12. Introduction to Neural Computation in Clifford Algebra*

Sven Buchholz and Gerald Sommer

Institute of Computer Science and Applied Mathematics,
Christian-Albrechts-University of Kiel

12.1 Introduction

This is the first of two chapters on neural computation in Clifford algebra. The name Clifford algebra refers to its inventor William K. Clifford (1845-1879). We will restrict ourselves on Clifford algebras generated by non–degenerate quadratic forms. Thus, Clifford algebras are non–degenerated geometric algebras hereafter.

Any degenerate geometric algebra can be embedded in a larger dimensional non-degenerate geometric (Clifford) algebra as stated in the first chapter of this book. In that sense, our restriction will not be significant. However, it is necessary because it is not possible to derive learning algorithms for neural networks in degenerate algebras. We will explain this in full detail in the second chapter on Clifford multilayer perceptrons (Clifford MLPs).

The idea of developing neural networks in other than the real domain is not new. Complex valued networks were already introduced at the beginning of our decade, see e.g. [90]. Recently, [7] proposed a quaternionic valued neural network. A first attempt of developing neural networks in Clifford algebras was made by [187]. Unfortunately, the approach proposed there had many drawbacks. We showed this in [33], [15], where we also sketched an alternative that leads to correct learning algorithms for Clifford–valued neural networks.

* This work has been supported by DFG Grants So-320-2-1 and So-320-2-2.

In this chapter we will not speak of Clifford neural networks yet. Instead, the center of our studies will be the Clifford neuron itself. Thus, we will develop Clifford neural networks directly from their building blocks. At the level of neurons it is much easier to understand the principles of neural computation in Clifford algebras. The main principle is, that neural computation in Clifford algebras can be seen as model-based in comparison to that in the real case. To show this theoretically and experimentally is the main goal of this chapter.

The model–based approach will allow us to interpret the non–commutativity of Clifford algebras in general as a feature of Clifford neural computation. That aspect is not mentioned in the previous work of [7] and [187]. This will lead us to a special type of Clifford neuron called spinor neuron.

The split–up of the discussion of Clifford neural computation in one chapter on Clifford neurons and one other on Clifford MLPs follows also a classical road. That is the design from linear units to non–linear networks. The last section of this chapter is therefore dedicated to linearization with Clifford neurons. We will work out this completely on the example of Möbius transformations.

We start with an outline of Clifford algebra now.

12.2 An Outline of Clifford Algebra

A compact and comprehensive introduction on geometric algebras (and therefore also on Clifford algebras) has already been given by the first chapter of this book. Here, we will only review those facts needed in the following.

In addition, we will put more emphasis to the direct generation of algebras by non-degenerate quadratic forms. This gives the signature of a vector space in a very natural way.

So let be Q a non–degenerate quadratic form on $\mathbb{R}^n$. For shortness, we will call $(\mathbb{R}^n, Q)$ a quadratic space.

By a theorem of linear algebra there exists a basis of $\mathbb{R}^n$ such that

$$Q(v) = -v_1^2 - v_2^2 - \ldots - v_p^2 + v_{p+1}^2 \ldots + v_{p+q}^2 \tag{12.1}$$

for all $v = (v_1, \ldots, v_n) \in \mathbb{R}^n$, $p, q \in \mathbb{N}$ and $p + q = n$. This allows us to work with quadratic forms in an abstract fashion. In that sense, a quadratic form is already determined by (p, q). Then, we will denote a quadratic space by $\mathbb{R}^{p,q}$ hereafter. For the vectors of an orthonormal basis $\{e_1, \ldots, e_n\}$ of $\mathbb{R}^n$ we get from (12.1)

$$-Q(e_i) = +1 \quad \text{if} \quad i \leq p \tag{12.2}$$

$$-Q(e_i) = -1 \quad \text{if} \quad i > p\,. \tag{12.3}$$

With the corresponding scalar product to Q in mind we can also speak of $\mathbb{R}^{0,q}$ as an Euclidean space, $\mathbb{R}^{p,0}$ as an anti–Euclidean space, and $\mathbb{R}^{p,q}$ ($p \neq 0 \wedge q \neq 0$) as an indefinite space, respectively.

Equations (12.1)-(12.3) together now allow the following definition of a Clifford algebra [159].

Definition 12.2.1. *An associative algebra with unity 1 over* $\mathbb{R}^{p,q}$ *containing* $\mathbb{R}^{p,q}$ *and* $\mathbb{R}$ *as distinct subspaces is called the Clifford algebra* $\mathcal{C}_{p,q}$ *of* $\mathbb{R}^{p,q}$*, iff*

(a) $v \otimes_{p,q} v = -Q(v)\,, \quad v \in \mathbb{R}^{p,q}$

(b) $\mathcal{C}_{p,q}$ *is generated as an algebra by* $\mathbb{R}^{p,q}$

(c) $\mathcal{C}_{p,q}$ *is not generated by any proper subspace of* $\mathbb{R}^{p,q}$ *.*

Examples of Clifford algebras are the real numbers $\mathbb{R}$ corresponding to $\mathcal{C}_{0,0}$, the complex numbers $\mathbb{C}$ corresponding to $\mathcal{C}_{0,1}$, and the quaternions $\mathbb{H}$ corresponding to $\mathcal{C}_{0,2}$, respectively.

An element of a Clifford algebra is called a *multivector*, due to the fact that it consists of objects of different types by definition. The algebra multiplication $\otimes_{p,q}$ of a Clifford algebra $\mathcal{C}_{p,q}$ is called the *geometric product*.

Condition (a) of the above definition implies equations (12.2),(12.3) and further for all $i, j \in \{1, \ldots, p+q\}$

$$e_i \otimes_{p,q} e_j = -e_j \otimes_{p,q} e_i\,. \tag{12.4}$$

Clearly, 2^{p+q} is then an upper bound for the dimension of a Clifford algebra. Condition (c) guarantees that no lower dimensional algebras are generated. For a complete proof see e.g. [194]. Hence, the dimension of a Clifford algebra is 2^{p+q}.

A further very important consequence of equation (12.4) is, that only Clifford algebras up to dimension 2 are commutative ones.

We will now give explicitly a basis of a Clifford algebra in terms of the basis vectors $\{e_1, \ldots, e_n\}$ of the underlying quadratic space. Using the canonical order of the power set $\mathcal{P}(\{1, \ldots, n\})$ to derive the index set

$$\mathcal{A} := \{\{a_1, \ldots, a_r\} \in \mathcal{P}(\{1, \ldots, n\}) \mid 1 \leq a_1 \leq \ldots \leq a_r \leq n\} \tag{12.5}$$

and then defining for all $A \in \mathcal{A}$

$$e_A := e_{a_1} \ldots e_{a_r}\,, \tag{12.6}$$

we achieve a basis $\{e_A \mid A \in \mathcal{A}\}$ of the Clifford algebra $\mathcal{C}_{p,q}$. Every $x \in \mathcal{C}_{p,q}$ can then be written as

$$x = \sum_{A \in \mathcal{A}} x_A e_A\,. \tag{12.7}$$

For every $r \in \{0, \ldots, 2^{p+q}-1\}$ the set $\{e_A \mid A \in \mathcal{A},\ |A| = r\}$ is spanning a linear subspace of $\mathcal{C}_{p,q}$. This linear subspace is called the *r–vector part* of the Clifford algebra $\mathcal{C}_{p,q}$. An element of such a subspace is then called an r–vector. For r running from 0 to 3 an r–vector is also often called a scalar, vector, bivector, or trivector, respectively.

The vector part of a Clifford algebra $\mathcal{C}_{p,q}$ should get its own notation $\mathbb{R}_{p,q}$ to be distinguished from the quadratic space $\mathbb{R}^{p,q}$ itself.

All even r-vectors form the even part $\mathcal{C}^+_{p,q}$ of the Clifford algebra $\mathcal{C}_{p,q}$. $\mathcal{C}^+_{p,q}$ is a subalgebra of $\mathcal{C}_{p,q}$ isomorphic to $\mathcal{C}_{p,q-1}$. Whereas the odd part $\mathcal{C}^-_{p,q}$ formed by all odd r-vectors is not a subalgebra.

Clifford algebras are $\mathbb{R}$–linear

$$\forall\lambda \in \mathbb{R}\, \forall x, y \in \mathcal{C}_{p,q} :\ (\lambda\, x)y = x(\lambda\, y) = \lambda\, (xy)\,, \tag{12.8}$$

so every Clifford algebra is isomorphic to some matrix algebra. The matrix representations of Clifford algebras $\mathcal{C}_{p,q}$ up to dimension 16 are given in Table 12.1. As we can see, there are many isomorphic Clifford algebras.

Table 12.1. Matrix representations of Clifford algebras up to dimension 16

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

Next, we will deal with involutions of Clifford algebras. An involution is an algebra mapping of order 2. Thus the set of all involutions of a Clifford algebra is given by

$$In(\mathcal{C}_{p,q}) := \{f : \mathcal{C}_{p,q} \to \mathcal{C}_{p,q} \mid f^2 = id\}\,. \tag{12.9}$$

The most important involutions of a Clifford algebra are the following ones. The first, called *inversion*

$$\hat{x} = \sum_{A\in\mathcal{A}} (-1)^{|A|}\, x_A e_A \tag{12.10}$$

is an automorphism ($\hat{x}\hat{y} = \widehat{xy}$), whereas *reversion*

$$\tilde{x} = \sum_{A\in\mathcal{A}} (-1)^{\frac{|A|\,(|A|-1)}{2}} x_A e_A\,, \tag{12.11}$$

and *conjugation*

$$\bar{x} = \sum_{A\in\mathcal{A}} (-1)^{\frac{|A|\,(|A|+1)}{2}} x_A e_A \tag{12.12}$$

are anti–automorphisms ($\tilde{x}\tilde{y} = \widetilde{yx}$, $\bar{x}\bar{y} = \overline{yx}$). Conjugation is obviously a composition of inversion and reversion. The conjugation of complex numbers results a special case of (12.12).

Finally, we want to analyse which Clifford algebras are division algebras. The answer is given by the famous FROBENIUS theorem. That theorem states, that there are no other real division algebras despite of $\mathbb{R}$, $\mathbb{C}$, and $\mathbb{H}$. A

finite-dimensional associative algebra $\mathcal{A}$ is a division algebra, iff it contains no divisors of zero. Therefore, any other Clifford algebras except the ones mentioned above will contain divisors of zero. Thereby, an element $a \in \mathcal{A}$ is a divisor of zero, iff there exists an element $b \in \mathcal{A}\backslash\{0\}$ with $ab = 0$ or $ba = 0$.

The existence of divisors of zero can cause many problems in the design of neural algorithms in the frame of Clifford algebras. We will see this already in outlines in the next section.

12.3 The Clifford Neuron

In this section we will start with a generic neuron as computational unit. From this, a standard real valued neuron is then derived. Finally, we will introduce the Clifford neuron based on the geometric product. Through this way, we will also introduce some basics of neural computation in general very briefly. To characterize the computation with Clifford neurons as model–based in relation to that with real neurons is the main goal of this section.

A generic neuron is a computational unit of the form shown in Figure 12.1. The computation within such a neuron is performed in two steps. Firstly, a propagation function f associates the input vector x with the parameters of the neuron comprised in the weight vector w. Then, the application of a activation function g follows. Thus, the output of a generic neuron is given by

$$y = g(f(x; w)) \,. \tag{12.13}$$

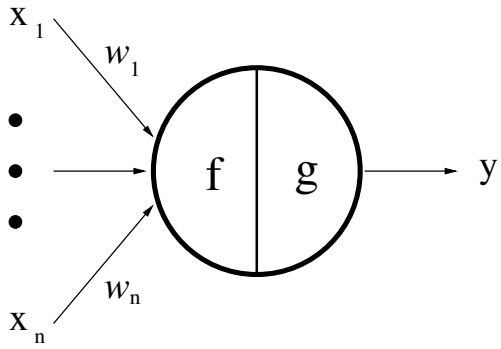

Fig. 12.1. Generic neuron

In general, the propagation function f is a mapping

$$f : D^n \to D \tag{12.14}$$

for a domain D. The activation function g is a mapping

$$g : D \to D' \tag{12.15}$$

to a domain D'. Mostly, D is a continuous domain. In this case, we have usually $D' = D$ for function approximation. On the other hand, the neuron computes a classification if D' is discrete.

From now on, we will assume if no other statement is made, that g is set to be the identity. We will also speak of a neuron with this in mind.

12.3.1 The Real Neuron

For a real neuron we have with our previous notation $D = \mathbb{R}$ and $w, x \in \mathbb{R}^n$. The most common propagation function for such a neuron simply computes a weighted sum of the inputs of a real neuron

$$f(x) = \sum_{i=1}^{n} w_i x_i + \theta \,, \tag{12.16}$$

with an additional parameter $\theta \in \mathbb{R}$, that works as a bias. By extending the domain by one dimension and then using an extended input vector $x^+ := (x, 1)$ and an extended weight vector $w^+ := (w, \theta)$ we can rewrite (12.16) in the form

$$f(x^+) = \sum_{i=1}^{n+1} w_i^+ x_i^+ \,. \tag{12.17}$$

A real neuron with the above propagation function is therefore a linear associator. Non-linearity of the neuron could be achieved by applying a non-linear activation function g.

As a linear associator we can use the real neuron for linear regression. This (neural computation) is done by formulating linear regression as a learning problem.

So let us consider a training set $T := \{(x^1, t^1), \ldots, (x^m, t^m)\}$ consisting of input–output pairs (x^i, t^i) with $x^i \in \mathbb{R}^n$, $t^i \in \mathbb{R}$. The aim of learning is to find a weight vector $w = (w_1, \ldots, w_n)$ that minimizes the sum-of-squared error (SSE)

$$E = \frac{1}{2} \sum_{i=1}^{m} (t^i - \sum_{j=1}^{n} w_j x_j^i)^2 \tag{12.18}$$

iteratively. A well known method to do so is using gradient descent. Then, at each step the following correction of the weights

$$\Delta w_j = -\frac{\partial E}{\partial w_j} \,. \tag{12.19}$$

has to be made. In terms of neural networks this is called *back–propagation*, due to the fact that the error is propagated back from the output.

Since the error function (12.18) is convex, back–propagation will always find the global minimum.

Provided with the above basic knowledge about generic and real neurons we are now able to study Clifford neurons in detail.

12.3.2 The Clifford Neuron

An abstract Clifford neuron is easily derived as a special case of a generic neuron by taking in (12.13) a Clifford algebra as domain. However, some care has to be taken already. The propagation function of a generic Clifford neuron should obviously be a mapping of the form

$$f : \mathcal{C}_{p,q} \to \mathcal{C}_{p,q}. \tag{12.20}$$

The above function is then just a special case of (12.14) with $D = \mathcal{C}_{p,q}$ and $n = 1$. In that case the illustration of a generic neuron in Figure 12.1 has no great strength anymore, because we have just one input and one weight. But through that, we can also see immediately that f has lost its independent function. More precisely, it is fully determined by the way the association of the one input with the one weight is done. Clearly, there is only one intented way of association — the geometric product.

The propagation function f of a Clifford neuron is given either by

$$f(x) = w \otimes_{p,q} x + \theta \tag{12.21}$$

or by

$$f(x) = x \otimes_{p,q} w + \theta\,. \tag{12.22}$$

All the entities are now multivectors, i.e. $x, w, \theta \in \mathcal{C}_{p,q}$.

Of course, we have to distinguish left–sided and right–sided weight multiplication in the general case of a non-commutative Clifford algebra.

Formally, we have just replaced the scalar product by the geometric product. As in the real case, we can interpret the parameter θ as a bias. However, now an extension of the form (12.17) is possible to treat θ as a normal weight.

The input-weight association of a Clifford neuron should now be made concretely. For the sake of simplicity let us choose the complex numbers $\mathcal{C}_{0,1}$ as an example. A complex neuron computes just a complex multiplication, say $xw = y$. Further, let be $x = x_1 + x_2\,\mathrm{i}$ and $y = y_1 + y_2\,\mathrm{i}$.

Now assume we want to compute a complex multiplication with real neurons. Clearly, this requires 2 real input and 2 real output neurons. We are looking then for a weight matrix $W \in \mathbb{R}(2)$ that fulfills $(x_1, x_2)\,W = (y_1, y_2)$. This is achieved by setting $w_{11} = w_{22}$ and $w_{12} = -w_{21}$, which just results in

the well-known matrix representation of complex numbers. Figure 12.2 gives an illustration of the situation.

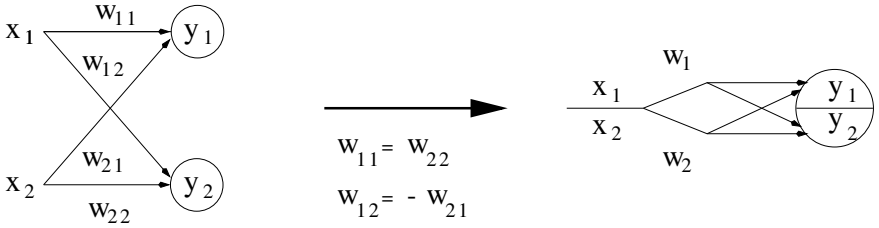

Fig. 12.2. Computation of a complex neuron (right) and simulation with real neurons (left)

Thus, complex multiplication is just a certain linear transformation, namely a dilatation-rotation, which easily follows from the polar form of complex numbers. This means in terms of neural computation, a complex neuron can be seen as model-based. Instead of an unspecified linear function (real neurons) we use a dilatation-rotation (complex neuron). If this model is applicable to given data, we would only need half of the parameters (see again Figure 12.2) for computation. Furthermore, the real neurons have to imitate the model "by finding the given weight constraints" with independent weights. This approach should then also be less efficient with respect to time complexity or less accurate.

To be able to verify this experimentally, we now need a correct learning algorithm for a complex neuron. Yet, we will give here the rule for updating the weight in the general case of an arbitrary Clifford neuron. So let be $T := \{(x^1, t^1), \ldots, (x^m, t^m)\}$ the training set consisting of input–output pairs (x^i, t^i) with $x^i, t^i \in \mathcal{C}_{p,q}$. The SSE defined analogously to (12.18) is then minimized by applying the correction step

$$\Delta w = \bar{x^i} \otimes_{p,q} (t^i - w \otimes_{p,q} x^i) \,. \tag{12.23}$$

for left–sided weight multiplication and

$$\Delta w = (t^i - x^i \otimes_{p,q} w) \otimes_{p,q} \bar{x^i} \,. \tag{12.24}$$

for right–sided weight multiplication, respectively. Here, the function $\bar{}$ stands for that univocally determined involution yielding

$$x \otimes_{p,q} \overline{y} = \sum_i x_i y_i \,. \tag{12.25}$$

Using this function avoids the appearance of divisors of zeros during back–propagation. This is necessary, otherwise learning could stop for an non–zero

error. The proof of correctness of the algorithm will be postponed to the next chapter.

Now, we can perform our first intented experiment.

Experiment 1 (Complex multiplication).
The task for a complex neuron and for real neurons as in Figure 12.2 was simply to learn the complex multiplication with $2 - 4\,\mathrm{i}$. As training set $T = \{(-0.3, 0), (-0.5, -0.3), (-0.6, 0)\}$ was used. After 116 epochs (which means after applying the training patterns 116 times) the SSE of the complex neuron where dropped under 0.000001. The learned weight of the complex neuron was $w = 2.0000 - 4.0000\,\mathrm{i}$. In contrast, the SSE of the real neurons dropped under 0.000001 after 246 steps but the weight matrix was

$$W = \begin{pmatrix} 1.99741 & -3.99738 \\ 4.00252 & 1.99700 \end{pmatrix}.$$

Thus, our very plain considerations are right. Simulation of a model seems worse than using a model directly.

In the case of complex numbers we have identified the input–weight association by the geometric (complex) product completely and characterized it as model-based. The generalization of this is quiet easy. Due to the $\mathbb{R}$-linearity of Clifford algebras (12.8), any geometric product can be expressed as a special matrix multiplication. This means that the computation of an arbitrary single Clifford neuron can also be performed by the corresponding number of real neurons. However, this point of view on the neuron level is too artificial. In practice we have to deal with *real* data of *any* dimension.

We have introduced Clifford algebras as the algebras of quadratic spaces in section 2. Therefore, a natural computation of a Clifford neuron should process (real) data of the underlying quadratic space. In fact, the complex multiplication of a Clifford neuron should also be seen in this way. As we know already, a complex neuron computes a dilatation–rotation. More precisely, it computes a transformation of vectors of $\mathbb{R}^2$ in such a manner. As real vector spaces of the same dimension $\mathbb{R}^2$ and $\mathbb{C}$ are isomorphic. In that sense a complex neuron processes also indeed points of $\mathbb{R}^2$. However, complex numbers are no vectors.

This interpretation problem will be easily resolved in the next section. That section will be fully dedicated to the processing of data drawn from quadratic spaces with Clifford neurons in a formally consistent manner. By doing so we will also get a better understanding of the model–based nature of Clifford neurons.

12.4 Clifford Neurons as Linear Operators

Following the ideas developed at the end of the previous section, we are now interested how a linear transformation of the form

$$f : \mathbb{R}_{p,q} \to \mathbb{R}_{p,q} \tag{12.26}$$

can be computed with Clifford neurons. To be able to do so, we need a theoretical method to describe such transformation in Clifford algebras.

Fortunately, any multivector that has a multiplicative inverse defines such a transformation already. Thus, the mathematical object we have to look at is the group formed by these multivectors. This group is called the Clifford group.

Applying a group to the elements of a set is generally formalized in the following way.

Definition 12.4.1. *Let G be a group and M be a non–empty set. The map*

$$\star : G \times M \to M; \quad (a, x) \mapsto a \star x \tag{12.27}$$

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

For example, the general linear group $GL(n, \mathbb{R})$ of $\mathbb{R}^n$ operates on (column) vectors by matrix multiplication

$$\cdot : GL(n, \mathbb{R}) \times \mathbb{R}^n \to \mathbb{R}^n; \quad (A, x) \mapsto Ax\,. \tag{12.28}$$

The Clifford case is more complicated than that. It will be studied in detail in the next subsection. The results of this study will then be transposed to the level of Clifford neurons and will be verified there experimentally.

12.4.1 The Clifford Group

Let us start directly with the definition of the Clifford group.

Definition 12.4.2. *The Clifford group $\Gamma_{\mathrm{p,q}}$ of a Clifford algebra $\mathcal{C}_{p,q}$ is defined as*

$$\Gamma_{\mathrm{p,q}} := \{s \in \mathcal{C}_{p,q} \mid \forall x \in \mathbb{R}_{p,q} :\ sx\hat{s}^{-1} \in \mathbb{R}_{p,q}\}\,. \tag{12.29}$$

From that definition we get immediately

$$\Gamma_{\mathrm{p,q}} \times \mathbb{R}_{p,q} \to \mathbb{R}_{p,q}; \quad (s, x) \mapsto sx\hat{s}^{-1} \tag{12.30}$$

as the operation of the Clifford group $\Gamma_{\mathrm{p,q}}$ on $\mathbb{R}_{p,q}$. Thus, the operation of $\Gamma_{\mathrm{p,q}}$ is not one single primitive operation, as it was the case in the example of $GL(n, \mathbb{R})$ (12.28). Another important difference to that case is, that the elements of the group are of the same type as the elements of the set on which the group is operating. Actually, this is one of the great advantages of Clifford algebra. We shall call an element of $\Gamma_{\mathrm{p,q}}$ a linear operator to distinguish it from an ordinary multivector. It is indeed a linear operator since the Clifford group $\Gamma_{\mathrm{p,q}}$ consists of linear transformations of $\mathbb{R}_{p,q}$ by definition (12.26).

Hence, $\Gamma_{\mathrm{p,q}}$ is isomorphic to a general linear group or one of its subgroups. The relation of $\Gamma_{\mathrm{p,q}}$ to those classical groups can be concluded from the map

$$\psi_s : \mathbb{R}^{p,q} \to \mathbb{R}^{p,q};\ x \mapsto sx\hat{s}^{-1}\,. \tag{12.31}$$

For all $x \in \mathbb{R}_{p,q},\ s \in \Gamma_{\mathrm{p,q}}$ we have

$$Q(\psi_s(x)) = (\widehat{sx\hat{s}^{-1}})sx\hat{s}^{-1} = \hat{s}\hat{x}s^{-1}sx\hat{s}^{-1} = \hat{x}x = Q(x)\,, \tag{12.32}$$

so ψ_s is an orthogonal map. In fact, it is easy to see that ψ_s is even an orthogonal automorphism of $\mathbb{R}_{p,q}$. Thereby, we have proofed the following theorem in principle.

Theorem 12.4.1. *The map $\Psi_s : \Gamma_{\mathrm{p,q}} \to O(p,q);\ s \mapsto \psi_s$ is a group epimorphism.*

Indeed, $\Gamma_{\mathrm{p,q}}$ is a multiple cover of the orthogonal group $O(p,q)$ since the kernel of Ψ_s is $\mathbb{R}\setminus\{0\}$.

Altogether, we know now that the Clifford group $\Gamma_{\mathrm{p,q}}$ is an orthogonal transformation group. However, it is still unnecessarily large. Therefore, we first reduce $\Gamma_{\mathrm{p,q}}$ to a two-fold cover of $O(p,q)$ by defining the so–called Pin group

$$\mathrm{Pin(p,q)} := \{s \in \Gamma_{\mathrm{p,q}} \mid s\tilde{s} = \pm 1\}\,. \tag{12.33}$$

The even elements of $\mathrm{Pin(p,q)}$ form the spin group

$$\mathrm{Spin(p,q)} := \mathrm{Pin(p,q)} \cap \mathcal{C}\ell^{+}_{p,q} \tag{12.34}$$

which is a double cover of the special orthogonal group $SO(p,q)$. Finally, those elements of $\mathrm{Spin(p,q)}$ with Clifford norm equal 1 form a further subgroup

$$\mathrm{Spin_{+}(p,q)} := \{s \in \mathrm{Spin(p,q)} \mid s\tilde{s} = 1\} \tag{12.35}$$

that covers $SO_{+}(p,q)$ twice. Thereby, $SO_{+}(p,q)$ is the connected component of the identity of $O(p,q)$.

As usual, we write $\mathrm{Pin(p)}$ for $\mathrm{Pin(p,q)}$ and so on. We shall remark here, that $\mathrm{Spin(p,q)} \simeq \mathrm{Spin(q,p)}$ and $\mathrm{Spin(p)} = \mathrm{Spin_{+}(p)}$. Both follows easily from the properties of the orthogonal groups together with $\mathcal{C}\ell^{+}_{p,q} \simeq \mathcal{C}\ell^{+}_{q,p}$.

For the spin group $\mathrm{Spin(p,q)}$ there exists another way besides the standard one (12.30) of operating as a dilatation–rotation operator. This way will allow the reduction of the dimension of the Clifford algebra in use. Also it will resolve the interpretation problem regarding complex multiplication noticed earlier in section 3.

$\mathrm{Spin(p,q)}$ consists by definition only of even elements. Remembering $\mathcal{C}\ell^{+}_{p,q} \simeq \mathcal{C}\ell_{p,q-1}$, we can interpret a spinor also as an element of $\mathcal{C}\ell_{p,q-1}$.

Let us denote by $\lambda\mathbb{R}_{p,q-1}$ both the scalar and vector part of $\mathcal{C}_{p,q-1}$. This space is called the space of paravectors. Then the operation of $\mathrm{Spin(p,q)}$ on $\mathbb{R}_{p,q-1}$ is the same as on $\mathbb{R}_{p,q}$ [194]. More precisely, for every $s \in \mathrm{Spin(p,q)}$ the map

$$\phi_s : \lambda\mathbb{R}_{p,q-1} \to \lambda\mathbb{R}_{p,q-1};\ x \mapsto sx\hat{s}^{-1} \tag{12.36}$$

is a dilatation–rotation of $\mathbb{R}_{p,q}$. If the underlained Clifford algebra in (12.36) is commutative in addition we have

$$\phi_s(x) = sx\hat{s}^{-1} = xs\hat{s}^{-1} = xs' \quad (s' := s\hat{s}^{-1} \in \mathrm{Spin(p,q)})\,. \tag{12.37}$$

In the special case of complex numbers the above relations together with $\mathcal{C}_{0,1} = \lambda\mathbb{R}_{0,1}$ implies that any complex multiplication is indeed a dilatation–rotation.

All the obtained results will be transposed to the computational level of Clifford neurons now.

12.4.2 Spinor Neurons

In the previous section we have studied the group of linear transformations $\Gamma_{\mathrm{p,q}}$ of a Clifford algebra. Actually, we have found out that $\Gamma_{\mathrm{p,q}}$ consists of orthogonal transformations only. The operation of $\Gamma_{\mathrm{p,q}}$ can be simulated by concatenation of a left–sided and a right-sided (or vice versa) Clifford neuron. This architecture is shown in Figure 12.3.

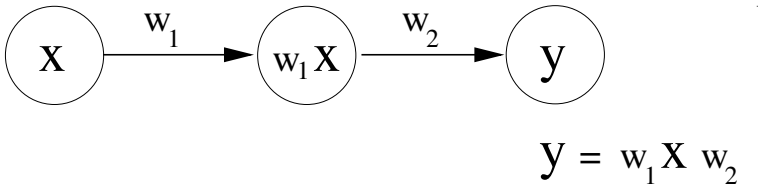

Fig. 12.3. Simulation of the operation of $\Gamma_{\mathrm{p,q}}$ with Clifford neurons

Every orthogonal transformation is computable by this architecture. This is just done by using the vector part of the input and output neuron to process the data. However, there might exist other suitable ways of representing the data. In general there are $\binom{n}{k}^2$ possibilities of input-output representations of k–dimensional data in n dimensions. Therefore, we will only study the case of plane transformations in Clifford algebras of dimension 4. In that case there are 36 possibilities of data representation.

Table 12.2. Used codes for 2 dimensional data in 4 dimension

	Representation
1	0xx0
2	0x0x
3	00xx
4	xx00
5	x0x0
6	x00x

Using the notations of Table 12.2 the number 11 then denotes input representation 1 and output representation 1 and thus input–output representation $0xx0 - 0xx0$. This is the representation corresponding directly to the definition of $\Gamma_{\mathrm{p,q}}$. The results for the computation of 2-D Euclidean transformations are listed in Table 12.3.

Table 12.3. Suitable data representations for SO(2) and O(2) computation

Algebra	Weight multiplication	Data representation
$\mathcal{C}_{0,2}$	left-right, right-left	all
	left	11,22,33,44,55,66,25,52
	right	11,22,33,44,55,66,16,61,34,43
$\mathcal{C}_{1,1}$	left, left-right, right-left	22,55,25,52
	right	55,22
$\mathcal{C}_{2,0}$	left, left-right, right-left	11,66,16,61
	right	11,66

As we can see, there is no difference between the computation of SO(2) and O(2). Remarkable, all representations with two weights in $\mathcal{C}_{0,2}$ are suitable. Due to the existence of complex number representations we get also representations that work with only one weight. In the case of an anti–Euclidean transformation we have to distinguish SO(1,1) and O(1,1). The suitable data representations can be found in Table 12.4 and Table 12.5, respectively.

Before starting to discuss the above listed results we should re-think the situation in general. All the reported results were obtained by applying data of a transformation of one of the mentioned types. So we actually just checked which representation will not work. Having in mind that all the transformations could also be computed by real neurons as in Fig. 12.2, we should extend our point of view again.A main idea of this introductory chapter is to develop interpretations of the computation of Clifford neurons. This should always be done by characterizing Clifford neurons as model–based ones as in section 3.

Table 12.4. Suitable data representations for SO(1,1) computation

Algebra	Weight multiplication	Data representation
$\mathcal{C}_{0,2}$	left, right, left-right, right-left	none
$\mathcal{C}_{1,1}$	left-right, right-left	44,64,14,34,46,66,16,36, 41,61,11,31,53,63,13,33
	left	44,66,16,61,11,33
	right	44,34,66,11,43,33
$\mathcal{C}_{2,0}$	left-right, right-left	44,54,24,34,45,55,25,35, 42,52,22,32,43,53,23,33
	left	44,55,25,52,22,33
	right	44,34,55,22,43,33

Table 12.5. Suitable data representations for O(1,1) computation

Algebra	Weight multiplication	Data representation
$\mathcal{C}_{0,2}$	left, right, left-right, right-left	none
$\mathcal{C}_{1,1}$	left-right, right-left	44,64,14,34,46,66,16,36 41,61,11,31,53,63,13,33
	left	34,43
	right	16,61
$\mathcal{C}_{2,0}$	left-right, right-left	44,54,24,34,45,55,25,35, 42,52,22,32,43,53,23,33
	left	34,43
	right	25,52

This step has to be made for the computation of orthogonal transformations with Clifford neurons now. That is, we have to determine the conditions so that the computation of Clifford neurons as in Figure 12.3 can be forced to be an orthogonal computation. In that case we would apply this model independent of the processed data. To be able to do so, we have to constrain the weights of the left–sided and right–sided Clifford neurons in Figure 12.3 together. This should result in one neuron with one weight that is multiplied from the left and from the right. But this will be not possible for an orthogonal transformation in general. However, it is possible for the operation of a spinor. The corresponding neuron is then named a *spinor neuron*. Computation with such neurons is always model–based. In the case of a 2–dimensional Clifford algebra this is also valid for any orthogonal transformation due to the commutativity of the algebra. However, this could require a special data representation as shown in Tables 12.3-12.5. So we use the notion of a spinor neuron in that sense that the operation of the neuron as a linear operator is performed by one weight. After we have reflected that spinor neurons are model–based, we will now perform simulations to compare them with real neurons.

12.4.3 Simulations with Spinor Neurons

With the following two experiments we want to test the strength of the model of single spinor neurons in comparison with multiple real neurons, especially in the presence of noise. We will just speak of real neurons, since the number of real input and output neurons to compute a linear transformation of $\mathbb{R}^n$ is always n. See again Figure 12.2.

Experiment 2 (Euclidean 2D similarity transformation).
The transformation that should be learned was a composition of a Euclidean 2D rotation about -55°, a translation of [+1, -0.8], and a scaling of factor 1.5. The training and test data is shown in Figure 12.4.

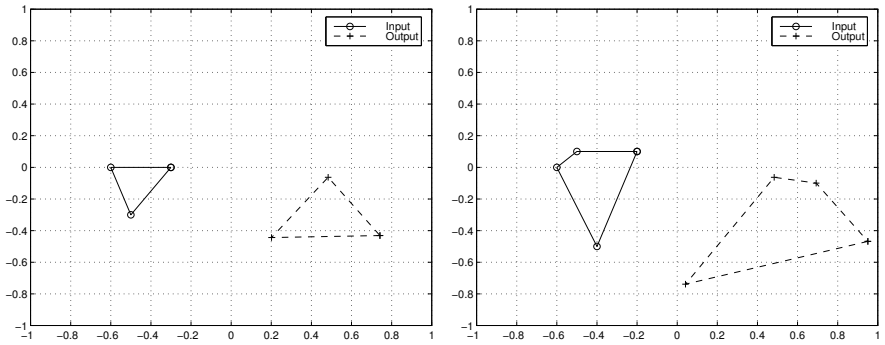

Fig. 12.4. Training data (left) and test data (right)

The experiment was performed using real neurons, a complex neuron, and a spinor neuron in $\mathcal{C}_{0,2}$. The convergence of the training is reported in Figure 12.5.

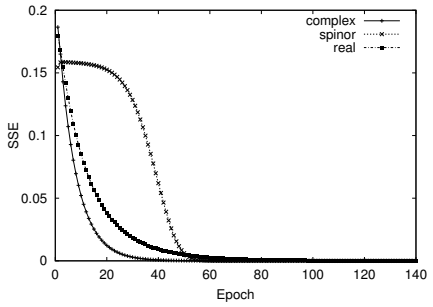

Fig. 12.5. Convergence of the learning

The spinor neuron learned indeed a spinor representation. Its weights in the odd components were zero. This required some more epochs of learning in comparison with the complex neuron. But it learned the task still faster than the real neurons. Besides the qualitative difference of the learning curve of the spinor neuron to the curves of the other neurons, no great quantitative difference could be noticed.

To test the generalization performance of the different neurons we also made simulations with noisy training data, by adding median-free uniform noise up to a level of 20%. The obtained results are shown in Figure 12.6.

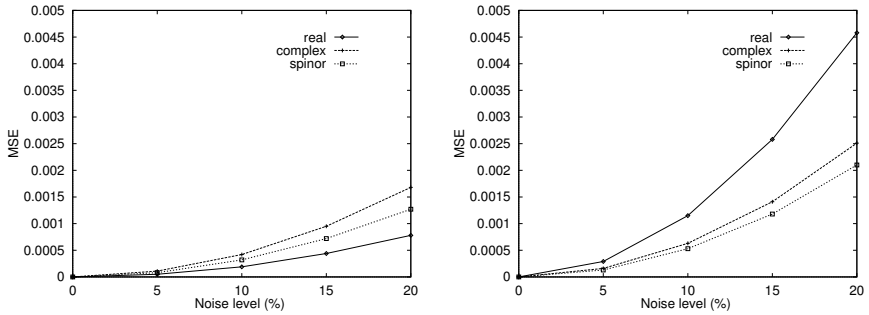

Fig. 12.6. Training errors (left) and generalization errors (right) by different noise levels

Due to the fact that the real neurons compute a general linear transformation, they have learned the noise better than the Clifford neurons. As a consequence, the generalization was then much worse in comparison with the Clifford neurons. There was no significant difference in generalization between the both Clifford neurons. The output on the test data of the real neurons and the complex neuron is shown in Figure 12.7.

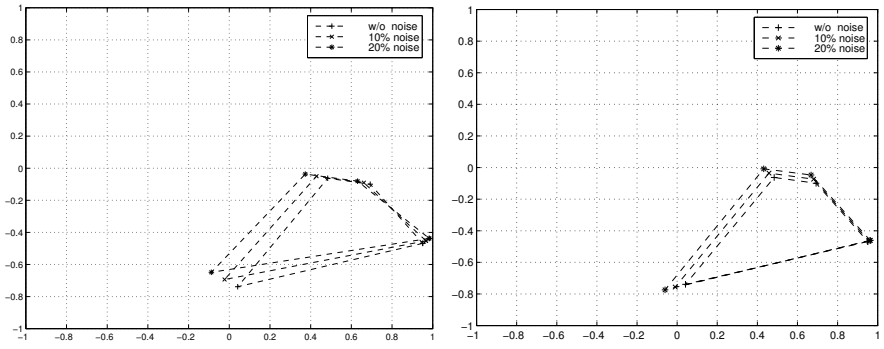

Fig. 12.7. Generalization obtained by the real neurons (left) and the complex neuron (right) by different noise levels

Using the model–based Clifford neurons for data fitting this model gave better results than using real neurons, especially on training with noisy data.

Experiment 3 (Euclidean 3D similarity transformation).
The only 4–dimensional Clifford algebra in which a Euclidean 3D rotation can be computed is $\mathcal{C}_{0,2}$. Thus, we can only compare experimentally the quaternionic and the real way of neural computation of such transformations. Actually, a quaternionic spinor neuron with any input-output representation can compute such a transformation. For the following experiment we use the standard spinor representation. That is we used the input–output representation $xxx0 - xxx0$. For this single quaternionic spinor neuron and a network of real neurons the task was to learn a rotation of -60° about the axis [0.5, $\sqrt{0.5}$, 0.5] with translation about [0.2,-0.2,0.3]. The data for training is shown in Figure 12.8.

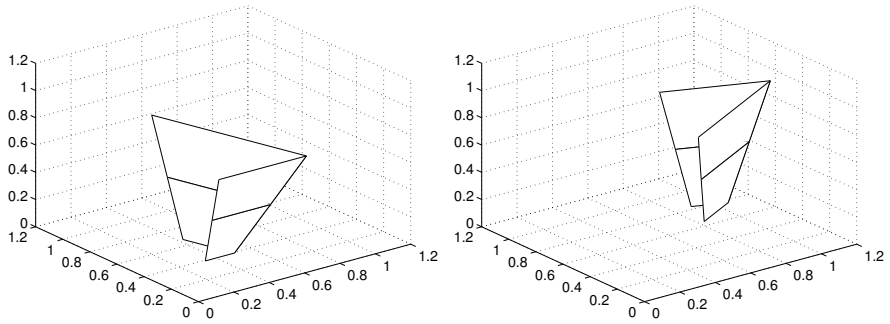

Fig. 12.8. Training data input (left) and training data output(right)

As test set we use a transformed version of the training data as shown in Figure 12.9.

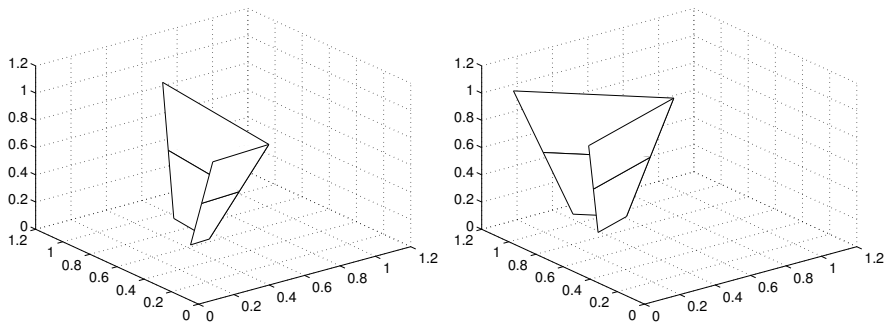

Fig. 12.9. Test data input (left) and test data output (right)

The convergence of the training is shown in Figure 12.10. As we can see, the quaternionic spinor neuron converges much faster than the real neurons. The real neurons have to learn the matrix representation of the quaternionic multiplication. Due to that fact, it was impossible to drop the SSE < 0.00001 for the real neurons. Thus, there exists already a numerical boundary value of reachable accuracy for the computation with real neurons.

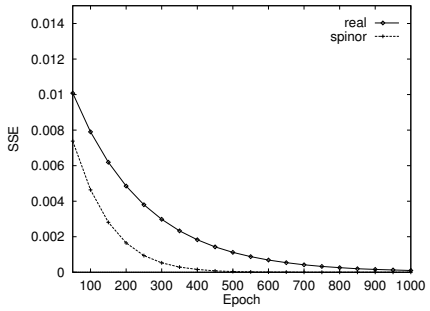

Fig. 12.10. Convergence of the learning

Clearly, this effects the performance of the real neuron on noisy training data in a quiet negative way. The errors for different noise level are shown in Figure 12.11.

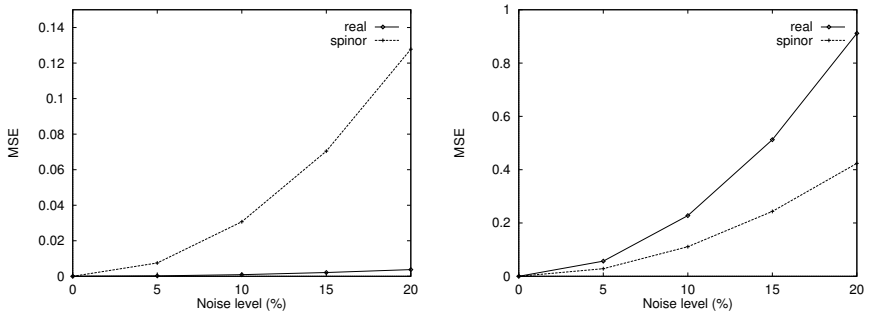

Fig. 12.11. Training errors (left) and generalization errors (right) by different noise levels

The real neurons simply learned the noise. Therefore, their generalization is worse than that of the Clifford neuron by a factor two. Actually, the real neurons performed much worse than indicated by that, as it can be seen by looking at the obtained generalization results shown in Figure 12.12 and Figure 12.13, where crosses indicate the desired output.

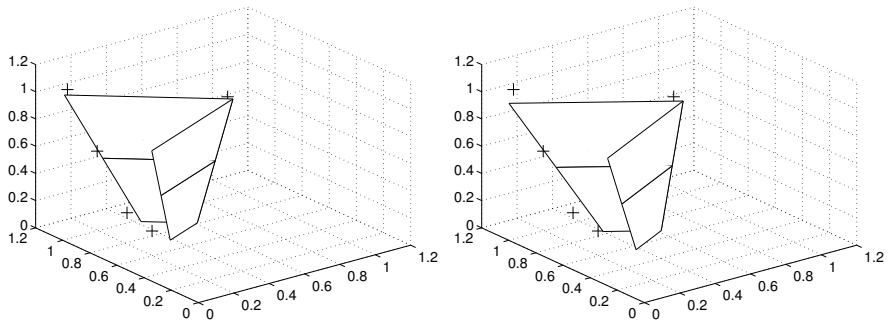

Fig. 12.12. Generalization obtained with spinor neurons by 10% noise (left) and by 20% noise (right)

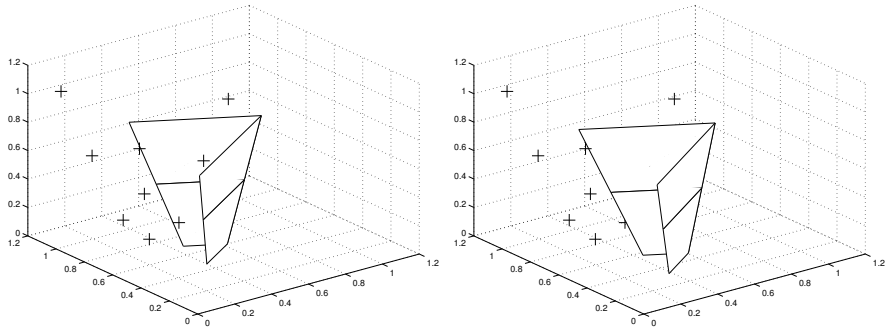

Fig. 12.13. Generalization obtained with real neurons by 10% noise (left) and by 20% noise (right)

The pose of the object generalized by the real neurons is completely wrong. In fact, this is already caused by the way the real neurons learned the task. The model applied is that of the noise. Instead of still separating the problem in a rotation part (weight matrix) and a translation part (biases) they will always use the biases strongly to fit the noise. Instead, a Clifford neuron applies this model of separation.

With this simulation we will finish our study of Clifford neurons as linear operators.

12.5 Möbius Transformations

In this section we will demonstrate, that Clifford neurons are able to learn transformations that are not learnable with real neurons. Clearly, this will require to linearize a non-linear transformation in a unique way in the framework of Clifford algebras.

The transformations in mind are the plane projective transformations. These are the most general transformations mapping lines to lines.

Of course, some theoretical preparations have to be made first. The idea is to relate the projective transformation groups to Möbius transformation groups. So let therefore the *complex general projective group*, denoted by PGL(2,$\mathbb{C}$), $\widehat{\mathbb{C}} := \mathbb{C} \cup \{\infty\}$, be the one-point compactification of $\mathbb{C}$.

The biholomorphic functions of $\widehat{\mathbb{C}}$ in itself are isomorphic to the group of the fractional-linear transformations

$$z \mapsto \tau_A(z) := \frac{az+b}{cz+d}\,,\ A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in GL(2,\mathbb{C})\,. \tag{12.38}$$

This group is also called the Möbius group of $\widehat{\mathbb{C}}$, denoted by $M(\widehat{\mathbb{C}})$. Further then, the map

$$GL(2,\mathbb{C}) \to M(\widehat{\mathbb{C}})\,,\ A \mapsto \tau_A \tag{12.39}$$

is a group isomorphism with kernel identical to the center of $GL(2,\mathbb{C})$. Due to the fact, that PGL(2,$\mathbb{C}$) is GL(2,$\mathbb{C}$) factorized to its center, we then have PGL(2,$\mathbb{C}$) $\simeq M(\widehat{\mathbb{C}})$.

The definition of Möbius transformations of the complex plane $\mathbb{C}$, can be easily generalized to the general case of a quadratic space $\mathbb{R}^{p,q}$, where no explicit notion of the corresponding one-point compactification will be made anymore.

Definition 12.5.1. *The map*

$$\mathcal{C}_{p,q} \to \mathcal{C}_{p,q}, x \mapsto (ax+b)(cx+d)^{-1} \quad a,b,c,d \in \mathcal{C}_{p,q}, \quad (cx+d) \in \Gamma_{\mathrm{p,q}}$$

is called a Möbius transformation of $\mathbb{R}^{p,q}$.

Again, the group formed by Möbius transformations of $\mathbb{R}^{p,q}$ is called the *Möbius group* and will be denoted by $M(p,q)$, that is, $M(\widehat{\mathbb{C}})$ is now denoted by $M(0,1)$. The Möbius group $M(p,q)$ is covered by the orthogonal group $O(p+1,q+1)$, and is therefore (section 3.2) four times covered by Pin(p + 1, q + 1). Clearly then, we have immediately $M(p,q) \simeq M(q,p)$. However, Pin(p + 1, q + 1) acts not directly on elements of $\mathcal{C}_{p,q}$ in $\mathcal{C}_{p+1,q+1}$.

To be able to achieve the intented embedding of $\mathcal{C}_{p,q}$ in $\mathcal{C}_{p+1,q+1}$, i.e. to find a way to let $M(p,q)$ (or Pin(p + 1, q + 1), respectively) operate on $\mathcal{C}_{p,q}$, we must proceed our study of Möbius transformations in terms of matrix groups. We will restrict ourselves thereby essentially to the case of interest, that is Möbius transformations of anti–Euclidean spaces $\mathbb{R}^{0,n}$. The following characterization theorem for that was given already by *Vahlen, 1902*.

Theorem 12.5.1. *A matrix* $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ *with entries in* $\mathcal{C}_{0,n}$ *represents a Möbius transformation of* $\mathbb{R}^{0,n}$, *iff*

(a) $a, b, c, d \in \Gamma_{0,n} \cup \{0\}$
(b) $\bar{a}b, b\bar{d}, \bar{d}c, c\bar{a} \in \mathbb{R}^{0,n}$
(c) $a\tilde{d} - b\tilde{c} \in \mathbb{R}\backslash\{0\}$.

Matrices fulfilling these conditions are called *Vahlen matrices*.

A characterization of Möbius transformations of Euclidean spaces is easily obtained by switching the signature $(0, n)$ to $(n, 0)$ in the above theorem. For the general case of a quadratic space with an arbitrary signature one has to allow all products of vectors (not only invertible) in $\mathbb{R}^{p,q}$ in condition (a) of Theorem 12.5.1, which is just the same if $p = 0$ or $q = 0$.

We will now develop the representation of Möbius transformations of the complex plane in detail. Due to the fact that $\mathcal{C}_{p,q}(2) \simeq \mathcal{C}_{p+1,q+1}$, the algebra to concern is $\mathcal{C}_{1,2}$, for which we need a matrix representation firstly. This is given by defining the following basis

$$e_0 := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \; e_1 := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \; e_2 := \begin{pmatrix} i & 0 \\ 0 & -i \end{pmatrix} \; e_3 := \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

and the remaining basis vectors are easily obtained by matrix multiplication, e.g.

$$e_{123} = \begin{pmatrix} -i & 0 \\ 0 & -i \end{pmatrix} .$$

A complex number z can therefore be represented as a matrix in an obvious way either by

$$Z' := \begin{pmatrix} z & 0 \\ 0 & z \end{pmatrix}$$

or equivalently by

$$Z'' := \begin{pmatrix} z & 0 \\ 0 & \bar{z} \end{pmatrix}$$

with the corresponding multivectors $(Re(z), 0, 0, 0, 0, 0, 0, -Im(z))$ and $(Re(z), 0, Im(z), 0, 0, 0, 0, 0)$, respectively. Although outside our main focus, we should remark as a warning, that none of them gives a multivector representation of complex numbers in $\mathcal{C}_{1,2}$, because complex multiplication is not preserved. For a complex Vahlen matrix V neither $VZ'V^{\tilde{}}$ nor $VZ''V^{\tilde{}}$ represent a Möbius transformation in general.

The right embedding to choose is

$$Z := \begin{pmatrix} z & z\bar{z} \\ 1 & \bar{z} \end{pmatrix} , \tag{12.40}$$

which can be deduced by using the concept of paravectors, mentioned earlier in section 3.2.
The corresponding multivector is then given by

$$(Re(z), \frac{1}{2}(1 + z\bar{z}), Im(z), \frac{1}{2}(1 - z\bar{z}), 0, 0, 0, 0)\,.$$

Applying now a complex Vahlen matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ as a spinor to Z one obtains

$$\begin{aligned} & \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} z & z\bar{z} \\ 1 & \bar{z} \end{pmatrix} \begin{pmatrix} a & b \\ c & d \end{pmatrix}^{\tilde{}} \\ = & \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} z & z\bar{z} \\ 1 & \bar{z} \end{pmatrix} \begin{pmatrix} \bar{d} & \bar{b} \\ \bar{c} & \bar{a} \end{pmatrix} \\ = & \lambda \begin{pmatrix} z' & z'\bar{z'} \\ 1 & \bar{z'} \end{pmatrix} \end{aligned}$$

where $\lambda = |bz + d|^2$ and $z' = (az + b)(cz + d)^{-1}$. Thus, we have found the spinor representation of a complex Möbius transformation in $\mathcal{C}_{1,2}$. With some effort λ could be expressed only in terms of the parameters of the Möbius transformation. Therefore, we can speak of it as a scaling factor.

Experiment 4 (Möbius transformation).
As an example, we will now study how the Möbius transformation

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

can be learned by $\mathcal{C}_{1,2}$-Neurons, but not by real valued neurons.

The training and test data used for this task is shown below in Fig. 12.14.

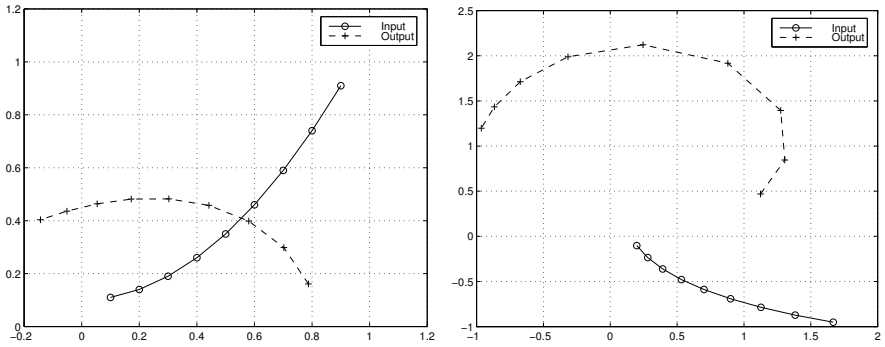

Fig. 12.14. Training data (left) and test data (right)

Neither on the pure complex input-output pairs nor on the coded data (12.40) an MLP can generalize the transformation. In the first case a training SSE of 0.00326 results in a generalization SSE of 6.15109 on the test set, in the second case a generalization SSE of 2.31305 was reached, although the training SSE was only 0.00015. So the MLP has in both cases just memorized the training examples, but not learned the structure within the data, because of its missing abilities to do so, namely the to embed the 4-dimensional data correctly in the required 8-dimensional space. This was done, as theoretically derived, by the Clifford neurons with nice robustness with respect to noise as shown in Figs. 12.15 and 12.16, respectively.

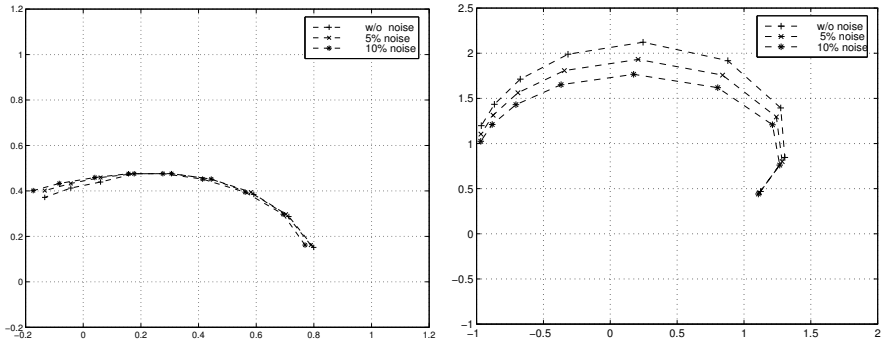

Fig. 12.15. Learned transformation on training data (left) and test data(right)

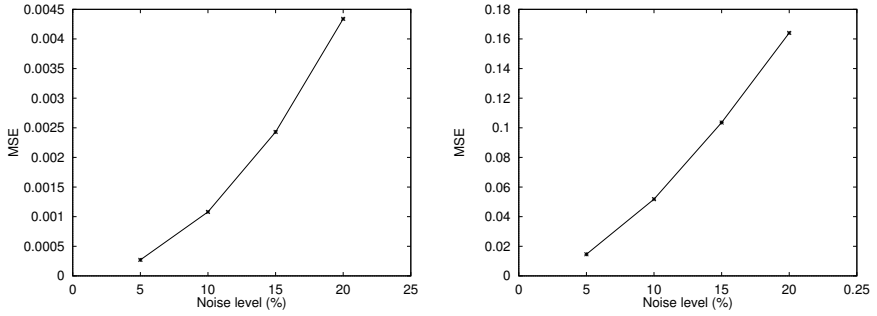

Fig. 12.16. MSE vs. noise level for training (left) and testing (right)

12.6 Summary

In this first of two chapters on Clifford neural computation we discussed the Clifford neuron in detail. We showed how the geometric product can be used with an associator. We introduced the special neuron model of spinor neurons that allows to compute orthogonal transformations very elegantly. This way of computation was proven to be faster and much more robust against noise as real single–layer neural networks. Moreover, we were able to show on the example of Möbius transformations that there exist geometric transformations that are only exclusively computable by Clifford neurons. This was done by using a non–linear coding of the data which resulted in a linearization in Clifford algebras.

We now will make the transition from the Clifford neuron and linearity to Clifford neural networks and non–linearity in the subsequent chapter.