Simplifying and Generalising Equivariant Geometric Algebra Networks

Christian Hockey, Yuxin Yao, Joan Lasenby
Department of Engineering, University of Cambridge

March 10, 2024

Abstract

In this paper we present and explain geometric algebra equivariant neural network mappings, and use these to construct a generalised equivariant geometric algebra transformer very similar to that in [4]. We then present tests conducted on an n-body dynamics problem and a protein structure prediction problem, using a conformal geometric algebra (CGA) implementation of the transformer blocks.

1 Introduction

Geometric algebra is an algebra defined for a vector space which allows invertible multiplications of elements of that vector space by unifying the concepts of scalars, vectors, and higher dimensional directed objects under one algebra using the geometric product. Recently, several papers [3] [4] [10] [11] [14] have indicated the usefulness of embedding geometric algebra objects, known as multivectors, into neural networks designed for solving geometric problems. A prominent current area of research is in Clifford group equivariant neural networks [10], and the goal of this paper is to simplify the literature surrounding the subject, as well as to generalise these neural networks to arbitrary geometric algebras. This paper will also demonstrate the value in using conformal geometric algebra in the solving of certain geometric problems using a custom transformer architecture similar to that proposed in [4].

2 Background

2.1 Geometric Algebra Equivariant Neural Networks

Several recent papers [3] [11] [14] have demonstrated that embedding multivectors into the structure of traditional neural network layers and correspondingly updating their mappings to reflect the geometric significance of their inputs can yield far better results on geometric problems than less intrinsically geometric methods. Much of the recent research into GANNs [4] [10] is centered around equivariant layer mappings being used in geometric algebra neural networks in order to maintain equivariance with respect to Euclidean transformations in 3D. A function f is equivariant with respect to the transformation ρ if the order in which the function and the transformation are applied does not matter. This is stated mathematically in equation 1.

$$f(\rho(x)) = \rho(f(x)) \tag{1}$$

If a network is able to preserve this property with respect to Euclidean transformations, then it gains the attractive property (for geometric problems) that any Euclidean transformation applied to its inputs are also applied to its outputs automatically - this is a very powerful inductive bias. If we then take a GA neural network and ensure that its mappings are equivariant with respect to the transformation $\rho(x) = Rx\tilde{R}$ (this is known as the sandwich product), where R is a versor in the

chosen algebra, then our network becomes equivariant with respect to any orthogonal transformation in the algebra. For conformal geometric algebra (CGA, $\mathcal{G}(4,1,0)$) and plane-based geometric algebra (PGA, $\mathcal{G}(3,0,1)$), this results in a network which is equivariant with respect to all Euclidean 3D transformations. "Clifford Group Equivariant" networks were investigated in general in [10], and many of the layer mappings used in this paper are drawn from here. This work will expand on and clarify the expressions underlying these mappings, in order to be as accessible as possible for applied researchers.

2.2 Geometric Algebra Transformer

Equivariant networks have also been applied to the popular transformer [13] architecture in [4]. This showed good results in application to several simple geometric problems, however it was implemented to be equivariant in PGA, meaning many of its layer operations are adapted to a basis vector which squares to 0, meaning they are more complicated than the mappings would need to be in cases where the basis vectors of the algebra square to only $+1$ and -1. Additionally, although creating a neural network which is equivariant in PGA allows equivariance to Euclidean transformations, an implementation in CGA would gain the advantage of equivariance with respect to dilations in 3D space, presenting a potential additional advantage of implementing an equivariant transformer network for general GA signatures (without basis vectors that square to zero). This paper therefore will present a generalised GA transformer architecture, with several notable differences from the architecture proposed in [4], and will explain the design choices made.

3 Implementation

This section will go through the methods used to implement GA equivariant neural networks for testing, as well as the architecture used for a generalised equivariant geometric algebra transformer, which was tested using conformal geometric algebra.

3.1 Use of TFGA

In all implementations of GA equivariant networks that we describe here, the TFGA package in TensorFlow [8] was used. This allowed for layers to be implemented using any geometric algebra, with any signature, due to abstraction of GA operations (such as the geometric product); this is in contrast to the implementations in [4], for which PGA ($\mathcal{G}(3,0,1)$) was enforced. However, use of this package meant that layers were not serialisable (meaning that saving trained models was less efficient), and the abstraction of operations prevented fast, hard-coded layer actions; this introduced a significant training time overhead compared to non-GA models, and in practical applications a more optimised package could be developed for optimal performance.

3.2 Generalised GA Transformer

3.2.1 Full Architecture

The design of a single generalised GA equivariant transformer block is shown in figure 1. This is very similar to that found in [4], with the most notable difference being the geometric product layer preceding the multi-head dot-product attention layer, as opposed to a standard linear layer. It was found that this layer resulted in better performance on the n-body dynamics problem than the linear layer, likely due to greater grade-mixing before the attention layer resulting in the dot-product attention having more effect. Additionally, a sigmoid gated non-linearity was found to perform better than a GELU as was used in [4], allowing for far better fitting of problems with fewer transformer blocks. Finally, a simple inner product attention mechanism was used in lieu of the more complex 'distance-aware' dot-product attention proposed in [4]; this was chosen as in other algebras, such as conformal geometric algebra ($\mathcal{G}(4,1,0)$), the scalar part of the inner product between queries and keys could straightforwardly be used without omission of information. The rest

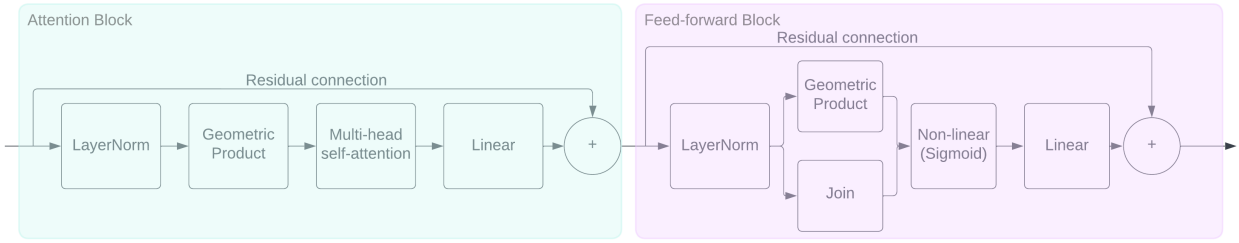

Figure 1: Transformer block architecture

of this section will go through each of the layers used in more detail and explain the logic behind each of the decisions made. It should be noted that although this architecture accepts any GA signature, some of the operations involved (such as the dual) would need to be altered in the case of a basis vector squaring to 0 to ensure proper functioning of the network.

3.2.2 Attention Block

The first part of the attention block is a layer norm; this has been implemented similarly to that in [4], and effectively divides by an average (computed over the layer dimension) of the invariant $\sqrt{|\langle x\tilde{x}\rangle_0|}$ (where x is a multivector, $\tilde{x}$ denotes the reversion of x, and $\langle\cdot\rangle_r$ denotes the grade-r projection of a multivector). The absolute value is taken here to guard against potential negative norms in some algebras (such as conformal geometric algebra) and the average is taken in order to avoid significant instability issues which occur if a more specific norm is used for each input multivector. A geometric product layer is then used. This is given by the geometric product of two linear layers; the equivariant linear layer mapping (from input multivectors x to output multivectors y) is shown in equation 2, and the subsequent geometric product mapping is shown in equation 3.

$$y_j = \sum_{i,r} a_{ijr}\langle x_i\rangle_r \tag{2}$$

$$y_i = \text{Linear}_1(x)_i \text{Linear}_2(x)_i \tag{3}$$

A multi-head equivariant self-attention mechanism is then used. This first projects its inputs into different query (q_i), key (k_i), and value (v_i) multivectors for each head, then uses the attention mechanism in equation 4 for each head (here, $\cdot$ denotes the inner product, and n_b the number of blades in the algebra). This mechanism is essentially a GA version of that proposed in [13], and is very similar to the attention mechanism used in [4]. A linear layer is then used to combine the outputs from the heads, completing the attention layer. Finally, a linear layer is used at the output of the attention block to project back to the input dimension so that the residual connection can be used.

$$y_i = \text{Softmax}\left(\frac{\langle q_i \cdot k_i\rangle_0}{\sqrt{n_b}}\right) v_i \tag{4}$$

3.2.3 Feed-forward Block

The first layer in the feed-forward block is a layer norm, as in the attention block, for stability in training when multiple transformer blocks are added. This is followed by a geometric product layer (as described in the previous section) concatenated with a join layer, which implements the operation given in equation 5, where I is the pseudoscalar of the algebra, given by the geometric product of all basis vectors in the algebra (post-multiplying by I^{-1} is also known as the dual, and multiplying by I is known as the anti-dual). This operation is implemented due to its usefulness in PGA [6] (but note there that the dual should be described differently), however its usefulness

in other algebras such as CGA remains to be investigated (preliminary results indicate it made no difference in CGA).

$$y_i = [(\text{Linear}_1(x)_i I^{-1}) \wedge (\text{Linear}_2(x)_i I^{-1})]I \tag{5}$$

The outputs from this are then put through a non-linear layer, implemented by applying a gated non-linearity, ϕ, to the multivectors, as in [10] and [4]. This layer's operation is given by the equation in 6, where h is some scalar invariant under the versor sandwich product of the input multivector x. For the transformer, the invariant chosen was the scalar part of the multivector (i.e. $h(x) = \langle x \rangle_0$) - this was chosen due to its stability over some expression of the form $x\tilde{x}$, which would have resulted in highly unstable chained transformer blocks. A sigmoid non-linearity was chosen over any others, as in testing this provided far superior performance and scaling with additional blocks than a GELU non-linearity, which was adopted in [4].

$$y_i = \phi(h(x_i))x_i \tag{6}$$

Finally, the feed-forward block ends with a linear layer to project back to the number of input multivectors so that the residuals can be added.

4 N-body Dynamics Problem

4.1 Problem set-up

The first problem on which the CGA was tested was an n-body dynamics problem which was identical to that in [4], for comparison purposes. The model was given as inputs the masses, initial velocities and initial positions of 4 bodies under gravity and needed to predict their positions 0.1 seconds ahead of the input positions. An Euler simulator was used to generate 10000 samples, simulating 100 time steps of length 10^{-3} seconds for each sample. CGA ($\mathcal{G}(4,1,0)$) was used for testing the generalised transformer, and an MLP was used for benchmarking and comparing results.

4.2 Embedding into CGA

In order to use a CGA in the processing of multivectors, the input information first needed to be embedded into CGA from 3D. The masses were embedded as scalar parts of the multivector inputs. The points were then embedded using the scheme described in [7] (used for rigid body motion), and the vectors were similarly transformed, before taking their dual. These transformations are described below in equations 7 and 8 where capital letters denote multivectors in CGA and lower-case letters denote 3D vectors with bases e_1, e_2, and e_3. We also define $n = e_+ + e_-$ and $\overline{n} = e_+ - e_-$, where $e_+^2 = 1$ and $e_-^2 = -1$ are the remaining two basis vectors in CGA (note that changing to the more commonly used mapping where $n_\infty = e_+ + e_-$ and $n_0 = \frac{1}{2}(e_- - e_+)$ would make no difference to any outcomes).

$$X = x + \frac{1}{2}x^2 n - \frac{1}{2}\overline{n} \tag{7}$$

$$V = (v + (x \cdot v)n)I^{-1} \tag{8}$$

The input multivector into the network is then given by $Z = m + X + V$ where m is the mass scalar. The dual of the velocity is taken in order to make use of the additional dimensions of the input multivector and avoid adding the points and velocity vectors into the same bases in the input, avoiding any ambiguity in the input and improving the effectiveness of the attention mechanism.

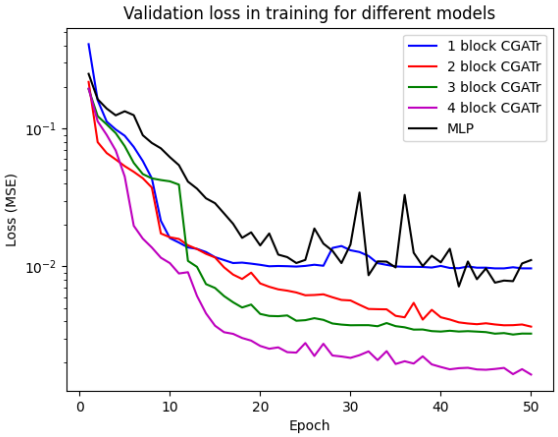

Figure 2: Validation loss for varying numbers of transformer blocks

4.3 Results

The scaling of the model with additional transformer blocks is compared with an MLP model in figure 2, with the corresponding number of parameters for each model and best validation loss shown in table 1. From these, we can see that the CGA transformer captures the attractive properties of an equivariant geometric algebra transformer: it scales very well with increasing numbers of blocks, and greatly outperforms a non-geometric architecture with a similar number of parameters on a geometric problem. It also trains in fewer epochs, with the 4-block transformer being a clear example of the fast generalisation which these networks can achieve. It should be noted that these networks still perform worse than those in [4], however this is likely due to our comparative computational restrictions (our networks have parameter budgets of the order of 5000, whereas the GA transformer in [4] had 1.9 million parameters).

Model	**No. parameters**	**Validation loss (MSE)**
MLP	5587	0.0111
CGA 1 block	1496	0.0097
CGA 2 block	2892	0.0037
CGA 3 block	4288	0.0033
CGA 4 block	5684	0.0016

Table 1: Results of n-body experiment

5 Protein Structure Prediction

Model	**No. Parameters**	**Test MAE**	**Test SSIM**	**Training MAE**	**Validation MAE**
MLP	107394	5.780	0.824	6.368	6.911
CGA 1 block	108753	5.700	0.821	6.101	6.995
CGA 2 block	110149	5.699	0.833	6.070	6.968

Table 2: Results of protein structure prediction experiment.

In [9], the problem of predicting protein structure (specifically amino acid inter-residue distances) using information about corresponding amino acid sequences was tackled using GA equivariant neural network layers. This was done using the PDNet dataset [1] and using a graph transformer [5]

[2] [12] to process a graph input consisting of node features (features of individual amino acids) and edges (features associated with pairs of amino acids) before feeding the output nodes from the graph transformer into an equivariant 3D GA ($\mathcal{G}(3,0,0)$) feed-forward network. For this investigation, a CGA transformer with a varying number of transformer blocks was used as a GA feed-forward network after the graph transformer, in order to investigate the capabilities of this architecture with a non-equivariant backbone.
The results in table 2 show the performance of the projector in the production of inter-residue distance maps. These show that the network scales reasonably well on the test set with increasing numbers of transformer blocks in the GA projector, providing a starting point for further investigation into the performance of the GA transformer on this problem. In a future work, the amino acid coordinates will be directly predicted using these layers; we expect that this will highlight the strength of a GA network over other architectures due to the more fundamentally geometric nature of the problem.

6 Conclusion

In this paper, we have explained the logic underlying GA equivariant neural networks and shown the generalised mappings which can be used to construct them. We also proposed a generalised equivariant geometric algebra transformer architecture, similar to that proposed in [4] but with several key differences and simplifications, and used a CGA implementation of the network to demonstrate its functioning on n-body dynamics and protein structure prediction problems. We trained the models with limited computational resources, demonstrating their effectiveness even when kept relatively lightweight in terms of parameter budget.

References

[1] Badri Adhikari. "A fully open-source framework for deep learning protein real-valued distances". In: *Scientific reports* 10.1 (2020), p. 13374.

[2] Minkyung Baek et al. "Accurate prediction of protein structures and interactions using a 3-track network". In: *bioRxiv* (2021). DOI: 10.1101/2021.06.14.448402. eprint: https://www.biorxiv.org/content/early/2021/06/15/2021.06.14.448402.full.pdf. URL: https://www.biorxiv.org/content/early/2021/06/15/2021.06.14.448402.

[3] Johannes Brandstetter et al. "Clifford neural layers for PDE modeling". In: *arXiv preprint arXiv:2209.04934* (2022).

[4] Johann Brehmer et al. "Geometric Algebra Transformers". In: *arXiv preprint arXiv:2305.18415* (2023).

[5] Allan Costa et al. "Distillation of MSA Embeddings to Folded Protein Structures with Graph Transformers". In: *bioRxiv* (2021). DOI: 10.1101/2021.06.02.446809. eprint: https://www.biorxiv.org/content/early/2021/06/02/2021.06.02.446809.full.pdf. URL: https://www.biorxiv.org/content/early/2021/06/02/2021.06.02.446809.

[6] Pim De Haan, Taco Cohen, and Johann Brehmer. "Euclidean, Projective, Conformal: Choosing a Geometric Algebra for Equivariant Transformers". In: *arXiv preprint arXiv:2311.04744* (2023).

[7] Leo Dorst and Joan Lasenby. *Guide to geometric algebra in practice*. Springer Science & Business Media, 2011.

[8] Robin Kahlow. *TensorFlow Geometric Algebra*. DOI: 10.5281/zenodo.3902404. URL: https://doi.org/10.5281/zenodo.3902404.

[9] Alberto Pepe, Sven Buchholz, and Joan Lasenby. "Clifford Group Equivariant Neural Network Layers for Protein Structure Prediction". In: *Northern Lights Deep Learning Conference*. PMLR. 2024, pp. 205–211.

[10] David Ruhe, Johannes Brandstetter, and Patrick Forré. "Clifford group equivariant neural networks". In: *arXiv preprint arXiv:2305.11141* (2023).

[11] David Ruhe et al. "Geometric clifford algebra networks". In: *arXiv preprint arXiv:2302.06594* (2023).

[12] Yunsheng Shi et al. "Masked label prediction: Unified message passing model for semi-supervised classification". In: *arXiv preprint arXiv:2009.03509* (2020).

[13] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[14] Maksim Zhdanov et al. "Clifford-Steerable Convolutional Neural Networks". In: *arXiv preprint arXiv:2402.14730* (2024).