Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 213–220

ISBN 978-83-60810-22-4

Preserivng pieces of information in a given order in HRR and GA_c

Agnieszka Patyk-Łońska

***Abstract*—Geometric Analogues of Holographic Reduced Representations (GA HRR or GA_c—the continuous version of discrete GA described in [16]) employ role-filler binding based on geometric products. Atomic objects are real-valued vectors in n-dimensional Euclidean space and complex statements belong to a hierarchy of multivectors. A property of GA_c and HRR studied here is the ability to store pieces of information in a given order by means of trajectory association. We describe results of an experiment: finding the alignment of items in a sequence without the precise knowledge of trajectory vectors.**

***Index Terms*—distributed representations, geometric algebra, HRR, BSC, word order, trajectory associations, bag of words.**

I. Introduction

OVER the years several attempts have been made to preserve the order in which the objects are to be remembered with the help of binding and superposition. While some solutions to the problem of preserving pieces of information in a given order have proved ingenious, others are obviously flawed. Let us consider the representation of the word eye—it has three letters, one of which occurs twice. The worst possible choice of binding and superposition would be to store quantities of letters, e.g.

$$eye = twice * e + once * y, \tag{1}$$

since we would not be able to distinguish eye from eey or yee. Another ambiguous representation would be to remember the neighborhood of each letter

$$eye = before_y * e + between_e * y + after_y * e. \tag{2}$$

Unfortunately, such a method of encoding causes words eye and $eyeye$ to have the same representation

$$\begin{aligned} eyeye &= before_y * e + 2 \cdot between_e * y + \\ &\quad (before_y + after_y) * e + after_y * e \\ &= 2\big(before_y * e + between_e * y + after_y * e\big) \\ &= 2\ eye. \end{aligned} \tag{3}$$

Real-valued vectors are normalized in most distributed representation models, therefore the factor of 2 would be most likely lost in translation. Such *contextual roles* (Smolensky [19]) cause problems when dealing with certain types of palindromes. Remembering positions of letters is also not a good solution

$$eye = letter_{first} * e + letter_{second} * y + letter_{third} * e \tag{4}$$

as we need to redundantly repeat the first letter as the third letter, otherwise we could not distinguish eye from ey or ye. Secondly, this method of encoding will not detect similarity between eye and $yeye$.

A quantum-like attempt to tackle the problem of information ordering was made in [1]—a version of semantic analysis, reformulated in terms of a Hilbert-space problem, is compared with structures known from quantum mechanics. In particular, an LSA matrix representation [1], [10] is rewritten by the means of quantum notation. Geometric algebra has also been used extensively in quantum mechanics ([2], [4], [3]) and so there seems to be a natural connection between LSA and GA_c, which is the ground for fututre work on the problem of preserving pieces of information in a given order.

As far as convolutions are concerned, the most interesting approach to remembering information in a given order has been described in [12]. Authors present a model that builds a holographic lexicon representing both word meaning and word order from unsupervised experience with natural language texts comprising altogether 90000 words. This model uses simple convolution and superposition to construct n-grams recording the frequency of occurrence of every possible word sequence that is encountered, a window of about seven words around the target word is usually taken into consideration. To predict a word in a completely new sentence, the model looks up the frequency with which the potential target is surrounded by words present in the new sentence. To be useful, n-gram models need to be trained on massive amounts of text and therefore require extensive storage space. We will use a completely different approach to remembering information order—trajectory association described by Plate in [18]. Originally, this technique also used convolution and correlation, but this time items stored in a sequence are actually superimposed, rather than being bound together.

II. Geometric analogues of HRR

Holographic Reduced Representations (HRR) and Binary Spatter Codes (BSC) are distributed representations of cognitive structures where binding of role–filler codevectors maintains predetermined data size. In HRR [17], [18] binding is performed by means of circular convolution

$$(x \circledast y)_j = \sum_{k=0}^{n-1} x_k y_{j-k \bmod n}. \tag{5}$$

of real n-tuples or, in 'frequency domain', by componentwise multiplication of (complex) n-tuples,

$$(x_1, \ldots, x_n) \circledast (y_1, \ldots, y_n) = (x_1 y_1, \ldots, x_n y_n). \tag{6}$$

978-83-60810-22-4/$25.00

Bound n-tuples are superposed by addition, and unbinding is performed by an approximate inverse. A dual formalism, where real data are bound by componentwise multiplication, was discussed by Gayler [9]. In BSC [13], [14] one works with binary n-tuples, bound by componentwise addition mod 2,

$$\begin{aligned}(x_1,\ldots,x_n)\oplus(y_1,\ldots,y_n) &= (x_1\oplus y_1,\ldots,x_n\oplus y_n),\\ x_j\oplus y_j &= x_j+y_j \bmod 2,\end{aligned} \tag{7}$$

and superposed by pointwise majority-rule addition; unbinding is performed by the same operation as binding.

One often reads that the above models represent data by *vectors*, which is not exactly true. Given two vectors one does not know how to perform, say, their convolution or componentwise multiplication since the result depends on basis that defines the components. Basis must be fixed in advance since otherwise all the above operations become ambiguous. Geometric Analogues of Holographic Reduced Representations (GA HRR) [5] can be constructed if one defines binding by the geometric product, a notion introduced in 19th century works of Grassmann [11] and Clifford [8].

In order to grasp the main ideas behind GA HRR let us consider an orthonormal basis $b_1,\ldots,b_n$ in some n-dimensional Euclidean space. Now consider two vectors $x=\sum_{k=1}^n x_k b_k$ and $y=\sum_{k=1}^n y_k b_k$. The *scalar*

$$x\cdot y = y\cdot x \tag{8}$$

is known as the *inner product*. The *bivector*

$$x\wedge y = -y\wedge x \tag{9}$$

is the *outer product* and may be regarded as an oriented plane segment (alternative interpretations are also possible, cf. [7]). $\mathbf{1}$ is the identity of the algebra. The geometric product of x and y then reads

$$xy = \underbrace{\sum_{k=1}^n x_k y_k\ \mathbf{1}}_{x\cdot y} + \underbrace{\sum_{k<l}(x_k y_l - y_k x_l) b_k b_l}_{x\wedge y}. \tag{10}$$

Grassmann and Clifford introduced geometric product by means of the basis-independent formula involving the *multivector*

$$xy = x\cdot y + x\wedge y \tag{11}$$

which implies the so-called Clifford algebra

$$b_k b_l + b_l b_k = 2\delta_{kl}\mathbf{1}. \tag{12}$$

when restricted to an orthonormal basis. Inner and outer product can be defined directly from xy:

$$x\cdot y = \frac{1}{2}(xy+yx), \qquad x\wedge y = \frac{1}{2}(xy-yx).$$

The most ingenious element of (11) is that it adds two apparently different objects, a scalar and a plane element, an operation analogous to addition of real and imaginary parts of a complex number. Geometric product for vectors x, y, z can be axiomatically defined by the following rules:

$$\begin{aligned}(xy)z &= x(yz),\\ x(y+z) &= xy+xz,\\ (x+y)z &= xz+yz,\\ xx = x^2 &= |x|^2,\end{aligned}$$

where $|x|$ is a positive scalar called the magnitude of x. The rules imply that $x\cdot y$ must be a scalar since

$$xy+yx = |x+y|^2 - |x|^2 - |y|^2.$$

Geometric algebra allows us to speak of inverses of vectors: $x^{-1}=x/|x|^2$. x is invertible (i.e. possesses an inverse) if its magnitude is nonzero. Geometric product of an arbitrary number of invertible vectors is also invertible. The possibility of inverting all nonzero-magnitude vectors is perhaps the most important difference between geometric and convolution algebras.

Geometric products of *different* basis vectors

$$b_{k_1\ldots k_j} = b_{k_1}\ldots b_{k_j},$$

$k_1<\cdots<k_j$, are called basis blades (or just blades). In n-dimensional Euclidean space there are 2^n different blades. This can be seen as follows. Let $\{x_1,\ldots,x_n\}$ be a sequence of bits. Blades in an n-dimensional space can be written as

$$c_{x_1\ldots x_n} = b_1^{x_1}\ldots b_n^{x_n}$$

where $b_k^0=\mathbf{1}$, which shows that blades are in a one-to-one relation with n-bit numbers. A general multivector is a linear combination of blades,

$$\psi = \sum_{x_1\ldots x_n=0}^{1} \psi_{x_1\ldots x_n} c_{x_1\ldots x_n}, \tag{13}$$

with real or complex coefficients $\psi_{x_1\ldots x_n}$. Clifford algebra implies that

$$c_{x_1\ldots x_n}c_{y_1\ldots y_n} = (-1)^{\sum_{k<l} y_k x_l} c_{(x_1\ldots x_n)\oplus(y_1\ldots y_n)}, \tag{14}$$

where $\oplus$ is given by (7). Multiplication of two basis blades is thus, up to a sign, in a one-to-one relation with exclusive alternative of two binary n-tuples. Accordingly, (14) is a projective representation of the group of binary n-tuples with addition modulo 2.

GA HRR is based on binding defined by geometric product (14) of blades while superposition is just addition of blades (13). The discrete GA_d is a version of GA HRR obtained if $\psi_{x_1\ldots x_n}$ in (13) equal ±1. The first recognition tests of GA_d, as compared to HRR and BSC, were described in [16]. In the present paper we go further and compare HRR and BSC with GA_c, a version of GA HRR employing "projected products" [5] and arbitrary real $\psi_{x_1\ldots x_n}$.

Throughout this paper we shall use the following notation: "$*$" denotes binding roles and fillers by means of the geometric

product and "+" denotes the superposition of sentence chunks, e.g.

$$\textit{"Fido bit Pat"} = bite_{agt} * Fido + bite_{obj} * Pat. \quad (15)$$

Additionally, "$\circledast$" will denote binding performed by circular convolution used in the HRR model and a^* denotes the involution of a HRR vector a. A "+"' in the superscript of x^+ denotes the operation of reversing a blade or a multivector x: $(b_{k_1 \ldots k_j})^+ = b_{k_j} \ldots b_{k_1}$. Asking a question will be denoted with "$\sharp$", as in

$$\begin{aligned} &\textit{"Who bit Pat?"} \\ &= (bite_{agt} * Fido + bite_{obj} * Pat) \sharp bite_{agt} \quad (16) \\ &\approx Fido. \end{aligned}$$

The *size* of a (multi)vector means the number of memory cells it occupies in computer's memory, while the *magnitude* of a (multi)vector $V = \{v_1, \ldots, v_n\}$ is its Euclidean norm $\sqrt{\sum_{i=1}^{n} v_i^2}$.

For our purposes it is important that geometric calculus allows us to define in a very systematic fashion a hierarchy of associative, non-commutative, and invertible operations that can be performed on 2^n-tuples. The resulting superpositions are less noisy than the ones based on convolutions, say. Geometric product preserves dimensionality at the level 2^n-dimensional *multivectors*, where n is the number of bits indexing basis vectors. Moreover, all nonzero vectors are invertible with respect to geometric product, a property absent for convolutions and important for unbinding and recognition. A detailed analysis of links between GA HRR, HRR and BSC can be found in [5].

III. The GA$_c$ model

The procedure we employ was suggested in [5]. The space of 2^n-tuples is split into subspaces corresponding to scalars (0-vectors), vectors (1-vectors), bivectors (2-vectors), and so on. At the bottom of the hierarchy lay vectors $V \in \mathbb{R}^n$, having rank 1 and being denoted as $\overset{1}{V}$. An object of rank 2 is created by multiplying two elements of rank 1 with the help of the geometric product. Let $\overset{1}{V} = \{\alpha_1, \alpha_2, \alpha_3\}$ and $\overset{1}{W} = \{\beta_1, \beta_2, \beta_3\}$ be vectors in $\mathbb{R}^3$. A multivector $\overset{2}{X}$ of rank 2 in $\mathbb{R}^3$ comprises the following elements (cf. [15])

$$\overset{2}{X} = \overset{1}{V}\overset{1}{W} = \begin{bmatrix}\alpha_1\\ \alpha_2\\ \alpha_3\end{bmatrix}\begin{bmatrix}\beta_1\\ \beta_2\\ \beta_3\end{bmatrix} = \begin{bmatrix}\alpha_1\beta_1 + \alpha_2\beta_2 + \alpha_3\beta_3\\ \alpha_1\beta_2 - \alpha_2\beta_1\\ \alpha_1\beta_3 - \alpha_3\beta_1\\ \alpha_2\beta_3 - \alpha_3\beta_2\end{bmatrix}, \quad (17)$$

the first entry in the array on the right being a scalar and the remaining three entries being 2-blades. For arbitrary vectors in $\mathbb{R}^n$ we would have obtained one scalar (or, more conviniently: $\binom{n}{0}$ scalars) and $\binom{n}{2}$ 2-blades.

Let $\overset{2}{X} = \{\gamma_1, \gamma_2, \gamma_3, \gamma_4\}$ and $\overset{1}{V} = \{\alpha_1, \alpha_2, \alpha_3\}$ be two multivectors in $\mathbb{R}^3$. A multivector $\overset{3}{Z}$ of rank 3 in $\mathbb{R}^3$ may be created in two ways: as a result of multiplying either $\overset{1}{V}$ by $\overset{2}{X}$ or $\overset{2}{X}$ by $\overset{1}{V}$. Let us concentrate on the first case

$$\overset{3}{Z} = \overset{1}{V}\overset{2}{X} = \begin{bmatrix}\alpha_1\\ \alpha_2\\ \alpha_3\end{bmatrix}\begin{bmatrix}\gamma_1\\ \gamma_2\\ \gamma_3\\ \gamma_4\end{bmatrix} = \begin{bmatrix}\alpha_1\gamma_1 - \alpha_2\gamma_2 - \alpha_3\gamma_3\\ \alpha_1\gamma_2 + \alpha_2\gamma_1 - \alpha_3\gamma_4\\ \alpha_1\gamma_3 + \alpha_2\gamma_4 + \alpha_3\gamma_1\\ \alpha_1\gamma_4 - \alpha_2\gamma_3 + \alpha_3\gamma_2\end{bmatrix}. \quad (18)$$

Here, the first three entries in the resulting matrix are 1-blades, while the last entry is a 3-blade. For arbitrary multivectors of rank 1 and 2 in $\mathbb{R}^n$ we would have obtained $\binom{n}{1}$ vectors and $\binom{n}{3}$ trivectors. We cannot generate multivectors of rank higher than 3 in $\mathbb{R}^3$, but it is easy to check that in spaces $\mathbb{R}^{n>3}$ a multivector of rank 4 would have $\binom{n}{0}$ scalars, $\binom{n}{2}$ bivectors and $\binom{n}{4}$ 4-blades. The number of k-blades in a multivector of rank r is described by Table I. It becomes clear that a multivector of rank r over $\mathbb{R}^n$ is actually a vector over a $\sum_{i=0}^{\lfloor \frac{r}{2} \rfloor} \binom{n}{2i + r \bmod 2}$-dimensional space.

As an example let us consider the following roles and fillers being normalized vectors drawn randomly from $\mathbb{R}^n$ with Gaussian distribution $\mathrm{N}(0, \frac{1}{n})$

$$\begin{array}{rclrcl} Pat &=& \{a_1, \ldots, a_n\}, & name &=& \{x_1, \ldots, x_n\}, \\ male &=& \{b_1, \ldots, b_n\}, & sex &=& \{y_1, \ldots, y_n\}, \\ 66 &=& \{c_1, \ldots, c_n\}, & age &=& \{z_1, \ldots, z_n\}. \end{array} \quad (19)$$

$PSmith$, who is a 66 year old male named Pat, is created by first multiplying roles and fillers with the help of the geometric product

$$\begin{aligned} PSmith &= name * Pat + sex * male + age * 66 \\ &= name \cdot Pat + name \wedge Pat + sex \cdot male + \\ &\quad sex \wedge male + age \cdot 66 + age \wedge 66 \quad (20) \end{aligned}$$

$$\begin{aligned} &= \begin{bmatrix} \sum_{i=1}^{n}(a_i x_i + b_i y_i + c_i z_i) \\ a_1x_2 - a_2x_1 + b_1y_2 - b_2y_1 + c_1z_2 - c_2z_1 \\ a_1x_3 - a_3x_1 + b_1y_3 - b_3y_1 + c_1z_3 - c_3z_1 \\ \vdots \\ a_{n-1}x_n - a_nx_{n-1} + b_{n-2}y_n - b_ny_{n-1} + c_{n-1}z_n - c_nz_{n-1} \end{bmatrix} \\ &= [d_0, d_{12}, d_{13}, \ldots, d_{(n-1)n}]^T \\ &= d_0 + d_{12}e_{12} + d_{13}e_{13} + \cdots + d_{(n-1)n}e_{(n-1)n}, \quad (21) \end{aligned}$$

where $e_1, \ldots, e_n$ are orthonormal basis blades. In order to be decoded as much correctly as possible, $PSmith$ should have the same magnitude as vectors representing atomic objects, therefore it needs to be normalized. Finally, $PSmith$ takes the form of

$$PSmith = [\hat{d}_0, \hat{d}_{12}, \hat{d}_{13}, \ldots, \hat{d}_{(n-1)n}]^T, \quad (22)$$

where $\hat{d}_i = \frac{d_i}{\sqrt{\sum_{j=0,12}^{(n-1)n} d_j^2}}$.

TABLE I
NUMBERS OF k-BLADES IN MULTIVECTORS OF VARIOUS RANKS IN $\mathbb{R}^n$

rank	scalars	vectors	bivectors	trivectors	4-blades	...	data size
1	0	$\binom{n}{1}$	0	0	0	...	$o(\binom{n}{1})$
2	$\binom{n}{0}$	0	$\binom{n}{2}$	0	0	...	$o(\binom{n}{0}+\binom{n}{2})$
3	0	$\binom{n}{1}$	0	$\binom{n}{3}$	0	...	$o(\binom{n}{1}+\binom{n}{3})$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$2r$	$\binom{n}{0}$	0	$\binom{n}{2}$	0	$\binom{n}{4}$	...	$o(\sum_{i=0}^{r}\binom{n}{2i})$
$2r+1$	0	$\binom{n}{1}$	0	$\binom{n}{3}$	0	...	$o(\sum_{i=0}^{r}\binom{n}{2i+1})$

$PSmith$ is now a multivector of rank 2. The decoding operation

$$\begin{aligned} & name^{+}PSmith \\ &= name^{+}(name\cdot Pat + name\wedge Pat + sex\cdot male \\ &\quad + sex\wedge male + age\cdot 66 + age\wedge 66) \end{aligned} \tag{23}$$

will produce a multivector of rank 3 consisting of vectors and trivectors. However, the original Pat did not contain any trivector components—they all belong to the noise part and the only interesting blades in $name^{+}PSmith$ are vectors. The expected answer is a vector, therefore there is no point in calculating the whole multivector $name^{+}PSmith$ and only then comparing it with items stored in the clean-up memory. To be efficient, one should generate only the vector-part while computing $name^{+}PSmith$ and skip the noisy trivectors.

Let $\langle\cdot\rangle_k$ denote the projection of a multivector on k-blades. To decode $PSmith$'s $name$ we need to compute

$$\begin{aligned} & \langle name^{+}PSmith\rangle_1 \\ &= name^{+}namePat + \langle\ name^{+}(name\wedge Pat \\ &\quad + sex\cdot male + sex\wedge male + age\cdot 66 + age\wedge 66)\ \rangle_1 \\ &= Pat + \text{noise} = Pat'. \end{aligned} \tag{24}$$

The resulting Pat' will still be noisy, but to a lesser degree than it would have been if the trivectors were present.

Formally, we are using a map $*^1_{1,2}$ that transforms a multivector of rank 1 (i.e. an n-tuple) and a multivector of rank 2 (i.e. a $(1+\frac{(n-1)n}{2})$-tuple) into a multivector of rank 1 without computing the unnecessary blades. Let X be a multivector of rank 2

$$X = \langle X\rangle_0 + \langle X\rangle_2 = x_0 + \sum_{l<m} x_{lm}e_l e_m, \tag{25}$$

where $x_{lm} = -x_{ml}$. If $A = (A_1,\ldots,A_n)$ is a decoding vector (actually, an inverse of a role vector), then

$$\begin{aligned} A *^1_{1,2} X &= x_0 A + \sum_{l,m} A_l x_{lm} e_m \\ &= \sum_k \big(xA_k + \sum_l A_l x_{lk}\big)e_k \\ &= \sum_k Y_k e_k = Y, \end{aligned} \tag{26}$$

with $Y = (Y_1,\ldots,Y_n)$ being an n-tuple, i.e. a multivector of rank 1. More explicitly,

$$Y_k = (A *^1_{1,2} X)_k = x_0 A_k + \sum_{l=1}^{k-1} A_l x_{lk} - \sum_{l=k+1}^{n} A_l x_{kl}. \tag{27}$$

The map $*^1_{1,2}$ is an example of a *projected product*, introduced in [5], reconstructing the vector part of AX without computing the unnecessary parts. The projected product is basis independent, as opposed to circular convolutions. In general, $*^m_{l,k}$ transforms the geometric product of two multivectors $\overset{l}{A}$ and $\overset{k}{B}$ into a multivector $\overset{m}{C}$.

We now need to compare Pat' with other items stored in the clean-up memory using the dot product, and since Pat' is a vector, we need to compare only the vector part. That means, if the clean-up memory contained a multivector $\overset{2t+1}{M}$ of an odd rank, we would also need to compute $Pat'\cdot\langle\overset{2t+1}{M}\rangle_1$ while searching for the right answer.

This method of decoding suggests that items stored in the clean-up memory should hold information about their ranks, which is dangerously close to employing fixed data slots present in localist architectures. However, a rank of a clean-up memory item can be "guessed" from its size. In a distributed model we also should not "know" for sure how many parts the projected product should reject, but it can certainly reject parts spanned by blades of highest grades.

Before providing formulas for encoding and decoding a complex statement we need to introduce additional notation for the projected product and the projection. We have already introduced the projected product $*^m_{l,k}$ transforming the geometric product of two multivectors of ranks l and k into a multivector of rank m. This will not always be the case for complex statements, since we can produce a multivector that will not be of any given rank. Let $*^m_{l,\{\alpha_1,\alpha_2,\ldots,\alpha_k\}}$ denote the projected product transforming the geometric product of a multivector $\overset{l}{A}$ and a multivector B containing α_1-blades, α_2-blades,... and α_k-blades into a multivector $\overset{m}{C}$. In this way, the projected product $*^1_{1,2}$ may be written down as $*^1_{1,\{0,2\}}$. By analogy, let $\langle\cdot\rangle_{\{\alpha_1,\alpha_2,\ldots,\alpha_k\}}$ denote the projection of a multivector on components spanned by α_1-blades, α_2-blades,... and α_m-blades.

Let Ψ denote the normalized multivector encoding the sentence "*Fido bit PSmith*", i.e.

$$\Psi = \underbrace{bite_{agt} * Fido}_{\text{rank 2}} + \underbrace{bite_{obj} * \underbrace{PSmith}_{\text{rank 2}}}_{\text{rank 3}}. \tag{28}$$

Multivector Ψ will contain scalars, vectors, bivectors and trivectors and can be written down as the following vector of dimension $\sum_{i=0}^{3} \binom{n}{i}$

$$\Psi = \underbrace{\alpha}_{\text{a scalar}} + \underbrace{\sum_{i=1}^{n} \beta_i e_i}_{\text{vectors}} + \underbrace{\sum_{1=i<j}^{n} \gamma_{ij} e_{ij}}_{\text{bivectors}} + \underbrace{\sum_{1=i<j<k}^{n} \delta_{ijk} e_{ijk}}_{\text{trivectors}} \tag{29}$$

IV. Trajectory association

In the HRR model vectors are normalized and therefore can be regarded as radii of a sphere of radius 1. If we attach a sequence of items, say A, B, C, D, E to arrowheads of five of those vectors, we obtain a certain $trajectory$ on the surface of a sphere, that is associated with sequence $ABCDE$. This is a geometric analogue to the $method\ of\ loci$ which instructs to remember a list of items by associating each term with a distinctive location along a familiar path. Let k be a randomly chosen HRR vector and let

$$k^i = k \circledast k^{i-1} = k^{i-1} \circledast k, \quad i > 1 \tag{30}$$

be its ith $power$, with $k^1 = k$. The sequence S_{ABCDE} is then stored as

$$S_{ABCDE} = A \circledast k + B \circledast k^2 + C \circledast k^3 + D \circledast k^4 + E \circledast k^5. \tag{31}$$

Of course, each power of k needs to be normalized before being bound with a sequence item. Otherwise, every subsequent power of k would be larger or smaller than its predecessor. As a result, every subsequent item stored in a sequence would have a bigger or a smaller share in vector S_{ABCDE}. Obviously, this method cannot be applied to the discrete GA model (decribed in [16]) or to BSC, since it is impossible to obtain more than two distinct powers of a vector with the use of XOR as a means of binding.

This technique has a few obvious advantages present in HRR but not in GA$_c$ had we wished to use ordinary vectors as first powers—different powers of a vector k would then be multivectors of different ranks. While k^i and $k^{i\pm1}$ are very similar in HRR, in GA$_c$ they would not even share the same blades. Further, the similarity of k^i and k^{i+m} in HRR is the same as the similarity of k^j and k^{j+m}, whereas in GA$_c$ that similarity would depend on the parity of i and j. In the light of these shortcomings, we need to use another structure acting as a first power in order to make trajectories work in GA$_c$. Let t be a random normalized full multivector over $\mathbb{R}^n$ and let us define powers of t in the following way

$$t^1 = t, \quad t^i = (t^{i-1})t \ \text{ for } i > 1. \tag{32}$$

We will store vectors $a_1 \dots a_l$ in a sequence $S_{a_1 \dots a_l}$ using powers of the multivector t

$$S_{a_1 \dots a_l} = a_1 t + a_2 t^2 + \dots + a_l t^l. \tag{33}$$

To answer a question "*What is the second item in a sequence?*" in GA$_c$ we need to use the projected product

$$\langle S_{a_1 \dots a_l} (t^2)^+ \rangle_1 \approx a_2, \tag{34}$$

and to find out the place of item a_i we need to compute

$$(a_i)^+ S_{a_1 \dots a_l} \approx t^i. \tag{35}$$

Some may argue that such encoding puts a demand on items in the clean-up memory to hold information if they are roles or fillers, which is dangerously close to employing fixed data slots present in localist architectures. Actually, elements of a sequence can be recognized by their size, relatively shorter than the size of multivector t and its powers.

V. Item alignment

We present an experiment using trajectory association and we comment on test results for HRR and GA$_c$ models. We tested whether the HRR and GA$_c$ models were capable of performing the following task:

> *Given only a set of letters A, B, C, D, E and an encoded sequence $S_{?????}$ comprised of those five letters find out the position of each letter in that sequence.*

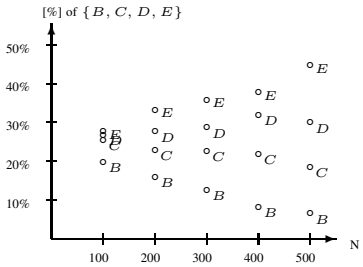
We assumed that no direct access to t or its powers is given—they do belong to the clean-up memory, but cannot be retrieved "by name". One may think of this problem as a "black box" that inputs randomly chosen letter vectors and in return outputs a (multi)vector representing always the same sequence, irrespectively of the dimension of data. Inside, the black box generates (multi)vectors t, t^2, t^3, t^4, t^5. Their values are known to the observer but their names are not. Since we can distinguish letters from non-letters, the naive approach would be to try out all 120 alignments of letters A, B, C, D and E using all possible combinations of non-letters as the powers of t. Unfortunately, powers of t are different each time the black box produces a sequence. We will use an algorithm based on two assumptions:

(a) t^x, if not recognized correctly, is more similar to highest powers of t,
(b) letters lying closer to the end of the sequence are often offered as the incorrect answer to questions concerning letters, as in $S \sharp t^n$.

Assumption (a) can be easily justified: since lower powers of t are recognized correctly more often, higher powers of t come up more often as the incorrect answer to $S \sharp A$. Vector t^3 is the correct answer to $S_{xxAxx} \sharp A$. However, if t^3 is not recognized, the next most similar answer will be t^5 because it contains three "copies" of t^3, indicated here by brackets

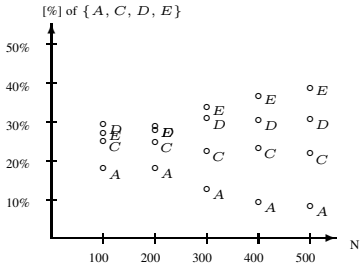
$$\Big\{ t * \Big(t * [t \Big\} * t \Big) * t]. \tag{36}$$

The second most similar item will be t^4 because it contains two "copies" of t, and so on. The item least similar to t^3 will



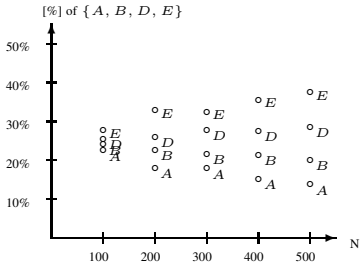
Asking with A

	N=100	N=200	N=300	N=400	N=500	
non A	1643	960	616	506	393	= 100%
B	19.90%	16.04%	12.66%	8.30%	6.62%	
C	25.56%	22.92%	22.73%	21.94%	18.58%	
D	26.78%	27.71%	28.73%	32.02%	30.03%	
E	27.75%	33.33%	35.88%	37.75%	44.78%	



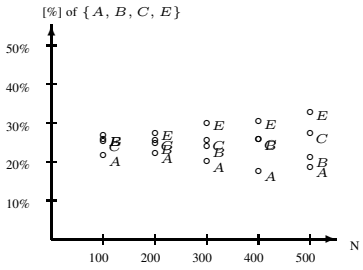
Asking with B

	N=100*	N=200*	N=300	N=400	N=500	
non B	2474	1588	1133	837	628	= 100%
A	18.03%	18.14%	12.71%	9.44%	8.28%	
C	25.14%	24.87%	22.51%	23.30%	21.97%	
D	29.59%	28.90%	30.98%	30.59%	30.89%	
E	27.24%	28.09%	33.80%	36.68%	38.85%	



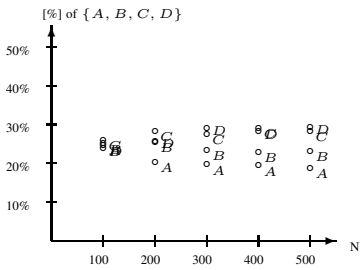
Asking with C

	N=100	N=200	N=300	N=400	N=500	
non C	3003	2214	1694	1346	1121	= 100%
A	22.51%	18.16%	18.00%	15.30%	13.83%	
B	24.18%	22.81%	21.55%	21.47%	20.07%	
D	25.51%	26.06%	27.86%	27.64%	28.55%	
E	27.81%	32.97%	32.59%	35.59%	37.56%	



Asking with D

	N=100*	N=200	N=300	N=400	N=500	
non D	3432	2686	2264	1808	1564	= 100%
A	21.71%	22.23%	20.19%	17.64%	18.67%	
B	26.84%	24.72%	24.03%	25.77%	21.16%	
C	25.44%	25.58%	25.66%	26.05%	27.37%	
E	26.02%	27.48%	30.12%	30.53%	32.80%	



Asking with E

	N=100*	N=200*	N=300	N=400*	N=500	
non E	3754	3116	2617	2353	1921	= 100%
A	24.77%	20.31%	19.79%	19.46%	18.84%	
B	25.12%	25.48%	23.42%	22.90%	23.27%	
C	26.08%	28.34%	27.59%	29.20%	28.37%	
D	24.03%	25.87%	29.19%	28.35%	29.52%	

Fig. 1. Finding letter alignment in a sequence S_{ABCDE} in HRR, 10000 trials.

be t. Assumption (b) comes exactly from the same fact—a letter multiplied by one of the highest powers of t will be stored more "prominently" than other letters.

The clean-up memory $\mathcal{C}$ for this experiment consists of all five letters and the five powers of t. We will also use an auxiliary clean-up memory $\mathcal{L}$ containing letters only.

Since the normalization using the square root of the number of chunks proved very noisy in initial tests on statements containing powers of the trajectory vector, we decided to improve the HRR model. The HRR vectors in our tests were normalized by dividing them with their magnitude.

TABLE II
FINDING LETTER ALIGNMENT IN A SEQUENCE S_{ABCDE} IN GA$_c$, 10000 TRIALS.

$\mathbb{R}^5$	f_A	f_B	f_C	f_D	f_E
asking with A	71.60%	**8.22%**	6.44%	2.72%	6.47%
asking with B	8.98%	65.92%	9.16%	6.76%	**9.18%**
asking with C	7.28%	9.52%	67.25%	**9.67%**	6.28%
asking with D	8.74%	6.75%	8.80%	66.83%	**8.88%**
asking with E	8.03%	**9.96%**	6.83%	9.28%	65.90%

$A \prec B$, $^*B \prec E$, $C \prec D$, $D \prec E$, $^*E \prec B$ $\Rightarrow$ $A \prec B$, $C \prec D \prec E$

$\mathbb{R}^6$	f_A	f_B	f_C	f_D	f_E
asking with A	80.34%	**5.34%**	4.61%	5.08%	4.63%
asking with B	6.56%	73.91%	**7.48%**	4.67%	7.38%
asking with C	6.71%	7.47%	72.83%	**7.68%**	5.31%
asking with D	6.79%	5.37%	7.42%	72.30%	**8.12%**
asking with E	5.52%	7.77%	5.58%	**8.54%**	72.59%

$A \prec B$, $B \prec C$, $C \prec D$, $^*D \prec E$, $^*E \prec D$ $\Rightarrow$ $A \prec B \prec C \prec D$

$\mathbb{R}^7$	f_A	f_B	f_C	f_D	f_E
asking with A	89.78%	2.42%	**2.91%**	2.37%	2.52%
asking with B	3.78%	83.92%	4.04%	**4.44%**	3.82%
asking with C	4.30%	4.79%	80.54%	5.11%	**5.26%**
asking with D	4.35%	5.07%	5.41%	79.09%	**6.08%**
asking with E	4.57%	5.07%	**5.85%**	5.68%	78.83%

$A \prec C$, $B \prec D$, $^*C \prec E$, $D \prec E$, $^*E \prec C$ $\Rightarrow$ $A \prec C$, $B \prec D \prec E$

$\mathbb{R}^8$	f_A	f_B	f_C	f_D	f_E
asking with A	95.33%	0.88%	1.26%	1.20%	**1.30%**
asking with B	1.34%	92.27%	1.72%	**2.93%**	1.74%
asking with C	2.15%	2.28%	88.99%	2.35%	**4.23%**
asking with D	1.93%	**3.75%**	2.82%	88.19%	3.31%
asking with E	2.60%	2.36%	**5.09%**	3.32%	86.63%

$A \prec E$, $^*B \prec D$, $^*C \prec E$, $^*D \prec B$, $^*E \prec C$ $\Rightarrow$ $A \prec E$

The algorithm for finding out the position of each letter begins with asking a question

$$\begin{aligned} S_{?????} \sharp L_x &= \left\{ \begin{array}{ll} S_{?????} \circledast (L_x)^* & \text{in HRR} \\ (L_x)^+ S_{?????} & \text{in GA}_c \end{array} \right\} \\ &= (t^x)' \approx t^x \end{aligned} \tag{37}$$

for each letter $L_x \in \mathcal{L}$. Next, we need to find the item in the clean-up memory $\mathcal{C} \setminus \mathcal{L}$ that is most similar to $(t^x)'$. Let us denote this item by z. With high probability, z is the power of t associated with the position of the letter L_x in the sequence $S_{?????}$, although, if recognized incorrectly, z will most likely point to some other $t^{y>x}$. Now let us ask a second question

$$\begin{aligned} S_{?????} \sharp z &= \left\{ \begin{array}{ll} S_{?????} \circledast z^* & \text{in HRR} \\ \langle S_{?????}\ z^+ \rangle_1 & \text{in GA}_c \end{array} \right\} \\ &= L' \approx L_x. \end{aligned} \tag{38}$$

We use the projected product in GA$_c$ because we are looking for a letter vector placed on the position indicated by z. In HRR the resulting L' should be compared with letters only. In most cases L' will point to the correct letter. However, in a small fraction of test results, L' will point to letters surrounding L_x, because z has been mistakenly decoded as t^y for some $y \neq x$. Also, letters preceding L_x should come up less often than letters proceeding L_x.

Figure 1 presents test results for HRR. The data in Figure 1 should be interpreted as follows: the first row of each table next to a graph contains the vector lengths of the data used in 5 consecutive experiments (10000 trials each). The second row contains the number of faulty answers within those 10000 trials. The next 4 rows present the percentage of occurence of a "faulty" letter within all faulty answers presented in the second row.

Faulty alignments (i.e. those, for which the percentages corressponding to letters do no align increasingly within a single column) have been marked with a "*" in the table headings. We used S_{ABCDE} as the mysterious encoded sequence $S_{?????}$. In each case we crossed out the most frequently occurring letter and we concentrated on the frequency of the remaining letters. In HRR, for sufficiently large vector sizes, the frequencies f_L of all letters $L \in \mathcal{L}$ aligned correctly

$$f_B < f_C < f_D < f_E \qquad \text{asking with } A, \tag{39}$$

$$f_A < f_C < f_D < f_E \qquad \text{asking with } B, \tag{40}$$

$$f_A < f_B < f_D < f_E \qquad \text{asking with } C, \tag{41}$$

$$f_A < f_B < f_C < f_E \qquad \text{asking with } D, \tag{42}$$

$$f_A < f_B < f_C < f_D \qquad \text{asking with } E. \tag{43}$$

It was straightforward that these inequalities lead to $f_A < f_B < f_C < f_D < f_E$ and correctly identify the encoded sequence as S_{ABCDE}. Test results are less accurate when we asked about letters lying closer to the end of a sequence, therefore the size of the vector should be adequately long. Moreover, the longer the vector, the larger the difference between the frequencies.

GA$_c$ was expected to perform worse in this experiment, because we can construct powers of a mulivector t^{i-1} by

multiplying it with t from one side only. Indeed, at the first glance Table II shows that letter frequencies do not align correctly at all. We therefore needed to slightly modify the algorithm for finding letter alignment in GA_c: we concentrated on two largest frequencies in each series of asking questions—the largest frequency represents the letter L that was used to ask the question and the second largest frequency indicates letter $\hat{L}$ that most likely proceeds letter L.

Table II presents the frequencies of letters recognized as the most probable answer to Equation (38), the second largest frequency in each row is printed in bold. Partial letter alignments have been placed next to each table and contradictory alignments have been preceded with a "*". When being asked with the last letter of the sequence, HRR provided less accurate answers and so did GA_c by yielding more contradictions than in case of previous letters. It is impossible to avoid contradictory alignments in GA_c because we do not know which letter is the last one and the algorithm for recovering letter alignment in GA_c instructs us to write down the partial alignment with that letter being proceeded by some other letter. The remaining alignments point correctly to the sequence S_{ABCDE}

$$\left.\begin{array}{c} A \prec B \\ C \prec D \prec E \\ A \prec B \prec C \prec D \\ A \prec C \\ B \prec D \prec E \\ A \prec E \end{array}\right\} \Rightarrow A \prec B \prec C \prec D \prec E. \qquad (44)$$

VI. Conclusion

We have shown that multivector powers in GAc have properties similar to convolutive powers of HRR vectors

- (multi)vectors t^{i-r} and t^i are similar in much the same way as t^i and t^{i+r},
- items placed near the beginning of a sequence are remembered more prominently and thus, are recognized correctly more often,
- items placed near the end of a sequence are remembered less precisely and often come up as the most probable answer when the correct item is not recognized.

We have used the last two properties to find the alignment of sequence items without the explicit knowledge of (multi)vector powers. While HRR retrieved the original alignment without greater problems, GA_c left us with an easily soluble logical puzzle providing fragmentary alignments.

These properties can be used to build holographic lexicons, dictionaries and other structures that require storing order information and word meaning in the same pattern.

Acknowledgment

This work was supported by grant G.0405.08 of the Research Programme of the Research Foundation-Flanders (FWO, Belgium)

References

[1] D. Aerts and M. Czachor, "Quantum aspects of semantic analysis and symbolic artificial intelligence", *J. Phys. A*, vol. 37, pp. L123-L13, 2004.
[2] D. Aerts and M. Czachor, "Cartoon computation: Quantum-like algorithms without quantum mechanics", *J. Phys. A*, vol. 40, pp. F259-F266, 2007.
[3] M. Czachor, "Elementary gates for cartoon computation", *J. Phys. A*, vol. 40, pp. F753-F759, 2007.
[4] D. Aerts and M. Czachor, "Tensor-product versus geometric-product coding", *Physical Review A*, vol. 77, id. 012316, 2008.
[5] D. Aerts, M. Czachor, and B. De Moor, "Geometric Analogue of Holographic Reduced Representation", *J. Math. Psychology*, vol. 53, pp. 389-398, 2009.
[6] D. Aerts, M. Czachor, and B. De Moor, "On geometric-algebra representation of binary spatter codes". preprint arXiv:cs/0610075 [cs.AI], 2006.
[7] D. Aerts, M. Czachor, and Ł. Orłowski, "Teleportation of geometric structures in 3D ", *J. Phys. A* vol. 42, 135307, 2009.
[8] W.K. Clifford, "Applications of Grassmann's extensive algebra", *American Journal of Mathematics Pure and Applied*, vol. 1, 350–358, 1878.
[9] R. W. Gayler, "Multiplicative binding, representation operators, and analogy", *Advances in Analogy Research: Integration of Theory and Data from the Cognitive, Computational, and Neural Sciences*, K. Holoyak, D. Gentner, and B. Kokinov, eds., Sofia, Bulgaria: New Bulgarian University, p. 405, 1998.
[10] S. Deerwester et al. "Indexing by Latent Semantic Analysis", *Journal of American Society for Information Science*, vol. 41, 391, 1990.
[11] H. Grassmann, "Der Ort der Hamilton'schen Quaternionen in der Ausdehnungslehre", *Mathematische Annalen*, vol. 3, 375–386, 1877.
[12] M.N. Jones & D.J.K. Mewhort, "Representing Word Meaning and Order Information in a Composite Holographic Lexicon", *Psychological Review*, vol. 114, No. 1, pp. 1-37, 2007.
[13] P. Kanerva, "Binary spatter codes of ordered k-tuples". In C. von der Malsburg et al. (Eds.), *Artificial Neural Networks ICANN Proceedings, Lecture Notes in Computer Science* vol. 1112, pp. 869-873, 1996.
[14] P. Kanerva, "Fully distributed representation". *Proc. 1997 Real World Computing Symposium (RWC97, Tokyo)*, pp. 358-365, 1997.
[15] N.G. Marchuk, and D.S. Shirokov, "Unitary spaces on Clifford algebras", *Advances in Applied Clifford Algebras*, vol 18, pp. 237-254, 2008.
[16] A. Patyk, "Geometric Algebra Model of Distributed Representations", in *Geometric Algebra Computing in Engineering and Computer Science*, E. Bayro-Corrochano and G. Scheuermann, eds. Berlin: Springer, 2010. Preprint arXiv:1003.5899v1 [cs.AI].
[17] T. Plate, "Holographic Reduced Representations", *IEEE Trans. Neural Networks*, vol. 6, no. 3, pp. 623-641, 1995.
[18] T. Plate, *Holographic Reduced Representation: Distributed Representation for Cognitive Structures*. CSLI Publications, Stanford, 2003.
[19] P. Smolensky, "Tensor product variable binding and the representation of symbolic structures in connectionist