SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks

Fabian B. Fuchs[*][†]
Bosch Center for Artificial Intelligence
A2I Lab, Oxford University
`fabian@robots.ox.ac.uk`

Daniel E. Worrall[*]
Amsterdam Machine Learning Lab, Philips Lab
University of Amsterdam
`d.e.worrall@uva.nl`

Volker Fischer
Bosch Center for Artificial Intelligence
`volker.fischer@de.bosch.com`

Max Welling
Amsterdam Machine Learning Lab
University of Amsterdam
`m.welling@uva.nl`

Abstract

We introduce the SE(3)-Transformer, a variant of the self-attention module for 3D point clouds and graphs, which is *equivariant* under continuous 3D roto-translations. Equivariance is important to ensure stable and predictable performance in the presence of nuisance transformations of the data input. A positive corollary of equivariance is increased weight-tying within the model. The SE(3)-Transformer leverages the benefits of self-attention to operate on large point clouds and graphs with varying number of points, while guaranteeing SE(3)-equivariance for robustness. We evaluate our model on a toy N-body particle simulation dataset, showcasing the robustness of the predictions under rotations of the input. We further achieve competitive performance on two real-world datasets, ScanObjectNN and QM9. In all cases, our model outperforms a strong, non-equivariant attention baseline and an equivariant model without attention.

1 Introduction

Self-attention mechanisms [31] have enjoyed a sharp rise in popularity in recent years. Their relative implementational simplicity coupled with high efficacy on a wide range of tasks such as language modeling [31], image recognition [18], or graph-based problems [32], make them an attractive component to use. However, their generality of application means that for specific tasks, knowledge of existing underlying structure is unused. In this paper, we propose the *SE(3)-Transformer* shown in Fig. 1, a self-attention mechanism specifically for 3D point cloud and graph data, which adheres to *equivariance constraints*, improving robustness to nuisance transformations and general performance.

Point cloud data is ubiquitous across many fields, presenting itself in diverse forms such as 3D object scans [29], 3D molecular structures [21], or N-body particle simulations [14]. Finding neural structures which can adapt to the varying number of points in an input, while respecting the irregular sampling of point positions, is challenging. Furthermore, an important property is that these structures should be invariant to global changes in overall input pose; that is, 3D translations and rotations of the input point cloud should not affect the output. In this paper, we find that the explicit imposition of equivariance constraints on the self-attention mechanism addresses these challenges. The SE(3)-Transformer uses the self-attention mechanism as a data-dependent filter particularly suited for sparse, non-voxelised point cloud data, while respecting and leveraging the symmetries of the task at hand.

---

[*]equal contribution
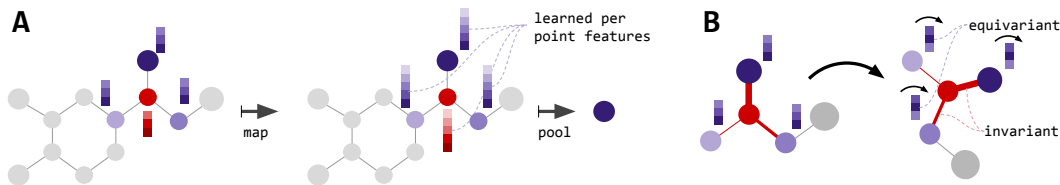[†]work done while at the Bosch Center for Artificial Intelligence

Figure 1: **A**) Each layer of the SE(3)-Transformer maps from a point cloud to a point cloud (or graph to graph) while guaranteeing equivariance. For classification, this is followed by an invariant pooling layer and an MLP. **B**) In each layer, for each node, attention is performed. Here, the red node attends to its neighbours. Attention weights (indicated by line thickness) are invariant w.r.t. input rotation.

Self-attention itself is a pseudo-linear map between sets of points. It can be seen to consist of two components: input-dependent *attention weights* and an embedding of the input, called a *value embedding*. In Fig. 1, we show an example of a molecular graph, where attached to every atom we see a value embedding vector and where the attention weights are represented as edges, with width corresponding to the attention weight magnitude. In the SE(3)-Transformer, we explicitly design the attention weights to be invariant to global pose. Furthermore, we design the value embedding to be equivariant to global pose. Equivariance generalises the translational weight-tying of convolutions. It ensures that transformations of a layer's input manifest as equivalent transformations of the output. SE(3)-equivariance in particular is the generalisation of translational weight-tying in 2D known from conventional convolutions to roto-translations in 3D. This restricts the space of learnable functions to a subspace which adheres to the symmetries of the task and thus reduces the number of learnable parameters. Meanwhile, it provides us with a richer form of invariance, since relative positional information between features in the input is preserved.

The works closest related to ours are tensor field networks (TFN) [28] and their voxelised equivalent, 3D steerable CNNs [37]. These provide frameworks for building SE(3)-equivariant convolutional networks operating on point clouds. Employing self-attention instead of convolutions has several advantages. (1) It allows a natural handling of edge features extending TFNs to the graph setting. (2) This is one of the first examples of a nonlinear equivariant layer. In Section 3.2, we show our proposed approach relieves the strong angular constraints on the filter compared to TFNs, therefore adding representational capacity. This constraint has been pointed out in the equivariance literature to limit performance severely [36]. Furthermore, we provide a more efficient implementation, mainly due to a GPU accelerated version of the spherical harmonics. The TFN baselines in our experiments leverage this and use significantly scaled up architectures compared to the ones used in [28].

Our contributions are the following:

- We introduce a novel self-attention mechanism, guaranteeably invariant to global rotations and translations of its input. It is also equivariant to permutations of the input point labels.

- We show that the SE(3)-Transformer resolves an issue with concurrent SE(3)-equivariant neural networks, which suffer from angularly constrained filters.

- We introduce a `Pytorch` implementation of spherical harmonics, which is 10x faster than `Scipy` on CPU and $100 - 1000\times$ faster on GPU. This directly addresses a bottleneck of TFNs [28]. E.g., for a ScanObjectNN model, we achieve $\approx 22\times$ speed up of the forward pass compared to a network built with SH from the `lielearn` library (see Appendix C).

- Code available at `https://github.com/FabianFuchsML/se3-transformer-public`

2 Background And Related Work

In this section we introduce the relevant background materials on self-attention, graph neural networks, and equivariance. We are concerned with point cloud based machine learning tasks, such as object classification or segmentation. In such a task, we are given a point cloud as input, represented as a collection of n coordinate vectors $\mathbf{x}_i \in \mathbb{R}^3$ with optional per-point features $\mathbf{f}_i \in \mathbb{R}^d$.

2.1 The Attention Mechanism

The standard *attention mechanism* [31] can be thought of as consisting of three terms: a set of query vectors $\mathbf{q}_i \in \mathbb{R}^p$ for $i = 1, ..., m$, a set of key vectors $\mathbf{k}_j \in \mathbb{R}^p$ for $j = 1, ..., n$, and a set of

value vectors $\mathbf{v}_j \in \mathbb{R}^r$ for $j = 1, ..., n$, where r and p are the dimensions of the low dimensional embeddings. We commonly interpret the key $\mathbf{k}_j$ and the value $\mathbf{v}_j$ as being 'attached' to the same point j. For a given query $\mathbf{q}_i$, the attention mechanism can be written as

$$\text{Attn}\left(\mathbf{q}_i, \{\mathbf{k}_j\}, \{\mathbf{v}_j\}\right) = \sum_{j=1}^{n} \alpha_{ij} \mathbf{v}_j, \qquad \alpha_{ij} = \frac{\exp(\mathbf{q}_i^\top \mathbf{k}_j)}{\sum_{j'=1}^{n} \exp(\mathbf{q}_i^\top \mathbf{k}_{j'})} \tag{1}$$

where we used a softmax as a nonlinearity acting on the weights. In general, the number of query vectors does not have to equal the number of input points [16]. In the case of *self-attention* the query, key, and value vectors are embeddings of the input features, so

$$\mathbf{q} = h_Q(\mathbf{f}), \qquad\qquad \mathbf{k} = h_K(\mathbf{f}), \qquad\qquad \mathbf{v} = h_V(\mathbf{f}), \tag{2}$$

where $\{h_Q, h_K, h_V\}$ are, in the most general case, neural networks [30]. For us, query $\mathbf{q}_i$ is associated with a point i in the input, which has a geometric location $\mathbf{x}_i$. Thus if we have n points, we have n possible queries. For query $\mathbf{q}_i$, we say that node i *attends* to all other nodes $j \neq i$.

Motivated by a successes across a wide range of tasks in deep learning such as language modeling [31], image recognition [18], graph-based problems [32], and relational reasoning [30, 9], a recent stream of work has applied forms of self-attention algorithms to point cloud data [44, 42, 16]. One such example is the Set Transformer [16]. When applied to object classification on ModelNet40 [41], the input to the Set Transformer are the cartesian coordinates of the points. Each layer embeds this positional information further while dynamically querying information from other points. The final per-point embeddings are downsampled and used for object classification.

Permutation equivariance A key property of self-attention is *permutation equivariance*. Permutations of point labels $1, ..., n$ lead to permutations of the self-attention output. This guarantees the attention output does not depend arbitrarily on input point ordering. Wagstaff et al. [33] recently showed that this mechanism can theoretically approximate *all* permutation equivariant functions. The SE(3)-transformer is a special case of this attention mechanism, inheriting permutation equivariance. However, it limits the space of learnable functions to rotation and translation equivariant ones.

2.2 Graph Neural Networks

Attention scales quadratically with point cloud size, so it is useful to introduce neighbourhoods: instead of each point attending to *all* other points, it only attends to its nearest neighbours. Sets with neighbourhoods are naturally represented as graphs. Attention has previously been introduced on graphs under the names of intra-, self-, vertex-, or graph-attention [17, 31, 32, 12, 26]. These methods were unified by Wang et al. [34] with the non-local neural network. This has the simple form

$$\mathbf{y}_i = \frac{1}{\mathcal{C}(\{\mathbf{f}_j \in \mathcal{N}_i\})} \sum_{j \in \mathcal{N}_i} w(\mathbf{f}_i, \mathbf{f}_j) h(\mathbf{f}_j) \tag{3}$$

where w and h are neural networks and $\mathcal{C}$ normalises the sum as a function of all features in the neighbourhood $\mathcal{N}_i$. This has a similar structure to attention, and indeed we can see it as performing attention per neighbourhood. While non-local modules do not explicitly incorporate edge-features, it is possible to add them, as done in Veličković et al. [32] and Hoshen [12].

2.3 Equivariance

Given a set of transformations $T_g : \mathcal{V} \to \mathcal{V}$ for $g \in G$, where G is an abstract group, a function $\phi : \mathcal{V} \to \mathcal{Y}$ is called equivariant if for every g there exists a transformation $S_g : \mathcal{Y} \to \mathcal{Y}$ such that

$$S_g[\phi(v)] = \phi(T_g[v]) \qquad \text{for all } g \in G, v \in \mathcal{V}. \tag{4}$$

The indices g can be considered as parameters describing the transformation. Given a pair (T_g, S_g), we can solve for the family of equivariant functions ϕ satisfying Equation 4. Furthermore, if (T_g, S_g) are linear and the map ϕ is also linear, then a very rich and developed theory already exists for finding ϕ [6]. In the equivariance literature, deep networks are built from interleaved linear maps ϕ and equivariant nonlinearities. In the case of 3D roto-translations it has already been shown that a suitable structure for ϕ is a *tensor field network* [28], explained below. Note that Romero et al. [24] recently introduced a 2D roto-translationally equivariant attention module for pixel-based image data.

Group Representations In general, the transformations (T_g, S_g) are called *group representations*. Formally, a group representation $\rho : G \to GL(N)$ is a map from a group G to the set of $N \times N$

invertible matrices $GL(N)$. Critically ρ is a *group homomorphism*; that is, it satisfies the following property $\rho(g_1 g_2) = \rho(g_1)\rho(g_2)$ for all $g_1, g_2 \in G$. Specifically for 3D rotations $G = SO(3)$, we have a few interesting properties: 1) its representations are orthogonal matrices, 2) all representations can be decomposed as

$$\rho(g) = \mathbf{Q}^\top \left[\bigoplus_\ell \mathbf{D}_\ell(g) \right] \mathbf{Q}, \tag{5}$$

where $\mathbf{Q}$ is an orthogonal, $N \times N$, change-of-basis matrix [5]; each $\mathbf{D}_\ell$ for $\ell = 0, 1, 2, ...$ is a $(2\ell+1) \times (2\ell+1)$ matrix known as a Wigner-D matrix[3]; and the $\bigoplus$ is the *direct sum* or concatenation of matrices along the diagonal. The Wigner-D matrices are *irreducible representations* of SO(3)— think of them as the 'smallest' representations possible. Vectors transforming according to $\mathbf{D}_\ell$ (i.e. we set $\mathbf{Q} = \mathbf{I}$), are called *type-ℓ* vectors. Type-0 vectors are invariant under rotations and type-1 vectors rotate according to 3D rotation matrices. Note, type-ℓ vectors have length $2\ell + 1$. They can be stacked, forming a feature vector $\mathbf{f}$ transforming according to Eq. (5).

Tensor Field Networks Tensor field networks (TFN) [28] are neural networks, which map point clouds to point clouds under the constraint of SE(3)-equivariance, the group of 3D rotations and translations. For point clouds, the input is a vector field $\mathbf{f} : \mathbb{R}^3 \to \mathbb{R}^d$ of the form

$$\mathbf{f}(\mathbf{x}) = \sum_{j=1}^N \mathbf{f}_j \delta(\mathbf{x} - \mathbf{x}_j), \tag{6}$$

where δ is the Dirac delta function, $\{\mathbf{x}_j\}$ are the 3D point coordinates and $\{\mathbf{f}_j\}$ are point features, representing such quantities as atomic number or point identity. For equivariance to be satisfied, the features of a TFN transform under Eq. (5), where $\mathbf{Q} = \mathbf{I}$. Each $\mathbf{f}_j$ is a concatenation of vectors of different *types*, where a subvector of type-ℓ is written $\mathbf{f}_j^\ell$. A TFN layer computes the convolution of a continuous-in-space, learnable weight kernel $\mathbf{W}^{\ell k} : \mathbb{R}^3 \to \mathbb{R}^{(2\ell+1) \times (2k+1)}$ from type-k features to type-ℓ features. The type-ℓ output of the TFN layer at position $\mathbf{x}_i$ is

$$\mathbf{f}_{\text{out},i}^\ell = \sum_{k \geq 0} \underbrace{\int \mathbf{W}^{\ell k}(\mathbf{x}' - \mathbf{x}_i)\mathbf{f}_{\text{in}}^k(\mathbf{x}')\, \mathrm{d}\mathbf{x}'}_{k \to \ell \text{ convolution}} = \sum_{k \geq 0}\sum_{j=1}^n \underbrace{\mathbf{W}^{\ell k}(\mathbf{x}_j - \mathbf{x}_i)\mathbf{f}_{\text{in},j}^k}_{\text{node } j \,\to\, \text{node } i \text{ message}}, \tag{7}$$

We can also include a sum over input channels, but we omit it here. Weiler et al. [37], Thomas et al. [28] and Kondor [15] showed that the kernel $\mathbf{W}^{\ell k}$ lies in the span of an equivariant basis $\{\mathbf{W}_J^{\ell k}\}_{J=|k-\ell|}^{k+\ell}$. The kernel is a linear combination of these basis kernels, where the Jth coefficient is a learnable function $\varphi_J^{\ell k} : \mathbb{R}_{\geq 0} \to \mathbb{R}$ of the radius $\|\mathbf{x}\|$. Mathematically this is

$$\mathbf{W}^{\ell k}(\mathbf{x}) = \sum_{J=|k-\ell|}^{k+\ell} \varphi_J^{\ell k}(\|\mathbf{x}\|)\mathbf{W}_J^{\ell k}(\mathbf{x}), \qquad \text{where } \mathbf{W}_J^{\ell k}(\mathbf{x}) = \sum_{m=-J}^J Y_{Jm}(\mathbf{x}/\|\mathbf{x}\|)\mathbf{Q}_{Jm}^{\ell k}. \tag{8}$$

Each basis kernel $\mathbf{W}_J^{\ell k} : \mathbb{R}^3 \to \mathbb{R}^{(2\ell+1) \times (2k+1)}$ is formed by taking a linear combination of Clebsch-Gordan matrices $\mathbf{Q}_{Jm}^{\ell k}$ of shape $(2\ell + 1) \times (2k + 1)$, where the J, m^{th} linear combination coefficient is the mth dimension of the Jth spherical harmonic $Y_J : \mathbb{R}^3 \to \mathbb{R}^{2J+1}$. Each basis kernel $\mathbf{W}_J^{\ell k}$ completely constrains the form of the learned kernel in the angular direction, leaving the only learnable degree of freedom in the radial direction. Note that $\mathbf{W}_J^{\ell k}(\mathbf{0}) \neq \mathbf{0}$ only when $k = \ell$ and $J = 0$, which reduces the kernel to a scalar w multiplied by the identity, $\mathbf{W}^{\ell\ell} = w^{\ell\ell}\mathbf{I}$, referred to as *self-interaction* [28]. As such we can rewrite the TFN layer as

$$\mathbf{f}_{\text{out},i}^\ell = \underbrace{w^{\ell\ell}\mathbf{f}_{\text{in},i}^\ell}_{\text{self-interaction}} + \sum_{k \geq 0}\sum_{j \neq i}^n \mathbf{W}^{\ell k}(\mathbf{x}_j - \mathbf{x}_i)\mathbf{f}_{\text{in},j}^k, \tag{9}$$

Eq. (7) and Eq. (9) present the convolution in message-passing form, where messages are aggregated from all nodes and feature types. They are also a form of nonlocal graph operation as in Eq. (3), where the weights are functions on edges and the features $\{\mathbf{f}_i\}$ are node features. We will later see how our proposed attention layer unifies aspects of convolutions and graph neural networks.

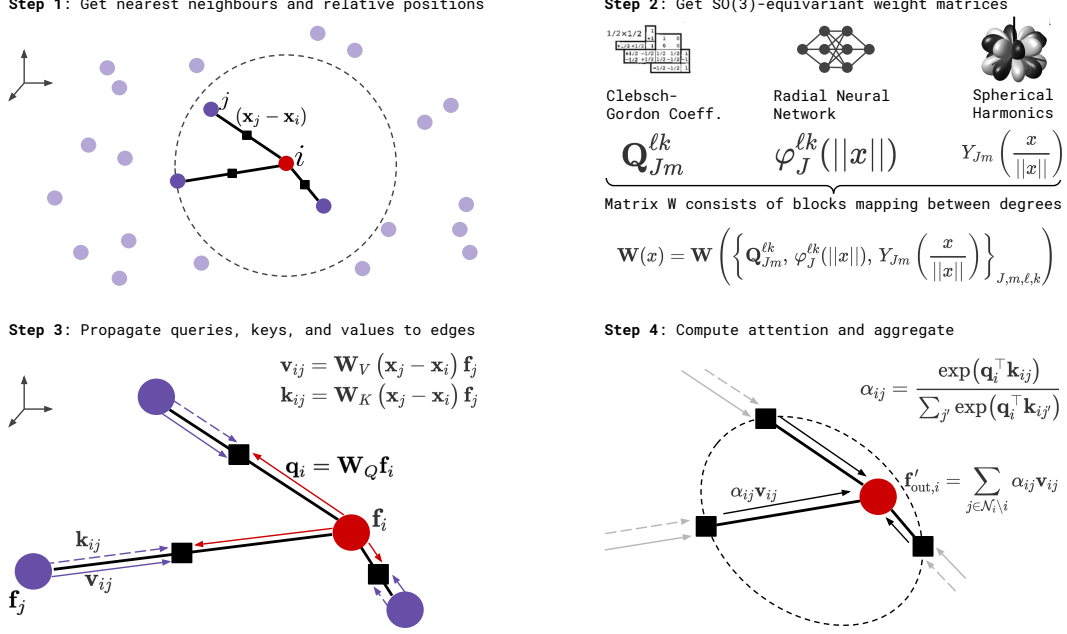[3]The 'D' stands for *Darstellung*, German for representation

Figure 2: Updating the node features using our equivariant attention mechanism in four steps. A more detailed description, especially of step 2, is provided in the Appendix. Steps 3 and 4 visualise a graph network perspective: features are passed from nodes to edges to compute keys, queries and values, which depend both on features and relative positions in a rotation-equivariant manner.

3 Method

Here, we present the *SE(3)-Transformer*. The layer can be broken down into a procedure of steps as shown in Fig. 2, which we describe in the following section. These are the construction of a graph from a point cloud, the construction of equivariant edge functions on the graph, how to propagate SE(3)-equivariant messages on the graph, and how to aggregate them. We also introduce an alternative for the self-interaction layer, which we call *attentive self-interaction*.

3.1 Neighbourhoods

Given a point cloud $\{(\mathbf{x}_i, \mathbf{f}_i)\}$, we first introduce a collection of neighbourhoods $\mathcal{N}_i \subseteq \{1, ..., N\}$, one centered on each point i. These neighbourhoods are computed either via the nearest-neighbours methods or may already be defined. For instance, molecular structures have neighbourhoods defined by their bonding structure. Neighbourhoods reduce the computational complexity of the attention mechanism from quadratic in the number of points to linear. The introduction of neighbourhoods converts our point cloud into a graph. This step is shown as Step 1 of Fig. 2.

3.2 The SE(3)-Transformer

The SE(3)-Transformer itself consists of three components. These are 1) edge-wise attention weights α_{ij}, constructed to be SE(3)-invariant on each edge ij, 2) edge-wise SE(3)-equivariant value messages, propagating information between nodes, as found in the TFN convolution of Eq. (7), and 3) a linear/attentive self-interaction layer. Attention is performed on a per-neighbourhood basis as follows:

$$\mathbf{f}_{\text{out},i}^\ell = \underbrace{\mathbf{W}_V^{\ell\ell} \mathbf{f}_{\text{in},i}^\ell}_{\text{③ self-interaction}} + \sum_{k \geq 0} \sum_{j \in \mathcal{N}_i \setminus i} \underbrace{\alpha_{ij}}_{\text{① attention}} \underbrace{\mathbf{W}_V^{\ell k}(\mathbf{x}_j - \mathbf{x}_i)\mathbf{f}_{\text{in},j}^k}_{\text{② value message}}. \tag{10}$$

These components are visualised in Fig. 2. If we remove the attention weights then we have a tensor field convolution, and if we instead remove the dependence of $\mathbf{W}_V$ on $(\mathbf{x}_j - \mathbf{x}_i)$, we have a conventional attention mechanism. Provided that the attention weights α_{ij} are invariant, Eq. (10) is equivariant to SE(3)-transformations. This is because it is just a linear combination of equivariant value messages. Invariant attention weights can be achieved with a dot-product attention structure shown in Eq. (11). This mechanism consists of a normalised inner product between a query vector $\mathbf{q}_i$

at node i and a set of key vectors $\{\mathbf{k}_{ij}\}_{j \in \mathcal{N}_i}$ along each edge ij in the neighbourhood $\mathcal{N}_i$ where

$$\alpha_{ij} = \frac{\exp(\mathbf{q}_i^\top \mathbf{k}_{ij})}{\sum_{j' \in \mathcal{N}_i \setminus i} \exp(\mathbf{q}_i^\top \mathbf{k}_{ij'})}, \quad \mathbf{q}_i = \bigoplus_{\ell \geq 0} \sum_{k \geq 0} \mathbf{W}_Q^{\ell k} \mathbf{f}_{\text{in},i}^k, \quad \mathbf{k}_{ij} = \bigoplus_{\ell \geq 0} \sum_{k \geq 0} \mathbf{W}_K^{\ell k}(\mathbf{x}_j - \mathbf{x}_i)\mathbf{f}_{\text{in},j}^k. \quad (11)$$

$\bigoplus$ is the direct sum, i.e. vector concatenation in this instance. The linear embedding matrices $\mathbf{W}_Q^{\ell k}$ and $\mathbf{W}_K^{\ell k}(\mathbf{x}_j - \mathbf{x}_i)$ are of TFN type (c.f. Eq. (8)). The attention weights α_{ij} are invariant for the following reason. If the input features $\{\mathbf{f}_{\text{in},j}\}$ are SO(3)-equivariant, then the query $\mathbf{q}_i$ and key vectors $\{\mathbf{k}_{ij}\}$ are also SE(3)-equivariant, since the linear embedding matrices are of TFN type. The inner product of SO(3)-equivariant vectors, transforming under the same representation $\mathbf{S}_g$ is invariant, since if $\mathbf{q} \mapsto \mathbf{S}_g \mathbf{q}$ and $\mathbf{k} \mapsto \mathbf{S}_g \mathbf{k}$, then $\mathbf{q}^\top \mathbf{S}_g^\top \mathbf{S}_g \mathbf{k} = \mathbf{q}^\top \mathbf{k}$, because of the orthonormality of representations of SO(3), mentioned in the background section. We follow the common practice from the self-attention literature [31, 16], and chosen a softmax nonlinearity to normalise the attention weights to unity, but in general any nonlinear function could be used.

Aside: Angular Modulation The attention weights add extra degrees of freedom to the TFN kernel in the angular direction. This is seen when Eq. (10) is viewed as a convolution with a data-dependent kernel $\alpha_{ij} \mathbf{W}_V^{\ell k}(\mathbf{x})$. In the literature, SO(3) equivariant kernels are decomposed as a sum of products of learnable radial functions $\varphi_J^{\ell k}(\|\mathbf{x}\|)$ and non-learnable angular kernels $\mathbf{W}_J^{\ell k}(\mathbf{x}/\|\mathbf{x}\|)$ (c.f. Eq. (8)). The fixed angular dependence of $\mathbf{W}_J^{\ell k}(\mathbf{x}/\|\mathbf{x}\|)$ is a strange artifact of the equivariance condition in noncommutative algebras and while necessary to guarantee equivariance, it is seen as overconstraining the expressiveness of the kernels. Interestingly, the attention weights α_{ij} introduce a means to modulate the angular profile of $\mathbf{W}_J^{\ell k}(\mathbf{x}/\|\mathbf{x}\|)$, while maintaining equivariance.

Channels, Self-interaction Layers, and Non-Linearities Analogous to conventional neural networks, the SE(3)-Transformer can straightforwardly be extended to multiple channels per representation degree ℓ, so far omitted for brevity. This sets the stage for self-interaction layers. The attention layer (c.f. Fig. 2 and circles 1 and 2 of Eq. (10)) aggregates information over nodes and input representation degrees k. In contrast, the self-interaction layer (c.f. circle 3 of Eq. (10)) exchanges information solely between features of the same degree and within one node—much akin to 1x1 convolutions in CNNs. Self-interaction is an elegant form of learnable skip connection, transporting information from query point i in layer L to query point i in layer $L + 1$. This is crucial since, in the SE(3)-Transformer, points do not attend to themselves. In our experiments, we use two different types of self-interaction layer: (1) linear and (2) attentive, both of the form

$$\mathbf{f}_{\text{out},i,c'}^\ell = \sum_c w_{i,c'c}^{\ell\ell} \mathbf{f}_{\text{in},i,c}^\ell. \quad (12)$$

Linear: Following Schütt et al. [25], output channels are a learned linear combination of input channels using one set of weights $w_{i,c'c}^{\ell\ell} = w_{c'c}^{\ell\ell}$ per representation degree, shared across all points. As proposed in Thomas et al. [28], this is followed by a norm-based non-linearity.

Attentive: We propose an extension of linear self-interaction, *attentive self-interaction*, combining self-interaction and nonlinearity. We replace the learned scalar weights $w_{c'c}^{\ell\ell}$ with attention weights output from an MLP, shown in Eq. (13) ($\bigoplus$ means concatenation.). These weights are SE(3)-invariant due to the invariance of inner products of features, transforming under the same representation.

$$w_{i,c'c}^{\ell\ell} = \text{MLP}\left(\bigoplus_{c,c'} \mathbf{f}_{\text{in},i,c'}^{\ell\top} \mathbf{f}_{\text{in},i,c}^\ell\right) \quad (13)$$

3.3 Node and Edge Features

Point cloud data often has information attached to points (node-features) and connections between points (edge-features), which we would both like to pass as inputs into the first layer of the network. Node information can directly be incorporated via the tensors $\mathbf{f}_j$ in Eqs. (6) and (10). For incorporating edge information, note that $\mathbf{f}_j$ is part of multiple neighbourhoods. One can replace $\mathbf{f}_j$ with $\mathbf{f}_{ij}$ in Eq. (10). Now, $\mathbf{f}_{ij}$ can carry different information depending on which neighbourhood $\mathcal{N}_i$ we are currently performing attention over. In other words, $\mathbf{f}_{ij}$ can carry information both about node j but also about edge ij. Alternatively, if the edge information is scalar, it can be incorporated into the weight matrices $\mathbf{W}_V$ and $\mathbf{W}_K$ as an input to the radial network (see step 2 in Fig. 2).

Table 1: Predicting future locations and velocities in an electron-proton simulation.

		Linear	DeepSet [46]	Tensor Field [28]	Set Transformer [16]	**SE(3)-Transformer**
Position	MSE x	0.0691	0.0639	0.0151	0.0139	**0.0076**
	std	-	0.0086	0.0011	0.0004	0.0002
	Δ_{EQ}	-	0.038	$1.9 \cdot 10^{-7}$	0.167	$3.2 \cdot 10^{-7}$
Velocity	MSE v	0.261	0.246	0.125	0.101	**0.075**
	std	-	0.017	0.002	0.004	0.001
	Δ_{EQ}	-	1.11	$5.0 \cdot 10^{-7}$	0.37	$6.3 \cdot 10^{-7}$

4 Experiments

We test the efficacy of the SE(3)-Transformer on three datasets, each testing different aspects of the model. The N-body problem is an equivariant task: rotation of the input should result in rotated predictions of locations and velocities of the particles. Next, we evaluate on a real-world object classification task. Here, the network is confronted with large point clouds of noisy data with symmetry only around the gravitational axis. Finally, we test the SE(3)-Transformer on a molecular property regression task, which shines light on its ability to incorporate rich graph structures. We compare to publicly available, state-of-the-art results as well as a set of our own baselines. Specifically, we compare to the Set-Transformer [16], a non-equivariant attention model, and Tensor Field Networks [28], which is similar to SE(3)-Transformer but does not leverage attention.

Similar to [27, 39], we measure the exactness of equivariance by applying uniformly sampled SO(3)-transformations to input and output. The distance between the two, averaged over samples, yields the equivariance error. Note that, unlike in Sosnovik et al. [27], the error is not squared:

$$\Delta_{EQ} = \|L_s\Phi(f) - \Phi L_s(f)\|_2 \, / \, \|L_s\Phi(f)\|_2 \tag{14}$$

4.1 N-Body Simulations

In this experiment, we use an adaptation of the dataset from Kipf et al. [14]. Five particles each carry either a positive or a negative charge and exert repulsive or attractive forces on each other. The input to the network is the position of a particle in a specific time step, its velocity, and its charge. The task of the algorithm is then to predict the relative location and velocity 500 time steps into the future. We deliberately formulated this as a regression problem to avoid the need to predict multiple time steps iteratively. Even though it certainly is an interesting direction for future research to combine equivariant attention with, e.g., an LSTM, our goal here was to test our core contribution and compare it to related models. This task sets itself apart from the other two experiments by not being invariant but equivariant: When the input is rotated or translated, the output changes respectively (see Fig. 3).

We trained an SE(3)-Transformer with 4 equivariant layers, each followed by an attentive self-interaction layer (details are provided in the Appendix). Table 1 shows quantitative results. Our model outperforms both an attention-based, but not rotation-equivariant approach (Set Transformer) and a equivariant approach which does not levarage attention (Tensor Field). The equivariance error shows that our approach is indeed fully rotation equivariant up to the precision of the computations.

(a) Set Transformer   (b) SE(3)-Transformer
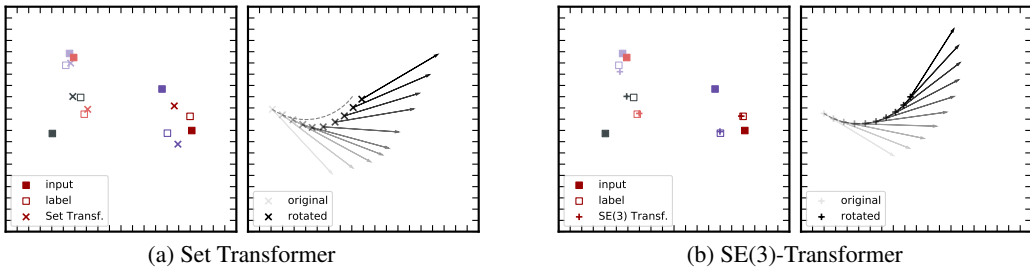
Figure 3: A model based on conventional self-attention (left) and our rotation-equivariant version (right) predict future locations and velocities in a 5-body problem. The respective left-hand plots show input locations at time step $t = 0$, ground truth locations at $t = 500$, and the respective predictions. The right-hand plots show predicted locations and velocities for rotations of the input in steps of 10 degrees. The dashed curves show the predicted locations of a perfectly equivariant model.

(a) Training *without* data augmentation.

(b) Training *with* data augmentation.
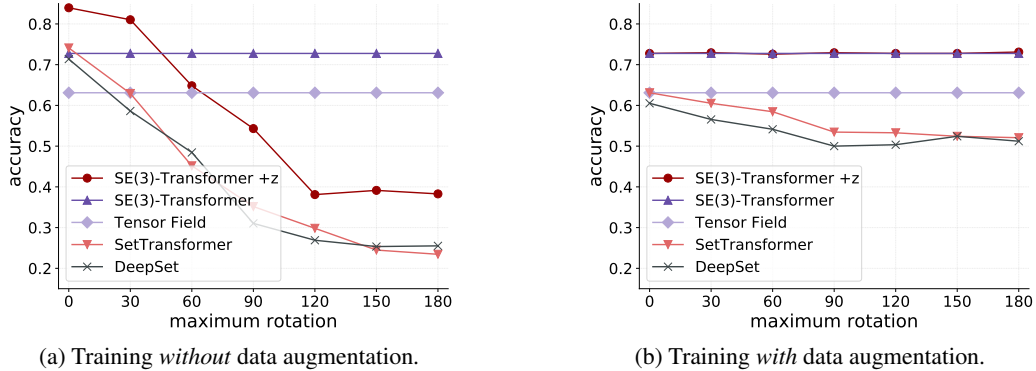
Figure 4: ScanObjectNN: x-axis shows data augmentation on the test set. The x-value corresponds to the maximum rotation around a random axis in the x-y-plane. If both training and test set are not rotated ($x = 0$ in **a**), breaking the symmetry of the SE(3)-Transformer by providing the z-component of the coordinates as an additional, scalar input improves the performance significantly. Interestingly, the model learns to ignore the additional, symmetry-breaking input when the training set presents a rotation-invariant problem (strongly overlapping dark red circles and dark purple triangles in **b**).

Table 2: Classification accuracy on the 'object only' category of the ScanObjectNN dataset[4]. The performance of the SE(3)-Transformer is averaged over 5 runs (standard deviation 0.7%).

	Tensor Field	DeepSet	SE(3)-Transf.	3DmFV	Set Transformer	PointNet	SpiderCNN	Tensor Field +z	PointNet++	SE(3)-Transf.+z	PointCNN	DGCNN	PointGLR
No. Points	128	1024	**128**	1024	1024	1024	1024	128	1024	**128**	1024	1024	1024
Accuracy	63.1%	71.4%	**72.8** %	73.8%	74.1%	79.2%	79.5%	81.0%	84.3%	**85.0%**	85.5%	86.2%	87.2%

4.2 Real-World Object Classification on ScanObjectNN

Object classification from point clouds is a well-studied task. Interestingly, the vast majority of current neural network methods work on scalar coordinates without incorporating vector specific inductive biases. Some recent works explore rotation invariant point cloud classification [45, 47, 4, 22]. Our method sets itself apart by using roto-translation equivariant layers acting directly on the point cloud without prior projection onto a sphere [22, 45, 7]. This allows for weight-tying and increased sample efficiency while transporting information about local and global orientation through the network - analogous to the translation equivariance of CNNs on 2D images. To test our method, we choose ScanObjectNN, a recently introduced dataset for real-world object classification. The benchmark provides point clouds of 2902 objects across 15 different categories. We only use the coordinates of the points as input and object categories as training labels. We train an SE(3)-Transformer with 4 equivariant layers with linear self-interaction followed by max-pooling and an MLP. Interestingly, the task is not fully rotation invariant, in a statistical sense, as the objects are aligned with respect to the gravity axis. This results in a performance loss when deploying a fully SO(3) invariant model (see Fig. 4a). In other words: when looking at a new object, it helps to know where 'up' is. We create an SO(2) invariant version of our algorithm by additionally feeding the z-component as an type-0 field and the x, y position as an additional type-1 field (see Appendix). We dub this model *SE(3)-Transformer +z*. This way, the model can 'learn' which symmetries to adhere to by suppressing and promoting different inputs (compare Fig. 4a and Fig. 4b). In Table 2, we compare our model to the current state-of-the-art in object classification[4]. Despite the dataset not playing to the strengths of our model (full SE(3)-invariance) and a much lower number of input points, the performance is competitive with models specifically designed for object classification - PointGLR [23], for instance, is pre-trained on the larger ModelNet40 dataset [41]. For a discussion of performance vs. number of input points used, see Appendix D.1.2.

---

[4]At time of submission, PointGLR was a recently published preprint [23]. The performance of the following models was taken from the official benchmark of the dataset as of June 4th, 2020 (`https://hkust-vgd.github.io/benchmark/`): 3DmFV [3], PointNet [19], SpiderCNN [43], PointNet++ [20], DGCN [35].

4.3 QM9

The QM9 regression dataset [21] is a publicly available chemical property prediction task. There are 134k molecules with up to 29 atoms per molecule. Atoms are represented as a 5 dimensional one-hot node embeddings in a molecular graph connected by 4 different chemical bond types (more details in Appendix). 'Positions' of each atom are provided.

Table 3: QM9 Mean Absolute Error. Top: Non-equivariant models. Bottom: Equivariant models. SE(3)-Tr. is averaged over 5 runs.

TASK UNITS	α bohr^3	$\Delta\varepsilon$ meV	$\varepsilon_{\text{HOMO}}$ meV	$\varepsilon_{\text{LUMO}}$ meV	μ D	C_ν cal/mol K
WaveScatt [11]	.160	118	85	76	.340	.049
NMP [10]	.092	69	43	38	.030	.040
SchNet [25]	.235	63	41	34	.033	.033
Cormorant [1]	.085	61	34	38	.038	.026
LieConv(T3) [8]	.084	49	30	25	.032	.038
TFN [28]	.223	58	40	38	.064	.101
SE(3)-Transformer	.142±.002	53.0±0.3	35.0±.9	33.0±.7	.051±.001	.054±.002

We show results on the test set of Anderson et al. [1] for 6 regression tasks in Table 3. Lower is better. The table is split into non-equivariant (top) and equivariant models (bottom). Our nearest models are Cormorant and TFN (own implementation). We see that while not state-of-the-art, we offer competitive performance, especially against Cormorant and TFN, which transform under irreducible representations of SE(3) (like us), unlike LieConv(T3), using a left-regular representation of SE(3), which may explain its success.

5 Conclusion

We have presented an attention-based neural architecture designed specifically for point cloud data. This architecture is guaranteed to be robust to rotations and translations of the input, obviating the need for training time data augmentation and ensuring stability to arbitrary choices of coordinate frame. The use of self-attention allows for anisotropic, data-adaptive filters, while the use of neighbourhoods enables scalability to large point clouds. We have also introduced the interpretation of the attention mechanism as a data-dependent nonlinearity, adding to the list of equivariant nonlinearties which we can use in equivariant networks. Furthermore, we provide code for a speed up of spherical harmonics computation of up to 3 orders of magnitudes. This speed-up allowed us to train significantly larger versions of both the SE(3)-Transformer and the Tensor Field network [28] and to apply these models to real-world datasets.

Our experiments showed that adding attention to a roto-translation-equivariant model consistently led to higher accuracy and increased training stability. Specifically for large neighbourhoods, attention proved essential for model convergence. On the other hand, compared to convential attention, adding the equivariance constraints also increases performance in all of our experiments while at the same time providing a mathematical guarantee for robustness with respect to rotations of the input data.

Broader Impact

The main contribution of the paper is a mathematically motivated attention mechanism which can be used for deep learning on point cloud based problems. We do not see a direct potential of *negative* impact to the society. However, we would like to stress that this type of algorithm is inherently suited for classification and regression problems on molecules. The SE(3)-Transformer therefore lends itself to application in drug research. One concrete application we are currently investigating is to use the algorithm for early-stage suitability classification of molecules for inhibiting the reproductive cycle of the coronavirus. While research of this sort always requires intensive testing in wet labs, computer algorithms can be and are being used to filter out particularly promising compounds from large databases of millions of molecules.

Acknowledgements and Funding Disclosure

References

[1] Brandon Anderson, Truong Son Hy, and Risi Kondor. Cormorant: Covariant molecular neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[2] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv Preprint*, 2016.

[3] Yizhak Ben-Shabat, Michael Lindenbaum, and Anath Fischer. 3dmfv: Three-dimensional point cloud classification in realtime using convolutional neural networks. *IEEE Robotics and Automation Letters*, 2018.

[4] Chao Chen, Guanbin Li, Ruijia Xu, Tianshui Chen, Meng Wang, and Liang Lin. ClusterNet: Deep Hierarchical Cluster Network with Rigorously Rotation-Invariant Representation for Point Cloud Analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[5] Gregory S Chirikjian, Alexander B Kyatkin, and AC Buckingham. Engineering applications of noncommutative harmonic analysis: with emphasis on rotation and motion groups. *Appl. Mech. Rev.*, 54(6):B97–B98, 2001.

[6] Taco S. Cohen and Max Welling. Steerable cnns. *International Conference on Learning Representations (ICLR)*, 2017.

[7] Taco S. Cohen, Mario Geiger, Jonas Koehler, and Max Welling. Spherical cnns. In *International Conference on Learning Representations (ICLR)*, 2018.

[8] Marc Finzi, Samuel Stanton, Pavel Izmailov, and Andrew Wilson. Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data. *Proceedings of the International Conference on Machine Learning, ICML*, 2020.

[9] Fabian B. Fuchs, Adam R. Kosiorek, Li Sun, Oiwi Parker Jones, and Ingmar Posner. End-to-end recurrent multi-object tracking and prediction with relational reasoning. *arXiv preprint*, 2020.

[10] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, and Oriol Vinyals George E. Dahl. Neural message passing for quantum chemistry. *Proceedings of the International Conference on Machine Learning, ICML*, 2020.

[11] Matthew J. Hirn, Stéphane Mallat, and Nicolas Poilvert. Wavelet scattering regression of quantum chemical energies. *Multiscale Model. Simul.*, 15(2):827–863, 2017.

[12] Yedid Hoshen. Vain: Attentional multi-agent predictive modeling. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations, ICLR*, 2015.

[14] Thomas N. Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard S. Zemel. Neural relational inference for interacting systems. In *Proceedings of the International Conference on Machine Learning, ICML*, 2018.

[15] Risi Kondor. N-body networks: a covariant hierarchical neural network architecture for learning atomic potentials. *arXiv preprint*, 2018.

[16] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the International Conference on Machine Learning, ICML*, 2019.

[17] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *International Conference on Learning Representations (ICLR)*, 2017.

[18] Niki Parmar, Prajit Ramachandran, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. In *Advances in Neural Information Processing System (NeurIPS)*, 2019.

[19] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[20] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[21] Raghunathan Ramakrishnan, Pavlo Dral, Matthias Rupp, and Anatole von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1, 08 2014.

[22] Yongming Rao, Jiwen Lu, and Jie Zhou. Spherical fractal convolutional neural networks for point cloud recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[23] Yongming Rao, Jiwen Lu, and Jie Zhou. Global-local bidirectional reasoning for unsupervised representation learning of 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[24] David W. Romero, Erik J. Bekkers, Jakub M. Tomczak, and Mark Hoogendoorn. Attentive group equivariant convolutional networks. *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

[25] K. T. Schütt, P.-J. Kindermans, H. E. Sauceda, S. Chmiela1, A. Tkatchenko, and K.-R. Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[26] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, 2018.

[27] Ivan Sosnovik, Michał Szmaja, and Arnold Smeulders. Scale-equivariant steerable networks. *International Conference on Learning Representations (ICLR)*, 2020.

[28] Nathaniel Thomas, Tess Smidt, Steven M. Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds. *ArXiv Preprint*, 2018.

[29] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Duc Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *International Conference on Computer Vision (ICCV)*, 2019.

[30] Sjoerd van Steenkiste, Michael Chang, Klaus Greff, and Jürgen Schmidhuber. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. *International Conference on Learning Representations (ICLR)*, 2018.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[32] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *International Conference on Learning Representations (ICLR)*, 2018.

[33] Edward Wagstaff, Fabian B. Fuchs, Martin Engelcke, Ingmar Posner, and Michael A. Osborne. On the limitations of representing functions on sets. *International Conference on Machine Learning (ICML)*, 2019.

[34] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[35] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 2019.

[36] Maurice Weiler and Gabriele Cesa. General E(2)-Equivariant Steerable CNNs. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[37] Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco Cohen. 3d steerable cnns: Learning rotationally equivariant features in volumetric data. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[38] Marysia Winkels and Taco S. Cohen. 3d g-cnns for pulmonary nodule detection. *1st Conference on Medical Imaging with Deep Learning (MIDL)*, 2018.

[39] Daniel E. Worrall and Max Welling. Deep scale-spaces: Equivariance over scale. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[40] Daniel E. Worrall, Stephan J. Garbin, Daniyar Turmukhambetov, and Gabriel J. Brostow. Harmonic networks: Deep translation and rotation equivariance. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[41] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[42] Saining Xie, Sainan Liu, and Zeyu Chen Zhuowen Tu. Attentional shapecontextnet for point cloud recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[43] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. *European Conference on Computer Vision (ECCV)*, 2018.

[44] Jiancheng Yang, Qiang Zhang, and Bingbing Ni. Modeling point clouds with self-attention and gumbel subset sampling. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[45] Yang You, Yujing Lou, Qi Liu, Yu-Wing Tai, Lizhuang Ma, Cewu Lu, and Weiming Wang. Pointwise Rotation-Invariant Network with Adaptive Sampling and 3D Spherical Voxel Convolution. In *AAAI Conference on Artificial Intelligence*, 2019.

[46] Manzil Zaheer, Satwik Kottur, Siamak Ravanbhakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander Smola. Deep Sets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[47] Zhiyuan Zhang, Binh-Son Hua, David W. Rosen, and Sai-Kit Yeung. Rotation invariant convolutions for 3d point clouds deep learning. In *International Conference on 3D Vision (3DV)*, 2019.

A Group Theory and Tensor Field Networks

Groups A *group* is an abstract mathematical concept. Formally a group $(G, \circ)$ consists of a set G and a binary composition operator $\circ : G \times G \to G$ (typically we just use the symbol G to refer to the group). All groups must adhere to the following 4 axioms

- **Closure**: $g \circ h \in G$ for all $g, h \in G$
- **Associativity**: $f \circ (g \circ h) = (f \circ g) \circ h = f \circ g \circ h$ for all $f, g, h \in G$
- **Identity**: There exists an element $e \in G$ such that $e \circ g = g \circ e = g$ for all $g \in G$
- **Inverses**: For each $g \in G$ there exists a $g^{-1} \in G$ such that $g^{-1} \circ g = g \circ g^{-1} = e$

In practice, we omit writing the binary composition operator $\circ$, so would write gh instead of $g \circ h$. Groups can be finite or infinite, countable or uncountable, compact or non-compact. Note that they are not necessarily *commutative*; that is, $gh \neq hg$ in general.

Actions/Transformations Groups are useful concepts, because they allow us to describe the structure of *transformations*, also sometimes called *actions*. A transformation (operator) $T_g : \mathcal{X} \to \mathcal{X}$ is an injective map from a space into itself. It is parameterised by an element g of a group G. Transformations obey two laws:

- **Closure**: $T_g \circ T_h$ is a valid transformation for all $g, h \in G$
- **Identity**: There exists at least one element $e \in G$ such that $T_e[\mathbf{x}] = \mathbf{x}$ for all $\mathbf{x} \in \mathcal{X}$,

where $\circ$ denotes composition of transformations. For the expression $T_g[\mathbf{x}]$, we say that T_g *acts* on $\mathbf{x}$. It can also be shown that transformations are associative under composition. To codify the structure of a transformation, we note that due to closure we can always write

$$T_g \circ T_h = T_{gh}, \tag{15}$$

If for any $x, y \in \mathcal{X}$ we can always find a group element g, such that $T_g[x] = y$, then we call $\mathcal{X}$ a *homogeneous space*. Homogeneous spaces are important concepts, because to each pair of points x, y we can always associate at least one group element.

Equivariance and Intertwiners As written in the main body of the text, equivariance is a property of functions $f : \mathcal{X} \to \mathcal{Y}$. Just to recap, given a set of transformations $T_g : \mathcal{X} \to \mathcal{X}$ for $g \in G$, where G is an abstract group, a function $f : \mathcal{X} \to \mathcal{Y}$ is called equivariant if for every g there exists a transformation $S_g : \mathcal{Y} \to \mathcal{Y}$ such that

$$S_g[f(\mathbf{x})] = f(T_g[\mathbf{x}]) \qquad \text{for all } g \in G, \mathbf{x} \in \mathcal{X}. \tag{16}$$

If f is linear and equivariant, then it is called an intertwiner. Two important questions arise: 1) How do we choose S_g? 2) once we have (T_g, S_g), how do we solve for f? To answer these questions, we need to understand what kinds of S_g are possible. For this, we review representations.

Representations A group representation $\rho : G \to GL(N)$ is a map from a group G to the set of $N \times N$ invertible matrices $GL(N)$. Critically ρ is a *group homomorphism*; that is, it satisfies the following property $\rho(g_1 g_2) = \rho(g_1)\rho(g_2)$ for all $g_1, g_2 \in G$. Representations can be used as transformation operators, acting on N-dimensional vectors $\mathbf{x} \in \mathbb{R}^N$. For instance, for the group of 3D rotations, known as $SO(3)$, we have that 3D rotation matrices, $\rho(g) = \mathbf{R}_g$ act on (i.e., rotate) 3D vectors, as

$$T_g[\mathbf{x}] = \rho(g)\mathbf{x} = \mathbf{R}_g \mathbf{x}, \qquad \text{for all } \mathbf{x} \in \mathcal{X}, g \in G. \tag{17}$$

However, there are many more representations of $SO(3)$ than just the 3D rotation matrices. Among representations, two representations ρ and ρ' of the same dimensionality are said to be *equivalent* if they can be connected by a similarity transformation

$$\rho'(g) = \mathbf{Q}^{-1}\rho(g)\mathbf{Q}, \qquad \text{for all } g \in G. \tag{18}$$

We also say that a representation is *reducible* if is can be written as

$$\rho(g) = \mathbf{Q}^{-1}(\rho_1(g) \oplus \rho_2(g))\mathbf{Q} = \mathbf{Q}^{-1} \begin{bmatrix} \rho_1(g) & \\ & \rho_2(g) \end{bmatrix} \mathbf{Q}, \qquad \text{for all } g \in G. \tag{19}$$

If the representations ρ_1 and ρ_2 are not reducible, then they are called *irreducible representations* of G, or *irreps*. In a sense, they are the atoms among representations, out of which all other representations can be constructed. Note that each irrep acts on a separate subspace, mapping vectors from that space back into it. We say that subspace $\mathcal{X}_\ell \in \mathcal{X}$ is *invariant under* irrep ρ_ℓ, if $\{\rho_\ell(g)\mathbf{x} \mid \mathbf{x} \in \mathcal{X}_\ell, g \in G\} \subseteq \mathcal{X}_\ell$.

Representation theory of $SO(3)$ As it turns out, all linear representations of compact groups[5] (such as $SO(3)$) can be decomposed into a direct sum of irreps, as

$$\rho(g) = \mathbf{Q}^\top \left[\bigoplus_J \mathbf{D}_J(g) \right] \mathbf{Q}, \tag{20}$$

where $\mathbf{Q}$ is an orthogonal, $N \times N$, change-of-basis matrix [5]; and each $\mathbf{D}_J$ for $J = 0, 1, 2, ...$ is a $(2J + 1) \times (2J + 1)$ matrix known as a Wigner-D matrix. The Wigner-D matrices are the irreducible representations of $SO(3)$. We also mentioned that vectors transforming according to $\mathbf{D}_J$ (i.e. we set $\mathbf{Q} = \mathbf{I}$), are called *type-J* vectors. Type-0 vectors are invariant under rotations and type-1 vectors rotate according to 3D rotation matrices. Note, type-J vectors have length $2J + 1$. In the previous paragraph we mentioned that irreps act on orthogonal subspaces $\mathcal{X}_0, \mathcal{X}_1, ...$. The orthogonal subspaces corresponding to the Wigner-D matrices are the space of *spherical harmonics*.

The Spherical Harmonics   The spherical harmonics $\mathbf{Y}_J : S^2 \to \mathbb{C}^{2J+1}$ for $J \geq 0$ are square-integrable complex-valued functions on the sphere S^2. They have the satisfying property that they are rotated directly by the Wigner-D matrices as

$$\mathbf{Y}_J(\mathbf{R}_g^{-1}\mathbf{x}) = \mathbf{D}_J^*(g)\mathbf{Y}_J(\mathbf{x}), \qquad \mathbf{x} \in S^2, g \in G, \tag{21}$$

where $\mathbf{D}_J$ is the Jth Wigner-D matrix and $\mathbf{D}_J^*$ is its complex conjugate. They form an orthonormal basis for (the Hilbert space of) square-integrable functions on the sphere $L^2(S^2)$, with inner product given as

$$\langle f, h \rangle_{S^2} = \int_{S^2} f(\mathbf{x})h^*(\mathbf{x}) \, d\mathbf{x}. \tag{22}$$

So $\langle Y_{Jm}, Y_{J'm'} \rangle_{S^2} = \delta_{JJ'}\delta_{mm'}$, where Y_{Jm} is the mth element of $\mathbf{Y}_J$. We can express any function in $L^2(S^2)$ as a linear combination of spherical harmonics, where

$$f(\mathbf{x}) = \sum_{J \geq 0} \mathbf{f}_J^\top \mathbf{Y}_J(\mathbf{x}), \qquad \mathbf{x} \in S^2, \tag{23}$$

where each $\mathbf{f}_J$ is a vector of coefficients of length $2J + 1$. And in the opposite direction, we can retrieve the coefficients as

$$\mathbf{f}_J = \int_{S^2} f(\mathbf{x})\mathbf{Y}_J^*(\mathbf{x}) \, d\mathbf{x} \tag{24}$$

following from the orthonormality of the spherical harmonics. This is in fact a Fourier transform on the sphere and the the vectors $\mathbf{f}_J$ can be considered Fourier coefficients. Critically, we can represent rotated functions as

$$f(\mathbf{R}_g^{-1}\mathbf{x}) = \sum_{J \geq 0} \mathbf{f}_J^\top \mathbf{D}_J^*(g)\mathbf{Y}_J(\mathbf{x}), \qquad \mathbf{x} \in S^2, g \in G. \tag{25}$$

The Clebsch-Gordan Decomposition   In the main text we introduced the *Clebsch-Gordan coefficients*. These are used in the construction of the equivariant kernels. They arise in the situation where we have a tensor product of Wigner-D matrices, which as we will see is part of the equivariance constraint on the form of the equivariant kernels. In representation theory a tensor product of representations is also a representation, but since it is not an easy object to work with, we seek to decompose it into a direct sum of irreps, which are easier. This decomposition is of the form of Eq. (20), written

$$\mathbf{D}_k(g) \otimes \mathbf{D}_\ell(g) = \mathbf{Q}^{\ell k \top} \left[\bigoplus_{J=|k-\ell|}^{k+\ell} \mathbf{D}_J(g) \right] \mathbf{Q}^{\ell k}. \tag{26}$$

In this specific instance, the change of basis matrices $\mathbf{Q}^{\ell k}$ are given the special name of the Clebsch-Gordan coefficients. These can be found in many mathematical physics libraries.

---

[5]Over a field of characteristic zero.

Tensor Field Layers   In Tensor Field Networks [28] and 3D Steerable CNNs [37], the authors solve for the intertwiners between SO(3) equivariant point clouds. Here we run through the derivation again in our own notation.

We begin with a point cloud $f(\mathbf{x}) = \sum_{j=1}^{N} \mathbf{f}_j \delta(\mathbf{x} - \mathbf{x}_j)$, where $\mathbf{f}_j$ is an equivariant point feature. Let's say that $\mathbf{f}_j$ is a type-k feature, which we write as $\mathbf{f}_j^k$ to remind ourselves of the fact. Now say we perform a convolution $*$ with kernel $\mathbf{W}^{\ell k} : \mathbf{R}^3 \to \mathbb{R}^{(2\ell+1)\times(2k+1)}$, which maps from type-$k$ features to type-ℓ features. Then

$$\mathbf{f}_{\text{out,i}}^\ell = [\mathbf{W}^{\ell k} * \mathbf{f}_{\text{in}}^k](\mathbf{x}) \tag{27}$$

$$= \int_{\mathbb{R}^3} \mathbf{W}^{\ell k}(\mathbf{x}' - \mathbf{x}_i) \mathbf{f}_{\text{in}}^k(\mathbf{x}') \, \mathrm{d}\mathbf{x}' \tag{28}$$

$$= \int_{\mathbb{R}^3} \mathbf{W}^{\ell k}(\mathbf{x}' - \mathbf{x}_i) \sum_{j=1}^{N} \mathbf{f}_{\text{in},j}^k \delta(\mathbf{x}' - \mathbf{x}_j) \, \mathrm{d}\mathbf{x}' \tag{29}$$

$$= \sum_{j=1}^{N} \int_{\mathbb{R}^3} \mathbf{W}^{\ell k}(\mathbf{x}' - \mathbf{x}_i) \mathbf{f}_{\text{in},j}^k \delta(\mathbf{x}' - \mathbf{x}_j) \, \mathrm{d}\mathbf{x}' \qquad \text{change of variables } \mathbf{x}'' = \mathbf{x}' - \mathbf{x}_j \tag{30}$$

$$= \sum_{j=1}^{N} \int_{\mathbb{R}^3} \mathbf{W}^{\ell k}(\mathbf{x}'' + \mathbf{x}_j - \mathbf{x}_i) \mathbf{f}_{\text{in},j}^k \delta(\mathbf{x}'') \, \mathrm{d}\mathbf{x}'' \qquad \text{sifting theorem} \tag{31}$$

$$= \sum_{j=1}^{N} \mathbf{W}^{\ell k}(\mathbf{x}_j - \mathbf{x}_i) \mathbf{f}_{\text{in},j}^k. \tag{32}$$

Now let's apply the equivariance condition to this expression, then

$$\mathbf{D}_\ell(g)\mathbf{f}_{\text{out,i}}^\ell = \sum_{j=1}^{N} \mathbf{W}^{\ell k}(\mathbf{R}_g^{-1}(\mathbf{x}_j - \mathbf{x}_i))\mathbf{D}_k(g)\mathbf{f}_{\text{in},j}^k \tag{33}$$

$$\implies \mathbf{f}_{\text{out,i}}^\ell = \sum_{j=1}^{N} \mathbf{D}_\ell(g)^{-1}\mathbf{W}^{\ell k}(\mathbf{R}_g^{-1}(\mathbf{x}_j - \mathbf{x}_i))\mathbf{D}_k(g)\mathbf{f}_{\text{in},j}^k \tag{34}$$

Now we notice that this expression should also be equal to Eq. (32), which is the convolution with an unrotated point cloud. Thus we end up at

$$\boxed{\mathbf{W}^{\ell k}(\mathbf{R}_g^{-1}\mathbf{x}) = \mathbf{D}_\ell(g)\mathbf{W}^{\ell k}(\mathbf{x})\mathbf{D}_k(g)^{-1},} \tag{35}$$

which is sometimes refered to as the *kernel constraint*. To solve the kernel constraint, we notice that it is a linear equation and that we can rearrange it as

$$\text{vec}(\mathbf{W}^{\ell k}(\mathbf{R}_g^{-1}\mathbf{x})) = (\mathbf{D}_k(g) \otimes \mathbf{D}_\ell(g))\text{vec}(\mathbf{W}^{\ell k}(\mathbf{x})) \tag{36}$$

where we used the identity $\text{vec}(\mathbf{A}\mathbf{X}\mathbf{B}) = (\mathbf{B}^\top \otimes \mathbf{A})\text{vec}(\mathbf{X})$ and the fact that the Wigner-D matrices are orthogonal. Using the Clebsch-Gordan decomposition we rewrite this as

$$\text{vec}(\mathbf{W}^{\ell k}(\mathbf{R}_g^{-1}\mathbf{x})) = \mathbf{Q}^{\ell k \top} \left[\bigoplus_{J=|k-\ell|}^{k+\ell} \mathbf{D}_J(g) \right] \mathbf{Q}^{\ell k}\text{vec}(\mathbf{W}^{\ell k}(\mathbf{R}_g^{-1}\mathbf{x})). \tag{37}$$

Lastly, we can left multiply both sides by $\mathbf{Q}^{\ell k}$ and denote $\boldsymbol{\eta}^{\ell k}(\mathbf{x}) \triangleq \mathbf{Q}^{\ell k}\text{vec}(\mathbf{W}^{\ell k}(\mathbf{x}))$, noting the the Clebsch-Gordan matrices are orthogonal. At the same time we

$$\boldsymbol{\eta}^{\ell k}(\mathbf{R}_g^{-1}\mathbf{x}) = \left[\bigoplus_{J=|k-\ell|}^{k+\ell} \mathbf{D}_J(g) \right] \boldsymbol{\eta}^{\ell k}(\mathbf{x}). \tag{38}$$

Thus we have that $\boldsymbol{\eta}_J^{\ell k}(\mathbf{R}_g^{-1}\mathbf{x})$ the Jth subvector of $\boldsymbol{\eta}^{\ell k}(\mathbf{R}_g^{-1}\mathbf{x})$ is subject to the constraint

$$\boldsymbol{\eta}_J^{\ell k}(\mathbf{R}_g^{-1}\mathbf{x}) = \mathbf{D}_J(g)\boldsymbol{\eta}_J^{\ell k}(\mathbf{x}), \tag{39}$$

15

which is exactly the transformation law for the spherical harmonics from Eq. (21)! Thus one way how $\mathbf{W}^{\ell k}(\mathbf{x})$ can be constructed is

$$\mathrm{vec}\left(\mathbf{W}^{\ell k}(\mathbf{x})\right) = \mathbf{Q}^{\ell k\top} \bigoplus_{J=|k-\ell|}^{k+\ell} \mathbf{Y}_J(\mathbf{x}). \tag{40}$$

B   Recipe for Building an Equivariant Weight Matrix

One of the core operations in the SE(3)-Transformer is multiplying a feature vector $\mathbf{f}$, which transforms according to $SO(3)$, with a matrix $\mathbf{W}$ while preserving equivariance:

$$S_g[\mathbf{W} * \mathbf{f}](\mathbf{x}) = [\mathbf{W} * T_g[\mathbf{f}]](\mathbf{x}), \tag{41}$$

where $T_g[\mathbf{f}](\mathbf{x}) = \rho_{\mathrm{in}}(g)\mathbf{f}(\mathbf{R}_g^{-1}\mathbf{x})$ and $S_g[\mathbf{f}](\mathbf{x}) = \rho_{\mathrm{out}}(g)\mathbf{f}(\mathbf{R}_g^{-1}\mathbf{x})$. Here, as in the previous section we showed how such a matrix $\mathbf{W}$ could be constructed when mapping between features of type-k and type-ℓ, where $\rho_{\mathrm{in}}(g)$ is a block diagonal matrix of type-k Wigner-D matrices and similarly $\rho_{\mathrm{in}}(g)$ is made of type-ℓ Wigner-D matrices. $\mathbf{W}$ is dependent on the relative position $\mathbf{x}$ and underlies the linear equivariance constraints, but is also has learnable components, which we did not show in the previous section. In this section, we show how such a matrix is constructed in practice.

Previously we showed that

$$\mathrm{vec}\left(\mathbf{W}^{\ell k}(\mathbf{x})\right) = \mathbf{Q}^{\ell k\top} \bigoplus_{J=|k-\ell|}^{k+\ell} \mathbf{Y}_J(\mathbf{x}), \tag{42}$$

which is an equivariant mapping between vectors of types k and ℓ. In practice, we have multiple input vectors $\{\mathbf{f}_c^k\}_c$ of type-k and multiple output vectors of type-ℓ. For simplicity, however, we ignore this and pretend we only have a single input and single output. Note that $\mathbf{W}^{\ell k}$ has no learnable components. Note that the kernel constraint only acts in the angular direction, but not in the radial direction, so we can introduce scalar radial functions $\varphi_J^{\ell k} : \mathbf{R}_{\geq 0} \to \mathbf{R}$ (one for each J), such that

$$\mathrm{vec}\left(\mathbf{W}^{\ell k}(\mathbf{x})\right) = \mathbf{Q}^{\ell k\top} \bigoplus_{J=|k-\ell|}^{k+\ell} \varphi_J^{\ell k}(\|\mathbf{x}\|)\mathbf{Y}_J(\mathbf{x}), \tag{43}$$

The radial functions $\varphi_J^{\ell k}(\|\mathbf{x}\|)$ act as an independent, learnable scalar factor for each degree J. The vectorised matrix has dimensionality $(2\ell+1)(2k+1)$. We can unvectorise the above yielding

$$\mathbf{W}^{\ell k}(\mathbf{x}) = \mathrm{unvec}\left(\mathbf{Q}^{\ell k\top} \bigoplus_{J=|k-\ell|}^{k+\ell} \varphi_J^{\ell k}(\|\mathbf{x}\|)\mathbf{Y}_J(\mathbf{x})\right) \tag{44}$$

$$= \sum_{J=|k-\ell|}^{k+\ell} \varphi_J^{\ell k}(\|\mathbf{x}\|)\mathrm{unvec}\left(\mathbf{Q}_J^{\ell k\top}\mathbf{Y}_J(\mathbf{x})\right) \tag{45}$$

where $\mathbf{Q}_J^{\ell k}$ is a $(2\ell+1)(2k+1) \times (2J+1)$ slice from $\mathbf{Q}^{\ell k}$, corresponding to spherical harmonic $\mathbf{Y}_J$. As we showed in the main text, we can also rewrite the unvectorised Clebsch-Gordan–spherical harmonic matrix-vector product as

$$\mathrm{unvec}\left(\mathbf{Q}_J^{\ell k\top}\mathbf{Y}_J(\mathbf{x})\right) = \sum_{m=-J}^{J} \mathbf{Q}_{Jm}^{\ell k\top}Y_{Jm}(\mathbf{x}). \tag{46}$$

In contrast to Weiler et al. [37], we do not voxelise space and therefore $\mathbf{x}$ will be different for each pair of points in each point cloud. However, the same $\mathbf{Y}_J(\mathbf{x})$ will be used multiple times in the network and even multiple times in the same layer. Hence, precomputing them at the beginning of each forward pass for the entire network can significantly speed up the computation. The Clebsch-Gordan coefficients do not depend on the relative positions and can therefore be precomputed once and stored on disk. Multiple libraries exist which approximate those coefficients numerically.

(a) Speed comparison on the CPU.
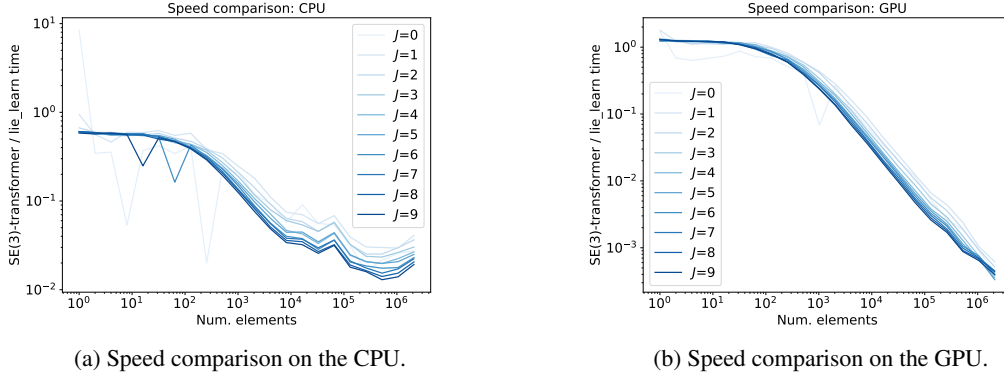
(b) Speed comparison on the GPU.

Figure 5: Spherical harmonics computation of our own implementation compared to the `lie-learn` library. We found that speeding up the computation of spherical harmonics is critical to scale up both Tensor Field Networks [28] and SE(3)-Transformers to solve real-world machine learning tasks.

C Accelerated Computation of Spherical Harmonics

The spherical harmonics (SH) typically have to be computed on the fly for point cloud methods based on irreducible computations, a bottleneck of TFNs [28]. Thomas et al. [28] ameliorate this by restricting the maximum type of feature to type-2, trading expressivity for speed. Weiler et al. [37] circumvent this challenge by voxelising the input, allowing them to pre-compute spherical harmonics for fixed relative positions. This is at the cost of detail as well as exact rotation and translation equivariance.

The number of spherical harmonic lookups in a network based on irreducible representations can quickly become large (number of layers $\times$ number of points $\times$ number of neighbours $\times$ number of channels $\times$ number of degrees needed). This motivates parallelised computation on the GPU - a feature not supported by common libraries. To that end, we wrote our own spherical harmonics library in `Pytorch`, which can generate spherical harmonics on the GPU. We found this critical to being able to run the SE(3)-Transformer and Tensor Field network baselines in a reasonable time. This library is accurate to within machine precision against the `scipy` counterpart `scipy.special.sph_harm` and is significantly faster. E.g., for a ScanObjectNN model, we achieve $\sim 22\times$ speed up of the forward pass compared to a network built with SH from the `lielearn` library. A speed comparison isolating the computation of the spherical harmonics is shown in Fig. 5. Code is available at `https://github.com/FabianFuchsML/se3-transformer-public`. In the following, we outline our method to generate them.

The tesseral/real spherical harmonics are given as

$$Y_{Jm}(\theta, \phi) = \sqrt{\frac{2J+1}{4\pi} \frac{(J-m)!}{(J+m)!}} P_J^{|m|}(\cos\theta) \cdot \begin{cases} \sin(|m|\phi) & m < 0, \\ 1 & m = 0, \\ \cos(m\phi) & m > 0, \end{cases} \quad (47)$$

where $P_J^{|m|}$ is the associated Legendre polynomial (ALP), $\theta \in [0, 2\pi)$ is azimuth, and $\phi \in [0, \pi]$ is a polar angle. The term $P_J^{|m|}$ is by far the most expensive component to compute and can be computed recursively. To speed up the computation, we use dynamic programming storing intermediate results in a memoization.

We make use of the following recursion relations in the computation of the ALPs:

$$P_J^{|J|}(x) = (-1)^{|J|} \cdot (1 - x^2)^{|J|/2} \cdot (2|J| - 1)!! \qquad \text{boundary: } J = m \quad (48)$$

$$P_J^{-m}(x) = (-1)^J \frac{(\ell - m)!}{(\ell + m)!} P_J^m(x) \qquad \text{negate } m \quad (49)$$

$$P_J^{|m|}(x) = \frac{2J-1}{J-|m|} x P_{J-1}^m(x) + \mathbb{I}[J - |m| > 1] \frac{J + |m| - 1}{J - |m|} P_{J-2}^m(x) \quad \text{recurse} \quad (50)$$

where the *semifactorial* is defined as $x!! = x(x-2)(x-4)\cdots$, and $\mathbb{I}$ is the indicator function. These relations are helpful because they define a recursion.

17

m   -4 -3 -2 -1 0 1 2 3 4

J

● ● ● ● ● ● ● ● ●   4

● ● ● ● ● ● ●   3

● ● ● ● ●   2

$P_3^{-1}(x)$ ● ● ● ● ●   1

●   0

Figure 6: Subproblem graph for the computatin of the associated Legendre polynomials. To compute $P_3^{-1}(x)$, we compute $P_3^1(x)$, for which we need $P_2^1(x)$ and $P_1^1(x)$. We store each intermediate computation, speeding up average computation time by a factor of ~ 10 on CPU.

To understand how we recurse, we consider an example. Fig. 6 shows the space of J and m. The black vertices represent a particular ALP, for instance, we have highlighted $P_3^{-1}(x)$. When $m < 0$, we can use Eq. (49) to compute $P_3^{-1}(x)$ from $P_3^1(x)$. We can then use Eq. (50) to compute $P_3^1(x)$ from $P_2^1(x)$ and $P_1^1(x)$. $P_2^1(x)$ can also be computed from Eq. (50) and the boundary value $P_1^1(x)$ can be computed directly using Eq. (48). Crucially, all intermediate ALPs are stored for reuse. Say we wanted to compute $P_4^{-1}(x)$, then we could use Eq. (49) to find it from $P_4^{-1}(x)$, which can be recursed from the stored values $P_3^1(x)$ and $P_2^1(x)$, without needing to recurse down to the boundary.

D   Experimental Details

D.1   ScanObjectNN

D.1.1   SE(3)-Transformer and Tensor Field Network

A particularity of object classification from point clouds is the large number of points the algorithm needs to handle. We use up to 200 points out of the available 2024 points per sample and create neighbourhoods with up to 40 nearest neighbours. It is worth pointing out that especially in this setting, adding self-attention (i.e. when comparing the SE(3) Transformer to Tensor Field Networks) significantly increased the stability. As a result, whenever we swapped out the attention mechanism for a convolution to retrieve the Tensor Field network baseline, we had to decrease the model size to obtain stable training. However, we would like to stress that all the Tensor Field networks we trained were significantly bigger than in the original paper [28], mostly enabled by the faster computation of the spherical harmonics.

For the ablation study in Fig. 4, we trained networks with 4 hidden equivariant layers with 5 channels each, and up to representation degree 2. This results in a hidden feature size per point of

$$5 \cdot \sum_{\ell=0}^{2} (2\ell + 1) = 45 \tag{51}$$

We used 200 points of the point cloud and neighbourhood size 40. For the Tensor Field network baseline, in order to achieve stable training, we used a smaller model with 3 instead of 5 channels, 100 input points and neighbourhood size 10, but with representation degrees up to 3.

We used 1 head per attention mechanism yielding one attention weight for each pair of points but across all channels and degrees (for an implementation of multi-head attention, see Appendix D.3). For the query embedding, we used the identity matrix. For the key embedding, we used a square equivariant matrix preserving the number of degrees and channels per degree.

For the quantitative comparison to the start-of-the-art in Table 2, we used 128 input points and neighbourhood size 10 for both the Tensor Field network baseline and the SE(3)-Transformer. We used farthest point sampling with a random starting point to retrieve the 128 points from the overall

point cloud. We used degrees up to 3 and 5 channels per degree, which we again had to reduce to 3 channels for the Tensor Field network to obtain stable training. We used a norm based non-linearity for the Tensor Field network (as in [28]) and no extra non-linearity (beyond the *softmax* in the self-attention algorithm) for the SE(3) Transformer.

For all experiments, the final layer of the equivariant encoder maps to 64 channels of degree 0 representations. This yields a 64-dimensional SE(3) *invariant* representation per point. Next, we pool over the point dimension followed by an MLP with one hidden layer of dimension 64, a ReLU and a 15 dimensional output with a cross entropy loss. We trained for 60000 steps with batch size 10. We used the Adam optimizer [13] with a start learning of `1e-2` and a reduction of the learning rate by 70% every 15000 steps. Training took up to 2 days on a system with 4 CPU cores, 30 GB of RAM, and an NVIDIA GeForce GTX 1080 Ti GPU.

The input to the Tensorfield network and the Se(3) Transformer are relative x-y-z positions of each point w.r.t. their neighbours. To guarantee equivariance, these inputs are provided as fields of degree 1. For the '+z' versions, however, we deliberately break the SE(3) equivariance by providing additional and relative z-position as two additional scalar fields (i.e. degree 0), as well as relative x-y positions as a degree 1 field (where the z-component is set to 0).

D.1.2   Number of Input Points

Limiting the input to 128/200 points in our experiments on ScanObjectNN was not primarily due to computational limitations: we conducted experiments with up to 2048 points, but without performance improvements. We suspect this is due to the global pooling. Examining cascaded pooling via attention is a future research direction. Interestingly, when limiting other methods to using 128 points, the SE(3)-Transformer outperforms the baselines (PointCNN: $80.3 \pm 0.8\%$, PointGLR: $81.5 \pm 1.0\%$, DGCNN: $82.2 \pm 0.8\%$, **ours**: $85.0 \pm 0.7\%$). It is worth noting that these methods were explicitly designed for the well-studied task of point classification whereas the SE(3)-Transformer was applied as is. Combining different elements from the current state-of-the-art methods with the geometrical inductive bias of the SE(3)-Transformer could potentially yield additional performance gains, especially with respect to leveraging inputs with more points. It is also worth noting that the SE(3)-Transformer was remarkably stable with respect to lowering the number of input points in an ablation study (16 points: 79.2%, 32 points: 81.4%, 64 points: 82.5%, 128 points: 85.0%, 256 points: 82.6%).

D.1.3   Sample Complexity

Equivariance is known to often lead to smaller sample complexity, meaning that less training data is needed (Fig. 10 in Worrall et al. [40], Fig. 4 in Winkels and Cohen [38], Fig. 4 in Weiler et al. [37]). We conducted experiments with different amounts of training samples $N_{samples}$ from the ScanObjectNN dataset. The results showed that for all $N_{samples}$, the SE(3)-Transformer outperformed the Set Transformer, a non-equivariant network based on attention. The performance delta was also slightly higher for the smallest $N_{samples}$ we tested (3.1% of the samples available in the training split of ScanObjectNN) than when using all the data indicating that the SE(3)-Transformer performs particularly well on small amounts of training data. However, performance differences can be due to multiple reasons. Especially for small datasets, such as ScanObjectNN, where the performance does not saturate with respect to amount of data available, it is difficult to draw conclusions about sample complexity of one model versus another. In summary, we found that our experimental results are in line with the claim that equivariance decreases sample complexity in this specific case but do not give definitive support.

D.1.4   Baselines

DeepSet Baseline   We originally replicated the implementation proposed in [46] for their object classification experiment on ModelNet40 [41]. However, most likely due to the relatively small number of objects in the ScanObjectNN dataset, we found that reducing the model size helped the performance significantly. The reported model had 128 units per hidden layer (instead of 256) and no dropout but the same number of layers and type of non-linearity as in [46].

Set Transformer Baseline   We used the same architecture as [16] in their object classification experiment on ModelNet40 [41] with an ISAB (induced set attention block)-based encoder followed by PMA (pooling by multihead attention) and an MLP.

D.2 Relational Inference

Following Kipf et al. [14], we simulated trajectories for 5 charged, interacting particles. Instead of a 2d simulation setup, we considered a 3d setup. Positive and negative charges were drawn as Bernoulli trials ($p = 0.5$). We used the provided code base `https://github.com/ethanfetaya/nri` with the following modifications: While we randomly sampled initial positions inside a $[-5, 5]^3$ box, we removed the bounding-boxes during the simulation. We generated 5k simulation samples for training and 1k for testing. Instead of phrasing it as a time-series task, we posed it as a regression task: The input data is positions and velocities at a random time step as well as the signs of the charges. The labels (which the algorithm is learning to predict) are the positions and velocities 500 simulation time steps into the future.

Training Details   We trained each model for 100,000 steps with batch size 128 using an Adam optimizer [13]. We used a fixed learning rate throughout training and conducted a separate hyper parameter search for each model to find a suitable learning rate.

SE(3)-Transformer Architecture   We trained an SE(3)-Transformer with 4 equivariant layers, where the hidden layers had representation degrees $\{0, 1, 2, 3\}$ and 3 channels per degree. The input is handled as two type-1 fields (for positions and velocities) and one type-0 field (for charges). The learning rate was set to `3e-3`. Each layer included attentive self-interaction.

We used 1 head per attention mechanism yielding one attention weight for each pair of points but across all channels and degrees (for an implementation of multi-head attention, see Appendix D.3). For the query embedding, we used the identity matrix. For the key embedding, we used a square equivariant matrix preserving the number of degrees and channels per degree.

Baseline Architectures   All our baselines fulfill permutation invariance (ordering of input points), but only the Tensor Field network and the linear baseline are SE(3) equivariant. For the **Tensor Field Network**[28] baseline, we used the same hyper parameters as for the SE(3) Transformer but with a linear self-interaction and an additional norm-based nonlinearity in each layer as in Thomas et al. [28]. For the **DeepSet**[46] baseline, we used 3 fully connected layers, a pooling layer, and two more fully connected layers with 64 units each. All fully connected layers act pointwise. The pooling layer uses max pooling to aggregate information from all points, but concatenates this with a skip connection for each point. Each hidden layer was followed by a LeakyReLU. The learning rate was set to `1e-3`. For the **Set Transformer**[16], we used 4 self-attention blocks with 64 hidden units and 4 heads each. For each point this was then followed by a fully connected layer (64 units), a LeakyReLU and another fully connected layer. The learning rate was set to `3e-4`.

For the **linear baseline**, we simply propagated the particles linearly according to the simulation hyperparamaters. The linear baseline can be seen as removing the interactions between particles from the prediction. Any performance improvement beyond the linear baseline can therefore be interpreted as an indication for relational reasoning being performed.

D.3 QM9

The QM9 regression dataset [21] is a publicly available chemical property prediction task consisting of 134k small drug-like organic molecules with up to 29 atoms per molecule. There are 5 atomic species (Hydrogen, Carbon, Oxygen, Nitrogen, and Flourine) in a molecular graph connected by chemical bonds of 4 types (single, double, triple, and aromatic bonds). 'Positions' of each atom, measured in ångtröms, are provided. We used the exact same train/validation/test splits as Anderson et al. [1] of sizes 100k/18k/13k.

The architecture we used is shown in Table 4. It consists of 7 multihead attention layers interspersed with norm nonlinearities, followed by a TFN layer, max pooling, and two linear layers separated by a ReLU. For each attention layer, shown in Fig. 7, we embed the input to half the number of feature channels before applying multiheaded attention [31]. Multiheaded attention is a variation of attention, where we partition the queries, keys, and values into H *attention heads*. So if our embeddings have dimensionality $(4, 16)$ (denoting 4 feature types with 16 channels each) and we use $H = 8$ attention heads, then we partition the embeddings to shape $(4, 2)$. We then combine each of the 8 sets of shape $(4, 2)$ queries, keys, and values individually and then concatenate the results into a single vector of the original shape $(4, 16)$. The keys and queries are edge embeddings, and thus the embedding matrices are of TFN type (c.f. Eq. (8)). For TFN type layers, the radial functions are learnable maps. For these we used neural networks with architecture shown in Table 5.

For the norm nonlinearities [40], we use

$$\texttt{Norm ReLU}(\mathbf{f}^\ell) = \texttt{ReLU}(\texttt{LN}\left(\|\mathbf{f}^\ell\|\right)) \cdot \frac{\mathbf{f}^\ell}{\|\mathbf{f}^\ell\|}, \qquad \text{where } \|\mathbf{f}^\ell\| = \sqrt{\sum_{m=-\ell}^{\ell} (f_m^\ell)^2}, \qquad (52)$$

where $\texttt{LN}$ is layer norm [2] applied across all features within a feature type. For the TFN baseline, we used the exact same architecture but we replaced each of the multiheaded attention blocks with a TFN layer with the same output shape.

The input to the network is a sparse molecular graph, with edges represented by the molecular bonds. The node embeedings are a 6 dimensional vector composed of a 5 dimensional one-hot embedding of the 5 atomic species and a 1 dimension integer node embedding for number of protons per atom. The edges embeddings are a 5 dimensional vector consisting of a 4 dimensional one-hot embedding of bond type and a positive scalar for the Euclidean distance between the two atoms at the ends of the bond. For each regression target, we normalised the values by mean and dividing by the standard deviation of the training set.

We trained for 50 epochs using Adam [13] at initial learning rate $\texttt{1e-3}$ and a single-cycle cosine rate-decay to learning rate $\texttt{1e-4}$. The batch size was 32, but for the TFN baseline we used batch size 16, to fit the model in memory. We show results on the 6 regression tasks not requiring thermochemical energy subtraction in Table 3. As is common practice, we optimised architectures and hyperparameters on $\varepsilon_{\text{HOMO}}$ and retrained each network on the other tasks. Training took about 2.5 days on an NVIDIA GeForce GTX 1080 Ti GPU with 4 CPU cores and 15 GB of RAM.

Table 4: QM9 Network architecture: d_{out} is the number of feature types of degrees $0, 1, ..., d_{\text{out}} - 1$ at the output of the corresponding layer and C is the number of multiplicities/channels per feature type. For the norm nonlinearity we use ReLUs, preceded by equivariant layer norm [37] with learnable affine transform.

No. repeats	Layer Type	d_{out}	C
1x	Input	1	6
1x	Attention: 8 heads	4	16
	Norm Nonlinearity	4	16
6x	Attention: 8 heads	4	16
	Norm Nonlinearity	4	16
1x	TFN layer	1	128
1x	Max pool	1	128
1x	Linear	1	128
1x	ReLU	1	128
1x	Linear	1	1

Table 5: QM9 Radial Function Architecture. C is the number of output channels at each layer. Layer norm [2] is computed per pair of input and output feature types, which have C_{in} and C_{out} channels each.

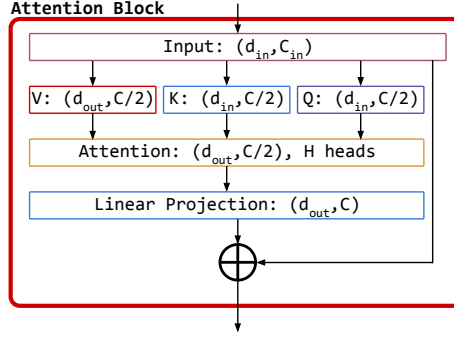| Layer Type	C
Input	6
Linear	32
Layer Norm	32
ReLU	32
Linear	32
Layer Norm	32
ReLU	32
Linear	$d_{\text{out}} * C_{\text{in}} * C_{\text{out}}$

Figure 7: Attention block for the QM9 dataset. Each component is listed with a tuple of numbers representing the output feature types and multiplicities, so $(4, 32)$ means feature types $0, 1, 2, 3$ (with dimensionalities $1, 3, 5, 7$), with 32 channels per type.

D.4   General Remark

Across experiments on different datasets with the SE(3)-Transformer, we made the observation that the number of representation degrees have a significant but saturating impact on performance. A big improvement was observed when switching from degrees $\{0, 1\}$ to $\{0, 1, 2\}$. Adding type-3 latent representations gave small improvements, further representation degrees did not change the performance of the model. However, higher representation degrees have a significant impact on memory usage and computation time. We therefore recommend representation degrees up to 2, when computation time and memory usage is a concern, and 3 otherwise.