GEOMETRIC TRANSFORMERS

Daniele Affinita (1885790), Flavio Maiorana (2051396)
Sapienza, University of Rome
{affinita.1885790, maiorana.2051396}@studenti.uniroma1.it

1 Introduction

Geometric Deep Learning refers to a class of neural network architectures that incorporate inductive biases into the model. Many objects present geometric structures that can be leveraged by deep learning models. In this project, we implemented a deep learning pipeline based on the paper Geometric Algebra Transformer [1] to perform binary classification on arteries, following the general Geometric Deep Learning methodology.

The first step was to identify the geometric structure that best describes the elements of the dataset. Based on the paper, we used the projective geometric (Clifford) algebra $\mathbb{G}_{3,0,1}$, which is capable of representing both geometric objects and their transformations as multivectors. We then encoded all dataset elements into an algebraic structure suitable for processing by neural network layers.

Subsequently, we implemented equivariant model layers, ensuring the entire architecture is equivariant with respect to the symmetry group of three-dimensional space, $E(3)$. Finally, we demonstrated this equivariance.

1.1 Dataset

We tackled a binary classification task on arterial mesh and simulated wall shear stress data published by Suk et al. [2]. The dataset includes two classes of arteries: single and bifurcating. Each artery consists of a set of multiple meshes, each described by five features: **wws**, which represents wall shear stress, the force per unit area exerted by the wall on the fluid in a direction parallel to the local tangent plane [3]; **pos**, indicating the mesh position; **face**, the 3D plane representing the mesh; **pressure**, the blood pressure exerted on the mesh; and **inlet_idcs**, the index of the artery inlet, which is mainly used for visualization purposes and not included as a feature since it does not carry additional information.

The dataset contains 4000 samples, split into 75% for training, 15% for validation, and 10% for testing. Each sample consists of a collection of mesh data points representing detailed features for a specific artery.

The dataset is balanced, with an equal number of samples for each class (2000 samples per class), meaning 50% of the samples represent single arteries and 50% represent bifurcating arteries.

Since each artery is characterized by a different number of meshes, and we needed to comply with the constraints of Google Colab, we decided to downsample the number of meshes for each artery to a fixed size. After conducting experiments, we found that using 300 meshes provided the best balance between model accuracy and storage requirements.

For exploratory data analysis, we projected each feature into 2D space using Principal Component Analysis (PCA) to see if any clear patterns emerged. The plots shown in Figure 1 reveal that the data is almost linearly separable, especially in the Wall Shear Stress feature. This finding led us to anticipate very high accuracy from a deep-learning model. The clear pattern observed further justifies the mesh downsampling, as the original artery shape can still be inferred without needing the complete set of meshes.

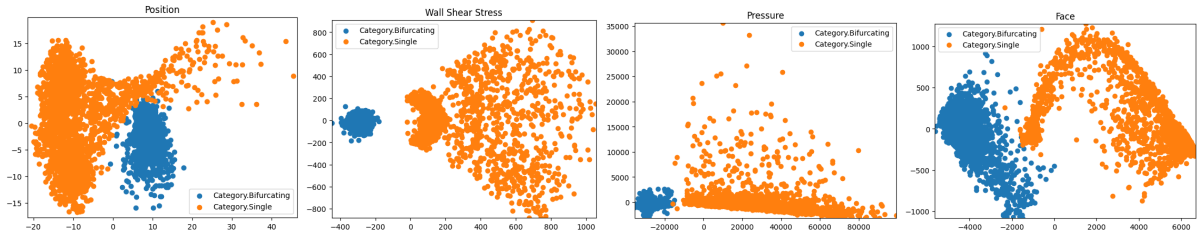

Figure 1: PCA projection of features

1.2 Notes on the code

You can find the project's code in the following repository: `https://github.com/DaniAffCH/Vessel-Geometric-Transformers`. The code adheres to the PEP-8 standard (validated with black and flake8 checks) and is fully type-checked (validated with mypy). It can be set up either locally or run on Google Colab. For the Colab setup, simply open the `main.ipynb` notebook and click the 'Open in Colab' button. The notebook will automatically clone the repository and set up the project, after which you only need to execute the cells. Notice that the notebook should be seen just as an entry point for the project. All the backend logic is distributed across different files to maintain a clean and organized codebase. It is possible to configure the project from `config.yaml`, for instance, you may want to setup your wandb account.

2 Methodology

2.1 Data encoding

Considering $\mathbb{G}_{3,0,1}$ geometric algebra, each multivector representing both objects and operations can be generated by four orthogonal basis e_0, e_1, e_2, e_3. As a result, a multivector is expressed as:

$$x = x_s + \sum_i x_i e_i + \sum_{i<j} x_{ij} e_{ij} + \sum_{i<j<k} x_{ijk} e_{ijk} + x_{0123} e_{0123}$$

Thus, we decided to store just the set of coefficients $(x_s, x_0, ..., x_{0123})$. By employing 4 basis we have 2^4 possible combinations, so 16 coefficients encoding a multivector. Since we identified the algebraic structure, the remaining problem is to embed geometric elements and operations into it. To achieve this, we relied on an embedding dictionary depicted in Table 1 provided by the original paper, which has been proven to retain all the object's symmetries.

We identified the most likely geometric object for each feature and applied the corresponding embedding vector. In particular, **pos** was associated with a Point, **pressure** represents a Scalar value, **plane** was encoded as a Plane, and finally, since **wss** represents a force over an area, the closest geometric object describing the situation was a translation. Although the latter is not completely physically accurate, it aligns with the intuition that wss represents a deformation of the area.

Object / operator	Scalar	Vector		Bivector		Trivector		PS
	1	e_0	e_i	e_{0i}	e_{ij}	e_{0ij}	e_{123}	e_{0123}
Scalar $\lambda \in \mathbb{R}$	λ	0	0	0	0	0	0	0
Plane w/ normal $n \in \mathbb{R}^3$, origin shift $d \in \mathbb{R}$	0	d	n	0	0	0	0	0
Line w/ direction $n \in \mathbb{R}^3$, orthogonal shift $s \in \mathbb{R}^3$	0	0	0	s	n	0	0	0
Point $p \in \mathbb{R}^3$	0	0	0	0	0	p	1	0
Pseudoscalar $\mu \in \mathbb{R}$	0	0	0	0	0	0	0	μ
Reflection through plane w/ normal $n \in \mathbb{R}^3$, origin shift $d \in \mathbb{R}$	0	d	n	0	0	0	0	0
Translation $t \in \mathbb{R}^3$	1	0	0	$\frac{1}{2}t$	0	0	0	0
Rotation expressed as quaternion $q \in \mathbb{R}^4$	q_0	0	0	0	q_i	0	0	0
Point reflection through $p \in \mathbb{R}^3$	0	0	0	0	0	p	1	0

Table 1: Geometric objects and transformations $\mathbb{G}_{3,0,1}$ embedding dictionary

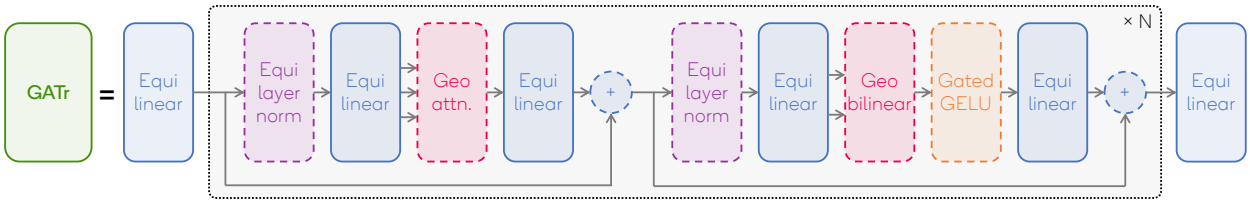

Figure 2: Overall GATr architecture.

2.2 GATr Layers

To tackle the binary classification task, we implemented the GATr architecture as shown in Figure 2. The only difference is that we placed an additional Linear layer at the end, to project the features extracted from the GATr backbone, into a probability.

2.2.1 Equi Linear Layer

The equi-linear layer is the main block of the GATr architecture. It represents a learnable linear map $\phi : \mathbb{G}_{3,0,1} \to \mathbb{G}_{3,0,1}$. In order to implement it we need the *blade* $\langle x \rangle_k$ multivector primitive that sets all the all the non-grade-k elements to 0. The equi-linear layer is defined as follows:

$$\phi(x) = \sum_{k=0}^{d+1} w_k \langle x \rangle_k + \sum_{k=0}^{d} v_k e_0 \langle x \rangle_k$$

In order to speed up the computations we decided to bring this computation in tensor form, by defining a *Equi-linear Basis Tensor* B. This tensor consists of a collection of diagonal matrices. However, these diagonal matrices are not fully populated; instead, their diagonal elements are determined by the grade of the corresponding basis elements. For the operation involving $e_0 \langle x \rangle_k$, we computed the corresponding basis vector and populated the matrices according to the grade of the elements. Specifically, considering the multivector property $e_0 \cdot e_0 = 0$, the basis vector, divided by grade, is represented as:

$$(e_0 \mid 0\, e_0 e_1\, e_0 e_2\, e_0 e_3 \mid 0\,0\,0\, e_{012}\, e_{013}\, e_{023} \mid 0\,0\,0\, e_{123} \mid 0)$$

Given this, the learnable parameters consist of 9 elements in total: 5 w_k, and 4 v_k. This approach applies to a single multivector, however, it can be generalized to a batch of multivectors by altering the number of features from a shape of $[\text{batch}, \text{in\_features}, 16]$ to $[\text{batch}, \text{out\_features}, 16]$. The corresponding vectorized Equi-Linear Layer, which is fully equivalent to the previous formulation and was used in our project, is given by:

$$y_{o,g} = \sum_{i,b} \sum_{g'} W_{o,i,b} \cdot B_{b,g,g'} \cdot x_{i,g'}$$

2.2.2 BiLinear Layer

The main limitation of the Equi Linear Layer is that it allows only minimal grade mixing. To address this and enhance the model's expressiveness, a BiLinear Layer is introduced, which captures additional relationships between multivectors, such as vector translation. To implement this layer, two Geometric Algebra primitives are required: the geometric product and Equi-Join. To implement the geometric product, denoted as xy, in vectorial form, we used the precomputed basis B from the original paper repository [4].

The Join operation is not equivariant to reflections, so an additional component was added. The Equi-Join uses a reference multivector z, which ensures the function changes sign correctly under input mirroring.

$$\text{EquiJoin}(x, y; z) = z_{0123}(x^* \wedge y^*)^*$$

where x^* represents the dualization operator, and $x \wedge y$ is the outer product.

Since the original paper suggests retrieving the reference z from the original data rather than hidden representations, we computed it as the average of input multivectors for each batch.

The final BiLinear layer is obtained by concatenating the results of the geometric product and of the EquiJoin. This layer is crucial for enhancing the model's expressiveness; however, its basic definition does not contain any learnable parameters. To address this, we modified the layer proposed by the paper by adding an Equi Linear transformation to the input multivectors, with a different transformation for each operator, and adding a final projection. The modified version of the layer becomes:

$$\phi_f(\text{Concatenate}_{\text{channels}}(\phi_1(x)\phi_2(y), \text{EquiJoin}(\phi_3(x), \phi_4(y); z)))$$

2.2.3 Geometric Attention Layer

The Geometric Attention layer modifies the original attention mechanism [5] by replacing the dot product with the invariant inner product and summing the attention scores over the channel dimension. Having three multivectors: q, k, and v the Geometric Attention is defined as:

$$\text{Attention}(q, k, v)_{i'c'} = \sum_i \text{Softmax}_i \left(\frac{\sum_c \langle q_{i'c}, k_{ic} \rangle}{\sqrt{8n_c}} \right) v_{ic'} \, .$$

We generalized the proposed attention mechanism to its multi-head counterpart by concatenating the outputs of multiple heads. However, this version of Geometric Attention is not fully equivariant. The paper's author introduced a variant, called Distance-aware dot-product attention, in the paper notes to address this.

2.2.4 Equi Norm Layer & Gated GELU Activation

The Batch Normalization layer is extended by normalizing the multivectors according to the expected value of the self-inner product: $x/\sqrt{\mathbb{E}\langle x, x \rangle}$. Since the outer product is an equivariant operator that returns a scalar, the entire normalization process remains equivariant. Finally, an equivariant activation function is introduced: the Gated GELU, which is obtained by applying the GELU only to the scalar part of the multivector and then multiplying the result by the original multivector, $\text{GELU}(x_1)x$.

3 Results

3.1 Equivariance check

The original paper claims that each introduced layers are equivariant to the $E(3)$ symmetry group. The equivariance of a layer f with respect to a group action, denoted by $\rho(x)$, can be formalized by the following equation:

$$f(\rho(x)) = \rho(f(x))$$

This means that applying a transformation ρ to the input x and then passing it through the layer f yields the same result as first applying f to x and then applying the transformation ρ to the output.

To verify the equivariance of the proposed layers, we leveraged a property of orthogonal transformations: they can be expressed as sequences of reflections. Given this, it's possible to verify the equivariance of any group action by verifying the equivariance with respect to reflections. Specifically, we compute a versor u as a sequence of reflections, $u = u_1 \cdots u_n$. If u is the product of an even number of reflections, it belongs to the Pin group. Otherwise, it belongs to the Spin group.

To apply a versor u to an element x, we use the sandwich product:

$$\rho_u(x) = \begin{cases} uxu^{-1} & \text{if } u \in \text{Pin} \\ u\hat{x}u^{-1} & \text{if } u \in \text{Spin} \end{cases}$$

where $\hat{x}$ denotes the grade involution operator and u^{-1} is the shirokov inverse of a multivector.

We verified the equivariance of the layers by generating a large number of random reflections and testing the equivariance equation described in 3.1. In our experiments, we generated 100 random versors u, each composed of a random number of reflections, and verified the equation for each generated u. Since exact equality between the tensors is not feasible

due to floating-point precision, we set a tolerance of 10^{-5}. We repeated this experiment for 10 random multivectors from the dataset, performing a total of 10×100 checks for each layer.

The results show that all layers passed the equivariance test except the Attention layer. This is most likely due to an extension proposed in the paper, which suggests using a distance-aware dot product to make the attention layer fully equivariant.

3.2 Training

We performed a comparison between three different models: a multilayer perceptron, a vanilla transformer and the geometric transformer. We performed hyperparameter optimization using Optuna. More specifically, we optimized the learning rate and the batch size, besides the number of attention heads and attention layers for the two transformers, and the hidden size for the MLP. Given this binary classification task is relatively easy due to the almost linear separability of the data, none of the models had notable difficulties during training.

3.3 Results comparison

Model	Accuracy	F1-score	Loss	Parameters	Training Time [min]
MLP	0.9925	0.9925	1.17e-2	307K	15
Transformer	0.99	0.99	2.24e-2	24K	16
GATR	**1.0**	**1.0**	**1.73e-6**	**153K**	22

Table 2: Comparison Across Models on Test Set with random seed `0xDEADBEEF`

All models solve the task with ease, although one notable observation is that the validation loss, thus also the validation accuracy, has a much better trend with the geometric transformer during training, demonstrating the fact that it is able to better interpret the input data by exploiting the inductive bias coming from the geometric algebra principles.

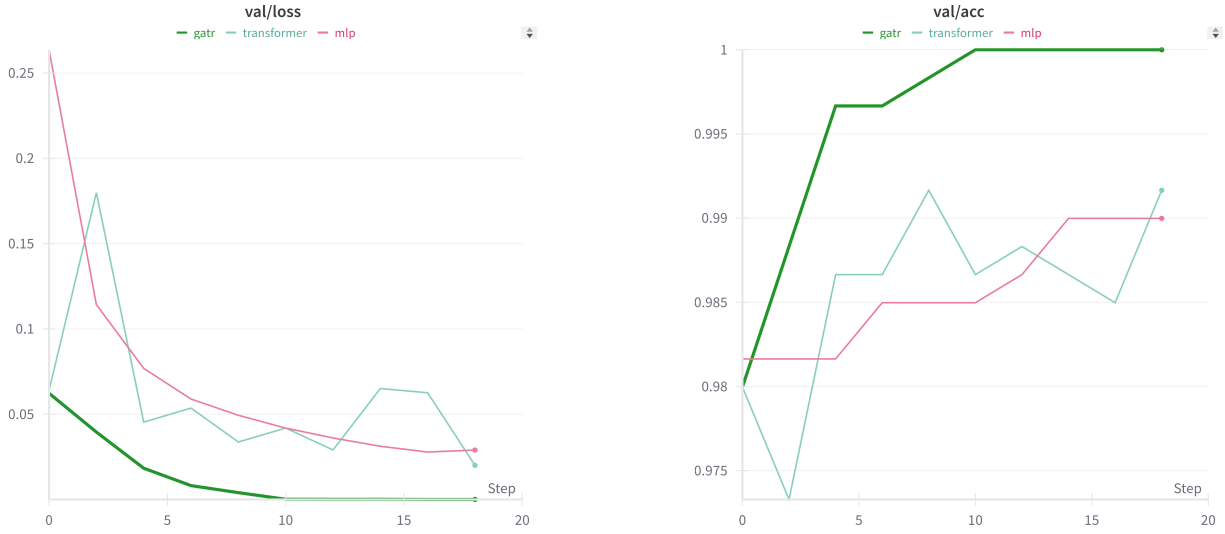

Figure 3: Trend for validation loss and accuracy

References

[1] Johann Brehmer, Pim De Haan, Sönke Behrends, and Taco S Cohen. Geometric algebra transformer. *Advances in Neural Information Processing Systems*, 36, 2024.

[2] Julian Suk, Pim de Haan, Phillip Lippe, Christoph Brune, and Jelmer M Wolterink. Mesh neural networks for se (3)-equivariant hemodynamics estimation on the artery wall. *Computers in Biology and Medicine*, 173:108328, 2024.

[3] Demosthenes Katritsis, Lambros Kaiktsis, Andreas Chaniotis, John Pantos, Efstathios P Efstathopoulos, and Vasilios Marmarelis. Wall shear stress: theoretical considerations and methods of measurement. *Progress in cardiovascular diseases*, 49(5):307–329, 2007.

[4] Qualcomm AI Research. Precomputed geometric product basis. `https://github.com/Qualcomm-AI-research/geometric-algebra-transformer/blob/main/gatr/primitives/data/geometric_product.pt`, 2024.

[5] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.