_2 + b)e_2\,. \tag{3.40}$$

Thus (3.39) is the multiplication of complex numbers and (3.40) is the multiplication of hyperbolic numbers. Both types are therefore encoded in the $BCN_{1,1}$.

In the remainder of this section we will now deal with representations of affine transformations of the plane. This is a useful class of propagation functions and a nice way to summarize all the topics of this section in one example.

3.2.3 Example: Affine Transformations of the Plane

An affine transformation of the plane is a mapping from $\mathbb{R}^2$ to $\mathbb{R}^2$ of the form

$$(x, y)^T \mapsto \begin{pmatrix} a & b \\ c & d \end{pmatrix} (x, y)^T + \begin{pmatrix} t_x \\ t_y \end{pmatrix} . \tag{3.41}$$

In Clifford algebra such transformation can be expressed in either of the two isomorphic algebras $\mathcal{C}_{1,1}$ and $\mathcal{C}_{2,0}$. The algebra $\mathcal{C}_{2,0}$ will be treated first. The direct approach of using the vector representation $x_1e_1 + x_2e_2$, however, does not work. Suitable representations can be found by utilizing the fact $\mathbb{R}(2) \cong \mathcal{C}_{2,0}$ (table 2.2).

Proposition 3.15 *The map*

$$\mathbb{R}(2) \to \mathcal{C}_{2,0}, \begin{pmatrix} a & b \\ c & d \end{pmatrix} \mapsto \frac{1}{2}((a+d)e_0 + (a-d)e_1 + (b+c)e_2 + (b-c)e_{12}) \tag{3.42}$$

is an isomorphism.

The application of the transformation matrix in (3.41) can also be written in the slightly more complicated way

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} (x, y)^T = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x & 0 \\ y & 0 \end{pmatrix} . \tag{3.43}$$

Transforming $\begin{pmatrix} x & 0 \\ y & 0 \end{pmatrix}$ by (3.42) yields $\frac{1}{2}(xe_0 + xe_1 + ye_2 - ye_{12})$, which allows the following statement[4].

Proposition 3.16 *For any real data* $\{(x_1^p, x_2^p), (d_1^p, d_2^p)\}_{p=1}^P$ *the* $BCN_{2,0}$ *trained on the training set* $\{(x_1e_0 + x_1e_1 + x_2e_2 - x_2e_{12})^p, (d_1e_0 + d_1e_1 + d_2e_2 - d_2e_{12})^p\}_{p=1}^P$ *converges to the best affine approximation of the data (assuming the use of an appropriate learning rate).*

Note that nothing inside the neuron has to be changed. Applying the isomorphism (3.42) to the multivector weight w of the trained neuron gives the transformation matrix of (3.41). Similarly, applying (3.42) to the multivector bias θ yields the translation vector of (3.41).

From the definition of an affine transformation (3.41) and with the methods outlined above the following result is easily obtained.

[4]The factor $\frac{1}{2}$ can be neglected.

Proposition 3.17 *The representations*

$$\{(x_1e_0 + x_1e_1 + x_2e_2 - x_2e_{12})^p, (d_1e_0 + d_1e_1 + d_2e_2 - d_2e_{12})^p\}_{p=1}^P \tag{3.44}$$

and

$$\{(x_2e_0 + x_2e_1 + x_1e_2 - x_1e_{12})^p, (d_2e_0 + d_2e_1 + d_1e_2 - d_1e_{12})^p\}_{p=1}^P \tag{3.45}$$

are the only isomorphic representations for a $BCN_{2,0}$ representing affine transformations (in the sense of Proposition 3.16).

The existence of suitable presentations of affine transformations for the $BCN_{1,1}$ is guaranteed by Theorem 3.10, and, all such presentation can be directly derived by applying the inverse isomorphism of (3.29) to (3.44) and (3.45).

Proposition 3.18 *The representations*

$$\{(x_1e_0 + x_2e_1 + x_2e_2 + x_1e_{12})^p, (d_1e_0 + d_2e_1 + d_2e_2 + d_1e_{12})^p\}_{p=1}^P \tag{3.46}$$

and

$$\{(x_2e_0 + x_1e_1 + x_1e_2 + x_2e_{12})^p, (d_2e_0 + d_1e_1 + d_1e_2 + d_2e_{12})^p\}_{p=1}^P \tag{3.47}$$

are the only isomorphic representations for a $BCN_{1,1}$ representing affine transformations (in the sense of Proposition 3.16).

Of course, the same representations derive from the following isomorphism from $\mathbb{R}(2)$ to $\mathcal{C}_{1,1}$.

Proposition 3.19 *The map*

$$\mathbb{R}(2) \to \mathcal{C}_{1,1},\ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \mapsto \frac{1}{2}((a+d)e_0 + (b+c)e_1 + (c-b)e_2 + (a-d)e_{12}) \tag{3.48}$$

is an isomorphism.

Applying (3.48) to $\begin{pmatrix} x & 0 \\ y & 0 \end{pmatrix}$ and $\begin{pmatrix} y & 0 \\ x & 0 \end{pmatrix}$ yields (3.46) and (3.47), respectively.

A lot of new technical terms for Clifford neural computation were introduced in this section. Most important, we saw how Basic Clifford Neurons incorporate different propagation functions, which are "selectable" from the outside by applying different representations.

3.3 The Clifford Associator

So far we have been studying only single Basic Clifford Neurons. Technically, a Basic Clifford Neuron (Definition 3.2) was introduced as one–dimensional special case of a Clifford Neuron (Definition 3.1). The latter will be used now to introduce the first Clifford neural network of this thesis.

Definition 3.20 (Clifford Associator) *A Clifford Associator (CA) is a Clifford neural network consisting of* n *inputs and* m *output Clifford Neurons computing the following function from* $(\mathcal{C}_{p,q,r})^n$ *to* $(\mathcal{C}_{p,q,r})^m$

$$y = W \otimes x\,, \tag{3.49}$$

with x and y both being tuples of multivectors and W being a weight matrix of size $m \times n$ *having multivector entries. An entry* w_{ij} *represents the multivector weight connecting the* i*–th input to the* j*–th output Clifford Neuron. The weight matrix W is applied to the input* x *by means of the tensor product* $\otimes$*.*

The above definition is only heavy–weighted with respect to the amount of needed formalism. A Clifford Associator is the natural generalization of a Linear Associator as introduced in Definition 3.4. In particular, the latter is obtained by plugging $\mathcal{C}_{0,0,0}$ into Definition 3.20. From the pure technical point of view the Clifford Associator can also be seen as a tensor network [63].

Obviously, isomorphic Clifford algebras give isomorphic Clifford Associators. The number of real parameters of a Clifford Associator of size $m \times n$ is given by

$$\mathcal{N}_{CA_{m,n}} = m \cdot n \cdot 2^{p+q+r}\,. \tag{3.50}$$

The update rule for the multivector weight w_{ij} of a Clifford Associator with underlying algebra $\mathcal{C}_{p,q,r}$ is the same as for the Basic Clifford Neuron $BCN_{p,q,r}$ having x_i, y_j and d_j as its input, actual computed output and expected output, respectively.

In this section only one particular Clifford Associator will be studied. This will be the Dual Clifford Associator of size 3×3 (shortly referred to as $DCA_{3,3}$). This specific architecture is chosen in order to highlight another feature of Clifford neural computation. It offers the possibility to directly process other geometric entities besides points. In the following an example for the processing of lines is worked out.

There are many ways to represent lines [68]. A very old one are Plücker coordinates [66].

Definition 3.21 (Plücker Coordinates) *Let* l *be a line in Euclidean 3D space represented by a point* $p = \{p_1, p_2, p_3\}$ *lying on* l *and the direction vector* $q = \{q_1, q_2, q_3\}$ *of* l. *Let* $q\prime := p \times q$ *be the moment vector. Then the six element vector*

$$L := (q, q\prime) \tag{3.51}$$

gives the so–called Plücker coordinates of l.

The following connection between Plücker coordinates and dual numbers trace back to Clifford himself [19].

Proposition 3.22 *Let* $L_1 = (q_{11}, q_{12}, q_{13}, q_{11}\prime, q_{12}\prime, q_{13}\prime), L_2 = (q_{21}, q_{22}, q_{23}, q_{21}\prime, q_{22}\prime, q_{23}\prime)$ *be Plücker coordinates of two lines related by an Euclidean transformation. Then there exists a* 3×3 *dual number matrix* W *such that*

$$\begin{pmatrix} q_{21} + q_{21}\prime e_1 \\ q_{22} + q_{22}\prime e_1 \\ q_{23} + q_{23}\prime e_1 \end{pmatrix} = W \begin{pmatrix} q_{11} + q_{11}\prime e_1 \\ q_{12} + q_{12}\prime e_1 \\ q_{13} + q_{13}\prime e_1 \end{pmatrix} . \tag{3.52}$$

The explicit matrix representation of the 3D Euclidean group $E(3)$ in terms of dual number matrices is given, for example, in [78]. A detailed discussion about the advantages of (3.52) over the common representation of E(3) by homogenous coordinates and 4×4 matrices for robotics can be found in [71].

Transformations of lines under $E(3)$ can be learned by a Dual Clifford Associator $DCA_{3,3}$ if the lines are presented in accordance to Proposition 3.22. This is also possible by a Linear Associator $LA_{6,6}$. Note that the $DCA_{3,3}$ has 18 real parameters and the $LA_{6,6}$ has 36 real parameters (3.50). Both architectures were compared in the following experimental setup.

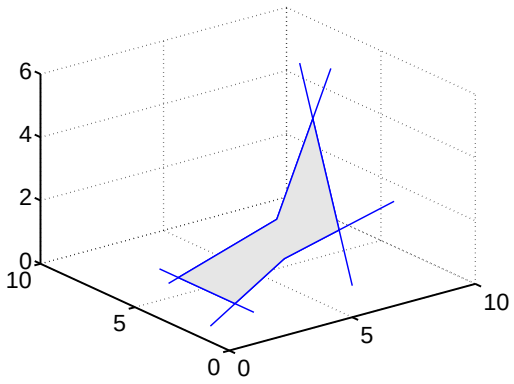

Figure 3.18: Training set for the Clifford Associator $DCA_{3,3}$ and the Linear Associator $LA_{6,6}$ consisting of 6 lines.

The input set consisted of the 6 lines shown in figure 3.18. A rotation about the axis $(2, 2, 3)$ and through 30 degrees followed by a translation by $(1, 2, 3)$ was chosen as learning task. Using batch learning with optimal learning rate the Linear Associator needed around 12000 epochs to reach a training error (MSE) less than 0.01. The same error level was reached by the $DCA_{3,3}$ in about 100 epochs. A comparison of the two learning curves as provided by figure 3.19 also shows, that the $DCA_{3,3}$ came up with a very good approach already after the first epoch. For the $DCA_{3,3}$ the same learning rate as for the $LA_{6,6}$ was used.

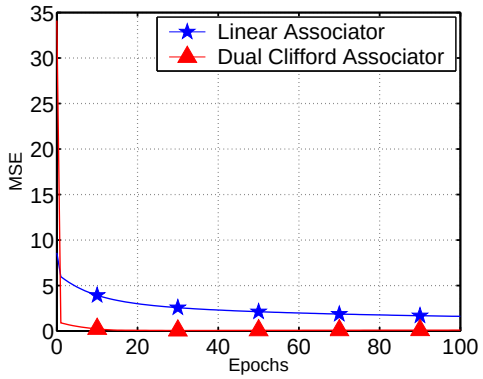

Figure 3.19: Training errors of the Dual Clifford Associator $DCA_{3,3}$ and the Linear Associator $LA_{6,6}$.

The Dual Clifford Associator also outperformed the Linear Associator in terms of generalization. Both networks were trained on data corrupted by Gaussian noise of different levels and then tested on the original noise–free data. The obtained results averaged over 20 runs are reported in figure 3.20. Not surprisingly, the Linear Associator basically learned the noise and only weak generalization did happen. In contrast, the Dual Clifford Associator was able to still generalize well even for higher levels of noise. This can be concluded from the fact that the MSE on the test set was always below the MSE on the training set, which has the following simple explanation. The $DCA_{3,3}$ computes the best fitting dual–number matrix W (3.52), and therefore representing the data in terms of Plücker coordinates (lines) forces a constraint optimization. Applying the same data to the $LA_{6,6}$, however, has noch such effect.

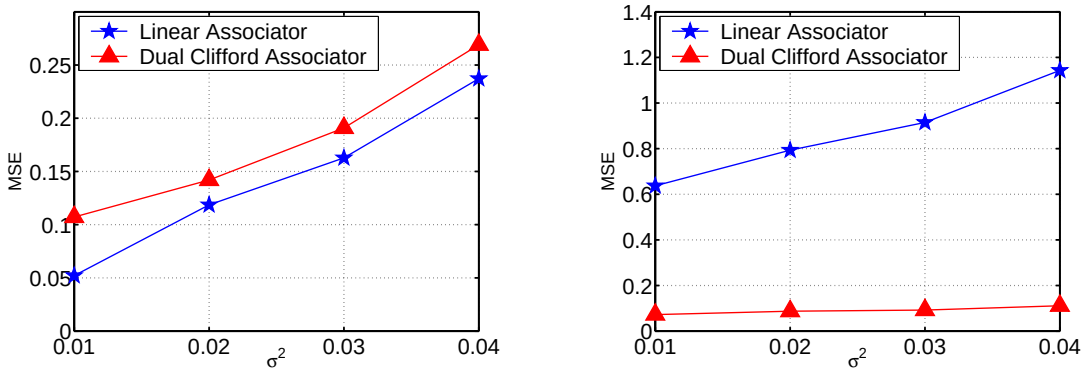

Figure 3.20: Performance of the Dual Clifford Associator $DCA_{3,3}$ and the Linear Associator $LA_{6,6}$ when trained on data with added Gaussian noise with variance σ^2. The left plot shows training performance and the right plot shows test performance (also in correspondence to the level of noise on the training set) on the original noise–free data.

The above experiment showed that, even if it is always theoretical possible to use a Linear Associator instead of a Clifford Associator, it may be totally unpractical for a particular task. Also, there is no generic way to present lines in an only real–valued architecture. However, this is exactly the sort of prior knowledge one might wish to communicate to a neural architecture. For example, many algorithms in computer vision are line based and deliver results in terms of lines (e.g. edge detection). One Clifford neural architecture can handle different geometric entities, or, more precisely, this feature is encoded via the geometric product inside the propagation function. The Dual Clifford Associator as introduced above can handle points as well. Everything only depends on the way the data is represented.

3.4 Summary of Chapter 3

In this chapter we undertook the first steps into the theory of Clifford neural computation. As its atom the Basic Clifford Neuron (BCN) was introduced. The computation of a BCN is solely determined by one geometric product associating one multivector weight with the received input. Particulary, the BCNs corresponding to the three two–dimensional Clifford algebras have been studied in great detail. Those neurons were compared in several experiments to the Linear Associator in order to highlight the model–based nature of Clifford neural computation. Theoretically, this was demonstrated by deriving the different optimal learning rates for those neurons (Proposition 3.7, Corollary 3.8, and Proposition 3.10) based on the

classical work of LeCun [51] for the Linear Associator. The standard error functions (SSE/MSE) were shown to be also the appropriate choice for the Clifford framework. From the particular nature of the derived update rule (3.28) for the Dual BCN the influence of the underlying degenerate Clifford algebra could be made visible.

The new theoretical concepts of isomorphic Basic Clifford Neurons and isomorphic representations have been presented. The first allows to identify equivalent neural Clifford architectures. This was proven theoretically and shown experimentally for the isomorphic neurons $\mathrm{BCN}_{1,1}$ and $\mathrm{BCN}_{2,0}$. Equivalence was shown to be hold up to the level of learning dynamics (Theorem 3.10) and (Corollary 3.11). Any Clifford neuron applies to the data it sees a certain geometric model induced by the underlying Clifford algebra. That fact allows a rich variety of data representations which then can be classified by the concept of isomorphic representations. As an example different representations for affine transformations have been studied. With both mentioned concepts the core set of tools for the systematic study of Clifford neural computation has been laid.

Finally, the Linear Associator was generalized to the Clifford Associator. The Dual Clifford Associator based on the algebra $\mathcal{C}_{0,0,1}$ was compared with the Linear Associator on data represented by lines. In an experiment that illustrates the capability of a Clifford architecture to intrinsical process lines, the Dual Clifford Associator outperformed the Linear Associator, with respect to both the number of needed training epochs and generalization capability.

Chapter 4

Spinor Clifford Neurons

In the previous chapter we utilized Clifford algebra to introduce neurons allowing for the computation of transformations of different geometric entities. Both the type of entities to process and the transformations to compute could be communicated to the neuron just by presenting the data in a certain specific way. Thus, although everything remained on a numerical level, structural a priori knowledge could be integrated. The study of this key feature of Clifford neural computation continues in this chapter. A second type of Clifford neuron is introduced, which has the following motivation.

The step from 2D Basic Clifford Neurons to Basic Clifford Neurons of higher dimensions yields only generalization in algebraical terms. What is still missing is generalization in terms of geometry. The propagation function of a Complex Basic Clifford Neuron, for example, is representing a 2D Euclidean transformation plus scaling, or, equivalently, the action of the corresponding transformation group on points in the plane. No Basic Clifford Neuron has a 3D Euclidean transformation as propagation function. This is due to the in general two–sided operation of the Clifford group (section 2.4), which prompts for the following new type of Clifford neuron.

Definition 4.1 (Spinor Clifford Neuron) *For any* $p + q > 1$ *a Spinor Clifford Neuron (SCN) computes the following function from* $\mathcal{C}_{p,q}$ *to* $\mathcal{C}_{p,q}$

$$y = w \otimes_{p,q} x \otimes_{p,q} \phi(w) + \theta\,, \tag{4.1}$$

with ϕ *being either inversion* $\tilde{}$*, reversion* $\hat{}$*, or conjugation* $\bar{}$*, respectively. For a particular Spinor Clifford Neuron the acronym* $\mathrm{SCN}^{\phi}_{p,q}$ *is used.*

A Spinor Clifford Neuron, basically, mimics the action of the Clifford group. There-

fore a SCN is only defined for non–degenerate Clifford algebras. Also, the propagation function (4.1) is only useful for non–commutative algebras, hence the condition $p + q > 1$. A SCN is a neural architecture based on two geometric products. However, its multivector weights, coupled by the involution ϕ, are not independent. Therefore a Spinor Clifford Neuron has actually only one independent multivector weight. Consequently, a SCN has the same number of free parameters as a BCN of same dimension. It is of course possible to study Clifford neurons with two uncoupled weights

$$y = w_1 \otimes_{p,q} x \otimes_{p,q} w_2 + \theta \,. \tag{4.2}$$

Here we will only consider SCNs due to their distinctive geometric properties, which do not apply for the more general neurons of the form (4.2).

To every Spinor Clifford Neuron the standard error function

$$\begin{aligned} E^s_{p,q} &= \frac{1}{2} \parallel d - w \otimes_{p,q} x \otimes_{p,q} \phi(w) + \theta \parallel_2 \\ &= \frac{1}{2} \parallel d - y \parallel_2 \\ &= \frac{1}{2} \parallel \text{error} \parallel_2 \end{aligned} \tag{4.3}$$

is associated. The bias term in (4.1) is common to all SCNs. The generic bias update rule for the Basic Clifford Neuron (3.3) does also apply in case of a SCN. The dynamics of the BCN and the SCN, however, differ as follows.

Proposition 4.2 *Let* $H_{BCN_{p,q}}$ *be the Hessian of a Basic Clifford Neuron* $BCN_{p,q}$*. Then*

$$H_{SCN^{\phi}_{p,q}} = 2\, H_{BCN_{p,q}} \tag{4.4}$$

is the Hessian of the Spinor Clifford Neuron $SCN^{\phi}_{p,q}$*.*

This gives immediately the following result.

Corollary 4.3 *Let* η_{opt} *be the optimal learning rate for a Basic Clifford Neuron* $BCN_{p,q}$*.Then* $\frac{1}{2}\eta_{opt}$ *is the optimal learning rate for the Spinor Clifford Neuron* $SCN^{\phi}_{p,q}$*.*

Both statements above are very intuitive because a SCN is linear and based on two geometric products.

The following discussion of Spinor Clifford Neurons follows the same structure used in chapter three for Basic Clifford Neurons. First, we will study the most

simple representative of a Spinor Clifford Neuron. That way all relevant aspects of Spinor Clifford Neurons can be introduced in an illustrative manner. Afterwards, the topic of isomorphic SCNs and isomorphic representations is studied in detail. The last section of this chapter presents a method for linearizing the computation of Möbius transformations by a single SCN using a conformal data representation.

4.1 The Quaternionic Spinor Clifford Neuron

The most simple Spinor Clifford Neurons are those of dimension four. Among them the SCN derived from the algebra of quaternions $\mathcal{C}_{0,2}$ is of particular interest and importance.

Definition 4.4 (Quaternionic Spinor Clifford Neuron) *The Quaternionic Spinor Clifford Neuron (Quaternionic SCN) computes the following function from $\mathcal{C}_{0,2}$ to $\mathcal{C}_{0,2}$*

$$y = w \otimes_{0,2} x \otimes_{0,2} \bar{w} + \theta \,. \tag{4.5}$$

The Quaternionic SCN is also the most exceptional Spinor neuron since it has the following unique properties. First, it can be used in the following canonical way (in accordance to the given motivation of SCNs). For any vector $x_1 e_1 + x_2 e_2$ the Quaternionic SCN computes a 2D orthogonal transformation followed by a translation. If the multivector weight w of a Quaternionic SCN is a unit quaternion then the propagation function is a 4D Euclidean transformation [27, 56]. If in the latter case a multivector $x_1 e_1 + x_2 e_2 + x_{12} e_{12}$ is presented a 3D Euclidean transformation results. Taken into account that quaternions are nowadays established as a fast way for computing 3D rotations the last presented fact is of most practical relevance.

Surprisingly, only ordinary quaternionic multiplication was considered as propagation function by Arena et al. [1] when proposing their Quaternionic MLP. Networks based on (4.5) will be introduced later in chapter five. At the moment we are only interested in the Quaternionic Spinor Clifford Neuron itself, for which the following update rule can be derived

$$\begin{aligned}
\Delta w_{0,2}{}^{-} &= \big(\mathrm{error}_0 w_0 x_0 + \mathrm{error}_1(w_0 x_1 + w_2 x_{12} - w_{12} x_2) \\
&+ \mathrm{error}_2(w_0 x_2 - w_1 x_{12} + w_{12} x_1) + \mathrm{error}_{12}(w_0 x_{12} + w_1 x_2 - w_2 x_1) e_0\big) \\
&+ \big(\mathrm{error}_0 w_1 x_0 + \mathrm{error}_1(w_1 x_1 + w_2 x_2 + w_{12} x_{12}) \\
&+ \mathrm{error}_2(w_2 x_1 - w_0 x_{12} - w_1 x_2) + \mathrm{error}_{12}(w_{12} x_1 + w_0 x_2 - w_1 x_{12}) e_1\big) \\
&+ \big(\mathrm{error}_0 w_2 x_0 + \mathrm{error}_1(-w_2 x_1 + w_1 x_2 + w_0 x_{12}) \\
&+ \mathrm{error}_2(w_1 x_1 + w_2 x_2 + w_{12} x_{12}) + \mathrm{error}_{12}(w_{12} x_2 - w_0 x_1 - w_2 x_{12}) e_2\big) \\
&+ \big(\mathrm{error}_0 w_{12} x_0 + \mathrm{error}_1(w_1 x_{12} - w_0 x_2 - w_{12} x_1) \\
&+ \mathrm{error}_2(w_0 x_1 + w_2 x_{12} - w_{12} x_2) + \mathrm{error}_{12}(w_1 x_1 + w_2 x_2 + w_{12} x_{12}) e_{12}\big) .
\end{aligned} \tag{4.6}$$

This rule is rather lengthly. Also, it is more complicated than necessary. As already mentioned, the architecture of a Spinor Clifford Neuron allows to think of the one multivector weight as two coupled multivector weights. In particular, it is then sufficient to derive only an update rule for the "right" weight in (4.1), and, then just set the "left" weight accordingly. That way only the update rule for one single geometric product is needed. This means that in principle the same update rule can be used for a BCN and a SCN with the same underlying algebra. For the Quaternionic SCN the $\mathrm{BCN}_{0,2}$ update rule applies, which reads

$$\begin{aligned}
\Delta w_{0,2} &= -\nabla E_{0,2} \\
&= (\mathrm{error} \cdot (x_0, x_1, x_2, x_{12})^T)\, e_0 \\
&+ (\mathrm{error} \cdot (-x_1, x_0, -x_{12}, x_2)^T)\, e_1 \\
&+ (\mathrm{error} \cdot (-x_2, x_{12}, x_0, -x_1)^T)\, e_2 \\
&+ (\mathrm{error} \cdot (-x_{12}, -x_2, x_1, x_0)^T)\, e_{12} \\
&= \mathrm{error} \otimes_{0,2} (x_0 e_0 + x_1 e_1 + x_2 e_2 - x_{12} e_{12}) \\
&= \mathrm{error} \otimes_{0,2} \bar{x} .
\end{aligned} \tag{4.7}$$

The precise procedure for determining $\Delta w_{0,2}{}^{-}$ from $\Delta w_{0,2}$ is then as follows. First determine the update of the "right" weight according to (4.7), whereby the order of the factors has to be reversed. This yields

$$w_c := \overline{(w \otimes_{0,2} x)} \otimes_{0,2} \mathrm{error} , \tag{4.8}$$

and, finally

$$\Delta w_{0,2}{}^{-} = \overline{w_c} . \tag{4.9}$$

In order to demonstrate the equivalence of the two update methods (4.6) and (4.7) the following experiment was performed using online learning[1]. A random distribution of 100 4D points $\{(x_1^p, x_2^p, x_3^p, x_4^p)\}_{p=1}^{100}$ with zero mean, variance one and standard deviation one was created. Interpreted as quaternions, this set of points was conjugated by a unit quaternion corresponding to a rotation about the axis $(2, 2, 3)$ and through 30 degrees and finally translated about $(0, 2, 2, -1)$. On that created training set several runs of a Quaternionic SCN with the two different update methods (4.6) and (4.7) were performed using a fixed initial multivector weight and a fixed learning rate $\eta = 0.2$. All runs converged to the solution in the same number of iterations. Differences of the learning curves could only be observed by explicitly studying absolute deviation between runs. Two examples are reported in figure 4.1 below. As can be seen from that plot the deviation for runs with different update methods was in the same range as that in between runs using (4.6). Thus both update methods are indeed equivalent. The observed differences between the runs were caused by using training data with precision of 15 digits, which means that training took place close to machine precision.

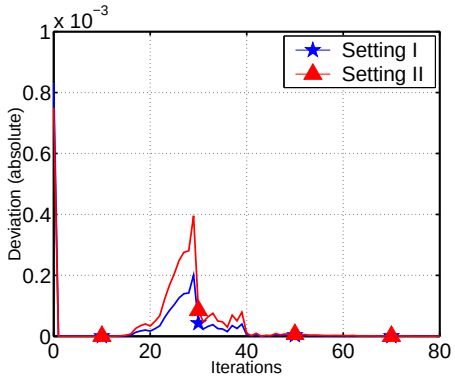

Figure 4.1: Absolute difference of training errors between two runs of the Quaternionic Spinor Clifford Neuron. In one case (referred to as setting I) update rule (4.6) was used for both runs, in the other case (referred to as setting II) update rules (4.6) and (4.7) were used in the first and second run, respectively.

In a second experiment only the last three components of the above training data sets were used. Thus the learning task was changed to be a 3D Euclidean transformation. On that task the Quaternionic SCN was compared with a Linear Associator $LA_{4,4}$ for which the bias term was incorporated as described in chapter three. Online learning was used with a learning rate $\eta = 0.2$ for the $LA_{4,4}$ and a learning rate

[1]Remember that in the regime of online learning a weight update is performed after every single presentation of a pattern, which is referred to as one iteration of the training.

$\eta = 0.1$ for the Quaternionic SCN, respectively. The resulting learning curves are presented in figure 4.2.

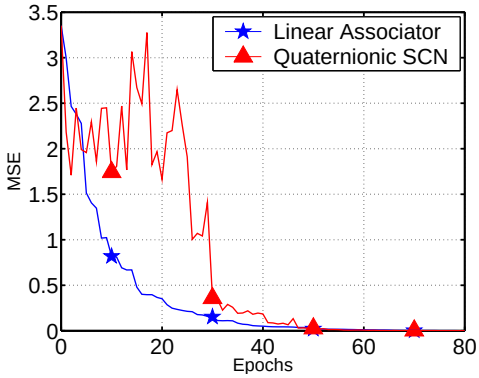

Figure 4.2: Training errors of the Quaternionic SCN and the Linear Associator $LA_{4,4}$.

Although oscillating at the beginning of training the Quaternionic SCN converged faster than the $LA_{4,4}$. After 50 training patterns the Quaternionic SCN has reached an error below 10^{-5}, for which the $LA_{4,4}$ needed about 70 patterns. Numerical results for noisy data are reported below in figure 4.3. The results are pretty similar to all the ones obtained in previous experiments. The Quaternionic SCN outperforms the general linear architecture in terms of generalization from noisy data. Clearly, this is due to the Quaternionic SCN possesses the right model for the data.

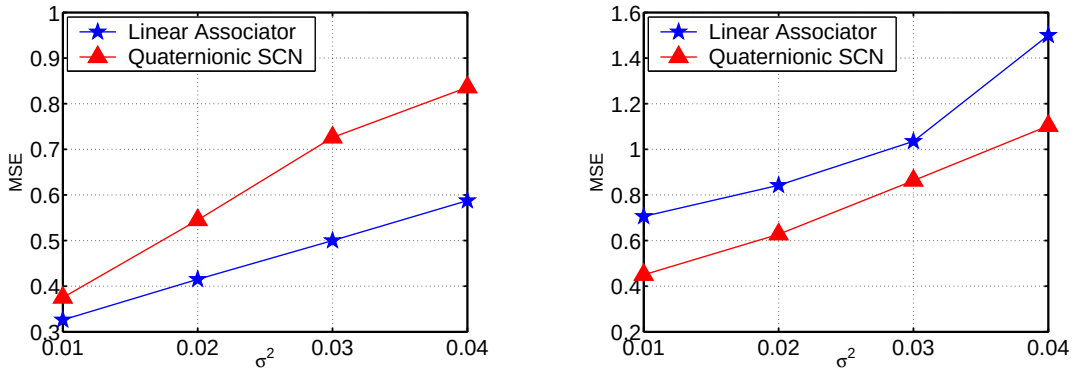

Figure 4.3: Performance of the Quaternionic SCN and the Linear Associator $LA_{4,4}$ when trained on data with added Gaussian noise with variance σ^2. The left plot shows training performance and the right plot shows test performance (in correspondence to the level of noise on the training set) on the original noise–free data.

The amount of outperforming, however, becomes really impressive when looking at the transformed test data itself, which consisted of the vertices of the 3D object already known from section 3.3, figure 3.18. The position and orientation of the object is not well recovered by the $LA_{4,4}$ (see figure 4.4 (right)), whereas the task was still managed satisfactory by the Quaternionic SCN.

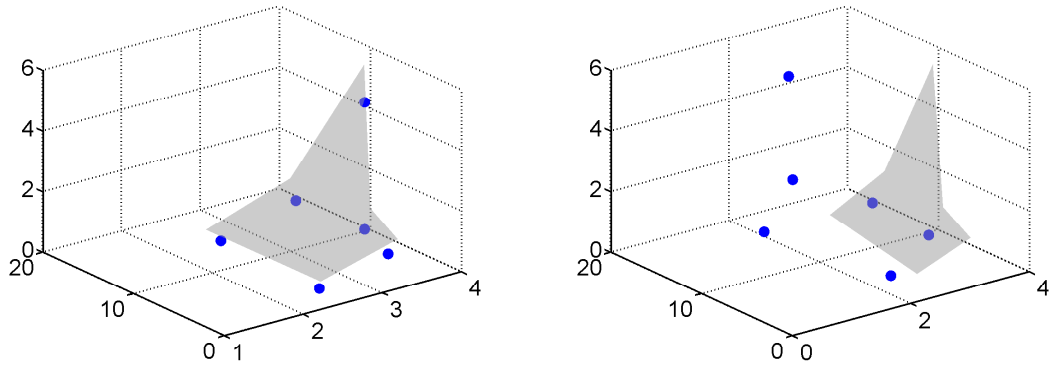

Figure 4.4: Learned transformations by the Quaternionic SCN (left) and the Linear Associator $LA_{4,4}$ (right) from noisy data ($\sigma^2 = 0.03$). Points (learned output) should lie on the vertices of the shaded object (expected output).

The main result of this section, however, is that the update rule for a SCN can be expressed very efficiently by using that of the BCN with same underlying algebra.

4.2 Isomorphic Spinor Clifford Neurons

One of the main results of chapter three was that Clifford neural architectures which only differ by isomorphic Basic Clifford Neurons are identical. Spinor Clifford Neurons, however, are not based on Basic Clifford Neurons. This would only be true for neurons of the form (4.2) with two independent weights. A Spinor Clifford Neuron, however, has two dependent weights which are coupled by an involution as introduced in Definition 4.1. This makes the SCN a Clifford neuron of its own rights. Therefore, the question of isomorphic Spinor Clifford Neurons is not covered by the aforementioned framework.

Since Spinor Clifford Neurons are determined by both a particular geometric product and a particular involution it is helpful to have an explicit notion of the latter. The involutions occurring for SCNs of dimension four and eight are given below in table 4.1 and table 4.2, respectively.

	e_0	e_1	e_2	e_{12}
Reversion ($\hat{}$)	+	+	+	-
Inversion ($\tilde{}$)	+	-	-	+
Conjugation ($\bar{}$)	+	-	-	-

Table 4.1: Involutions for Clifford algebras of dimension four.

	e_0	e_1	e_2	e_3	e_{12}	e_{13}	e_{23}	e_{123}
Reversion ($\hat{}$)	+	+	+	+	-	-	-	-
Inversion ($\tilde{}$)	+	-	-	-	+	+	+	-
Conjugation ($\bar{}$)	+	-	-	-	-	-	-	+

Table 4.2: Involutions for Clifford algebras of dimension eight.

For every Clifford algebra of dimension greater than two the occurring involutions are distinct from each other. This easily prompts the following statement.

Proposition 4.5 *Let* $\mathrm{SCN}^{\phi}_{p,q}$ *and* $\mathrm{SCN}^{\psi}_{p,q}$ *be two isomorphic Spinor Clifford Neurons. Then* $\phi = \psi$.

Thereby, two SCNs are regarded as being isomorphic in the same sense as before (section 3.2). Particularly, Theorem (3.10) holds analogously. The statement of Proposition 4.5 can be further generalized as follows.

Proposition 4.6 *Let* $\mathrm{SCN}^{\phi}_{p,q}$ *and* $\mathrm{SCN}^{\psi}_{p',q'}$ *be two isomorphic Spinor Clifford Neurons. Then* $\phi = \psi$ *and* $\mathcal{C}_{p,q} \cong \mathcal{C}_{p',q'}$.

Thus, two SCNs can only be isomorphic if the underlying Clifford algebras are isomorphic too. For example, $\mathrm{SCN}^{\tilde{}}_{2,0}$ is not isomorphic to the Quaternionic SCN. Otherwise, $\mathcal{C}_{2,0}$ would be isomorphic to the Clifford algebra of quaternions $\mathcal{C}_{0,2}$. Underlying isomorphic Clifford algebras are necessary but not sufficient for two SCNs to be isomorphic. This can be seen from the example of the $\mathrm{SCN}^{\tilde{}}_{1,1}$ and the $\mathrm{SCN}^{\tilde{}}_{2,0}$. To be able to actually perform computations the following two isomorphisms from the underlying Clifford algebras of these neurons to $\mathbb{R}(2)$ are needed first.

Proposition 4.7 *The map* ψ_{11}

$$\mathcal{C}_{1,1} \to \mathbb{R}(2),\ x_0e_0 + x_1e_1 + x_2e_2 + x_{12}e_{12} \mapsto \begin{pmatrix} x_0 + x_{12} & x_1 - x_2 \\ x_1 + x_2 & x_0 - x_{12} \end{pmatrix} \tag{4.10}$$

is an isomorphism.

Proposition 4.8 *The map* ψ_{20}

$$\mathcal{C}_{2,0} \to \mathbb{R}(2),\ x_0e_0 + x_1e_1 + x_2e_2 + x_{12}e_{12} \mapsto \begin{pmatrix} x_0 + x_1 & x_2 + x_{12} \\ x_2 - x_{12} & x_0 - x_1 \end{pmatrix} \tag{4.11}$$

is an isomorphism.

Let ϕ_{11} denote the isomorphism from $\mathbb{R}(2)$ to $\mathcal{C}_{1,1}$ (3.42) and ϕ_{20} denote the isomorphism from $\mathbb{R}(2)$ to $\mathcal{C}_{2,0}$ (3.48), respectively.
Furthermore, set $W := \begin{pmatrix} 3 & 4 \\ -7 & 8 \end{pmatrix}$ and $X := \begin{pmatrix} 2 & 12 \\ 3 & -5 \end{pmatrix}$. Then

$$\psi_{11}\big((\phi_{11}(W) \otimes_{1,1} \phi_{11}(X)) \otimes_{1,1} \phi_{11}(W)^{\tilde{}}\big) = \begin{pmatrix} 32 & 120 \\ -460 & -1062 \end{pmatrix},$$

whereas

$$\psi_{20}\big((\phi_{20}(W) \otimes_{2,0} \phi_{20}(X)) \otimes_{2,0} \phi_{20}(W)^{\tilde{}}\big) = \begin{pmatrix} 118 & 2 \\ -466 & -1032 \end{pmatrix}.$$

Therefore, the two Spinor Clifford Neurons $\mathrm{SCN}^{\tilde{}}_{1,1}$ and $\mathrm{SCN}^{\tilde{}}_{2,0}$ are not isomorphic. The same values for W and X as above can be used to show that $\mathrm{SCN}^{\hat{}}_{1,1}$ and $\mathrm{SCN}^{\hat{}}_{2,0}$ are also not isomorphic. Contrary to that the following result holds.

Proposition 4.9 *The neurons* $SCN^{\bar{}}_{1,1}$ *and* $SCN^{\bar{}}_{2,0}$ *are isomorphic.*

Proof: Let $w_{11} := w_0e_0 + w_1e_1 + w_2e_2 + w_{12}e_{12}$ be the weight of the $\mathrm{SCN}^{\bar{}}_{1,1}$. Then

$$\psi_{11}(\overline{w_{11}}) = \begin{pmatrix} w_0 - w_{12} & -w_1 + w_2 \\ w_1 - w_2 & w_0 + w_{12} \end{pmatrix} =: \overline{W_{11}}\,.$$

Furthermore, $w_{20} := w_0e_0 + w_{12}e_1 + w_1e_2 - w_2e_{12}$ is the image of w_{11} under the algebra isomorphism (3.29) from $\mathcal{C}_{1,1}$ to $\mathcal{C}_{2,0}$. Now direct computation shows

$$\begin{aligned} \psi_{20}(\overline{w_{20}}) &= \psi_{20}(w_0e_0 - w_{12}e_1 - w_1e_2 + w_2e_{12}) \\ &= \begin{pmatrix} w_0 - w_{12} & -w_1 + w_2 \\ w_1 - w_2 & w_0 + w_{12} \end{pmatrix} \\ &= \overline{W_{11}}\,. \end{aligned}$$

□

All of the above can be summarized as follows.

Theorem 4.10 *The only 4–dimensional isomorphic Spinor Clifford Neurons are* $\mathrm{SCN}^{-}_{1,1}$ *and* $\mathrm{SCN}^{-}_{2,0}$.

The fact that $\mathcal{C}_{1,2} \cong \mathbb{C}(2) \cong \mathcal{C}_{3,0}$ can be utilized to derive a similar theorem for the eight–dimensional case, for which the easy proof by direct computation will be omitted here.

Theorem 4.11 *The only 8–dimensional isomorphic Spinor Clifford Neurons are* $\mathrm{SCN}^{-}_{1,2}$ *and* $\mathrm{SCN}^{-}_{3,0}$.

Since explicitly designed for modelling the operation of the Clifford group, the standard data for Spinor Clifford Neurons are vectors. Therefore, for example,

$$(4) \;:\; \{(0e_0 + x_1e_1 + x_2e_2 + 0e_{12})^p, (0e_0 + d_1e_1 + d_2e_2 + 0e_{12})^p\}$$

is the corresponding vector representation of four–dimensional Spinor Clifford Neurons. From the isomorphism of the Clifford groups

$$\Gamma_{0,2} \cong \Gamma_{2,0} \tag{4.12}$$

follows immediately

$$\mathrm{SCN}^{-}_{2,0}(4) \cong \mathrm{SCN}^{-}_{0,2}(4)\,. \tag{4.13}$$

Note that in contrast $\mathrm{BCN}_{0,2} \ncong \mathrm{BCN}_{2,0}$ as well as $\mathrm{SCN}_{0,2} \ncong \mathrm{SCN}_{2,0}$. Hence (4.12) only induces isomorphic vector representations. Moreover, the vector representation is well distinguished from other possible representations, for example, those given by

$$\begin{aligned}
(1) &\;:\; \{(x_1e_0 + x_2e_1 + 0e_2 + 0e_{12})^p, (d_1e_0 + d_2e_1 + 0e_2 + 0e_{12})^p\}\\
(2) &\;:\; \{(x_1e_0 + 0e_1 + x_2e_2 + 0e_{12})^p, (d_1e_0 + 0e_1 + d_2e_2 + 0e_{12})^p\}\\
(3) &\;:\; \{(x_1e_0 + 0e_1 + 0e_2 + x_2e_{12})^p, (d_1e_0 + 0e_1 + 0e_2 + d_2e_{12})^p\}\\
(5) &\;:\; \{(0e_0 + x_1e_1 + 0e_2 + x_2e_{12})^p, (0e_0 + d_1e_1 + 0e_2 + d_2e_{12})^p\}\\
(6) &\;:\; \{(0e_0 + 0e_1 + x_1e_2 + x_2e_{12})^p, (0e_0 + 0e_1 + d_1e_2 + d_2e_{12})^p\}\,.
\end{aligned}$$

Proposition 4.12 *The vector representation* $SCN^{-}_{2,0}(4)$ *is unique in the following sense*

$$\mathrm{SCN}^{-}_{2,0}(4) \ncong \mathrm{SCN}^{-}_{0,2}(\mathfrak{i}) \tag{4.14}$$

for all $\mathfrak{i} \in \{(1), (2), (3), (5), (6)\}$.

Proof: The scalar component of

$$(w_0e_0+w_1e_1+w_2e_2+w_{12}e_{12})\otimes_{2,0}(x_1e_1+x_2e_2)\otimes_{2,0}(w_0e_0-w_1e_1-w_2e_2-w_{12}e_{12})$$

is zero where as it is nonzero for the spinor multiplications corresponding to the representations (1),(2),(3),(5) and (6). □

The above generalizes in that sense that there are much less isomorphic representations for Spinor Clifford Neurons than for Basic Clifford Neurons. When proceeding to Clifford neural networks later in this thesis we will concentrate on vector representations. For Euclidean transformations as propagation functions it is sufficient to study only Spinor Clifford Neurons based on conjugation. An application of a SCN based on reversion is given in the subsequent section.

4.3 Linearizing Möbius Transformations

Spinor Clifford Neurons allow the computation of general orthogonal transformations. For that purpose many different linear representations can be chosen as demonstrated before. A very interesting application of a nonlinear representation will be studied in the following. By using a so–called conformal representation[2] a Spinor Clifford Neuron based on reversion can compute a Möbius transformation in a linear way.

Definition 4.13 (Möbius Transformation) *For every* $a, b, c, d \in \mathbb{C}$ *with* $ad-bc \neq 0$ *the mapping*

$$\mathbb{C} \to \mathbb{C},\ z \mapsto \frac{az+b}{cz+d} \tag{4.15}$$

is called a Möbius transformation.

Möbius transformations are a specific class of holomorphic functions. They are of great importance in many mathematical branches. For example, they belong to the heart of monogenic functional calculus [45]. Moreover, Möbius transformations form the isometries of the Poincare model (unit disk) of the hyperbolic plane. Every Möbius transformation is a conformal mapping, that is, it preserves angles. Hence there is a close relation to orthogonal mappings. More precisely, the Möbius transformations (4.15) form a group M isomorphic to $O(1,2)/\{\pm 1\}$ (see e.g. [67]).

[2] In terms of Geometric Algebra this concept is referred to as the conformal model of the Euclidean Geometric Algebra (CGA) [40].

The Möbius group M can also be studied in terms of complex 2×2 matrices

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{4.16}$$

with $ad - bc \neq 0$ and ordinary matrix multiplication as group operation. Then, according to table 2.2, this is equivalently possible in either $\mathcal{C}_{1,2}$ or $\mathcal{C}_{3,0}$. Naturally, the Möbius group M acts on the Euclidean plane represented by complex numbers $\mathbb{C}$. To let M act properly on multivectors in $\mathcal{C}_{1,2}$ the following specific nonlinear embedding is needed.

Definition 4.14 (Conformal Compactification [67]) *For every* $z = x + iy \in \mathbb{C}$

$$xe_0 + \frac{1}{2}(1 + z\bar{z})e_1 + ye_2 + \frac{1}{2}(1 - z\bar{z})e_3 \tag{4.17}$$

is called the conformal compactification of $\mathbb{R}^2$ *(identified with* $\mathbb{C}$*) in* $\mathcal{C}_{1,2}$*.*

Figuratively, the set of paravectors $e_0+e_1+e_2+e_3$ in $\mathcal{C}_{1,2}$ is a projective space for $\mathbb{R}^2$ where points are represented as quadric cones. It is easier to work with the matrix representation of (4.17), for which an isomorphism from $\mathcal{C}_{1,2}$ to $\mathbb{C}(2)$ has first to be established. The basic idea is to use the following correspondences

$$\begin{aligned} e_0 &\mapsto \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ e_1 &\mapsto \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ e_2 &\mapsto \begin{pmatrix} i & 0 \\ 0 & -i \end{pmatrix} \\ e_3 &\mapsto \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \end{aligned}$$

which induce the following result.

Proposition 4.15 *The map* $\phi : \mathcal{C}_{1,2} \to \mathbb{C}(2)$

$$x_0 + x_1e_1 + x_2e_2 + x_3e_3 + x_{12}e_{12} + x_{13}e_{13} + x_{23}e_{23} + x_{123}e_{123} \mapsto$$

$$\begin{pmatrix} (x_0 + x_{23}) + i(x_2 - x_{123}) & (x_1 - x_3) + i(-x_{12} - x_{23}) \\ (x_1 + x_3) + i(x_{12} - x_{23}) & (x_0 - x_{23}) + i(-x_2 - x_{123}) \end{pmatrix} \tag{4.18}$$

is an isomorphism.

The image of the conformal compactification (4.17) of a point z under the above isomorphism ϕ reads

$$\begin{pmatrix} z & z\bar{z} \\ 1 & \bar{z} \end{pmatrix} . \tag{4.19}$$

The following linearization of Möbius transformations in terms of the Clifford algebra $\mathcal{C}_{1,2}$ is then easily verified by direct computation (see e.g. [67]).

Proposition 4.16 *Let* $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ *be a complex matrix with* $ad - bc \neq 0$. *Let* z *be the conformal compactification of a point* $(x, y) \in \mathbb{R}^2$ *according to* (4.17). *This yields*

$$A\,\phi(z)\,\tilde{A} = \lambda \begin{pmatrix} z' & z'\bar{z'} \\ 1 & \bar{z'} \end{pmatrix} , \tag{4.20}$$

with $\frac{1}{\lambda}z'$ *being the conformal compactification of the image of* (x, y) *under the Möbius transformation induced by* A *and* $\lambda = \|cz + d\|^2$.

The general result for $\mathbb{R}^n$ was proven by Vahlen [87] more than hundred years ago. However, results on 2×2 Clifford matrices inducing Möbius transformations for arbitrary quadratic spaces $\mathbb{R}^{p,q}$ became known not before around the 1990s (see e.g. [20]).

The realization of Möbius transformations by the Spinor Clifford Neuron $\mathrm{SCN}^{\sim}_{1,2}$ is not so straightforward due to the existence of the real factor λ in (4.20). Of course, λ can always be integrated as activation function. This, however, renders the whole Clifford algebra approach more complicated than a direct implementation of (4.15). It is also possible to scale in preprocessing either the input or the output by λ. The drawback of that method is that it requires knowledge of the transformation parameters in advance. This can be overcome by performing the scaling inside the neuron. More precisely, the first component and the third component of the current weight w can be used to compute λ [12], by which the original input x is divided in a first step. If x' denotes such a scaled term the computation $w \otimes_{1,2} x' \otimes_{1,2} \tilde{w}$ then follows. Therefore only the weight update for the last geometric product has actually to be computed. This can be done by using the update rule

$$
\begin{aligned}
\Delta w_{1,2} &= -\nabla E_{1,2} \\
&= (\text{error} \cdot (x_0, x_1, x_2, x_3, x_{12}, x_{13}, x_{23}, x_{123})^T)\, e_0 \\
&+ (\text{error} \cdot (x_1, x_0, x_{12}, x_{13}, x_2, x_3, x_{123}, x_{23})^T)\, e_1 \\
&+ (\text{error} \cdot (-x_2, x_{12}, x_0, -x_{23}, -x_1, x_{123}, x_3, -x_{13})^T)\, e_2 \\
&+ (\text{error} \cdot (-x_3, x_{13}, x_{23}, x_0, -x_{123}, -x_1, -x_2, x_{12})^T)\, e_3 \\
&+ (\text{error} \cdot (x_{12}, -x_2, -x_1, x_{123}, x_0, -x_{23}, -x_{13}, x_3)^T)\, e_{12} \\
&+ (\text{error} \cdot (x_{13}, -x_3, -x_{123}, -x_1, x_{23}, x_0, x_{12}, -x_2)^T)\, e_{13} \\
&+ (\text{error} \cdot (-x_{23}, -x_{123}, -x_3, x_2, -x_{13}, x_{12}, x_0, x_1)^T)\, e_{23} \\
&+ (\text{error} \cdot (-x_{123}, -x_{23}, -x_{13}, x_{12}, -x_3, x_2, x_1, x_0)^T)\, e_{123} \\
&= \text{error} \otimes_{1,2} (x_0 e_0 + x_1 e_1 - x_2 e_2 - x_3 e_3 + x_{12} e_{12} + x_{13} e_{13} - x_{23} e_{23} - x_{123} e_{123})\,.
\end{aligned}
\tag{4.21}
$$

and then compute the dependent weight in a similar way as demonstrated for the Quaternionic SCN before (section 4.1).

For testing the above procedure the Möbius transformation

$$
z \mapsto \frac{0.5(1+i)z + 0.5(1-i)}{-0.5(1+i)z + 0.5(1-i)} \tag{4.22}
$$

was chosen as learning task. The input set consisted of 100 data points $\{(x_1^p, x_2^p)\}_{p=1}^{100}$ from a distribution having zero mean, variance one and standard deviation one. From those points the output set was created by applying transformation (4.22). For both sets the conformal compactification (4.17) was then computed resulting in the final sets for training. For stable convergence of batch learning a rather small learning rate $\eta = 0.001$ was necessary. The learning curve of the $\tilde{\text{SCN}}_{1,2}$ is presented in figure 4.5.

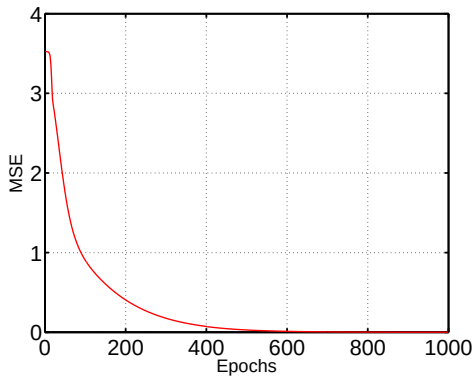

Figure 4.5: Training error of the $\tilde{\text{SCN}}_{1,2}$.

Although a lot more epochs were needed, compared to the typical numbers reported in the experiments on BCNs and SCNs we have done so far, learning succeeded with zero training error. Thus the exact parameters of the transformation were learned by the $\widetilde{\mathrm{SCN}}_{1,2}$. This demonstrates the indented capability of the neuron to perform Möbius transformations in a linear way when data is provided according to (4.17).

Another such application of the $\widetilde{\mathrm{SCN}}_{1,2}$ arises from the well–known relation between Möbius transformations and the cross–ratio of an ordered quadruple of points.

The cross–ratio of 4 points $z, q, r, s \in \mathbb{R}^2$ (identified with complex numbers) is given by [60]

$$[z, q, r, s] = \frac{(z-q)(r-s)}{(z-s)(r-q)}\,. \tag{4.23}$$

It is a well known projective invariant from which many others can be derived [90]. The precise relation to Möbius transformations is given by the next theorem.

Theorem 4.17 ([60]) *The cross–ratio* $[z, q, r, s]$ *is the image of* z *under that Möbius transformation that maps* q *to 0,* r *to 1 and* s *to* ∞*, respectively.*

The task now for the $\widetilde{\mathrm{SCN}}_{1,2}$ was to learn to compute the cross–ratio(s) $[z, q, r, s]$ with $q := 0.2 + \mathrm{i}\,0.3$, $r := 0.4 - \mathrm{i}\,0.7$, and $s := 0.6 - \mathrm{i}\,0.2$. That is after being trained on the set $\{(q, 0), (r, 1), (s, \infty)\}$[3] the neuron should be able to compute the cross–ratio $[z, q, r, s]$ when presented the test point z. Training was performed in batch mode with learning rate set to $\eta = 10^{-7}$. Due to the necessarily approximative coding of ∞ the task could not be learned exactly. Therefore training was stopped after the SSE dropped below 0.00001. This was the case after approximately 30000 epochs. A comparison between theoretical and learned transformation parameters is given in table 4.3.

Parameter	Theoretical value	Learned value
a	+0.20+ i 0.50	+0.20019 + i 0.50010
b	+0.11- i 0.16	+0.11001 - i 0.15992
c	-0.20+ i 1.00	-0.20079 + i 0.99939
d	-0.08- i 0.64	-0.07930 - i 0.64007

Table 4.3: Comparison between theoretical and learned transformation parameters.

[3] The entity ∞ was coded as 10^{15}.

The difference between the actual values and those learned by the neuron is quite small. Therefore an excellent performance on test points could be achieved as can be seen from table 4.4.

Test point	Theoretical value	$\widetilde{SCN}_{1,2}$ output value
2.0+i 3.0	+0.3578-i 0.3357	+ 0.3577-i 0.3364
4.0-i 7.0	+0.4838-i 0.3044	+ 0.4838-i 0.3051
0.3+i 0.1	+0.0077-i 0.6884	+ 0.0082-i 0.6884

Table 4.4: Comparison between theoretical cross–ratio values (rounded) and actual output cross–ratio values of the $\widetilde{SCN}_{1,2}$ for some test points.

4.4 Summary of Chapter 4

In this chapter we introduced the class of Spinor Clifford Neurons (SCN). This second fundamental type of Clifford neuron generalizes the two–dimensional Basic Clifford Neurons in a geometrical sense. The computation of a Spinor Clifford Neuron is an orthogonal transformation achieved by mimicking the two–sided operation of the Clifford group. Although therefore technically involving two Clifford products, a SCN has only one independent multivector weight. Thus a BCN and a SCN of same dimension have the same number of free parameters. A result relating the optimal learning rates of both types of Clifford neurons was presented (Corollary 4.3). A general method for reducing the complexity of the update rule for a SCN to that of BCN was derived. All of the above was demonstrated both theoretically and by experiments for the Quaternionic SCN. This particular neuron was also used to illustrate the general model–based nature of SCNs.

The question of isomorphic Spinor Clifford Neurons and isomorphic representations was studied. Necessary conditions for isomorphic SCNs were derived (Proposition 4.7), and the two neurons $\widetilde{SCN}_{1,1}$ and $\widetilde{SCN}_{2,0}$ were proven to be isomorphic. The so–called conformal representation was shown to enable a Spinor Clifford Neuron to compute Möbius transformations in a linear way. This unique property of the Clifford framework was utilized for neural computing of the cross–ratio.

Chapter 5

Clifford MLPs with Real–Valued Activation Functions

At the very beginning of chapter 3 we introduced $y = g(f(w; x))$ as the expression for a generic neuron. In that setting the Basic Clifford Neuron (BCN) reads (see Definition 3.2)

$$y = id(w \otimes_{p,q,r} x + \theta)\,, \tag{5.1}$$

and the Spinor Clifford Neuron (SCN) (see Definition 4.1) reads

$$y = id(w \otimes_{p,q} x \otimes_{p,q} \phi(w) + \theta)\,. \tag{5.2}$$

The Linear Associator, which was introduced in Definition 3.4, consists of one fully connected layer of real BCNs and, therefore, computes a linear mapping from $\mathbb{R}^n$ to $\mathbb{R}^m$

$$y = Wx\,, \tag{5.3}$$

with weight matrix $W := (w_{i,j})_{m,n}$. Adding another layer U is not beneficial since $y = U(Wx)$ would be of the same (linear) computational power as (5.3). If, however, a nonlinear activation function g in the first layer is used everything changes dramatically. For understanding the new neural architecture derived in that way

$$y = U(\sum_i g(w_{ij}x_j)) \tag{5.4}$$

it is totally sufficient to restrict (5.4) to the case of one–dimensional output (because only one type of activation function is used). Then the weight matrix U reduces to a weight vector u yielding the simpler expression

$$y = u_i(\sum_i g(w_{ij}x_j))\,. \tag{5.5}$$

Asking for the functional power of that expression is a purely mathematical problem, which is stated outside the actual neural framework. One possible mathematical view on (5.5) is that of a series with basis functions parameterised by the individual weights w_{ij}.

Suppose now we want to actually train the architecture (5.5) by gradient descent. For that purpose the nonlinear activation function g has to be bounded and differentiable everywhere as well. All this is fulfilled by the class of the so–called sigmoidal ("s–shaped") functions.

Definition 5.1 (Sigmoidal Function) *For every $\beta \neq 0$ the function $\sigma_\beta : \mathbb{R} \to \mathbb{R}$ defined by*

$$x \mapsto \frac{1}{1 + \exp(-\beta x)} \tag{5.6}$$

is called a sigmoidal function.

Of course, the "s–shaped" nature of sigmoidal functions is preserved under scaling and translation. The functions $2\sigma_2 - 1 = \tanh$ and σ_1 are plotted in figure 5.1 . The function σ_1 is known as the logistic function.

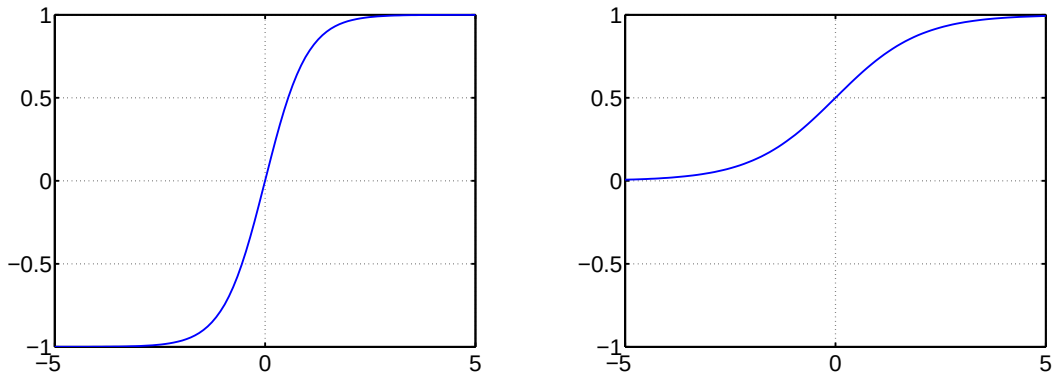

Figure 5.1: Plot of the tanh function (left) and the logistic function (right).

The (re)–discovery[1] of gradient descent for neural networks of type (5.4) with sigmoidal activation functions by Rumelhart et al. [57] gave rebirth to the whole area of neural computation and it meant nothing less than a revolution for the whole field of artificial intelligence. Since that time training such a network by gradient descent is known as Backpropagation and the network itself is famous as Multilayer Perceptron[2] (MLP). The number of published papers on MLPs have become

[1] See [36] for more historical details.

[2] Unfortunately, that name is quite misleading since the individual neurons are not Perceptrons.

"uncountable" with respect to both the range of applications (image processing, pattern recognition, speech recognition et cetera) and of research.

Soon after the publication of the Backpropagation algorithm the mathematical capabilities of MLPs have also been studied. A MLP with only one hidden layer of sigmoidal functions was proven to be a universal approximator for the class of continuous functions [23], which means that, theoretically, any such function can be learned arbitrarily well. Even more important for the fast growing popularity of the MLP was its ability to generalise well from training, which has been reported for many different tasks (see e.g. [9, 25, 69]). A comprehensive treatment of generalisation by MLPs can be found in [89].

From the formal point of view the MLP according to (5.4) can be easily generalised to a Clifford architecture by just replacing the real $\mathrm{BCN}_{0,0,0}$ with BCNs or SCNs of higher dimension. That is the activation function remains unchanged in the following sense. The same real–valued activation function g is applied to every component of the multivector argument. This yields (using the same notations as in Proposition 2.24) the following expression for the multivector output of a Clifford neuron (based on a BCN) in the first layer

$$\sum_{I\in\mathcal{I}} g((w \otimes_{p,q,r} x + \theta)_I)e_I\,. \tag{5.7}$$

A further step of generalisation could be the use of a Clifford–valued activation function. For example, the complex logistic function

$$z \mapsto \frac{1}{1+\exp(z)} \tag{5.8}$$

could be used as complex–valued activation function. Clifford Multilayer Perceptrons with Clifford–valued activation functions are studied in the subsequent chapter. In this chapter we will deal exclusively with the case of Clifford MLPs with real–valued activation functions. The straightforward definition of such a Clifford neural network is given below.

Definition 5.2 (Clifford MLP with Real–Valued Activation Functions) *A Clifford MLP with real–valued activation functions (CMLP) computes a function from $(\mathcal{C}_{p,q,r})^n$ to $(\mathcal{C}_{p,q,r})^m$ by passing the input through a set $\{1,\cdots,L\}$ of fully connected consecutive layers. The output of the j-th neuron in the l-th layer reads*

$$\sum_{I\in\mathcal{I}} g^{(l)}(\sum_k (w_{kj}^{(l)} \otimes_{p,q,r} x_k^{(l-1)} + \theta_j^{(l)})_I)e_I\,. \tag{5.9}$$

Thereby $x_k^{(l-1)}$ *denotes the* k*-th input from the previous layer (*$x_k^{(0)}$ *referring to the input of the network itself) and* $g^{(l)}$ *denotes the real–valued activation function used for all neurons in the* l*-th layer.*

Using SCNs instead of BCNs gives a Spinor Clifford MLP.

Definition 5.3 (Spinor Clifford MLP with Real–Valued Activation Functions) *A Spinor Clifford MLP with real–valued activation functions (SCMLP) computes a function from* $(\mathcal{C}_{p,q})^n$ *to* $(\mathcal{C}_{p,q})^m$ *by passing the input through a set* $\{1, \cdots, L\}$ *of fully connected consecutive layers. The output of the* j*-th neuron in the* l*-th layer reads*

$$\sum_{I \in \mathcal{I}} g^{(l)}\Big(\sum_k (w_{kj}^{(l)} \otimes_{p,q} x_k^{(l-1)} \otimes_{p,q} \phi(w_{kj})^{(l)} + \theta_j^{(l)})_I\Big) e_I \,, \tag{5.10}$$

and ϕ *refers to one of the three possible involutions (conjugation, reversion, and inversion).*

The last layer of a (S)CMLP is called the output layer, all other layers are called hidden ones[3]. The neurons of a (S)CMLP are also called nodes. Mostly, we will deal with two–layer (S)CMLPs. Two–layer CMLPs will be abbreviated as $\mathrm{MLP}_{p,q,r}^{g_1,g_2}$ and two–layer SCMLPs which are based on conjugation will be abbreviated as $\mathrm{SMLP}_{p,q}^{g_1,g_2}$, respectively. In both cases g_i stands for the function in the i–th layer. If no functions are given $g_1 = \sigma_1$ and $g_2 = \mathrm{id}$ is assumed. The only parameter controlling the complexity of a two–layer (S)CMLP is the number of hidden nodes, h. The total number of real parameters of such a network equals (set $r = 0$ for a SCMLP)

$$\mathcal{N}_{(S)CMLP} = 2^{p+q+r} \cdot h \cdot (n+1) + 2^{p+q+r} \cdot m \cdot (h+1) \,. \tag{5.11}$$

The chapter now proceeds as follows. The first section deals with the derivation of the Backpropagation algorithm for (S)CMLPs, followed by discussing universal approximation in the second one. In the last section experimental results are reported.

5.1 Backpropagation Algorithm

Training a Multilayer Perceptron using Backpropagation basically means to perform steepest gradient descent on a given error function with respect to the weights. Thereby the errors are propagated backwards through the network, thus the name

[3]Remember that we do not count the network input as layer.

is meant literally. For the existence of all involved partial derivatives it is sufficient that all real–valued activation functions are differentiable everywhere. To actually apply the algorithm its is additionally necessary that those functions and their derivatives are bounded. Complex Backpropagation has been first published in [54], the quaternionic extension was introduced in [1].

Backpropagation for Clifford MLPs will be studied first. So define for the activation of the j-th node in the l-th layer

$$s_j^{(l)} := \sum_k w_{kj}^{(l)} \otimes_{p,q,r} x_k^{(l)} + \theta_j^{(l)}\,, \tag{5.12}$$

and for the output of that node[4]

$$y_j^{(l)} := \sum_{I\in\mathcal{I}} g([s_j^{(l)}]_I) e_I\,. \tag{5.13}$$

The standard sum–of–squares error function for the CMLP then reads

$$E := \frac{1}{2} \parallel d - y \parallel_2\,, \tag{5.14}$$

whereby $y = (y_1^{(L)}, y_2^{(L)}, \ldots)$ is the vector of the single outputs. Starting at the output layer, we have to compute

$$\nabla E_{w_{kj}^{(L)}} = \sum_{I\in\mathcal{I}} \frac{\partial E}{\partial [w_{kj}^{(L)}]_I} e_I\,. \tag{5.15}$$

Applying the chain rule to each term of (5.15) yields

$$\frac{\partial E}{\partial [w_{kj}^{(L)}]_I} = \sum_{B\in\mathcal{I}} \frac{\partial E}{\partial [s_j^{(L)}]_B} \frac{\partial [s_j^{(L)}]_B}{\partial [w_{kj}^{(L)}]_I}\,. \tag{5.16}$$

Therefore the computation can be split into a activation function part and a propagation function part. For a single partial derivative of the error function with respect to the node activation we obtain

$$\frac{\partial E}{\partial [s_j^{(L)}]_B} = \frac{\partial E}{\partial [y_j]_B} \frac{\partial [y_j]_B}{\partial [s_j^{(L)}]_B} = ([d_j]_B - [y_j]_B)\,(g^{(L)})'([s_j^{(L)}]_B)\,, \tag{5.17}$$

and hence

$$\sum_{B\in\mathcal{I}} \frac{\partial E}{\partial [s_j^{(L)}]_B} = ([d_j]_B - [y_j]_B)\,(g^{(L)})'([s_j^{(L)}]_B) e_B =: \delta_j^{(L)}. \tag{5.18}$$

[4]To apply projection to a multivector with subindexes we write $[\,]_I$ for better readability from now on.

For many Clifford algebras we already know the remaining propagation function term $\frac{\partial [s_j^{(L)}]_B}{\partial [w_{kj}^{(L)}]_I}$ from chapter 3 and thus (5.16) completely. For those algebras the update rules for the weights of the output layer of the corresponding Clifford MLPs are then as follows

$$\mathcal{C}_{1,0,0}: \quad \Delta w_{kj}^{(L)} \underset{(3.24)}{=} \delta_j^{(L)} \otimes_{1,0,0} y_k^{(L-1)} \tag{5.19}$$

$$\mathcal{C}_{0,1,0}: \quad \Delta w_{kj}^{(L)} \underset{(3.19)}{=} \delta_j^{(L)} \otimes_{0,1,0} \overline{y_k^{(L-1)}} \tag{5.20}$$

$$\mathcal{C}_{0,0,1}: \quad \Delta w_{kj}^{(L)} \underset{(3.28)}{=} ([\delta_j^{(L)}]_0 [y_k^{(L-1)}]_0 + [\delta_j^{(L)}]_1 [y_k^{(L-1)}]_1) e_0 + ([\delta_j^{(L)}]_1 [y_k^{(L-1)}]_0) e_1 \tag{5.21}$$

$$\mathcal{C}_{0,2,0}: \quad \Delta w_{kj}^{(L)} \underset{(4.7)}{=} \delta_j^{(L)} \otimes_{0,2,0} \overline{y_k^{(L-1)}} \tag{5.22}$$

$$\mathcal{C}_{1,1,0}: \quad \Delta w_{kj}^{(L)} \underset{(3.30)}{=} \delta_j^{(L)} \otimes_{1,1,0} (y_0^{(L-1)} e_0 + y_1^{(L-1)} e_1 - y_2^{(L-1)} e_2 + y_{12}^{(L-1)} e_{12}) \tag{5.23}$$

$$\mathcal{C}_{2,0,0}: \quad \Delta w_{kj}^{(L)} \underset{(3.31)}{=} \delta_j^{(L)} \otimes_{2,0,0} (y_0^{(L-1)} e_0 + y_1^{(L-1)} e_1 + y_2^{(L-1)} e_2 - y_{12}^{(L-1)} e_{12}) \,. \tag{5.24}$$

For updating the weights in a hidden layer l we have to compute analogously to (5.15)

$$\nabla E_{w_{kj}^{(l)}} = \sum_{I \in \mathcal{I}} \frac{\partial E}{\partial [w_{kj}^{(l)}]_I} e_I \,, \tag{5.25}$$

yielding now

$$\frac{\partial E}{\partial [w_{kj}^{(l)}]_I} = \sum_{B \in \mathcal{I}} \frac{\partial E}{\partial [s_j^{(l)}]_B} \frac{\partial [s_j^{(l)}]_B}{\partial [w_{kj}^{(l)}]_I} \,. \tag{5.26}$$

Comparing (5.26) with (5.16) we see that the only[5] remaining task is the computation of the new error terms $\frac{\partial E}{\partial [s_j^{(h)}]_B}$, which now involves the error terms from the $(h+1)$-th layer. This results in slightly more complicated expressions. For example, for the Clifford MLP with underlying degenerate Clifford algebra $\mathcal{C}_{0,0,1}$ the hidden layer error terms read

$$\delta_j^{(l)} := \sum_{B \in \mathcal{I}} \frac{\partial E}{\partial [s_j^{(l)}]_B} = \frac{\partial E}{\partial [s_j^{(l)}]_0} + \frac{\partial E}{\partial [s_j^{(l)}]_1} \,, \tag{5.27a}$$

with

$$\begin{aligned} \frac{\partial E}{\partial [s_j^{(l)}]_0} &= \Big(\big(\sum_m [w_{jm}^{(l)}]_0 [\delta_m^{(l+1)}]_0 \big) \big((g^{(l)})'([s_j^{(l)}]_m) \big) \\ &+ \big(\sum_m [w_{jm}^{(l)}]_1 [\delta_m^{(l+1)}]_1 \big) \big((g^{(l)})'([s_j^{(l)}]_m) \big) \Big) e_0 \end{aligned} \tag{5.27b}$$

[5] This is because all neurons have the same propagation function regardless of the layer.

and

$$\frac{\partial E}{\partial [s_j^{(l)}]_1} = \Big(\big(\sum_m [w_{jm}^{(l)}]_0 [\delta_m^{(l+1)}]_1 \big) \big((g^{(l)})'([s_j^{(l)}]_m) \big) \Big) e_1 \,. \tag{5.27c}$$

Finally, putting together (5.21) and (5.27a), the whole Backpropagation algorithm for the $\mathrm{MLP}_{0,0,1}$ then reads

$$\Delta w_{kj}^{(l)} = ([\delta_j^{(l)}]_0 [y_k^{(l-1)}]_0 + [\delta_j^{(l)}]_1 [y_k^{(l-1)}]_1) e_0 + ([\delta_j^{(l)}]_1 [y_k^{(l-1)}]_0) e_1 \tag{5.28a}$$

$$\Delta \theta_j^{(l)} = \delta_j^{(l)} \,, \tag{5.28b}$$

with $\delta_j^{(l)}$ as in (5.18) if $l = L$, and $\delta_j^{(l)}$ as in (5.27a) otherwise. The simple update rule (5.28b) for the bias terms comes from the fact that such nodes can be viewed as having constant input 1. Hence (5.28b) holds for any (Spinor) Clifford MLP. The complete Backpropagation algorithm for Clifford MLPs with arbitrary underlying non–degenerate Clifford algebra is stated next.

Theorem 5.4 (Backpropagation for Non–Degenerate Clifford Algebras) *For every Clifford MLP defined over a non–degenerate Clifford algebra $\mathcal{C}_{p,q}$ there exists a unique involution*

$$^* : \mathcal{C}_{p,q} \to \mathcal{C}_{p,q}, x \mapsto x^* \tag{5.29}$$

such that

$$\Delta w_{kj}^{(l)} = \delta_j^{(l)} \otimes_{p,q} (y_k^{(l-1)})^* \tag{5.30a}$$

$$\Delta \theta_j^{(l)} = \delta_j^{(l)} \,, \tag{5.30b}$$

with

$$\delta_j^{(l)} = \begin{cases} \sum_{I \in \mathcal{I}} (g^{(L)})' \, (d_j - y_j^{(L)})_I e_I & \text{if } \; l = L, \\ \sum_{I \in \mathcal{I}} (g^{(l)})' \, (\sum_m (w_{jm}^{(l+1)})^* \otimes_{p,q} \delta_m^{(l+1)})_I e_I & \text{otherwise.} \end{cases} \tag{5.31}$$

The involution * fulfills

$$[a \otimes_{p,q} b^*]_0 = < a, b > \tag{5.32}$$

for all $a, b \in \mathcal{C}_{p,q}$.

Proof: Setting $^* = \mathrm{id}$ for the real case $\mathcal{C}_{0,0}$ gives the ordinary Backpropagation. Using conjugation ($^* = \bar{}$) for $\mathcal{C}_{0,1}$ and $\mathcal{C}_{0,2}$ gives complex Backpropagation and quaternionic Backpropagation, respectively. The involution * is already determined by any of the partial derivatives $\frac{\partial [s_j^{(l)}]_B}{\partial [w_{kj}^{(l)}]_B}$, particularly by $\frac{\partial [s_j^{(l)}]_0}{\partial [w_{kj}^{(l)}]_0}$. From that follows that * is the conjugation for every algebra $\mathcal{C}_{0,q}$. The product of any two fixed basis vectors in two Clifford algebras of the same dimensions differs only about the sign (due to the basis construction (2.18)). This is then also true for the partial

derivatives $\frac{\partial[s_j^{(l)}]_B}{\partial[w_{kj}^{(l)}]_B}$ in the same setting. Hence there always exists an involution as asserted.

□

Computing the uniquely determined involution fulfilling (5.32) for a particular Clifford algebra is a rather simple task. This was done in table 5.1 for the four–dimensional case and in table 5.2 for the eight–dimensional case, respectively. This finishes our discussion of Backpropagation for Clifford MLPs.

	e_0	e_1	e_2	e_{12}
$CMLP_{2,0}$	+	**+**	**+**	-
$CMLP_{1,1}$	+	**+**	**-**	+
$CMLP_{0,2}$	+	**-**	**-**	-

Table 5.1: The uniquely determined involutions fulfilling (5.32) for all non–degenerate Clifford algebras of dimension four. The signs of the vector components are printed in boldface in order to highlight the generic nature of the computation.

	e_0	e_1	e_2	e_3	e_{12}	e_{13}	e_{23}	e_{123}
$CMLP_{3,0}$	+	**+**	**+**	**+**	-	-	-	-
$CMLP_{2,1}$	+	**+**	**+**	**-**	-	+	+	+
$CMLP_{1,2}$	+	**+**	**-**	**-**	+	+	-	-
$CMLP_{0,3}$	+	**-**	**-**	**-**	-	-	-	+

Table 5.2: The uniquely determined involutions fulfilling (5.32) for all non–degenerate Clifford algebras of dimension eight. The signs of the vector components are printed in boldface in order to highlight the generic nature of the computation.

The derivation of the Backpropagation algorithm for Spinor Clifford MLPs is somehow more complicated. The update rule for a Spinor Clifford Neuron could be simplified by using the update rule of the corresponding Basic Clifford Neuron "twice" (section 4.1). This, however, was only possible because of the absence of a nonlinear activation function in a SCN. Hence this "trick" cannot be applied in case of a Spinor Clifford MLP, which renders the formulation of one general Spinor Backpropagation algorithm impossible. Also, any Backpropagation algorithm for

a particular SCMLP becomes rather lengthy. As an example, the update rule for the Quaternionic SCMLP is given in appendix A.2.

5.2 Universal Approximation

Universal approximation is a rather abstract and complicated topic. Hence many different opinions regarding its relevance can be found in the literature. It is surely not the most important property of a neural network from the practical point of view. However, it was and is of great importance for the development of the theory of neural computation. Therefore, besides proving results on the function approximation capabilities of Clifford MLPs, we also want to give an idea of the concept itself in this section.

The concept of universal approximation first entered the neural network world in 1987 [36], where the following famous theorem by Kolmogorov has been used for its motivation.

Theorem 5.5 (Kolmogorov [47]) *Any continuous function of* d *variables from a closed and bounded input domain can be exactly represented as a superposition of the form*

$$y = \sum_{j=1}^{2d+1} g\Big(\sum_{i=1}^{d} \lambda_i h_j(x_i) \Big) , \tag{5.33}$$

with h_j *being strictly monotonic functions and constants* $\lambda_i \in (0, 1)$.

The connection to neural computation is now as follows. Obviously, (5.33) can be seen as a feed–forward single hidden layer neural network. Also, with Kolmogorov's theorem in mind, one might ask for the representation capabilities of single hidden layer MLPs. Of course, the similarities are only limited. A MLP uses the same activation function, whereas (5.33) involves different functions. Moreover, Kolmogorov's theorem states exact representation. This is only possibly due to the unspecified nature of g, hence this function depends on the problem. On the other hand, a MLP uses prior specified (i.e. problem independent) activation functions. More on the relation between Kolmogorov's theorem and neural networks can be found in [48, 49].

Anyway, exact representation is too narrowed to capture the representation capabilities of neural networks. Therefore they are commonly studied in terms of function approximation, which is a much broader framework. In fact, if one can

approximate well, then one can expect good interpolation as well. The inverse, in general, does not hold. The ability to approximate (theoretically) well, can be formalised in the following way.

Definition 5.6 (Universal Approximator) *Let* F *and* L *be two families of functions of a normed space* (X, p). *Let* d *be the metric induced by the norm* p. *Then* F *is said to be an universal approximator for* L *in the norm* p *if, for any* $l \in L$ *and any* $\epsilon > 0$, *there exists some* $f \in F$ *such that* $d(l, f) < \epsilon$. *If additionally* $L = (X, p)$, *then* F *is said to be dense in* (X, p).

The most relevant norms are the L_p norms $(1 \leq p < \infty)$

$$\|f\|_p = \left(\int_X |f(x)|^p dx \right)^{1/p} \tag{5.34}$$

and the L_∞ norm

$$\|f\|_\infty = \sup_{x \in X} |f(x)| . \tag{5.35}$$

These two types of norms then lead to two different kinds of function spaces. Spaces of continuous functions, say $C(\mathbb{R}^n)$ (the space of all continuous functions from $\mathbb{R}^n$ to $\mathbb{R}$), are naturally equipped with the norm L_∞. The L_∞ norm is very strong, for example the following relation holds (see e.g. [75]).

Proposition 5.7 *Let* μ *be a nonnegative finite Borel measure with compact support* K. *Then* $C(K)$ *is dense in* $L^p(K, \mu)$ *for any* $(1 \leq p < \infty)$.

Hereby $L^p(K, \mu)$ refers to the space of all functions from K to $\mathbb{R}$ having finite L_p norm.

Roughly speaking, the L^∞ universe is continuous, whereas the L^p universe is discontinuous. This can also be seen from all the now classical papers on the universal approximation properties of MLPs [23, 41, 91][6]. In the following we review the results stated in [23].

Theorem 5.8 (Cybenko [23]) *For any continuous function* g *fulfilling* $\lim_{x \to -\infty} g(x) 0 =$ *and* $\lim_{x \to +\infty} g(x) = 1$ *finite sums of the form*

$$S_\infty = \sum_{j=1}^{N} \lambda_j g(y_j^T x + \theta_j) \tag{5.36}$$

are dense in $C([0, 1]^n)$ *with respect to* L_∞, *i.e. in the topology of uniform convergence on compacta.*

[6] Recent surveys on universal approximation are [65, 79].

In fact, density already holds for any compact domain and any two different squashing values $a \neq b$ [53].

Corollary 5.9 *For any continuous sigmoidal function σ the $\mathrm{MLP}^{\sigma,\mathrm{id}}_{0,0}$ is a universal approximator.*

For discontinuous functions the following holds.

Theorem 5.10 (Cybenko [23]) *Let g be a bounded measurable function which also fulfils $\lim_{x\to-\infty} g(x) = 0$ and $\lim_{x\to+\infty} g(x) = 1$. Then finite sums of the form*

$$S_1 = \sum_{j=1}^{N} \lambda_j g(y_j^T x + \theta_j) \tag{5.37}$$

are dense in $L^1([0,1]^n)$.

Discontinuous activation functions are merely of theoretical interest. First, continuous activation functions are a much more naturally choice for many function approximation tasks arising in applications. Secondly, there is a lack of good training algorithms for discontinuous activation functions.

Here we will only consider continuous activation functions and hence approximation in the $L_\infty\mathtt{norm}$. Our next goal is to state universal approximation for Clifford MLPs by generalising Theorem 5.8. Elements of Clifford Analysis needed for it are provided in appendix A.3.

The original proof of Theorem 5.8 consists of two parts. The first is to prove that the activation function has the property of being discriminatory. The generalized Clifford notion thereof is as follows.

Definition 5.11 *A function $g : \mathcal{C}_{p,q} \to \mathcal{C}_{p,q}$ is said to be discriminatory if*

$$\int_X g(w \otimes_{p,q} x + \theta)\, d\mu = 0 \tag{5.38}$$

implies that $\mu = 0$ for any finite regular Clifford measure μ with proper Clifford module X as support.

The second part then consists of utilising function analysis arguments, among them the Hahn–Banach theorem. This part can be generalised to prove the following statement.

Theorem 5.12 *The* $\mathrm{MLP}^{g,\mathrm{id}}_{p,q}$ *is a universal approximator for the class of continuous functions* $C(\mathcal{C}^n_{p,q}, \mathcal{C}_{p,q})$ *in the* L_∞ *norm (i.e. in the topology of uniform convergence on compacta), if* g *is continuous and discriminatory. In other words, finite sums of the form*

$$F := \sum_{j=1}^{N} \lambda_j \sum_{I \in \mathcal{I}} g\Big(\sum_k (w_{kj} \otimes_{p,q} x_k + \theta_j)_I\Big) e_I \tag{5.39}$$

are dense (w.r.t. L_∞*) in the space* $C(\mathcal{C}^n_{p,q}, \mathcal{C}_{p,q})$ *for any discriminatory continuous activation function* g*.*

Proof: Let X be a proper $\mathcal{C}_{p,q}$–module. Suppose F is not dense in $C(\mathcal{C}^n_{p,q}, \mathcal{C}_{p,q})$, i.e. its closure $\overline{F}$ is not the whole space $C(\mathcal{C}^n_{p,q}, \mathcal{C}_{p,q})$. Then the Clifford Hahn–Banach theorem (Theorem A.6) guarantees the existence of a bounded linear functional $T : F \to \mathcal{C}_{p,q}$, such that $T(\overline{F}) = T(F)$ and $T \neq 0$. Furthermore, by the Clifford Riesz representation theorem (Theorem A.7) there then exists a unique Clifford measure μ on X such that

$$T(g) = \int_X g \, d\mu \,. \tag{5.40}$$

Since g is discriminatory it follows that $\mu(X) = 0$. That is (5.40) vanishes on X. This is only possible if $T = 0$, which, however, is a contradiction. Hence our assumption is false, and F is dense in $C(\mathcal{C}^n_{p,q}, \mathcal{C}_{p,q})$. □

The logistic function σ_1 is discriminatory for $\mathcal{C}_{0,1}$ [2] and $\mathcal{C}_{0,2}$ [1]. Proving this for an arbitrary non–degenerate Clifford algebra $\mathcal{C}_{p,q}$ can only be sketched.

Theorem 5.13 *For any non–degenerate Clifford algebra* $\mathcal{C}_{p,q}$ *the function*

$$\sum_{I \in \mathcal{I}} \sigma_1([w \otimes_{p,q} x + \theta])_I e_I \tag{5.41}$$

is discriminatory.

Proof (Sketch): Let μ be a finite regular Clifford measure on $[0,1]^{2^{p+q}}$ such that

$$\sigma_1(w \otimes_{p,q} x + \theta) d\mu = 0 \,, \tag{5.42}$$

for some $w, x \in (\mathcal{C}_{p,q})^n$, $\theta \in \mathcal{C}_{p,q}$. Let us then consider the point–wise limit

$$\phi_i(w \otimes_{p,q} x + \theta) := \lim_{\lambda \to \infty} \sigma_1(\lambda [w \otimes_{p,q} x + \theta]_i) \,, \tag{5.43}$$

with $\lambda \in \mathbb{R}$. This limit evaluates to

$$\phi_i(w \otimes_{p,q} x + \theta) = \begin{cases} 1 & : \quad \text{if} \quad [w \otimes_{p,q} x + \theta]_i > 0 \\ 0 & : \quad \text{if} \quad [w \otimes_{p,q} x + \theta]_i \leq 0 \end{cases} \tag{5.44}$$

With the Lesbesgue-dominated convergence theorem of Clifford analysis follows

$$\begin{aligned} 0 &= \int_{[0,1]^{2^{p+q}}} \sigma_1(w \otimes_{p,q} x + \theta) d\mu \\ &= \int_{[0,1]^{2^{p+q}}} (\sum_{I \in \mathcal{I}} \phi_I(w \otimes_{p,q} x + \theta) e_I)\, d\mu \\ &= \lim_{\lambda \to \infty} \sigma_1(\lambda [w \otimes_{p,q} x + \theta]_i)\,. \end{aligned}$$

For all index vectors $j \in \{0,1\}^{2^{p+q}}$ define the following sets

$$H_j := \bigcap_{i \in \{1,\ldots,2^{p+q}\}, j[i]=1} \{[w \otimes_{p,q} x + \theta]_i > 0\} \cap \bigcap_{i \in \{1,\ldots,2^{p+q}\}, j[i]=0} \{[w \otimes_{p,q} x + \theta]_i \leq 0\}\,. \quad (5.45)$$

The half-spaces H_j yield a partition of the interval $[0,1]^{2^{p+q}}$. Therefore we have with (5.44), (5.45)

$$\mu(\cup H_j) = 0\,. \quad (5.46)$$

Unfortunately, no further assumptions on μ can be made. That means one has to check (5.46) for every single $j \in \{0,1\}^{2^{p+q}}$. By Theorem 5.8 we know that

$$\mu(H_{10\ldots0}) = 0\,. \quad (5.47)$$

This can be extended with some effort to show

$$\mu(\{H_j \mid \sum_{i=1}^{2^{p+q}} j[i] = 1\}) = 0\,. \quad (5.48)$$

□

A (Spinor) Clifford MLP is a universal approximator if and only if universal approximation holds for any of its component functions (i.e. all projections $[\,]_i$ are dense in $\mathbb{R}$). The "only if"–direction will be used in section 5.3. to show that the Dual Clifford MLP is not a universal approximator. In [11] we used that direction together with the universal approximation property of the Complex MLP to show that universal approximation also holds for the Hyperbolic MLP. The next result for the Quaternionic Spinor MLP also relies on this component–wise argumentation.

Theorem 5.14 *Let X be a compact subset of $\mathbb{H}^n$. Then there exists a natural number N such that the space*

$$\left\{ \sum_{j=1}^{N} \lambda_j\, \sigma_1((\sum_{i=1}^{n} w_i \otimes_{0,2} x_i \otimes_{0,2} \bar{w_i}) + \theta_j) \right\} \quad (5.49)$$

is dense in the space of all continuous functions from X to $\mathbb{H}$ (w.r.t. L_∞).

Proof: In [1] the $\mathrm{MLP}_{0,2}^{\sigma_1,\mathrm{id}}$ was proven to be a universal approximator. Thus we only have to show that any projection $[w \otimes_{0,2} x]_i$ can be written as a finite sum of spinor multiplications $w_j \otimes_{0,2} x_j \otimes_{0,2} \bar{w}_j$. Since $\mathbb{H}$ is a skew field there always exists $u \in \mathbb{H}$ such that for all $i \in \{0, \ldots, 3\}$

$$[w \otimes_{0,2} x]_i = [w \otimes_{0,2} x \otimes_{0,2} \bar{w} + u \otimes_{0,2} x \otimes_{0,2} \bar{u}]_i \,. \tag{5.50}$$

□

Universal approximation states nothing about the quality of approximation (i.e. approximation rates). This topic beyond our scope has been studied for the real MLP in [6, 42].

As a final remark, notice that if a (S)CMLP is universal approximator then also any network which is isomorphic to it.

5.3 Experimental Results

The first experiments reported for a Complex CMLP may be found in [7], in which the XOR–problem was studied. Function approximation by Complex CMLPs in a wider sense was first studied by Arena et al. in [2]. The Complex CMLP was compared with the ordinary MLP on a couple of 2D function approximation tasks. We will use the same problem set but consider, of course, the Hyperbolic CMLP and the Dual CMLP as well. In a later publication [1] Arena et al. tested their Quaternionic CMLP on the task of predicting the Lorentz Attractor [55]. This is actually a 3–dimensional problem, which renders it a perfect task for testing Spinor architectures.

5.3.1 2D Function Approximation

The following set of complex functions for the experimental evaluation of the Complex CMLP have been proposed in [2].

$$f(x + iy) = \exp(iy)(1 - x^2 - y^2) \tag{5.51}$$
$$g(x + iy) = x^2 + y^2 + 2ixy \tag{5.52}$$
$$h(x + iy) = \sin(x)\cosh(y) + i\cos(x)\sinh(y) \tag{5.53}$$

For every function $\phi \in \{f, g, h\}$ let ϕ_1 denote the real part of the function and ϕ_2 denote the imaginary part of the function, respectively. When speaking of training and test data, however, it is more precise to speak of 2D vector data rather than of complex numbers. In what follows all of the above functions are studied using the same experimental setup as reported in [2] if not mentioned otherwise.

We start with the function f. In order to perform its approximation 500 points were randomly drawn from $[0, 1]^2$ with uniform distribution. From that set 200 points were used for training while the remaining 300 points were kept for testing. However, as can be seen from figure 5.2, the domain $[0, 1]^2$ is not quite characteristic for the function f. The second component function f_2 shows a nearly linear behaviour in $[0, 1]^2$.

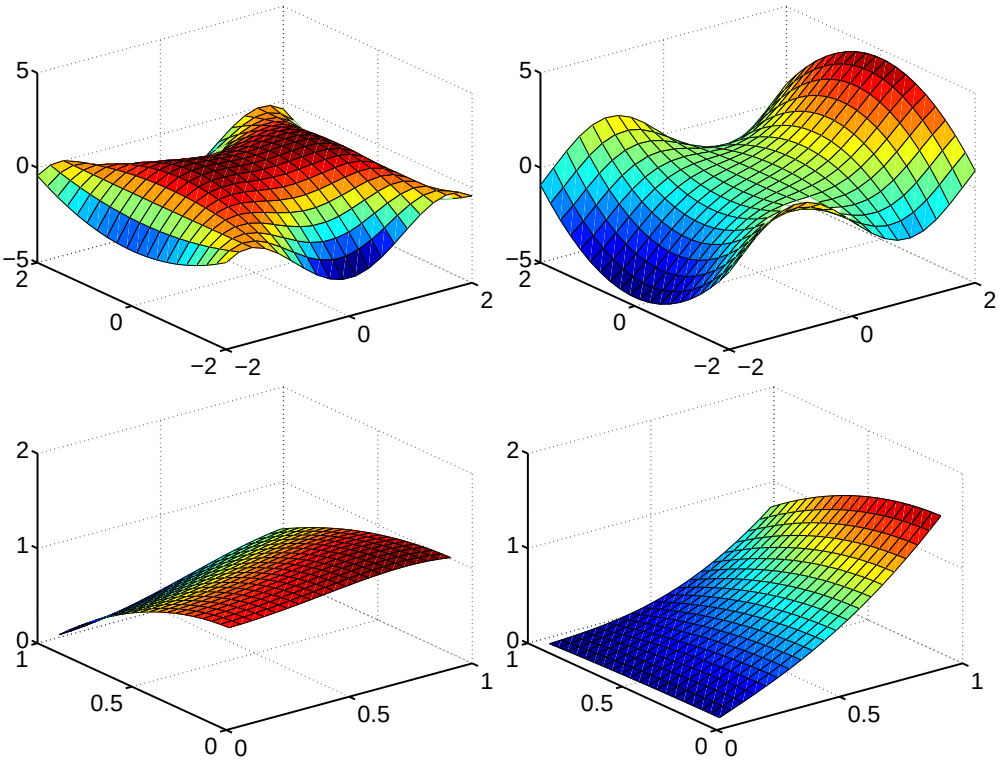

Figure 5.2: Plots of the function f (5.51). f_1 in $[-2, 2]^2$ (top left), f_2 in $[-2, 2]^2$ (top right), f_1 in $[0, 1]^2$ (bottom left), and f_2 in $[0, 1]^2$ (bottom right).

The networks were trained over 1100 iterations using online learning with learning rate 0.1 and momentum term 0.01 . All results are summarised in table 5.3, which shows mean values over 100 runs for each setup for the experiments performed by us. No statement about the number of runs is made in [2].

For determining the size of the networks, i.e. the number of hidden nodes, the

Network	Hidden Nodes	Error [2]	MSE Training	MSE Test	Parameters
$MLP_{0,0,0}$	4	0.1260	0.0005	0.0005	22
	5	0.0791	0.0004	0.0004	27
	6	0.1289	0.0003	0.0003	32
$MLP_{1,0,0}$	2	—	0.0088	0.0095	14
	3	—	0.0009	0.0009	20
$MLP_{0,1,0}$	2	0.4240	0.0019	0.0019	14
	3	0.0723	0.0003	0.0003	20
$MLP_{0,0,1}$	2	—	0.0662	0.0568	14
	3	—	0.0739	0.0709	20

Table 5.3: Results for the approximation of f.

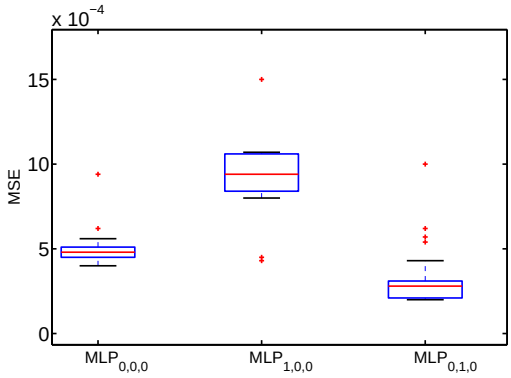

Figure 5.3: Box plot over 100 runs for the $MLP_{0,0,0}$ with 4 hidden nodes, the $MLP_{1,0,0}$ with 3 hidden nodes and the $MLP_{0,1,0}$ with 3 hidden nodes.

following strategy was applied. The $MLP_{1,0,0}$ was trained first with 2 and 3 hidden neurons, respectively. Followed by the other Clifford MLPs in the same way and then the real $MLP_{0,0,0}$ was trained with a growing number of hidden nodes until the training performance of the $MLP_{1,0,0}$ was reached. Assuming that the SSE is reported in [2], the obtained results are roughly in accordance. With the notable exception that no overfitting occurred in our experiments with the $MLP_{0,0,0}$ with 6 hidden nodes. The training of the $MLP_{0,0,1}$ did not converge to a reasonable error[7]. For 2 and 3 hidden nodes the $MLP_{0,1,0}$ totally outperformed the $MLP_{1,0,0}$ in each case. The training performance of the $MLP_{0,1,0}$ could not be met by the real $MLP_{0,0,0}$ with 4 hidden nodes (roughly the same number of parameters), however it outperformed the $MLP_{1,0,0}$. The detailed training errors of all the networks with the mentioned setup are shown as box plot in figure 5.3.

[7]Therefore, in the following discussion it will be omitted.

In all cases only a small number of outliers occurred, however, the $MLP_{1,0,0}$ showed a rather big variation over the runs. To meet the performance of the $MLP_{0,1,0}$ the $MLP_{0,0,0}$ needed 6 hidden nodes (approximately 50% more parameters). Also, the $MLP_{0,0,0}$ took more time for converging as can be seen from figure 5.4.

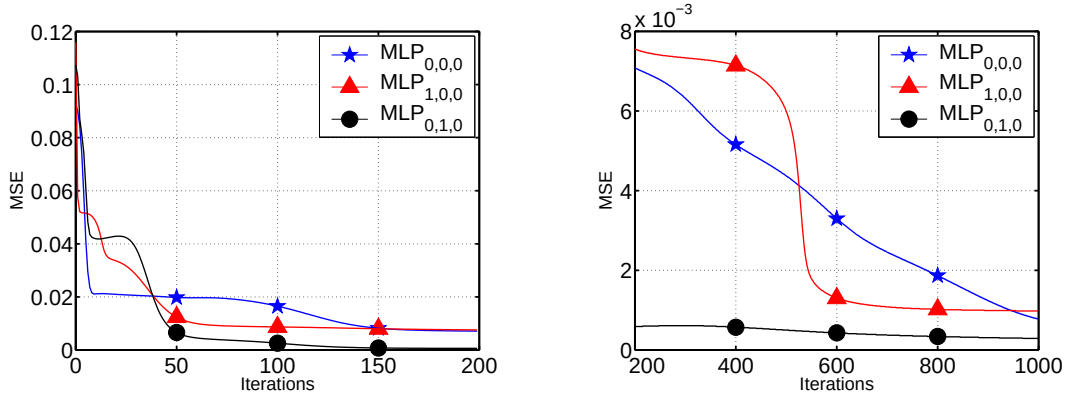

Figure 5.4: Learning curve for a typical run of the $MLP_{0,0,0}$ with 6 hidden nodes, the $MLP_{1,0,0}$ with 3 hidden nodes and the $MLP_{0,1,0}$ with 3 hidden nodes.

Since nothing about the generalisation performance can be concluded from the test errors (table 5.3) additional experiments with the above configurations on noisy data have been performed. As can be seen from the result in figure 5.5 the $MLP_{0,1,0}$ outperformed the $MLP_{0,0,0}$ in terms of generalization.

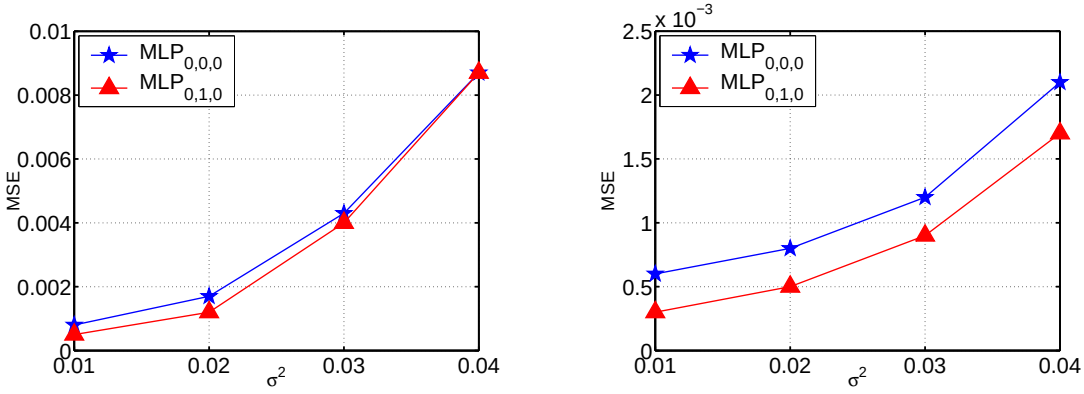

Figure 5.5: Training error (left) and test error (right) of the $MLP_{0,0,0}$ with 6 hidden nodes and the $MLP_{0,1,0}$ with 3 hidden nodes on noisy data.

To summarise all the results, the function f was best approximated by the $MLP_{0,1,0}$ both in terms of generalisation and efficiency. This was somehow predictable from its equation (5.51). However, the experiments have been performed on rather not

characteristic domain of the function. Fitting a wrong model to the data ($MLP_{1,0,0}$) resulted in the worst (but by no means critical) performance.

With the results for the approximation of the function f in mind the following prediction for the approximation of the function g (5.52) seems reasonable. All networks should show more or less the same good performance (excluding the $MLP_{0,0,1}$).

For the approximation of the function g 300 points were randomly drawn from $[0,1]^2$ with uniform distribution. From which then 100 points were selected for training while the remaining 200 points were kept for testing. The function g is plotted in figure 5.6. Of course, $[0,1]^2$ is not a domain which reveals the radial symmetry of the component function g_1 (parabola).

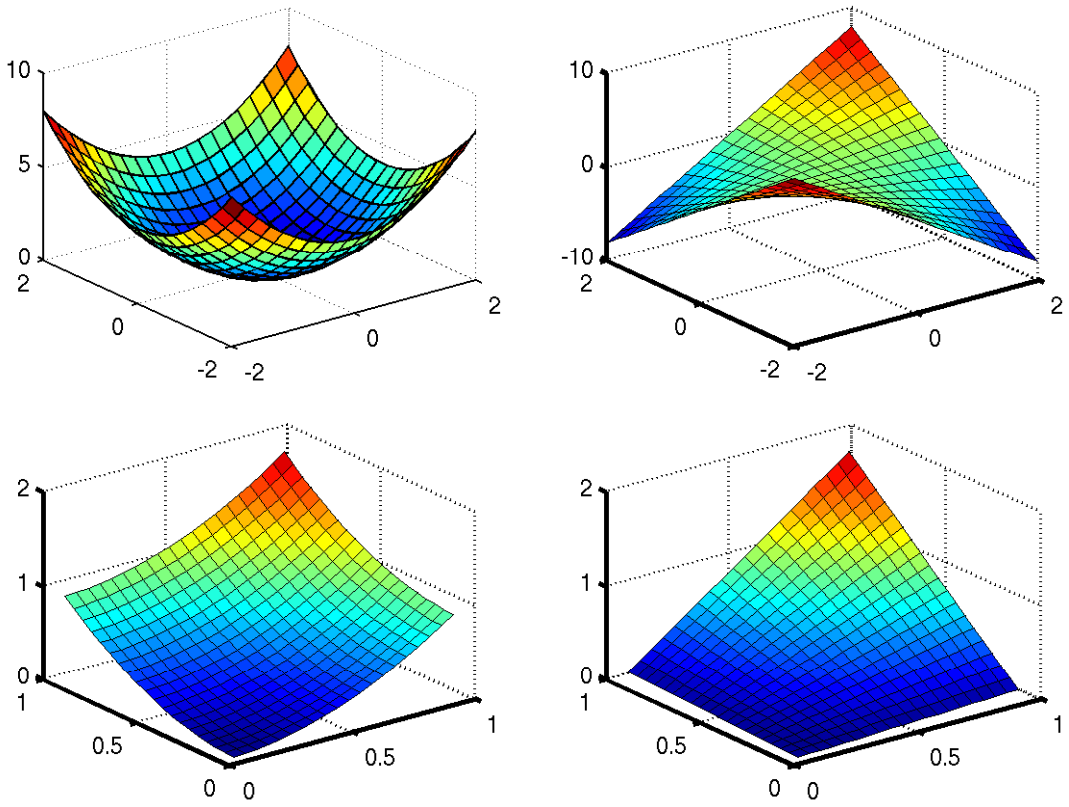

Figure 5.6: Plots of the function g (5.52). g_1 in $[-2,2]^2$ (top left), g_2 in $[-2,2]^2$ (top right), g_1 in $[0,1]^2$ (bottom left), and g_2 in $[0,1]^2$ (bottom right).

All networks were trained over 1100 iterations using again online learning with and momentum term 0.01. In [2] the learning rate was 0.1. Here, however, the learning rate was set to 0.2 since giving much better results for the $MLP_{0,0,0}$. Again,

Network	Hidden Nodes	Error [2]	MSE Training	MSE Test	Parameters
$MLP_{0,0,0}$	2	1.5585	0.0095	0.0094	12
	3	—	0.0002	0.0002	17
	4	0.0280	0.0001	0.0001	22
$MLP_{1,0,0}$	2	—	0.0109	0.0101	14
	3	—	0.0002	0.0002	20
	4	—	0.0001	0.0001	26
$MLP_{0,1,0}$	2	0.4240	0.0002	0.0002	14
	3	0.0723	0.0001	0.0001	20
	4	0.0770	0.0000	0.0001	26
$MLP_{0,0,1}$	2	—	0.0963	0.0904	14
	3	—	0.0925	0.0909	20

Table 5.4: Results for the approximation of g.

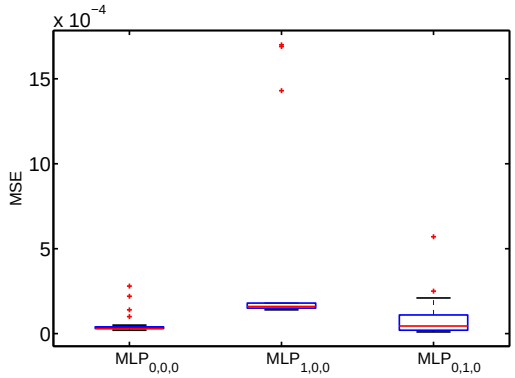

Figure 5.7: Box plot over 100 runs for the $MLP_{0,0,0}$ with 4 hidden nodes, the $MLP_{1,0,0}$ with 3 hidden nodes and the $MLP_{0,1,0}$ with 3 hidden nodes.

all of our results in table 5.4 are mean values from 100 runs.

The task turned out to be pretty simple for all networks with the exception of the $MLP_{0,0,1}$. Only considering the results of our experiments all remaining networks did indeed perform equally well. The slight differences between the Clifford MLPs and the real $MLP_{0,0,0}$ are mostly due to the different numbers of parameters, or, are simply not significant. The box plots in figure 5.7 show an example how the performance of the $MLP_{1,0,0}$ with 3 hidden nodes was "ruined" by a few outliers.

The learning curves provided by figure 5.8 show that both the $MLP_{0,0,0}$ and the $MLP_{1,0,0}$ reached a plateau for the specific run but both finally escaped successfully. Notice that plateaus are not uncommon for low error levels.

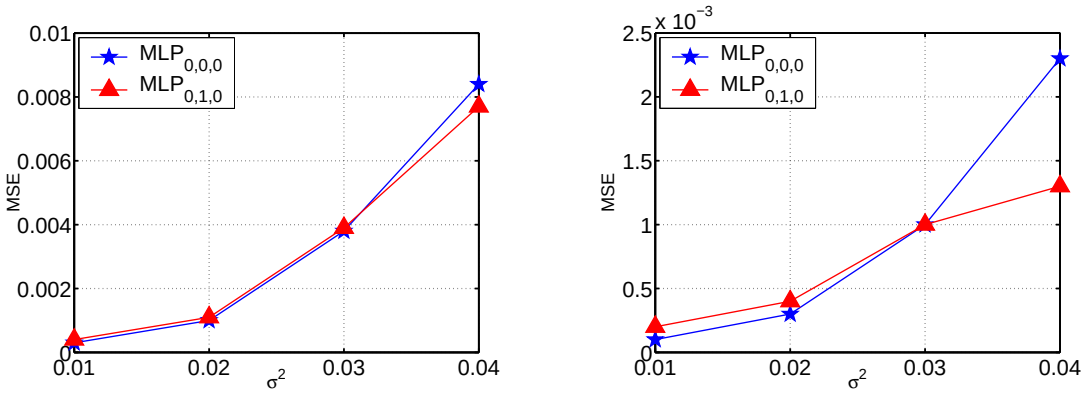

Figure 5.9: Training error (left) and test error (right) of the $MLP_{0,0,0}$ with 4 hidden nodes and the $MLP_{0,1,0}$ with 3 hidden nodes on noisy data.

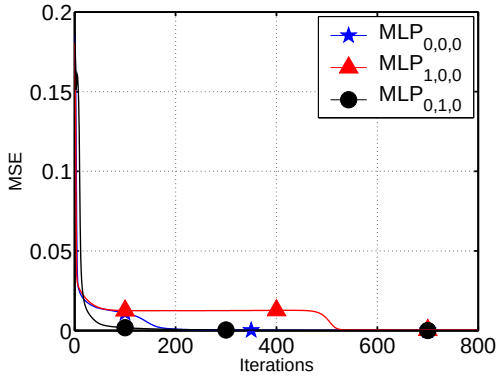

Figure 5.8: Learning curve for a typical run of the $MLP_{0,0,0}$ with 4 hidden nodes, the $MLP_{1,0,0}$ with 3 hidden nodes and the $MLP_{0,1,0}$ with 3 hidden nodes.

Even on noisy data, for which some results are given in figure 5.9, no significant differences could be observed [8].

All in all a summary for the approximation of g is quite easy. All networks performed as predicted equally well (excluding the $MLP_{0,0,1}$ as usual).

The last function h is by far the most interesting. This is because it is not such much a complex function but more a hyperbolical one. Actually, it is the sine function for hyperbolic numbers[9].

For the approximation of h 100 points were randomly drawn from $[0, 1]^2$ with uniform distribution to make up the training set. Contrary to [2], 500 points were

[8] The value for the highest level of noise in the test error of the $MLP_{1,0,0}$ is not significant.

[9] It is also a complex analytical function, which is why it was chosen in [2]. This aspect will be discussed in more detail in the next chapter.

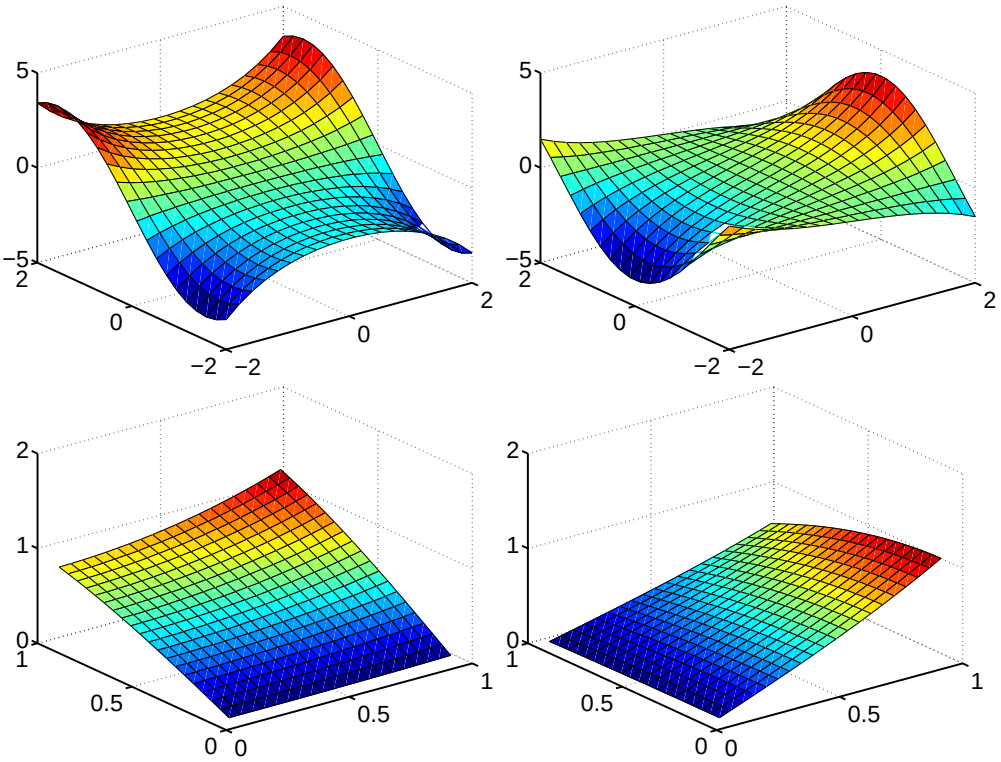

Figure 5.10: Plots of the function h (5.53). h_1 in $[-2,2]^2$ (top left), h_2 in $[-2,2]^2$ (top right), h_1 in $[0,1]^2$ (bottom left), and h_2 in $[0,1]^2$ (bottom right).

randomly drawn from $[-2,2]^2$ with uniform distribution for building up the test set. That actually changes the task from interpolation to extrapolation. Having identified g as the hyperbolic sine the $\mathrm{MLP}_{1,0,0}$ should be able to outperform the other networks in terms of extrapolation. As can be seen from the plots of the training and test data in figure 5.10 this task is rather ambitious.

All networks were trained over 1100 iterations using online learning with learning rate 0.1 and momentum term 0.01. As always 100 runs were performed. Training was started with the real $\mathrm{MLP}_{0,0,0}$, which already showed a good training performance for 4 hidden notes. All results are printed in table 5.5.

Note that our training results do not correspond in any sense with the results reported in [2]. The discussion of the results of the $\mathrm{MLP}_{0,0,1}$ will be postponed. With 2 hidden nodes both the $\mathrm{MLP}_{1,0,0}$ and the $\mathrm{MLP}_{0,1,0}$ outperformed the $\mathrm{MLP}_{0,0,0}$ with 3 hidden nodes having already more parameters. Not only the mean value but also the box plots (figure 5.11) for the $\mathrm{MLP}_{1,0,0}$ and the $\mathrm{MLP}_{0,1,0}$ turned out to be identical.

Network	Hidden Nodes	Error [2]	MSE Training	MSE Test	Parameters
$MLP_{0,0,0}$	2	0.2996	0.0016	1.1056	12
	3	—	0.0006	1.0214	17
	4	0.0457	0.0002	0.9462	22
$MLP_{1,0,0}$	1	—	0.0047	1.1054	8
	2	—	0.0002	0.7534	14
$MLP_{0,1,0}$	1	0.3482	0.0021	1.1278	8
	2	0.1586	0.0002	1.0806	14
$MLP_{0,0,1}$	1	—	0.0082	0.8784	8
	2	—	0.0071	0.8457	14

Table 5.5: Results for the approximation of h.

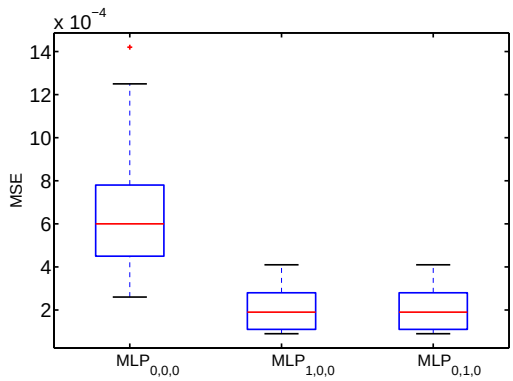

Figure 5.11: Box plot over 100 runs for the $MLP_{0,0,0}$ with 3 hidden nodes, the $MLP_{1,0,0}$ with 2 hidden nodes and the $MLP_{0,1,0}$ with 2 hidden nodes.

Using 4 hidden nodes the $MLP_{0,0,0}$ reached the same training performance as the two Clifford MLPs. Surprisingly, figure 5.12 reveals that the $MLP_{1,0,0}$ needed the highest number of iterations for convergence.

Looking at the test errors the $MLP_{1,0,0}$ totally outperformed the other networks. Also the largest drop in the test error occurred for the $MLP_{1,0,0}$ when changing from one hidden node to two hidden nodes. The test set output of all networks (except that of the $MLP_{0,0,1}$) is given in figure 5.13. Obviously, all networks extrapolate better in $[0,2]^2$ than in $[-2,0]^2$, just because the first is closer to the training domain $[0,1]^2$ than the latter. Extrapolation by the $MLP_{0,0,0}$ is rather bad (compared to the other two networks), particularly for the second component function h_2. Although the outputs of the $MLP_{1,0,0}$ and the $MLP_{0,1,0}$ look roughly similar at first sight, there are notable differences. For the first component function h_1 (left column of figure 5.13) the $MLP_{1,0,0}$ performed better than the $MLP_{0,1,0}$ in $[-2,+1]\times[+1,+2]$.

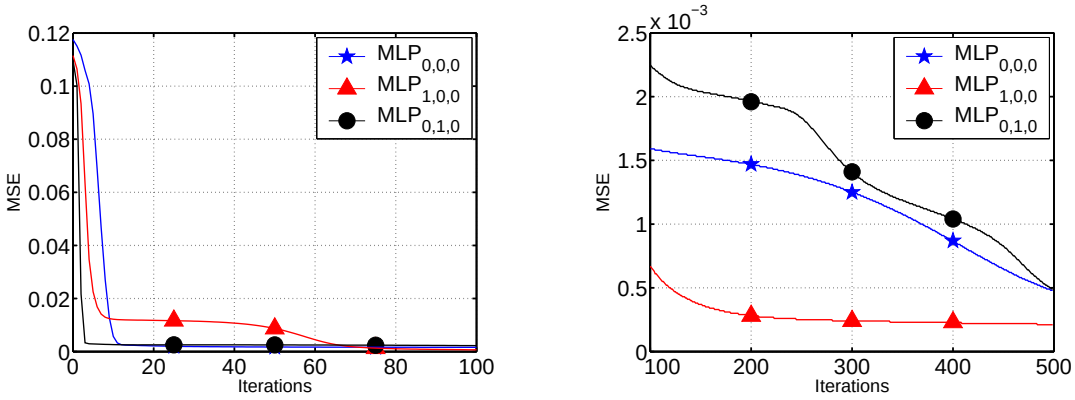

Figure 5.12: Learning curve for a typical run of the $\mathrm{MLP}_{0,0,0}$ with 4 hidden nodes, the $\mathrm{MLP}_{1,0,0}$ with 2 hidden nodes and the $\mathrm{MLP}_{0,1,0}$ with 2 hidden nodes.

Regarding the second component function h_2, the extrapolation in $[-2,0]^2$ is better done by the $\mathrm{MLP}_{1,0,0}$ than by the $\mathrm{MLP}_{0,1,0}$ (relatively speaking). Comparing the results with the true values in figure 5.10 (top row) the overall extrapolation on the whole test domain $[-2,2]^2$ is not that good. Nevertheless, the $\mathrm{MLP}_{1,0,0}$ did the best job, because of processing the most right model for the data. The $\mathrm{MLP}_{0,1,0}$ looks second best, however, its test error is actually larger than that of the $\mathrm{MLP}_{0,0,0}$ (table 5.5). Also from that table, the actual second best test performance (in pure numerical terms) was achieved by the $\mathrm{MLP}_{0,0,1}$.

The learned functions of the $\mathrm{MLP}_{0,0,1}$ for all the considered test functions are plotted in figure 5.14. The good test error result of the $\mathrm{MLP}_{0,0,1}$ for the function h is only due to its good extrapolation of the second component function h_2. Nevertheless, the $\mathrm{MLP}_{0,0,1}$ is not able to approximate the function h arbitrarily well, nor any of the other test functions. That becomes ultimately clear when looking at the first component functions. Here the computational power of the $\mathrm{MLP}_{0,0,1}$ is limited to functions of the type

$$F(x,y) = F(x)\,, \tag{5.54}$$

or, equivalently,

$$F(\cdot,y) = \mathrm{const}\,. \tag{5.55}$$

This is a direct consequence of the degenerate nature of the underlying geometric product. As outlined in section 5.2, a Clifford MLP is a universal approximator if and only if universal approximation holds for all the component functions. Universal approximation does not hold for the first component function of the $\mathrm{MLP}_{0,0,1}$, hence the $\mathrm{MLP}_{0,0,1}$ is not a universal approximator.

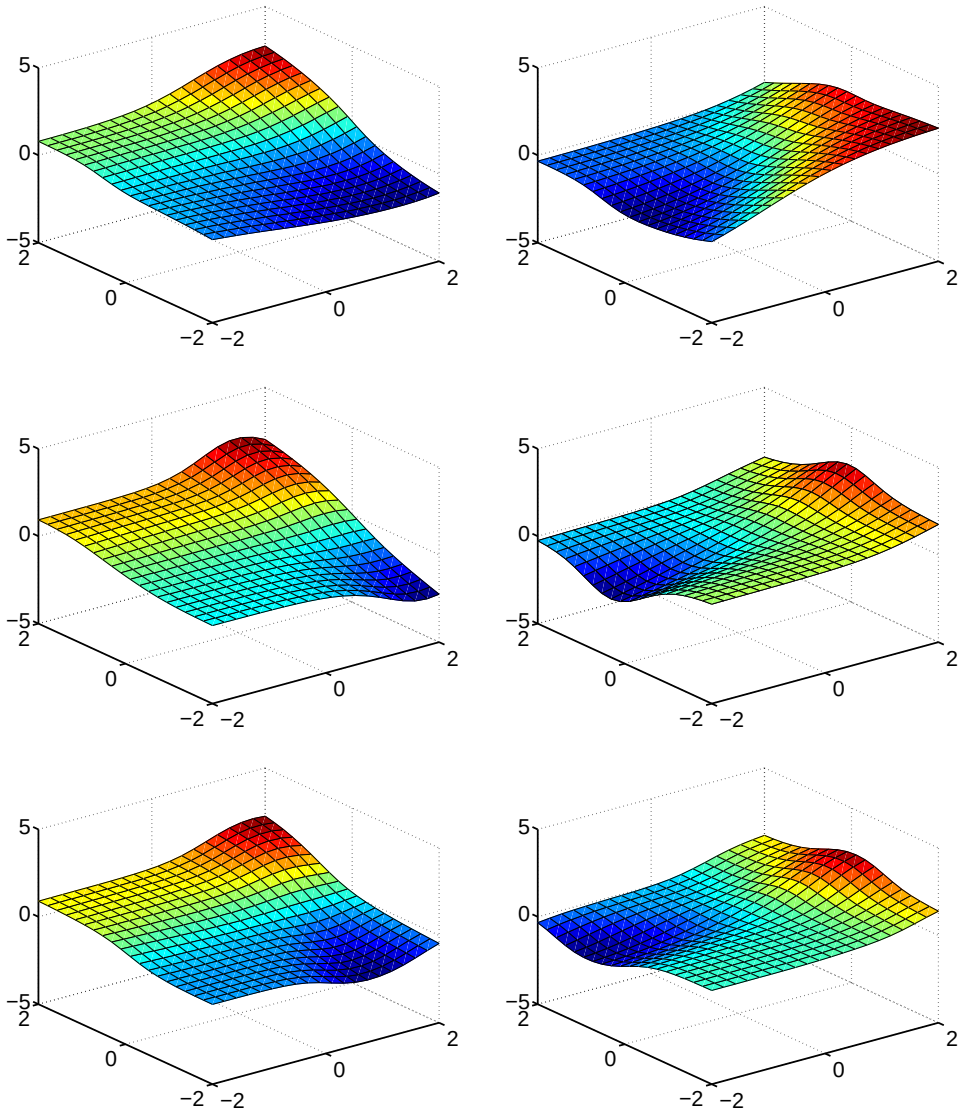

Figure 5.13: Extrapolation of the test functions h by the $MLP_{0,0,0}$ with 4 hidden nodes (top row), the $MLP_{1,0,0}$ with 2 hidden nodes (middle row) and the $MLP_{0,1,0}$ with 2 hidden nodes (bottom row).

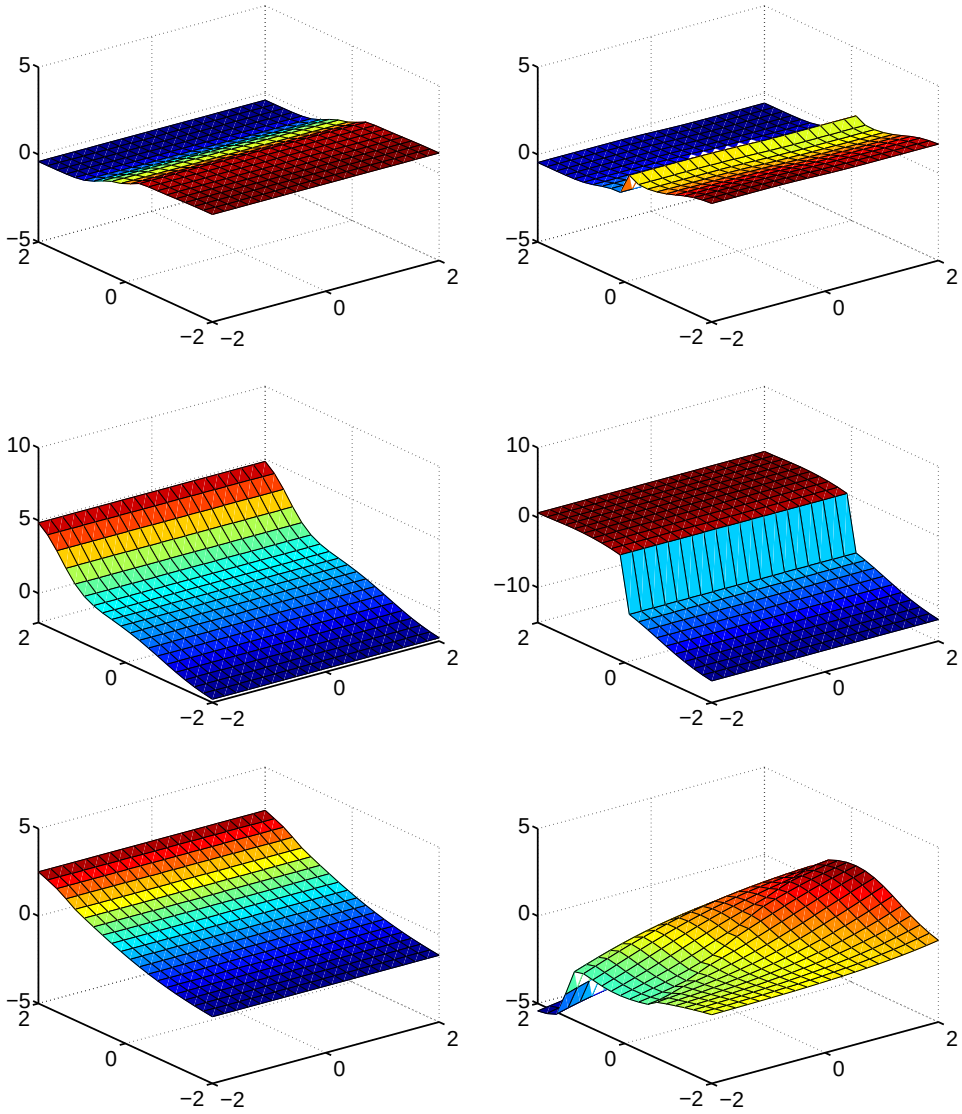

Figure 5.14: Learned function by the $\mathrm{MLP}_{0,0,1}$ for the approximation of f (top row), the approximation of g (middle row) and the approximation of h (bottom row).

While the Dual CMLP dropped out of the competition, the Complex CMLP and the Hyperbolic CMLP are both powerful networks. In fact, model–based effects could be well observed in the conducted experiments. Of course, which network to chose in advance for a particular task (with unknown characteristics) seems still an open question. However, there is some kind of answer in our framework. The algebra $\mathcal{C}_{1,1}$ contains both the complex and the hyperbolic numbers. Therefore a $\mathrm{CMLP}_{1,1}$ cannot perform worse than the Complex or Hyperbolic CMLP if given the

same number of hidden nodes. The drawback is of course the that way increased number of (real) parameters.

5.3.2 Prediction of the Lorenz Attractor

For testing the Quaternionic MLP a series of experiments on short–term prediction of chaotic time series have been originally performed in [1]. Among them being the prediction of the Lorenz attractor [55] which is a 3–dimensional problem. Hence a Quaternionic SCMLP seems to be a more natural choice, and results comparing both architectures have been reported by us in [13]. In the following this is redone and extended to a broader basis of network architectures.

The Lorenz attractor results from integrating the following three coupled nonlinear differential equations

$$\dot{x} = \sigma(x - y) \tag{5.56a}$$
$$\dot{y} = xz + rx - y \tag{5.56b}$$
$$\dot{z} = xy - bz\,, \tag{5.56c}$$

with global parameter values $\sigma := 10$, $r := \frac{8}{3}$ and $b = 28$.

For the experiments $(x_0, y_0, z_0) = (0, 1, 0)$ was used as initial state and the time interval (12s,17s) was sampled with sampling rate $\Delta t = 0.005$ yielding 1000 points[10]. From that the first 250 points were used as training set and the remaining $750 - \tau$ points as test set. The prediction step size τ varied from 4 to 8 steps. Yielding for $\tau = 8$ the first training pattern $\{(x_0, y_0, z_0), (x_8, y_8, z_8)\}$ and the last test pattern $\{(x_{992}, y_{992}, z_{992}), (x_{1000}, y_{1000}, z_{1000})\}$ All generated points separated into training and test set are shown in figure 5.15.

[10]No information about the initial state, the sampling interval and the sampling rate are provided in [1]. Hence no comparison with the results reported therein is possible.

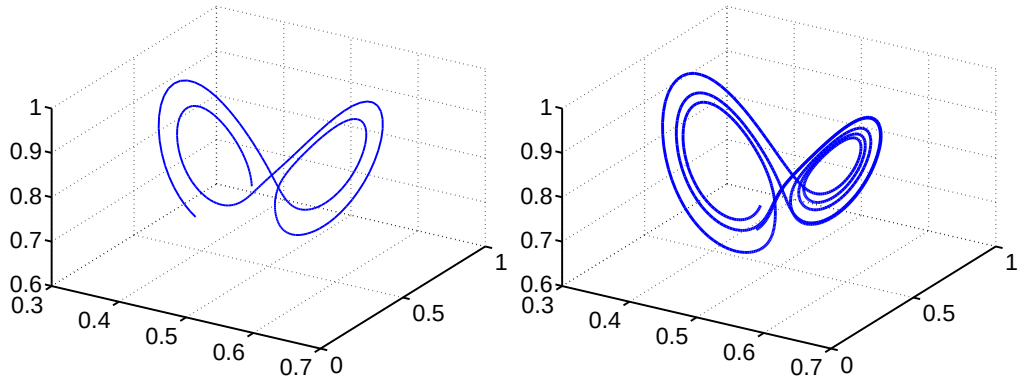

Figure 5.15: Lorenz attractor training set (left) and test set (right).

The following five architectures and representations have been chosen for a comparison. The $\mathrm{MLP}_{0,0,0}$, the Quaternionic $\mathrm{MLP}_{0,2,0}$, the Quaternionic Spinor $\mathrm{MLP}_{0,2,0}$ and the $\mathrm{MLP}_{1,1,0}$. For all the aforementioned Clifford architectures the data was presented as $\{(0, x_i, y_i, z_i), (0, x_{i+\tau}, y_{i+\tau}, z_{i+\tau})\}$, that is the scalar component was always set to zero. Additionally, for the Quaternionic Spinor $\mathrm{MLP}_{0,2,0}$ the data presentation $\{(x_i, y_i, z_i, 0), (x_{i+\tau}, y_{i+\tau}, z_{i+\tau}, 0)\}$ was also considered.

Training was always performed as online learning with learning rate $\eta = 0.1$ over 10000 iterations. The following strategy was applied. The $\mathrm{SMLP}_{0,2,0}$ was trained first for the smallest prediction step $\tau = 4$. Stable and best performance was achieved using 4 hidden nodes, corresponding to 52 real parameters. The same number of hidden nodes was then also used for the other Clifford MLPs. For the $\mathrm{MLP}_{0,0,0}$ 8 hidden nodes were chosen, corresponding to 59 real parameters and thus roughly meeting the parameter complexity of the Clifford MLPs for fair comparison. The number of hidden nodes was fixed and experiments increasing the prediction step size up to $\tau = 8$ followed. The aforementioned setup of learning parameters turned out to be also stable and optimal in that cases. All results averaged over 20 runs are reported in table 5.6.

Network	Phase	Prediction steps (τ)				
		4	5	6	7	8
$MLP_{0,0,0}$	Training	0.0003	0.0005	0.0012	0.0014	0.0022
	Test	0.0005	0.0009	0.0019	0.0022	0.0032
$MLP_{1,1,0}$	Training	0.0002	0.0004	0.0005	0.0014	0.0037
$(0, x, y, z)$	Test	0.0004	0.0007	0.0012	0.0024	0.0055
$MLP_{0,2,0}$	Training	0.0002	0.0003	0.0006	0.0008	0.0012
$(0, x, y, z)$	Test	0.0004	0.0005	0.0011	0.0012	0.0020
$SMLP_{0,2,0}$	Training	0.0002	0.0002	0.0003	0.0006	0.0007
$(0, x, y, z)$	Test	0.0003	0.0004	0.0004	0.0011	0.0012
$SMLP_{0,2,0}$	Training	0.0002	0.0003	0.0006	0.0009	0.0033
$(x, y, z, 0)$	Test	0.0005	0.0005	0.0011	0.0017	0.0051

Table 5.6: *Results for the prediction of the Lorenz attractor. See text for details.*

For the smallest prediction size all networks performed roughly similar, with the $SMLP_{0,2,0}(0, x, y, z)$ being slightly best on the test set. From step size $\tau = 6$ on the $SMLP_{0,2,0}(0, x, y, z)$ outperformed all the other networks. For $\tau = 8$ the generalisation of the $SMLP_{0,2,0}(0, x, y, z)$ is four times better than that of the $SMLP_{0,2,0}(x, y, z, 0)$ and that of the $MLP_{1,1,0}(0, x, y, z)$. This demonstrates the canonical role of the vector representation for the Quaternionic Spinor MLP and the importance of representation issues in the Clifford neural framework in general. The overall second best performance was achieved by the $MLP_{0,2,0}(0, x, y, z)$, which also uses vector representation, although not in connection with spinors. The real–valued $MLP_{0,0,0}$ was always among the worst for most prediction step sizes, with the notable exception for $\tau = 8$. In that case it outperformed at least the $SMLP_{0,2,0}(x, y, z, 0)$ and the $MLP_{1,1,0}(0, x, y, z)$.

For evaluating the prediction accuracy in time series problems correlation is often used as performance index [26]. A correlation value close to 1 means that the τ-step ahead zero–mean prediction is close to the actual series for a particular state variable on the test set. Performance evaluated in these terms is given in figure 5.16 for all single state variables, and additionally, in terms of overall correlation computed as the average over the single correlations.

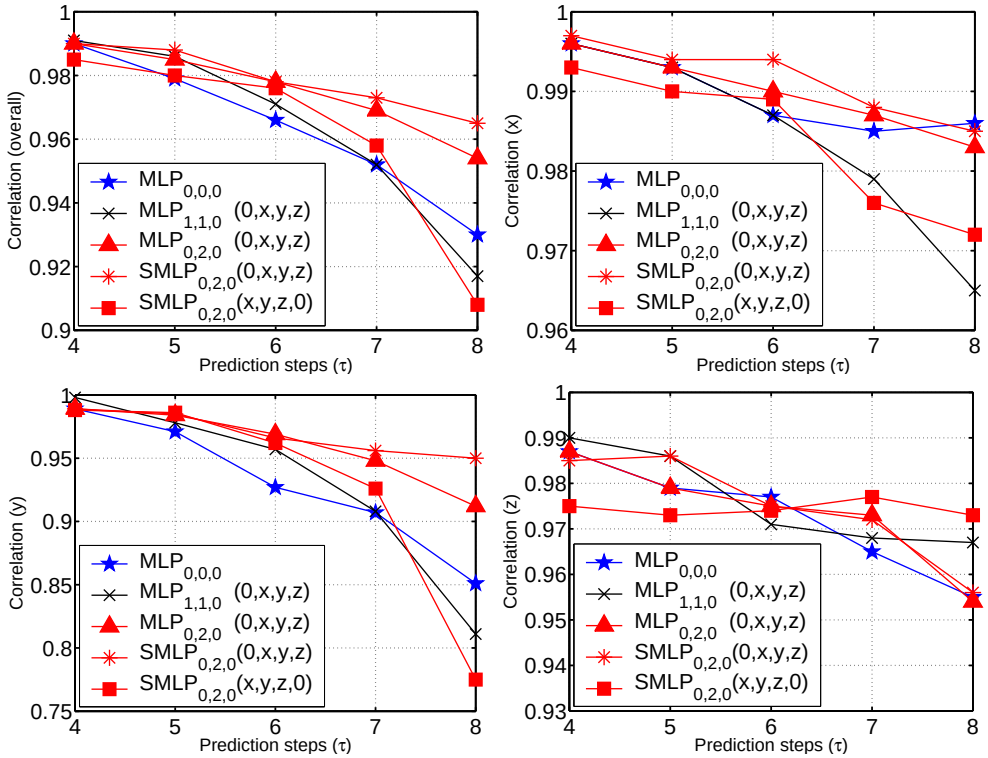

Figure 5.16: Plot of the overall correlation (top left), of the first state variable correlation (top right), the second state variable correlation (bottom left) and the third state variable correlation (bottom right). See text for details.

The results based on correlation performance are in accordance to that reported in table 5.6. All networks start with a roughly equal performance for small prediction sizes $\tau = 4$ and $\tau = 5$. From step size $\tau = 6$ the trend becomes visibly quite nice. The gap between the $\mathrm{MLP}_{0,2,0}(0, x, y, z)$ and the $\mathrm{SMLP}_{0,2,0}(0, x, y, z)$ on the one hand and the other three networks on the other hand gets bigger with increasing prediction size. The performance of the $\mathrm{SMLP}_{0,2,0}(0, x, y, z)$ is clearly better than that of the $\mathrm{MLP}_{0,2,0}(0, x, y, z)$ for $\tau = 8$ and already slightly better for $\tau = 7$. The overall trend is quite stable and there is no indication that it would not continue in the same way if τ would be increased further.

The performance of the $\mathrm{SMLP}_{0,2,0}(0, x, y, z)$ on the training set only can be achieved by the other networks if the number of hidden nodes is increased. Out of those we will only consider the consequences in terms of generalisation for the real $\mathrm{MLP}_{0,0,0}$ since the best model–based Clifford architecture has already be identified. The number of 13 hidden nodes was necessary for the $\mathrm{MLP}_{0,0,0}$ to achieve nearly the same performance as the $\mathrm{SMLP}_{0,2,0}(0, x, y, z)$. This is reported in table 5.7 together with the results of both architectures from training with data corrupted by mean–

free gaussian noise of different variance. The performance of the $MLP_{0,0,0}$ in the presence of noise was clearly worse than that of the $SMLP_{0,2,0}(0, x, y, z)$. In fact, the correlation obtained by the $SMLP_{0,2,0}(0, x, y, z)$ for a noise level of $\sigma^2 = 0.10$ is better than that of the $MLP_{0,0,0}$ for a noise level of $\sigma^2 = 0.05$.

Network	Noise (σ^2)	MSE Training	MSE Test	Correlation	Parameters
$MLP_{0,0,0}$	0.00	0.0007	0.0012	0.966	108
	0.05	0.0473	0.0488	0.939	
	0.10	0.0610	0.0918	0.917	
$SMLP_{0,2,0}$	0.00	0.0007	0.0012	0.965	72
	0.05	0.0113	0.0131	0.958	
	0.10	0.0287	0.0346	0.943	

Table 5.7: *Results for the prediction of the Lorenz attractor ($\tau = 8$) of the $MLP_{0,0,0}$ with 13 hidden nodes and the $SMLP_{0,2,0}(0, x, y, z)$ with 4 hidden nodes on different noise levels (σ).*

Thus giving the $MLP_{0,0,0}$ 50% more parameters only worked out in a noise–free setup, which can be also seen from figure 5.17 showing the actual outputs of both networks for the different levels of noise.

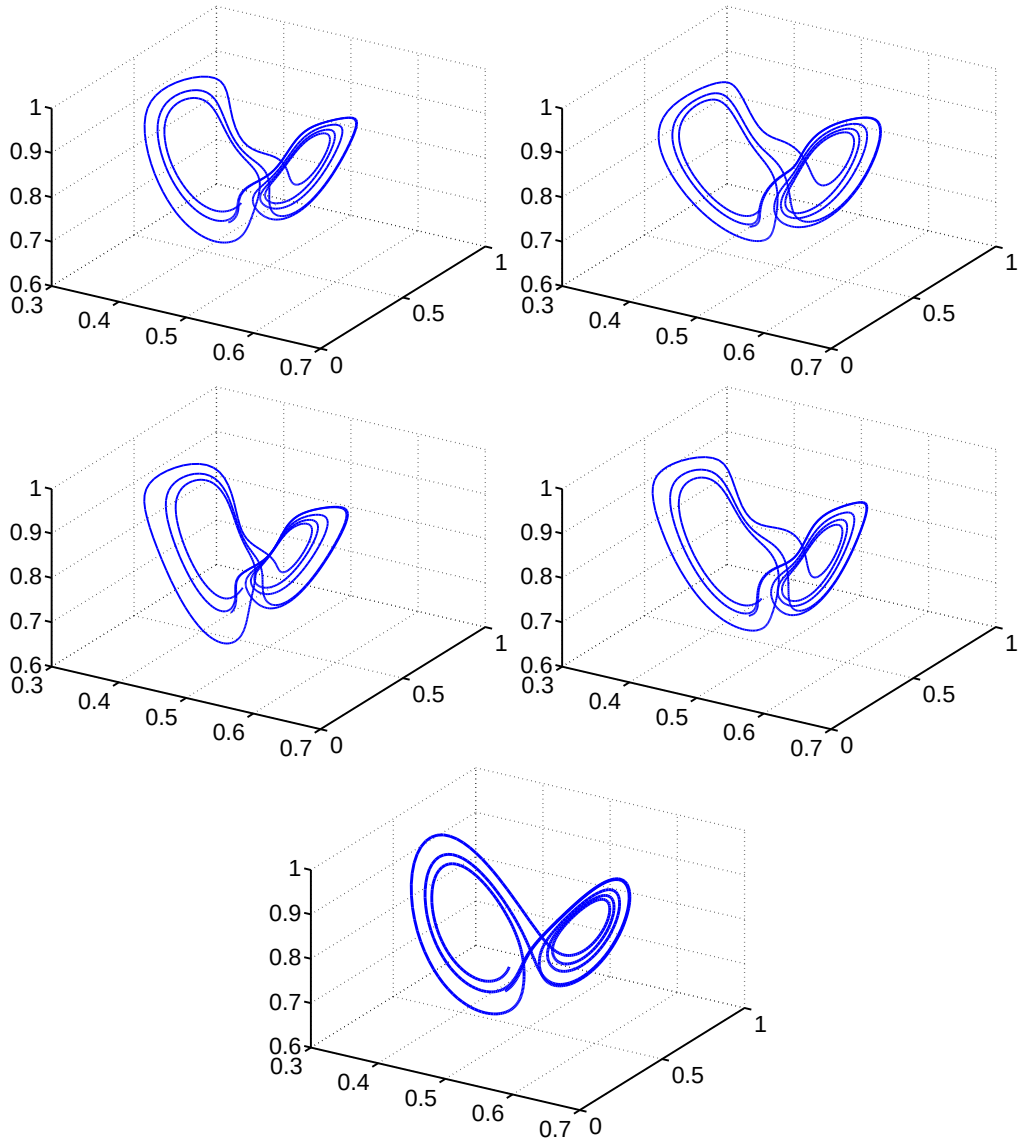

Figure 5.17: Test set output for the prediction of the Lorenz attractor ($\tau = 8$) of the $\mathrm{MLP}_{0,0,0}$ with 13 hidden nodes (left column) and the $\mathrm{SMLP}_{0,2,0}(0, x, y, z)$ with 4 hidden nodes (right column) on different noise levels ($\sigma = 0.05$ (top row) and $\sigma = 0.10$ (center row)). The bottom row shows the expected output again in order to simplify a comparison.

The $\mathrm{SMLP}_{0,2,0}(0, x, y, z)$ outperformed the other networks because of being the only one[11] using a representation forcing an intrinsically 3–dimensional processing of the data.

Although we did consider enough architectures to come up with this and other

[11] This follows from Theorem (4.10) and Proposition (4.12).

important conclusions the picture is not fully completed yet. Of course, it would be also interesting to extend the experiments to Clifford MLPs in eight dimensions. Theoretically, in terms of training error, the SMLP $_{0,3,0}(0, x, y, z, 0, 0, 0)$ can not perform worse than the $\mathrm{SMLP}_{0,2,0}(0, x, y, z)$ assuming the same number of hidden nodes. This is just because the former can mimic the computation of the latter by setting the "unnecessarily" weights to zero. Exactly the same argument holds for the $\mathrm{MLP}_{0,3,0}(0, x, y, z, 0, 0, 0)$ with respect to the $\mathrm{MLP}_{0,2,0}(0, x, y, z)$.

With 4 hidden nodes the $\mathrm{SMLP}_{0,3,0}(0, x, y, z, 0, 0, 0)$ achieved exactly the same correlation of 0.966 as the $\mathrm{SMLP}_{0,2}(0, x, y, z)$ (see table 5.7 again). However, with twice as much the number of real parameters, and, in a totally different way. This can be seen by comparing the single correlations values $(0.992, 0.931, 0.976)$ obtained for the $\mathrm{SMLP}_{0,3,0}(0, x, y, z, 0, 0, 0)$ with that of the $\mathrm{SMLP}_{0,2}(0, x, y, z)$ as reported in figure 5.16. The results for the former are only based on 10 different runs. Also averaged over 10 different runs the $\mathrm{MLP}_{0,3,0}(0, x, y, z, 0, 0, 0)$ with 4 hidden nodes achieved a correlation of about 0.964. This has to be considered as an equally well performance taken into consideration the relative small number of runs and the similar distribution of the single correlation values $(0.991, 0.922, 0.981)$. Both networks came up with a different solution than the $\mathrm{SMLP}_{0,2,0}(0, x, y, z)$, but failed to outperform the latter despite having twice as much parameters .

5.4 Summary of Chapter 5

The MLP is one of the most important and popular neural networks. In this chapter we dealt with (Spinor) Clifford MLPs with real–valued activation functions. In section 5.1 we derived a general Backpropagation algorithm applicable to Clifford MLPs with arbitrary underlying non–degenerate Clifford algebras. Additionally, Backpropagation algorithms for the Dual Clifford MLP ($\mathrm{MLP}_{0,0,1}$) and the Quaternionic Spinor CMLP ($\mathrm{SMLP}_{0,2,0}$) have been derived.

Then the important topic of universal approximation has been studied. The ideas of that concept together with the classical results have been discussed. In particular, we pointed out the differences between universal approximation in L_p and L_∞ spaces. Known approximation results from the literature for the Complex MLP, and the Quaternionic MLP have been generalized to all non–degenerate Clifford algebras. Also, for the Quaternionic Spinor CMLP universal approximation was proven.

Much attendance was given to the experimental study of Clifford MLPs. In a

first series of experiments on function approximation the two–dimensional Clifford MLPs and the real MLP have been compared. The Dual Clifford MLP ($MLP_{0,0,1}$) turned out to be inappropriate for general function approximation due to the degenerate nature of the underlying Clifford algebra. The MLP, as a model–free architecture, did not outperform the Complex CMLP on a rather simple unspecific task. However, the MLP was totally outperformed with respect to both efficiency and generalization by the respective expert (Clifford MLP) on more specific tasks. That way it was also demonstrated, that our generalization to Clifford algebras as unified framework is also useful for MLP–type architectures. Looking at both the $MLP_{1,0,0}$ and the $MLP_{0,1,0}$ as model–based architectures reveals much more than studying the $MLP_{0,1,0}$ as a singular extensions as done in the literature so far.

In a second experiment a better performance in a time series benchmark could be achieved by the Quaternionic Spinor CMLP than that of the original proposed Quaternionic CMLP. The observed outperforming was due to the fact that the given 3–dimensional problem was also 3–dimensional intrinsically processed by the Quaternionic Spinor MLP but not by the Quaternionic MLP. Both architectures did outperform the real MLP on that particular task. Summarizing the results of all experiments the validity and vitality of our approach could be demonstrated.

Chapter 6

Clifford MLPs with Clifford–Valued Activation Functions

The generalisation of a real MLP to a (Spinor) Clifford MLP can be split in two stages. The first stage generalises the propagation function of the neurons by replacing the scalar product with the Clifford product (CMLP) or the Spinor product (SCMLP). Both architectures still use real–valued activation functions and have been studied in detail in the previous chapter.

From our motivation of Clifford neural computation this is already a very powerful generalisation. The model–based nature of our approach comes from the different transformation properties of the different geometric products. The Spinor Clifford Neurons (and therefore also the SCMLP) have been actually designed from that perspective. Moreover we see our data as geometric entities (like points and lines). That is, from our understanding of Clifford neural computation, the kind of non-linearity added by means of activation functions is not so important. Also, the fundamental concepts of data representation and embedding are not affected by it.

Nevertheless it might be that Clifford–valued activation functions are even better suited and therefore will give better results. The study of MLPs with such activation functions is also indicated for reasons of completeness. In an informal way MLPs with Clifford–valued activation have been already introduced. Formally, such networks can be easily established by just changing the defining property (5.9) of the CMLP[1].

Definition 6.1 (Clifford MLP with Clifford–Valued Activation Functions) *A Clif-*

[1]The same notations and conventions will be used again here.

ford MLP with Clifford–valued activation functions (FCMLP[2]*,) computes a function from* $(\mathcal{C}_{p,q,r})^n$ *to* $(\mathcal{C}_{p,q,r})^m$ *by passing the input through a set of* $\{1, \cdots, L\}$ *fully connected consecutive layers. The output of the* j*-th neuron in the* l*-th layer reads*

$$G^{(l)}(\sum_k (w_{kj}^{(l)} \otimes_{p,q} x_k^{(l-1)} + \theta_j^{(l)}) \,. \tag{6.1}$$

At least for one layer $G^{(l)}$ *denotes a Clifford–valued activation function. That is, in that case, there is no function* $g : \mathbb{R} \to \mathbb{R}$ *such that*

$$G^{(l)}(\ldots) = \sum_{I \in \mathcal{I}} g^{(l)}(\ldots)_I e_I \,. \tag{6.2}$$

Contrary to all architectures before, there is no notion of isomorphic FCMLPs. Because of (6.2) FCMLPs differ about more than just Basic Clifford Neurons. Hence every FCMLP is (to large extents) its own case, and so there is no general treatment possible for this type of architecture.

Complex FCMLPs will be studied first. The common opinion in the neural network community, however, is that this case is already settled in a negative way due to the classical paper [32]. Very recently new ideas have been published [44] which will be incorporated in the discussion. All in all we will try to give a complete view of this case. The theory of complex functions is one of the best understood and most completed mathematical areas. This is not true for the function theory of other Clifford algebras. In the second section of this chapter we therefore concentrate on the other two–dimensional Clifford algebras (hyperbolic and dual numbers) and the corresponding FCMLPs. Such FCMLPs have not been discussed before in the literature. Due to the lack of a well developed function theory, however, we cannot go into same depth as for the Complex FCMLP. The basic aim is to come up with activation functions for the aforementioned Clifford algebras and evaluate them in experiments. This will be done in the final section of this chapter where all developed two–dimensional FCMLPs will be compared with the results of the corresponding CMLPs derived in section 5.3.1.

6.1 Complex–Valued Activation Functions

The basic idea behind a Multilayer Perceptron is the use of a nonlinear activation function as already motivated at the beginning of chapter 5. That way a powerful

[2]The capital ”F” in the acronym stands for ”full” as a shortcut for Clifford–valued activation function.

neural network is obtained capable of solving many practical tasks. Theoretically, this is guaranteed whenever an activation function is used such that the resulting MLP is an universal approximator (Definition 5.6). This, however, gives only one of the desirable properties of an activation function. Further requirements come from the applied training algorithm, i.e. Backpropagation.

As mentioned before, [54] can be seen as the first paper on Complex Multilayer Perceptrons. With this publication complex Backpropagation became widely known for the first time. Ordinary real Backpropagation requires a (real) differentiable activation function. Consequently, that condition was replaced in [54] by complex differentiability. Together with some general nomenclature [3] this notion should be studied first.

A function $G : D \subset \mathbb{C} \to \mathbb{C}$ is complex differentiable in $z_0 \in D$ if for $h \in D$

$$G'(z_0) = \frac{d}{dz_0} G(z_0) := \lim_{h \to 0} \frac{G(z_0 + h) - G(z)}{h} \tag{6.3}$$

exists. Formally, there is no visible difference to the ordinary expression for real differentiability. The idea of complex differentiation becomes more clear when looking at

$$G(z_0 + h) - G(z_0) = ah + o(|h|) \quad (\text{for } h \to 0)\,, \tag{6.4}$$

which is an equivalent formulation of (6.3) (see e.g. [60]). The nature of (6.4) differs from the known notion of differentiation in $\mathbb{R}^n$ in the following sense. In $\mathbb{R}^n$ the differential is a general linear function, whereas the complex differential (6.4) is always a dilation–rotation, because both a and h are complex numbers. Hence G has to be differentiable in the real sense and additional conditions have to be fulfilled by its partial derivatives for (6.3) to hold. With $G(z) = (u(x, y), v(x, y))$ and the standard notations

$$u_x := \frac{\partial u}{\partial x} \quad u_y := \frac{\partial u}{\partial y} \quad v_x := \frac{\partial v}{\partial x} \quad v_y := \frac{\partial v}{\partial y}$$

this then leads to the following well known characterization of complex differentiation by the famous Cauchy–Riemann equations.

Proposition 6.2 (Complex Differentiation) *The function* $G(z) = (u(x, y), v(x, y)) : D \subset \mathbb{C} \to \mathbb{C}$ *is complex differentiable in* $z_0 \in D$ *if and only if,* u *and* v *are real differentiable*

[3] As outlined in the introduction to this chapter what is mathematically essential for FCMLPs is the function theory of the underlying Clifford algebra. For complex numbers this is a well known classical theory and hence we will also use the classical notation here.

in z_0 and the Cauchy–Riemann equations

$$u_x = v_y \tag{6.5a}$$
$$u_y = -v_x \tag{6.5b}$$

hold.

If G is also complex differentiable in a neighborhood around z_0 then G is said to be analytic or holomorphic. Clearly, this is equivalent to the continuity of all partial derivatives of G . A function which is analytic in its whole domain is called entire.

For a Complex FCMLP with an entire activation function G the complex Backpropagation algorithm reads [54] (using the same notation as in the previous chapter)

$$\Delta w_{kj}^{(l)} = \delta_j^{(l)} \otimes_{0,1} \overline{(y_k^{(l-1)})} \tag{6.6a}$$
$$\Delta \theta_j^{(l)} = \delta_j^{(l)}, \tag{6.6b}$$

with

$$\delta_j^{(l)} = \begin{cases} \overline{(G^{(L)})'}\,(d_j - y_j^{(L)}) & \text{if } l = L, \\ \overline{(G^{(l)})'}\,(\sum_m \overline{(w_{jm}^{(l+1)})} \otimes_{0,1} \delta_m^{(l+1)}) & \text{otherwise.} \end{cases} \tag{6.7}$$

Compared with the corresponding CMLP algorithm (Proposition 5.4), obviously, only the error terms $\delta_j^{(l)}$ needed to be modified. If $u = v$ and also $u(x, y) = u(x)$, $v(x, y) = v(y)$ then (6.7) gives (5.31) back again. All in all deriving Backpropagation for analytic functions is not critical. The actual dilemma of Complex FCMLPs comes from the following fact.

Theorem 6.3 (Liouville) *Any bounded entire function is constant.*

An activation function has to be bounded, otherwise a software overflow may occur [32]. In the same paper it was also shown that all partial derivatives have to be bounded as well. Since being entire, the complex logistic function $1/(1 + \exp(-z))$ cannot be bounded. In fact, any value of the form $0 \pm i(n + 1/2)\pi$ is a singularity of that function as illustrated below in figure 6.1. For the same reason the complex tanh function is not bounded. Hence the most popular activation functions for the real MLP turned out to be unsuitable in the complex case.

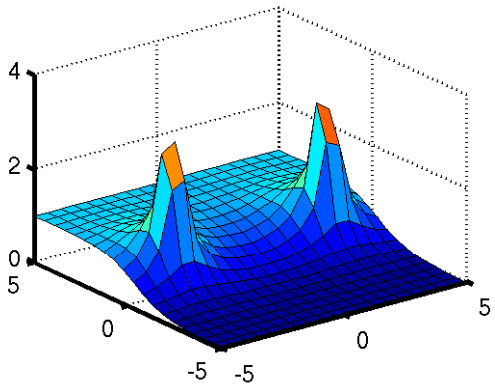

Figure 6.1: Plot of the magnitude of the complex logistic function $1/(1 + \exp(-z)$.

Therefore a new complex activation function was proposed by Georgiou & Koutsougeras [32]

$$\frac{z}{c + \frac{1}{r}\|z\|}\,. \tag{6.8}$$

This function is clearly bounded. However, this comes at a very high price. The nonlinearity of (6.8) reduces to a normalization of the argument. More precisely, the magnitude $\|z\|$ is monotonically squashed to a point in the interval $[0, r)$. The phase of the argument remains unchanged. This clearly is a very serious limitation.

The function (6.8) was generalised by Pearson [62] to Clifford algebras by using the norm $[x \otimes_{p,q} \bar{x}]_0$. This was undoubtable the first proposal of a Clifford neural network. Both papers [32, 62] did not provide any real experimental support for the usefulness of the proposed activation functions. Experiments in both papers were only performed for encoder–decoder problems with binary data, something clearly of limited validity. Moreover, even for these rather trivial setups the obtained results are not very convincing. If '1' is an expected output value for a certain pattern then very often the actual learned value is below '0.8' (page 34, [62]). Anyway, the function (6.8) has never been thought as a real useful alternative by the neural network community, because of keeping the argument phase constant. Hence people turned–back to component–wise activation functions like

$$g(z) = g(x + iy) = \frac{1}{1 + \exp(-x)} + i\frac{1}{1 + \exp(-y)}\,. \tag{6.9}$$

Often this 'retreat' was seen as the failure of complex neural networks, or, the resulting networks from (6.9) have been recognized as only light–weighted. In the

previous chapter we could refute this view, because of applying a wider framework and having identified the geometric product as the key for Clifford neural computation.

In [2] the following further backstroke for Complex FCMLPs has been proven.

Theorem 6.4 ([2]) *Complex FCMLPs having* (6.9) *as activation function are only universal approximators in* L_∞ *for the class of analytic functions, but not for the class of complex continuous functions.*

This very important result renders networks with the complex logistic function weaker than those applying the real logistic function separately to every component, and of course weaker than the classical real MLP with logistic activation function. Although Theorem 6.4 states only the case of a particular function it may be seen as an indication that this negative result could hold for all analytical functions. At least for closely related function like the complex tanh this is very likely. Note that the L_∞ norm is very strong and usually nicely resembles the practical power of the involved neural networks. Indeed, Theorem 6.4 was also illustrated experimentally in [2], to which we will come back later.

Summarising all of the above the case of Complex FCMLPs looks settled down in a negative way. In the remainder of this section we want to study the recent work of Kim & Adali [44] on those kind of networks and see if we have to rethink our opinion. In fact nothing less than the 9 functions listed in table 6.1 have been proposed in this paper.

$f(z)$	$\frac{d}{dz}f(z)$	Type of Singularity
$\tan z$	$\sec^2 z$	isolated
$\sin z$	$\cos z$	removable
$\arctan z$	$\frac{1}{1+z^2}$	isolated
$\arcsin z$	$(1-z^2)^{-1/2}$	removable
$\arccos z$	$-(1-z^2)^{-1/2}$	removable
$\tanh z$	$\operatorname{sech}^2 z$	isolated
$\sinh z$	$\cosh z$	removable
$\operatorname{arctanh} z$	$(1-z^2)^{-1}$	isolated
$\operatorname{arcsinh} z$	$(1+z^2)^{-1}$	removable

Table 6.1: List of activation functions proposed in [44].

Of course, all those functions have singularities. This is illustrated in figure 6.2 for

the complex sin function and in figure 6.3 for the complex tanh function. Contrary to the common opinion, this is not seen as very critical by the authors in [44].

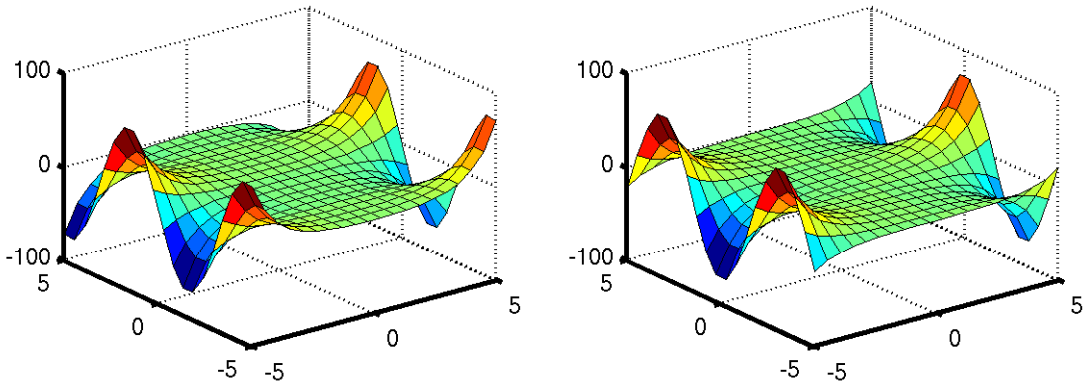

Figure 6.2: Plot of the complex sin function. Real part (left) and imaginary part (right).

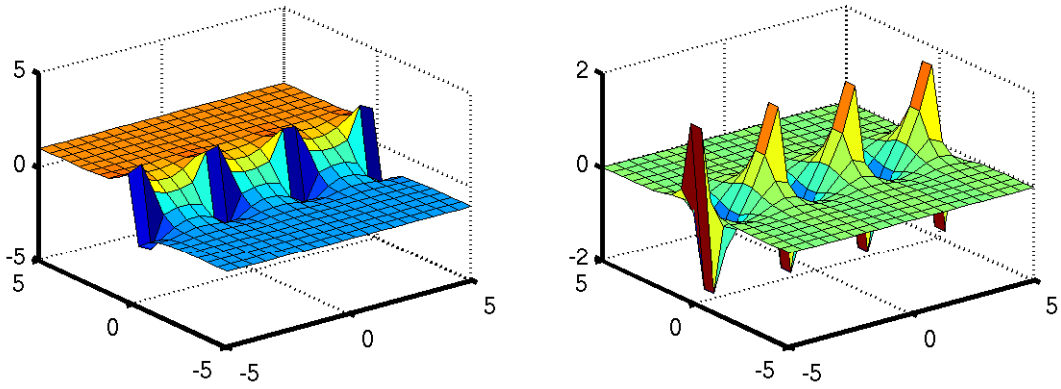

Figure 6.3: Plot of the complex tanh function. Real part (left) and imaginary part (right).

The authors motivate all the aforementioned activation functions by proving that they all give rise to neural networks that are universal approximators. This is impossible in the L_∞ norm. However, it is still possible in the weaker L_1 norm. In the latter removable singularities do no harm, since those points have zero measure. Roughly speaking, they do not count[4]. For activation functions having isolated singularities one already ends up with "density" only in the analytic deleted

[4]This is, of course, oversimplified. For the correct mathematics the reader is referred to the original paper [44], which is very precise.

neighborhood of the singularity. Practically, this means one has to avoid the singularities, for example by scaling the data [5].

All this may be possible. However, we do not see any advantage by doing so. Hence Complex FCMLPs remain not very promising.

6.2 General Clifford–Valued Activation Functions

From the previously studied complex case there seems to be little hope for FCMLPs in other Clifford algebras. Actually, we can already exclude many further algebras by the following argument. First note that a Clifford–valued activation function is suitable (e.g. bounded) if all of its component functions are suitable. This fact can be used in a bottom–up manner as follows. If the underlying algebra of a FCMLP contains a subalgebra for which the intended activation functions is known to be unsuitable, then this function is unsuitable for the FCMLP in question as well.

For example, the complex numbers ($\mathcal{C}_{0,1}$) are a subalgebra of the quaternions ($\mathcal{C}_{0,2}$). Hence the quaternionic logistic function is also unbounded. Neither could it give rise to universal approximation (w.r.t. L_∞) since this does not hold for the complex case. One may argue that such things become more and more less important when proceeding to higher dimensional algebras since less and less components are affected. This is somehow true, but it hardly justify the efforts. Particularly not from our point of view where the specific form of the nonlinearity is not that important.

According to the above subalgebra argument only two cases (hyperbolic and dual numbers) remain for the further study. Obviously, they are by no means affected by the former results for complex numbers. Every Clifford algebra has its own function theory. Hence we have to start from scratch again, beginning with the hyperbolic case first.

So let the function $G(z) = (u(x,y), v(x,y)) : D \subset \mathcal{C}_{1,0} \to \mathcal{C}_{1,0}$ be fixed[6]. Nothing has changed with regard to the notion of boundness. However, we need to define what is meant by "hyperbolic" differentiable. Formally, as in the complex case,

$$G'(z_0) = \frac{d}{dz_0} G(z_0) := \lim_{h \to 0} ((G(z_0 + h) - G(z)) \otimes_{1,0} h^{-1}) \tag{6.10}$$

has to exist for G being hyperbolic differentiable in z_0[7]. The limes (6.10) has to be

[5] This is also concluded and actually demonstrated in [44].

[6] We use a similar notation as for complex numbers since there is little danger of confusion.

[7] Some care has to be taken since $\mathcal{C}_{1,0}$ is not a field.

independent of the direction. In particular, we can use the orthogonal basis vectors to get a characterization by the partial derivatives similar to the Cauchy–Riemann equations (6.5).

Proposition 6.5 (Hyperbolic Differentiation) *The function* G *is hyperbolic differentiable in a point* z_0 *if and only if* u *and* v *are real differentiable in* z_0 *and the following equations hold*

$$u_x = v_y \tag{6.11a}$$
$$u_y = v_x\,. \tag{6.11b}$$

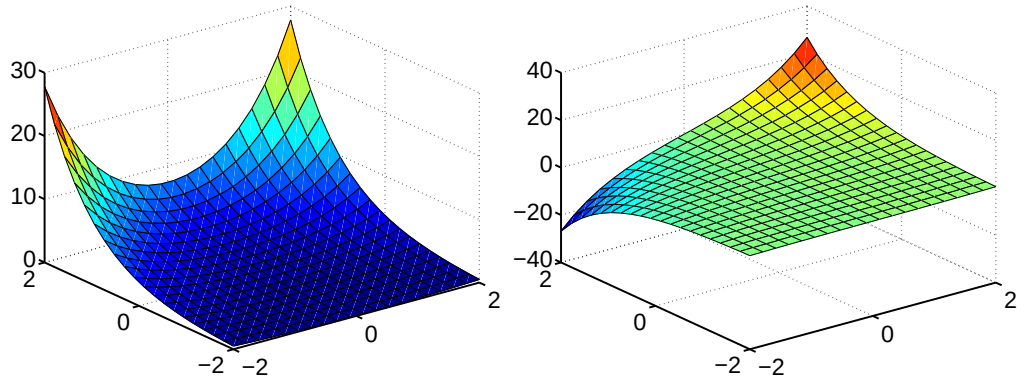

Figure 6.4: Plot of the hyperbolic exp function. Real part (left) and imaginary part (right).

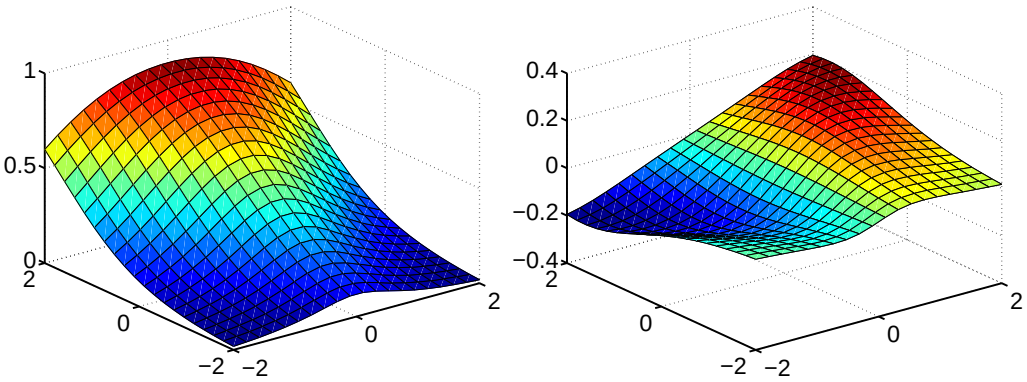

Figure 6.5: Plot of the hyperbolic logistic function. Real part (left) and imaginary part (right).

Remembering the matrix representation of hyperbolic numbers this comes as no surprise. If G is hyperbolic differentiable in a point z_0 and all partial derivatives

are continuous in z_0, then G is said to be holomorphic in this point. A function which is holomorphic everywhere is called entire as in the complex case before. For such a function the update part of the hyperbolic Backpropagation algorithm reads

$$\Delta w_{kj}^{(l)} = \delta_j^{(l)} \otimes_{0,1} y_k^{(l-1)} \tag{6.12a}$$

$$\Delta \theta_j^{(l)} = \delta_j^{(l)}, \tag{6.12b}$$

with

$$\delta_j^{(l)} = \begin{cases} (G^{(L)})' \, (d_j - y_j^{(L)}) & \text{if } \; l = L, \\ (G^{(l)})' \, (\sum_m w_{jm}^{(l+1)} \otimes_{0,1} \delta_m^{(l+1)}) & \text{otherwise.} \end{cases} \tag{6.13}$$

The above is basically the same algorithm as for the complex case where conjugation has been replaced by the identity now. That means a general version of Backpropagation for FCMLPs with underlying non–degenerate algebra could be easily derived similar to Proposition 5.4.

In every reasonable function theory the exponential function is entire [8]. Since every Clifford algebra is isomorphic to some matrix algebra the exponential function can always be defined as

$$\exp(A) = \sum_{n=0}^{\infty} \frac{A^n}{n!} \, . \tag{6.14}$$

Using the standard isomorphism (2.30) one gets for hyperbolic numbers [80]

$$\exp(x + y e_1) = \exp(x) + (\cosh(y) + \sinh(y) e_1) \tag{6.15}$$

as analogon to the famous Euler formula of complex numbers. The hyperbolic exponential function is illustrated in figure 6.4. The formula (6.15) can be used for the direct computation of the hyperbolic logistic function (in analogy to (5.8)), which is shown in figure 6.5.

Contrary to the complex case, the hyperbolic logistic function is bounded. This is due to the absence of singularities. Thus, in general terms, this seems to be a suitable activation function. Concretely, the following facts, however, might be of disadvantage. The real and imaginary part have different squashing values. Both component functions do only significantly differ from zero around the lines $x = y\, e_1 (x > 0)$ and $-x = y\, e_1 (x < 0)$. This finishes our study of Hyperbolic CMLPs.

[8] This is of course a rather personal statement from the practical point of view than a general established mathematical truth.

The dual numbers always showed exceptional behaviour due to the degenerate nature of the algebra. The function theory of dual numbers was already developed by Study at the beginning of the last century [84]. Unfortunately, the resulting theory is not very rich [9].

Starting as always with the differential quotient gives the following notion of differentiation for dual numbers.

Proposition 6.6 (Dual Differentiation) *The function* $G(z) = (u(x,y), v(x,y)) : D \subset \mathcal{C}_{0,0,1} \to \mathcal{C}_{0,0,1}$ *is dual differentiable in a point* z_0 *if and only if* u *and* v *are real differentiable in* z_0 *and the following equations hold*

$$u_x = v_y \tag{6.16a}$$
$$u_y = 0\,. \tag{6.16b}$$

Dual functions which are also differentiable in a neighborhood of a point are called synektic due to Study. Every such function can be expanded according to

$$G(x + ye_1) = G(x) + y\,\frac{\partial G(x)}{\partial x}\,e_1\,. \tag{6.17}$$

The dual exponential function and the thereof derived trigonometric functions are synektic. For the exponential function (6.17) yields

$$\exp(x + ye_1) = \exp(x) + y\exp(x)e_1\,, \tag{6.18}$$

and, for example, for the dual sine function we get

$$\sin(x + ye_1) = \sin(x) + y\cos(x)e_1\,. \tag{6.19}$$

Of course no further investigations are needed. For every synektic function we have

$$u(x,y) = u(x)\,. \tag{6.20}$$

Hence there is no Dual FCMLP. More precisely, it would always be identical to the ordinary Dual CMLP in the first component function. The latter being already identified to be responsible for the insufficiencies of the Dual CMLP as shown in section 5.4.

At the beginning of this chapter we neglected the usefulness of the quaternionic logistic function. Nevertheless the quaternionic case might still be of interest so that some further remarks seem to be in order. If one uses the standard differential

[9] This does not affect the richness of its applications.

quotient approach (likewise (6.3)) one ends up with a situation where only linear functions are holomorphic. In other Clifford algebras things are even more complicated. The function theory of Clifford algebras is an ongoing research area of mathematics. For an overview of its currents state we refer to [30].

6.3 Experimental Results

This section is rather short. However this could be anticipated from our theoretical study of FCMLPs in the prevoius sections. For the experiment we choose the function g (5.52) already known from section 5.3.1. Among the functions tested therein g turned out to be the most simple function to learn in the experiments for the CMLPs. For the $\mathrm{FMLP}_{0,1,0}$ with complex logistic activation function the results could be compared with those reported in [2]. From the activation functions proposed in [44] the complex sin and tanh function have been selected since both have been already discussed in section 6.1. Finally, the $\mathrm{FMLP}_{1,0,0}$ with hyperbolic logistic activation function was also tested.

The test setup was exactly the same as reported for the experiments in section 5.3.1. In table 6.2 all results averaged over 30 runs are reported.

Network	Activation	Nodes	Error [2]	MSE Training	MSE Test
$\mathrm{FMLP}_{0,1,0}$	logistic	2	13.96	0.1123	0.1243
		3	—	0.0987	0.0995
		4	12.94	0.0956	0.1013
$\mathrm{FCMLP}_{0,1,0}$	sin	2	—	1.0567	1.3723
		3	—	0.7652	0.5891
		4	—	0.7187	0.6439
$\mathrm{FMLP}_{0,1,0}$	tanh	2	—	0.5431	0.4812
		3	—	0.5292	0.5023
		4	—	0.4912	0.5010
$\mathrm{FMLP}_{1,0,0}$	logistic	2	—	0.0987	0.0873
		3	—	0.0782	0.0569
		4	—	0.0657	0.0682

Table 6.2: Results for the approximation of g for different FCMLPs.

Obviously, all networks failed to learn the function g in such a way that it makes no sense to study the in between differences. Actually, all results are worse than that

of the $CMLP_{0,0,1}$, with the only exception of the $MLP_{1,0,0}$. This however is only due to the fact that in this case the training data was scaled to lie in $[0, 0.2]$, which was necessary because of the output range of the hyperbolic logistic function (see figure figure 6.5 again). The failure for the $MLP_{0,1,0}$ with complex logistic function is in accordance to the results in [2]. The positive results reported in [44] may be due to the specific value range of the considered problem, avoiding the singularities of the used activation function (similar to the observed effect for the $MLP_{1,0,0}$).

6.4 Summary of Chapter 6

Historically, Complex MLPs have been first proposed with so–called fully complex activation functions such as the complex logistic function. This was also widely seen as mandatory. Only after recognizing several problems of that approach the attention turned towards Complex MLPs with real–valued activation functions.

Our investigations started with the review of the classical literature on Complex FMLPs. Particulary, we dealt with a bounded but phase–constant activation function proposed by Georgiou & Koutsougeras [32]. We argued why the latter property renders this function useless and therefore also its generalization to Clifford algebras as proposed by Pearson [62]. We had a closer look at a negative result [2] regarding the approximation capabilities of Complex FCMLPs with the complex logistic function as activation function. We also had a look at recently claimed positive results [44], for which we argued against practical relevance due to the involved weaker L_1 norm.

In the second section we studied FCMLPs in other Clifford algebras. For hyperbolic numbers we derived the necessary elements from hyperbolic function theory to define the hyperbolic logistic function. The Backpropagation algorithm for Hyperbolic FCMLPs has been derived and it was sketched how a unified Backpropagation algorithm for all non–degenerate Clifford algebras can be derived. The function theory of dual numbers showed elegantly that there is no such thing as a Dual FCMLP. Actually, we could prove that such a network would be of the same insufficient computational power as the ordinary Dual CMLP.

The experiments performed in the third section of this chapter then showed indeed the theoretically claimed drawbacks of FCMLPs. Therefore the case of FCMLPs can be seen as settled down in a negative way. To extend this common view on Complex FCMLPs to FCMLPs in other algebra is the main contribution of this chapter.

Chapter 7

Conclusion

In this final chapter we want to summarize what was done and outline what could be possible directions for further research.

7.1 Summary

The goal of this thesis was to design neural architectures that are capable of processing representations of geometric nature (like points and lines), are interpretable in a model–based sense, and therefore offer advantages over standard architectures. For that purpose the framework of Clifford algebra was used, and Clifford neural computation was developed as an algebraic theory.

We entered the world of Clifford neural computation at the most basic level of a single neuron. That Basic Clifford Neuron (BCN) "is" the geometric product. Therefore there was never a gap between the "algebraical" world and the "neural" one. Moreover, all the key properties of Clifford neural computation derived naturally from the mathematical theory.

Since every Clifford algebra is isomorphic to some matrix algebra every BCN performs a particular linear transformation. That way the model–based nature of Clifford algebra became visible immediately when comparing the BCNs with the Linear Associator. Also, differences among the two–dimensional BCNs could be detected from the different optimal learning rates. Motivated as the geometric generalization of the non–degenerate two–dimensional BCNs the so–called Spinor Clifford Neurons (SCNs) has been introduced. The SCN mimics the operation of the Clifford group and uses non-commutativity as a design feature.

The algebraical notion of isomorphic structures has been carried over to Clifford neurons. The meaning and the requirements for isomorphic Clifford neurons have been derived. Using this concept isomorphic Clifford architectures could be identified allowing the systematical study of Clifford neural computation.

The main motivation for our study of Clifford neural computation was that it allows to select different modes (internal representations) by just presenting different coded data (external representations). For example, using a certain linear representation we could "select" the propagation function of a $\mathrm{BCN}_{1,1}$ or $\mathrm{BCN}_{0,2}$ to be an affine transformation. Both turned out to be more robust than a Linear Associator. Using the conformal model of Clifford algebra a $\mathrm{SCN}^{\sim}_{1,2}$ was shown to be able to compute Möbius transformations in a linear way. This is impossible for any real–valued network. Another Clifford example utilized Plücker coordinates for the processing of lines. All this have been direct applications of Clifford algebra properties, particularly the graded subspace structure and isomorphisms of the Clifford group. For the systematic study of representation issues the notion of isomorphic representations has been introduced.

In the second part of the thesis we proceed from neurons to Clifford Multilayer Perceptrons (CMLPs). Two stages of generalization of an ordinary MLP have been discussed. First we studied CMLPs having real–valued activation functions. The cases of complex and quaternionic CMLPs are known from the literature. Theoretically, we extended these results by generalizing Backpropagation to arbitrary non–degenerate CMLPs. Also the learning algorithms for the Dual CMLP and the Quaternionic Spinor CMLP (QSCMLP) have been derived. The known universal approximation results have been extended to CMLPs with logistic activation function for arbitrary non–degenerate Clifford algebras and the QSCMLP.

An important part in the study of (S)CMLPs was devoted to experiments, the first being a series of 2D function approximation tasks. It turned out that the Dual CMLP is not a universal approximator and unsuitable. The model–free MLP did not outperform the Complex CMLP on a rather simple task. However it was outperformed by the matching Clifford expert both with respect to efficiency and generalization. Particularly, the Hyperbolic CMLP turned out to be an architecture of full value, compared to the Complex CMLP, which have been only studied in the literature. In a second experiment on the prediction of the chaotic Lorenz attractor the QSCMLP outperformed the original proposed Quaternionic MLP due to the fact of applying a three dimensional activation function. Therefore the experiments on (S)CMLPs showed some nice results which are in accordance to our view on Clifford neural computation. The model–based core is the (spinor) geometric

product, which also manifests itself in a (S)CMLP.

Finally Clifford MLPs with Clifford–valued activation functions (FCMLPs) have been studied. Different points of view found in the literature have been clarified. Nothing is gained by choosing such functions. Contrary, due the limitations from complex function theory, no suitable complex functions do exist. In case of hyperbolic numbers no general restrictions came from the underlying function theory, but the most natural activation function (the hyperbolic logistic function) did also fail practically.

The foundations of a theory of Clifford neural computation has been developed during the run of this thesis. The designed Clifford neurons were shown to process geometric entities like lines and points. The geometric interpretation of their propagation functions has been given. How to process different entities and use different geometric models by construction embedding has been illustrated. A systematic methodology for handling representation issues was introduced. The (Spinor) Clifford MLPs based on the developed Clifford neurons have showed advantages over standard MLPs. Hence the goal of designing useful architectures by processing representations has been reached. However, the provided basis of experiments was quite small. Also, no real world applications have been considered.

7.2 Outlook

Of course we only undertook the very first steps in this work. As shown in the summary, the theory of Clifford neural computation is not fully developed yet. Therefore future work can be directed to many different aspects.

First of all we only studied the geometric product and the spinor product. There is much more possible in Clifford algebra. It contains operations for computing union and intersection of all kinds of geometric objects. Every Lie group corresponds to a Clifford group. Every Clifford group is a manifold. All this mathematical notions are important in many areas of computer vision and robotics. For many of them promising Clifford algorithms already exist. Neural versions of such algorithms may be of interest and of practical benefit.

In the thesis we only dealt with supervised learning. There are some known results on complex Independent Component Analysis (ICA). This could be a further direction for more unsupervised learning approaches. Also, associative architecture may be of interest. Especially, from the theoretical point of view since many

known closed solutions exist for real–valued cases.

The training algorithm used by us has always been plain Backpropagation. This was sufficient for understanding the principles of Clifford neural computation, which was our main concern. For practical applications more sophisticated algorithms for training Clifford neural networks may be handy. For real world applications, such as control tasks, recurrent Clifford neural networks may be of interest too.

Every Clifford neuron can be seen as some kind of expert. In our opinion, thinking of mixtures of such experts seems to be the most natural next step. In the very end this might lead to a higher level of learning, where the actual structure is learned during training, or, structure changes as a consequence of interaction with the environment. Of course, this is the most ambiguous possible goal. However, it is one of the most urgent challenges for neural networks.

We hope the methods presented here provide a good basis for future work in the area of Clifford neural computation.

Appendix A

Supplemental Material

A.1 Dynamics of the Linear Associator

This is of course a standard topic in any text book on neural computation (see e.g. [8, 70]). A treatise above that level is given in the survey [3]. The facts reviewed in the following are mostly taken from [50, 51].

For a Linear Associator (LA) the standard mean–squared error function for batch learning reads

$$E_{LA} = \frac{1}{2P} \sum_{p=1}^{P} \| d^p - Wx^p \|_2 \ . \tag{A.1}$$

Applying gradient descent then results in the following update rule for the weights of a LA

$$w_{ij}(t+1) = w_{ij}(t) - \eta \frac{1}{P} \sum_{p=1}^{P} (d_j^p - w_{ij}(t) x_i^p) x_i^p \, , \tag{A.2}$$

where $x = (x_1, \ldots, x_n)^T$, $y = (y_1, \ldots, y_m)^T$ and $\eta > 0$ is the learning rate. In general, gradient descent only guarantees local convergence. Here, however, the error function (A.1) is convex. Therefore, there always exists a learning rate η yielding global convergence.

The dynamics of a one–dimensional Linear Associator is described in figure A.1. First of all, since only one direction has to be considered, convergence can always be achieved in one batch step by choosing the optimal learning rate η_{opt} (top right plot of figure A.1). Choosing a learning rate $\eta < \eta_{opt}$ results in slower convergence (top left plot of figure A.1). The same holds true for a learning rate $\eta \in (\eta_{opt}, 2\eta_{opt}]$

(bottom left plot of figure A.1). Finally, for $\eta > 2\eta_{opt}$ learning does not converge (bottom right plot of figure A.1).

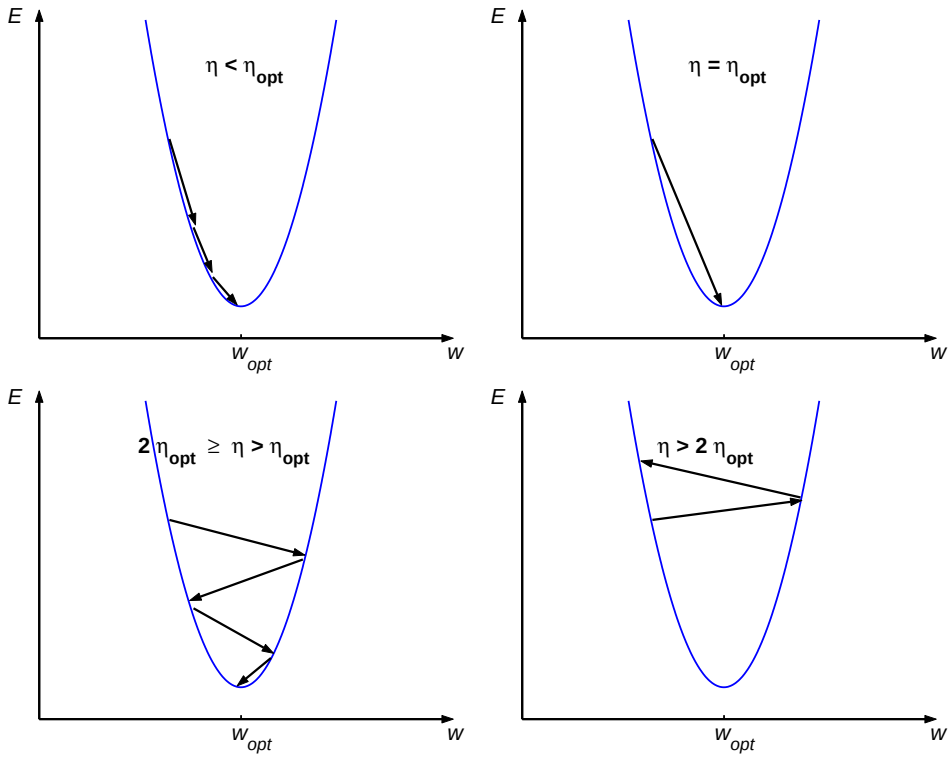

Figure A.1: Dynamics of a one–dimensional Linear Associator. Redrawn from [50]. See text for details.

All of the above relatively to η_{opt} remains also valid in higher dimensions. The optimal learning rate η_{opt} for a LA can be found by analysing the second–order properties of (A.1), i.e. by examining the corresponding Hessian matrix.

Proposition A.1 ([50]) *The Hessian matrix of* (A.1) *is given by*

$$H_{LA} = \frac{1}{P} \sum_{i=1}^{P} x^{p^T} x^p \,, \tag{A.3}$$

i.e. by the autocorrelation matrix of the inputs.

Note that H_{LA} is a symmetric and non–negative matrix. Hence all its eigenvalues are real and non–negative. Furthermore, if the matrix is of full rank then (A.1) has exactly one global minimum. The optimal learning rate is computed as follows.

Proposition A.2 ([50]) *Let λ_{max} and λ_{min} denote the largest and smallest eigenvalue of H_{LA}, respectively. Then the optimal learning rate for a LA using gradient descent is*

$$\eta_{opt} = \frac{1}{\lambda_{max}} . \tag{A.4}$$

The speed of convergence is proportional to

$$\frac{\lambda_{max}}{\lambda_{min}} . \tag{A.5}$$

It is therefore desirable to have a small eigenvalue spread (A.5). If all eigenvalues are equal then fastest convergence of learning is achieved. If additionally the coordinate system spanned by the eigenvectors is in alignment with the coordinate system of the data then convergence can be reached in one step. The alignment of the two coordinate systems can always be forced by applying the appropriate coordinate transformation to (A.1) [51]. In that new coordinate system the weight update equations would be decoupled allowing to use a separate learning rate for each direction. However, this is rather of theoretical interest since it practically means to solve everything in advance.

A.2 Update Rule for the Quaternionic Spinor MLP

For the Quaternionic Spinor MLP the weight update rule is given below[1]. The notations used are that of section 5.1. Setting $\delta_j^{(l)}$ as in (5.18) gives the update rule for a weight in the output layer and setting $\delta_j^{(l)}$ as in (5.27a) gives the update rule for a weight in a hidden layer.

$$
\begin{aligned}
\Delta w_{kj}^{(l)} &= \big(([\delta_j^{(l)}]_0 w_0 x_0 + ([\delta_j^{(l)}]_1 (w_0 x_1 + w_2 x_{12} - w_{12} x_2) \\
&+ ([\delta_j^{(l)}]_2 (w_0 x_2 - w_1 x_{12} + w_{12} x_1) + ([\delta_j^{(l)}]_{12} (w_0 x_{12} + w_1 x_2 - w_2 x_1) e_0\big) \\
&+ \big(([\delta_j^{(l)}]_0 w_1 x_0 + ([\delta_j^{(l)}]_1 (w_1 x_1 + w_2 x_2 + w_{12} x_{12}) \\
&+ ([\delta_j^{(l)}]_2 (w_2 x_1 - w_0 x_{12} - w_1 x_2) + ([\delta_j^{(l)}]_{12} (w_{12} x_1 + w_0 x_2 - w_1 x_{12}) e_1\big) \\
&+ \big(([\delta_j^{(l)}]_0 w_2 x_0 + ([\delta_j^{(l)}]_1 (-w_2 x_1 + w_1 x_2 + w_0 x_{12}) \\
&+ ([\delta_j^{(l)}]_2 (w_1 x_1 + w_2 x_2 + w_{12} x_{12}) + ([\delta_j^{(l)}]_{12} (w_{12} x_2 - w_0 x_1 - w_2 x_{12}) e_2\big) \\
&+ \big(([\delta_j^{(l)}]_0 w_{12} x_0 + ([\delta_j^{(l)}]_1 (w_1 x_{12} - w_0 x_2 - w_{12} x_1) \\
&+ ([\delta_j^{(l)}]_2 (w_0 x_1 + w_2 x_{12} - w_{12} x_2) + ([\delta_j^{(l)}]_{12} (w_1 x_1 + w_2 x_2 + w_{12} x_{12}) e_{12}\big)
\end{aligned}
$$

[1] Remember that updating the bias terms of a (S)CMLP is generic.

A.3 Some Elements of Clifford Analysis

This appendix provides some supplemented material on Clifford analysis which was used in the discussion of universal approximation by Clifford MLPs in section 5.2. All the beneath is taken from [10].

In general $(\mathcal{C}_{p,q})^n$ is not a linear space. A module is a generalization of a linear space, where the coefficients come from some ring $(\mathcal{C}_{p,q})$ instead of a field.

Definition A.3 (Left Module) *Let R be a ring with identity element 1. A left R-module X is an Abelian group $(X, +)$ together with a mapping $R \times X \to X : (r, g) \mapsto rg$ in such a way, that*

$$\begin{aligned}
&(a) \quad \forall g_1, g_2 \in X \ \forall r \in R : r(g_1 + g_2) = rg_1 + rg_2 \\
&(b) \quad \forall g \in X \ \ \forall r_1, r_2 \in R : (r_1 + r_2)g = r_1 g + r_2 g \\
&(c) \quad \forall g \in X \ \ \forall r_1, r_2 \in R : \quad (r_1 r_2)g = r_1(r_2 g) \\
&(d) \quad \forall g \in X : \qquad\qquad\qquad 1g = g
\end{aligned}$$

are fulfilled.

A left module may be equipped with a seminorm.

Definition A.4 (Seminorm) *Let X be a left $\mathcal{C}_{p,q}$-module. A function $p : X \to \mathbb{R}$ is called a seminorm on X if it fulfills for all $f, g \in X, \lambda \in \mathcal{C}_{p,q}$ and $\kappa \in \mathbb{R}$*

$$\begin{aligned}
&(a) \quad p(f + g) \leq p(f) + p(g) \\
&(b) \quad p(f) = 0 \Rightarrow f = 0 \\
&(c) \quad p(\lambda f) \leq C\|\lambda\| p(f) \\
&\qquad p(\kappa f) = |\kappa| p(f) \, .
\end{aligned}$$

A proper module is a left module equipped with a proper system of seminorms.

Definition A.5 (Proper Module) *Let X be a left $\mathcal{C}_{p,q}$-module. A family P of seminorms $p : X \to \mathbb{R}$ is called a proper system of seminorms on X if for any finite sequence $p_1, p_2, \ldots, p_k \in P$ there exist $p \in P$ and $C > 0$ such that for all $f \in X$*

$$\sup_{j=1,\ldots,k} p_j(f) \leq C p(f) \, . \tag{A.6}$$

After these preliminaries the Hahn–Banach theorem and the Riesz representation theorem of Clifford analysis can be stated.

Theorem A.6 (Hahn–Banach) *Let X be a proper $\mathcal{C}_{p,q}$-module, let Y be a submodule of X and let T be a bounded left $\mathcal{C}_{p,q}$-functional on Y. Then there exists a bounded left $\mathcal{C}_{p,q}$-functional T^* on X such that*

$$T^*_{|Y} = T\,. \tag{A.7}$$

Theorem A.7 (Riesz) *Let X be a proper $\mathcal{C}_{p,q}$-module, let $T : X \to \mathcal{C}_{p,q}$ be a bounded linear functional. Then there exists a unique Clifford measure μ on X such that for all continuous $f : X \to \mathcal{C}_{p,q}$*

$$\langle T, f\rangle = \int_X f\, d\mu\,. \tag{A.8}$$

Bibliography

[1] P. Arena, L. Fortuna, G. Muscato, and M. G. Xibilia. *Neural Networks in Multidimensional Domains*. Number 234 in LNCIS. Springer–Verlag, 1998.

[2] P. Arena, L. Fortuna, R. Re, and M. G. Xibilia. Multilayer perceptrons to approximate complex valued functions. *International Journal of Neural Systems*, 6:435–446, 1995.

[3] P. F. Baldi and K. Hornik. Learning in linear neural networks: A survey. *IEEE Transactions on Neural Networks*, 6(4):837–858, 1995.

[4] V. Banarer, C. Perwass, and G. Sommer. Design of a multilayered feed-forward neural network using hypersphere neurons. In N. Petkov and M. A. Westenberg, editors, *Proc. 10th Int. Conf. Computer Analysis of Images and Patterns, CAIP 2003, Groningen, The Netherlands, August 2003*, volume 2756 of *LNCS*, pages 571–578. Springer-Verlag, 2003.

[5] V. Banarer, C. Perwass, and G. Sommer. The hypersphere neuron. In *11th European Symposium on Artificial Neural Networks, ESANN 2003, Bruges*, pages 469–474. d-side publications, Evere, Belgium, 2003.

[6] A. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39:930–945, 1993.

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

[8] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, 1995.

[9] H. Boulard and N. Morgan. *Connectionist Speech Recognition*. Kluwer, 1993.

[10] F. Brackx, R. Delanghe, and F. Sommen. *Clifford Analysis*. Pitman, London, 1982.

[11] S. Buchholz and G. Sommer. A hyperbolic multilayer perceptron. In S.-I. Amari, C.L. Giles, M. Gori, and V. Piuri, editors, *International Joint Conference on Neural Networks, IJCNN 2000, Como, Italy*, volume 2, pages 129–133. IEEE Computer Society Press, 2000.

[12] S. Buchholz and G. Sommer. Learning geometric transformations with Clifford neurons. In G. Sommer and Y. Zeevi, editors, *2nd International Workshop on Algebraic Frames for the Perception-Action Cycle, AFPAC 2000, Kiel*, volume 1888 of *LNCS*, pages 144–153. Springer-Verlag, 2000.

[13] S. Buchholz and G. Sommer. Quaternionic spinor MLP. In *8th European Symposium on Artificial Neural Networks, ESANN 2000, Bruges*, pages 377–382, 2000.

[14] S. Buchholz and G. Sommer. Clifford algebra multilayer perceptrons. In Sommer [83], pages 315–334.

[15] S. Buchholz and G. Sommer. Introduction to neural computation in Clifford algebra. In Sommer [83], pages 291–314.

[16] T. Caelli, D. Squire, and T. Wild. Model–based neural networks. *Neural Networks*, 6:613–625, 1993.

[17] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363, 1936.

[18] T. Clarke. Generalization of neural network to the complex plane. *Proc. of the IJCNN*, 2:435–440, 1990.

[19] W. K. Clifford. Preliminary scetch of bi–quaternions. *Proc. London Math. Soc.*, 4:381–395, 1873.

[20] J. Cnops. *Hurwitz pairs and applications of Möbius transformations*. Habilitation dissertation, University of Gent, Belgium, 1994.

[21] B. J. Copeland. *Artificial Intelligence: A Philosophical Introduction*. Blackwell Publishers, Oxford, 1993.

[22] K. J. W. Craik. *The Nature of Explanation*. Cambridge University Press, Cambridge, 1943.

[23] G. Cybenko. Approximation by superposition of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

[24] K. Daniilidis. Using the algebra of dual quaternions for motion alignment. In Sommer [83], pages 489–499.

[25] B. Denby. The use of neural networks in high–energy physics. *Neural Computation*, 5:505–549, 1993.

[26] H. English and Y. Hiemstra. The correlation as cost function in neural networks. In *Proc. of the IEEE WCCI, Orlando*, pages 1370–1372, 1994.

[27] H.-D. Ebbinghaus et al. *Numbers*. Springer–Verlag, 3rd edition, 1995.

[28] B. G. Farley and W. A. Clark. Simulations of self–organizing systems by digital computer. *I.R.E. Transactions on Information Theory*, 4:76–84, 1954.

[29] M. Ferraro and T. Caelli. Neural computation of algebraic and geometrical structures. *Neural Networks*, 11:669–707, 1998.

[30] S. G. Gal. *Introduction to Geometric Function Theory of Hypercomplex Variables*. Nova Science Publishers, New York, 2002.

[31] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

[32] G. Georgiou and C. Koutsougeras. Complex domain backpropagation. *IEEE Trans. on Circuits and Systems II*, 39(5):330–334, 1992.

[33] W.R. Hamilton. *Elements of Quaternions*. Longmans Green, London 1866. Chelsea, New York, 1969.

[34] S. J. Hanson and C. R. Olson. Neural networks and natural intelligence: Notes from Mudville. *Connection Science*, 3:332–335, 1991.

[35] D. O. Hebb. *The Organization of Behaviour*. John Wiley & Sons, New York, 1949.

[36] R. Hecht-Nielsen. *Neurocomputing*. Addison–Wesley Publishing Company, Inc., 1991.

[37] W. Hein. *Struktur– und Darstellungstheorie der klassischen Gruppen*. Springer–Verlag, 1990.

[38] D. Hestenes. *Space–Time Algebra*. Gordon and Breach, New York, 1966.

[39] D. Hestenes. *New Foundations for Classical Mechanics*. D. Reidel Publishing, Dordrecht, 1986.

[40] D. Hestenes and G. Sobczyk. *Clifford Algebra to Geometric Calculus*. D. Reidel Publ. Comp., Dordrecht, 1984.

[41] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[42] L. Jones. A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression and neural networks. *Annals of Statistics*, 20:608–613, 1992.

[43] L. B. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 1996.

[44] T. Kim and T. Adali. Complex backpropagtion neural network using elementary transcendetal activation functions. *Proc. of IEEE ICASSP*, pages 1281–1284, 2001.

[45] V. V. Kisil. Möbius transformations and monogenic Functional Calculus. *Electronic Reasearch Announcements of the AMS*, 2(1):26–33, 1996.

[46] S. C. Kleene. General recursive functions of natural numbers. *Mathematische Annalen*, 112:727–742, 1936.

[47] A. N. Kolmogorov. On the representation of continous functions of several variables by superposition of continous functions of one variable and addition. *Doklady Akademiia Nauk SSSR*, 114(5):953–956, 1957.

[48] V. Kurkova. Kolmogorov's theorem is relevant. *Neural Computation*, 3(4):617–622, 1991.

[49] V. Kurkova. Kolmogorov's theorem and multilayer neural networks. *Neural Networks*, 5(3):501–506, 1992.

[50] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient BackProp. In G. B. Orr and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*. Springer–Verlag, 1998.

[51] Y. LeCun, I. Kanter, and S. A. Solla. Eigenvalues of covariance matrices: Application to neural–network learning. *Physical Review Letters*, 66(18):2396–2399, 1991.

[52] L. R. Leerink, C. L. Giles, B. G. Horne, and M. A. Jabri. Learning with product units. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 537–544. The MIT Press, 1995.

[53] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multliayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6:861–867, 1993.

[54] H. Leung and S. Haykin. The complex backpropagation algorithm. *IEEE Transactions on Signal Processing*, 3(9):2101–2104, 1991.

[55] E. N. Lorenz. Deterministic nonperiodic flow. *Journal Athmospheric Science*, 20:130–134, 1963.

[56] P. Lounesto. *Clifford Algebras and Spinors*. Cambridge University Press, 1997.

[57] J. L. McClelland, D. E. Rumelhart, and the PDP research group, editors. *Parallel Distributed Processing*, volume 2: Psychological and Biological Models. MIT Press, Cambridge MA, 1986.

[58] W. S. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[59] M. L. Minsky and S. A. Papert. *Perceptrons*. MIT Press, 1969.

[60] T. Needham. *Visual Complex Analysis*. Clarendon Press, Oxford, 1997.

[61] T. Nitta. An extension of the back–propagation algorithm to complex numbers. *Neural Networks*, 10(8):1391–1415, 1997.

[62] J. K. Pearson. *Clifford Networks*. PhD thesis, University of Kent, 1994.

[63] A. Pellionisz and R. Llinás. Tensor network theory of the metaorganization of functional geometries in the central nervous system. *Neuroscience*, 16(2):245–273, 1985.

[64] C. Perwass, V. Banarer, and G. Sommer. Spherical decision surfaces using conformal modelling. In B. Michaelis and G. Krell, editors, *25. Symposium für Mustererkennung, DAGM 2003, Magdeburg*, volume 2781 of *LNCS*, pages 9–16. Springer-Verlag, Berlin, 2003.

[65] A. Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999.

[66] P. Plücker. On a new geometry of space. *Phil. Trans. R. Soc. London*, 155, 1865.

[67] I. R. Porteous. *Clifford Algebras and the Classical Groups*. Cambridge University Press, Cambridge, 1995.

[68] H. Pottmann and J. Wallner. *Computational Line Geometry*. Springer–Verlag, 2001.

[69] H. Ritter, T. Martinetz, and K. Schulten. *Neuronale Netze*. Addison–Wesley, Bonn, 1990.

[70] R. Rojas. *Neural Networks*. Springer–Verlag, 1996.

[71] J. Rooney. A Comparison of Representations of General Screw Displacements. *Environment and Planning B*, 5:45–88, 1978.

[72] F. Rosenblatt. The perceptron: A probalistic model for information storage and organization in the brain. *Psycol. Rev.*, 65:386–408, 1958.

[73] F. Rosenblatt. *Principles of Neurodynamics*. Spartan, New York, 1962.

[74] B. Rosenhahn. Pose estimation revisited. Technical Report Number 0308, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, September 2003.

[75] W. Rudin. *Real and Complex Analysis*. McGraw–Hill, New York, 1966.

[76] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In Rumelhart et al. [77], pages 318–362.

[77] D. E. Rumelhart, J. L. McClelland, and the PDP research group, editors. *Parallel Distributed Processing*, volume 1: Foundations. MIT Press, Cambridge MA, 1986.

[78] A. E. Samuel, P. R. McAree, and K. H. Hunt. Unifying screw geometry and matrix transformation. *International Journal of Robotics Research*, 10(5):454–472, 1991.

[79] F. Scarselli and A. C. Tsoi. Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural Networks*, 11(1):15–37, 1998.

[80] V. G. Shervatov. *Hyperbolic Functions*. D.C. Heath and Company, Boston, 1963.

[81] B. F. Skinner. *The Behavior of Organisms: An Experimental Analysis*. Appleton–Century–Crofts, New York, 1938.

[82] G. Sommer. Algbraic aspects of designing behavior based systems. In G. Sommer and J. J. Koenderink, editors, *Proc. Int. Workshop AFPAC'97, vol. 1315 of LNCS*, pages 1–28. Springer–Verlag, 1997.

[83] G. Sommer, editor. *Geometric Computing with Clifford Algebras*. Springer–Verlag, 2001.

[84] E. Study. *Geometrie der Dynamen*. Leipzig, 1903.

[85] E. L. Thorndike. *The Fundamentals of Learning*. Teachers College Press, New York, 1933.

[86] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936.

[87] K. Th. Vahlen. Ueber Bewegungen und complexe Zahlen. *Math. Ann.*, 55:585–593, 1902.

[88] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer–Verlag, New York, 1995.

[89] M. Vidyasagar. *A Theory of Learning and Generalization*. Springer–Verlag, 1997.

[90] K. Voss and H. Süsse. *Adaptive Modelle und Invarianten für zweidimensionale Bilder*. Verlag Shaker, Aachen, 1995.

[91] H. White. Connectionist nonparametric regression: multilayer feedforward networks can learn arbitrary mappings. *Neural Networks*, 3(5):543–549, 1990.

[92] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Academic Press, 2000.

[93] I. M. Yaglom. *Complex Numbers in Geometry*. Academic Press, New York, 1968.