arXiv:2102.09844v2 [cs.LG] 9 May 2021

E(n) Equivariant Graph Neural Networks

Victor Garcia Satorras [1] **Emiel Hoogeboom** [1] **Max Welling** [1]

Abstract

This paper introduces a new model to learn graph neural networks equivariant to rotations, translations, reflections and permutations called E(n)-Equivariant Graph Neural Networks (EGNNs). In contrast with existing methods, our work does not require computationally expensive higher-order representations in intermediate layers while it still achieves competitive or better performance. In addition, whereas existing methods are limited to equivariance on 3 dimensional spaces, our model is easily scaled to higher-dimensional spaces. We demonstrate the effectiveness of our method on dynamical systems modelling, representation learning in graph autoencoders and predicting molecular properties.

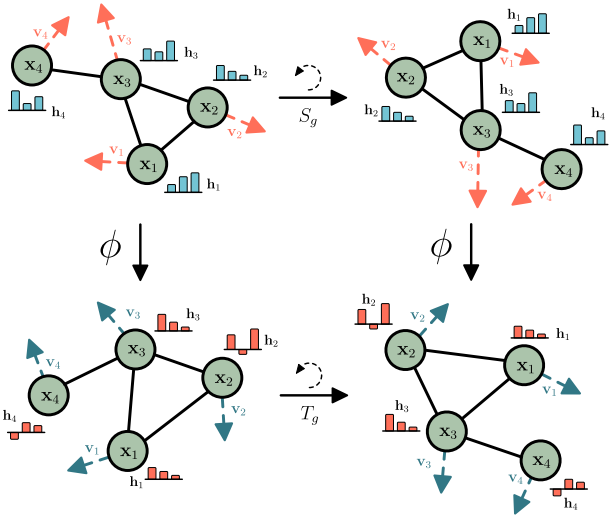

Figure 1. Example of rotation equivariance on a graph with a graph neural network ϕ

1. Introduction

Although deep learning has largely replaced hand-crafted features, many advances are critically dependent on inductive biases in deep neural networks. An effective method to restrict neural networks to relevant functions is to exploit the *symmetry* of problems by enforcing equivariance with respect to transformations from a certain symmetry group. Notable examples are translation equivariance in Convolutional Neural Networks and permutation equivariance in Graph Neural Networks (Bruna et al., 2013; Defferrard et al., 2016; Kipf & Welling, 2016a).

Many problems exhibit 3D translation and rotation symmetries. Some examples are point clouds (Uy et al., 2019), 3D molecular structures (Ramakrishnan et al., 2014) or N-body particle simulations (Kipf et al., 2018). The group corresponding to these symmetries is named the Euclidean group: SE(3) or when reflections are included E(3). It is often desired that predictions on these tasks are either equivariant or invariant with respect to E(3) transformations.

Recently, various forms and methods to achieve E(3) or SE(3) equivariance have been proposed (Thomas et al., 2018; Fuchs et al., 2020; Finzi et al., 2020; Köhler et al., 2020). Many of these works achieve innovations in studying types of higher-order representations for intermediate network layers. However, the transformations for these higher-order representations require coefficients or approximations that can be expensive to compute. Additionally, in practice for many types of data the inputs and outputs are restricted to scalar values (for instance temperature or energy, referred to as type-0 in literature) and 3d vectors (for instance velocity or momentum, referred to as type-1 in literature).

In this work we present a new architecture that is translation, rotation and reflection equivariant (E(n)), and permutation equivariant with respect to an input set of points. Our model is simpler than previous methods in that it does not require the spherical harmonics as in (Thomas et al., 2018; Fuchs et al., 2020) while it can still achieve competitive or better results. In addition, equivariance in our model is not limited to the 3-dimensional space and can be scaled to larger dimensional spaces without a significant increase in computation.

[1]UvA-Bosch Delta Lab, University of Amsterdam, Netherlands. Correspondence to: Victor Garcia Satorras <v.garciasatorras@uva.nl>, Emiel Hoogeboom <e.hoogeboom@uva.nl>, Max Welling <m.welling@uva.nl>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

We evaluate our method in modelling dynamical systems, representation learning in graph autoencoders and predicting molecular properties in the QM9 dataset. Our method reports the best or very competitive performance in all three experiments.

2. Background

In this section we introduce the relevant materials on equivariance and graph neural networks which will later complement the definition of our method.

2.1. Equivariance

Let $T_g : X \to X$ be a set of transformations on X for the abstract group $g \in G$. We say a function $\phi : X \to Y$ is equivariant to g if there exists an equivalent transformation on its output space $S_g : Y \to Y$ such that:

$$\phi(T_g(\mathbf{x})) = S_g(\phi(\mathbf{x})) \tag{1}$$

As a practical example, let $\phi(\cdot)$ be a non-linear function, $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_M) \in \mathbb{R}^{M \times n}$ an input set of M point clouds embedded in a n-dimensional space, $\phi(\mathbf{x}) = \mathbf{y} \in \mathbb{R}^{M \times n}$ the transformed set of point clouds, T_g a translation on the input set $T_g(\mathbf{x}) = \mathbf{x} + g$ and S_g an equivalent translation on the output set $S_g(\mathbf{y}) = \mathbf{y} + g$. If our transformation $\phi : X \to Y$ is translation equivariant, translating the input set $T_g(\mathbf{x})$ and then applying the function $\phi(T_x(\mathbf{x}))$ on it, will deliver the same result as first running the function $y = \phi(\mathbf{x})$ and then applying an equivalent translation to the output $T_g(\mathbf{y})$ such that Equation 1 is fulfilled and $\phi(\mathbf{x}+g) = \phi(\mathbf{x}) + g$. In this work we explore the following three types of equivariance on a set of particles $\mathbf{x}$:

1. Translation equivariance. Translating the input by $g \in \mathbb{R}^n$ results in an equivalent translation of the output. Let $\mathbf{x}+g$ be shorthand for $(\mathbf{x}_1+g, \dots, \mathbf{x}_M+g)$. Then $\mathbf{y} + g = \phi(\mathbf{x} + g)$
2. Rotation (and reflection) equivariance. For any orthogonal matrix $Q \in \mathbb{R}^{n \times n}$, let $Q\mathbf{x}$ be shorthand for $(Q\mathbf{x}_1, \dots, Q\mathbf{x}_M)$. Then rotating the input results in an equivalent rotation of the output $Q\mathbf{y} = \phi(Q\mathbf{x})$.
3. Permutation equivariance. Permuting the input results in the same permutation of the output $P(\mathbf{y}) = \phi(P(\mathbf{x}))$ where P is a permutation on the row indexes.

Note that velocities $\mathbf{v} \in \mathbb{R}^{M \times n}$ are unaffected by translations, but they transform equivalently under rotation (2) and permutation (3). Our method introduced in Section 3 will satisfy the three above mentioned equivariant constraints.

2.2. Graph Neural Networks

Graph Neural Networks are permutation equivariant networks that operate on graph structured data (Bruna et al., 2013; Defferrard et al., 2016; Kipf & Welling, 2016a). Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $v_i \in \mathcal{V}$ and edges $e_{ij} \in \mathcal{E}$ we define a graph convolutional layer following notation from (Gilmer et al., 2017) as:

$$\begin{aligned} \mathbf{m}_{ij} &= \phi_e(\mathbf{h}_i^l, \mathbf{h}_j^l, a_{ij}) \\ \mathbf{m}_i &= \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij} \\ \mathbf{h}_i^{l+1} &= \phi_h(\mathbf{h}_i^l, \mathbf{m}_i) \end{aligned} \tag{2}$$

Where $\mathbf{h}_i^l \in \mathbb{R}^{\text{nf}}$ is the nf-dimensional embedding of node v_i at layer l. a_{ij} are the edge attributes. $\mathcal{N}(i)$ represents the set of neighbors of node v_i. Finally, ϕ_e and ϕ_h are the edge and node operations respectively which are commonly approximated by Multilayer Perceptrons (MLPs).

3. Equivariant Graph Neural Networks

In this section we present Equivariant Graph Neural Networks (EGNNs). Following the notation from background Section 2.2, we consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $v_i \in \mathcal{V}$ and edges $e_{ij} \in \mathcal{E}$. In addition to the feature node embeddings $\mathbf{h}_i \in \mathbb{R}^{\text{nf}}$ we now also consider a n-dimensional coordinate $\mathbf{x}_i \in \mathbb{R}^{\text{n}}$ associated with each of the graph nodes. Our model will preserve equivariance to rotations and translations on these set of coordinates $\mathbf{x}_i$ and it will also preserve equivariance to permutations on the set of nodes $\mathcal{V}$ in the same fashion as GNNs.

Our Equivariant Graph Convolutional Layer (EGCL) takes as input the set of node embeddings $\mathbf{h}^l = \{\mathbf{h}_0^l, \dots, \mathbf{h}_{M-1}^l\}$, coordinate embeddings $\mathbf{x}^l = \{\mathbf{x}_0^l, \dots, \mathbf{x}_{M-1}^l\}$ and edge information $\mathcal{E} = (e_{ij})$ and outputs a transformation on $\mathbf{h}^{l+1}$ and $\mathbf{x}^{l+1}$. Concisely: $\mathbf{h}^{l+1}, \mathbf{x}^{l+1} = \text{EGCL}[\mathbf{h}^l, \mathbf{x}^l, \mathcal{E}]$. The equations that define this layer are the following:

$$\mathbf{m}_{ij} = \phi_e\left(\mathbf{h}_i^l, \mathbf{h}_j^l, \left\|\mathbf{x}_i^l - \mathbf{x}_j^l\right\|^2, a_{ij}\right) \tag{3}$$

$$\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + C \sum_{j \neq i} \left(\mathbf{x}_i^l - \mathbf{x}_j^l\right) \phi_x\left(\mathbf{m}_{ij}\right) \tag{4}$$

$$\mathbf{m}_i = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij} \tag{5}$$

$$\mathbf{h}_i^{l+1} = \phi_h\left(\mathbf{h}_i^l, \mathbf{m}_i\right) \tag{6}$$

Notice the main differences between the above proposed method and the original Graph Neural Network from equation 2 are found in equations 3 and 4. In equation 3 we now input the relative squared distance between two coordinates $\|\mathbf{x}_i^l - \mathbf{x}_j^l\|^2$ into the edge operation ϕ_e. The embeddings $\mathbf{h}_i^l$, $\mathbf{h}_j^l$, and the edge attributes a_{ij} are also provided as input to the edge operation as in the GNN case. In our case the edge attributes will incorporate the edge values $a_{ij} = e_{ij}$, but they could also include additional edge information.

In Equation 4 we update the position of each particle $\mathbf{x}_i$ as a vector field in a radial direction. In other words, the position of each particle $\mathbf{x}_i$ is updated by the weighted sum of all relative differences $(\mathbf{x}_i - \mathbf{x}_j)_{\forall j}$. The weights of this sum are provided as the output of the function $\phi_x : \mathbb{R}^{\text{nf}} \to \mathbb{R}^1$ that takes as input the edge embedding $\mathbf{m}_{ij}$ from the previous edge operation and outputs a scalar value. C is chosen to be $1/(M-1)$, which divides the sum by its number of elements. This equation is the main difference of our model compared to standard GNNs and it is the reason why equivariances 1, 2 are preserved (proof in Appendix A). Despite its simplicity, this equivariant operation is very flexible since the embedding $\mathbf{m}_{ij}$ can carry information from the whole graph and not only from the given edge e_{ij}.

Finally, equations 5 and 6 follow the same updates than standard GNNs. Equation 5 just aggregates all the incoming messages from neighbor nodes $\mathcal{N}(i)$ to node v_i and Equation 6 performs the node operation ϕ_v which takes as input the aggregated messages $\mathbf{m}_i$, the node emedding $\mathbf{h}_i^l$ and outputs the updated node embedding $\mathbf{h}_i^{l+1}$.

3.1. Analysis on E(n) equivariance

In this section we analyze the equivariance properties of our model for $E(3)$ symmetries (i.e. properties 1 and 2 stated in section 2.1). In other words, our model should be translation equivariant on $\mathbf{x}$ for any translation vector $g \in \mathbb{R}^n$ and it should also be rotation and reflection equivariant on $\mathbf{x}$ for any orthogonal matrix $Q \in \mathbb{R}^{n \times n}$. More formally our model satisfies:

$$Q\mathbf{x}^{l+1} + g, \mathbf{h}^{l+1} = \text{EGCL}(Q\mathbf{x}^l + g, \mathbf{h}^l)$$

We provide a formal proof of this in Appendix A. Intuitively, let's consider a $\mathbf{h}^l$ feature which is already $\text{E}(n)$ invariant, then we can see that the resultant edge embedding $\mathbf{m}_{ij}$ from Equation 3 will also be $\text{E}(n)$ invariant, because in addition to $\mathbf{h}^l$, it only depends on squared distances $\|\mathbf{x}_i^l - \mathbf{x}_j^l\|^2$, which are $\text{E}(n)$ invariant. Next, Equation 4 computes $\mathbf{x}_i^{l+1}$ by a weighted sum of differences $(\mathbf{x}_i - \mathbf{x}_j)$ which is added to $\mathbf{x}_i$, this transforms as a type-1 vector and preserves equivariance (see Appendix A). Finally the last two equations 5 and 6 that generate the next layer node-embeddings $\mathbf{h}^{l+1}$ remain $\text{E}(n)$ invariant since they only depend on $\mathbf{h}^l$ and $\mathbf{m}_{ij}$ which, as we saw above, are $\text{E}(n)$ invariant. Therefore the output $\mathbf{h}^{l+1}$ is $\text{E}(n)$ invariant and $\mathbf{x}^{l+1}$ is $\text{E}(n)$ equivariant to $\mathbf{x}^l$. Inductively, a composition of EGCLs will also be equivariant.

3.2. Extending EGNNs for vector type representations

In this section we propose a slight modification to the presented method such that we explicitly keep track of the particle's momentum. In some scenarios this can be useful not only to obtain an estimate of the particle's velocity at every layer but also to provide an initial velocity value in those cases where it is not 0. We can include momentum to our proposed method by just replacing Equation 4 of our model with the following equation:

$$\begin{aligned} \mathbf{v}_i^{l+1} &= \phi_v\left(\mathbf{h}_i^l\right)\mathbf{v}_i^l + C\sum_{j\neq i}\left(\mathbf{x}_i^l - \mathbf{x}_j^l\right)\phi_x\left(\mathbf{m}_{ij}\right) \\ \mathbf{x}_i^{l+1} &= \mathbf{x}_i^l + \mathbf{v}_i^{l+1} \end{aligned} \tag{7}$$

Note that this extends the EGCL layer as $\mathbf{h}^{l+1}, \mathbf{x}^{l+1}, \mathbf{v}^{l+1} = \text{EGCL}[\mathbf{h}^l, \mathbf{x}^l, \mathbf{v}^l, \mathcal{E}]$. The only difference is that now we broke down the coordinate update (eq. 4) in two steps, first we compute the velocity $\mathbf{v}_i^{l+1}$ and then we use this velocity to update the position $\mathbf{x}_i^l$. The velocity $\mathbf{v}^l$ is scaled by a new function $\phi_v : \mathbb{R}^N \to \mathbb{R}^1$ that maps the node embedding $\mathbf{h}_i^l$ to a scalar value. Notice that if the initial velocity is set to zero ($\mathbf{v}_i^{(0)} = 0$), both equations 4 and 7 become exactly the same for the first layer $l = 0$ and they become equivalent for the next layers since ϕ_v just re-scales all the outputs of ϕ_x from the previous layers with a scalar value. We proof the equivariance of this variant of the model in Appendix B.1. This variant is used in experiment 5.1 where we have to provide the initial velocity of the system, and predict a relative position change.

3.3. Inferring the edges

Given a point cloud or a set of nodes, we may not always be provided with an adjacency matrix. In those cases we can assume a fully connected graph where all nodes exchange messages with each other, in other words, the neighborhood operator $\mathcal{N}(i)$ from Equation 5 would include all other nodes of the graph except for i. This fully connected approach may not scale well to large graphs where we may want to locally limit the exchange of messages to avoid an overflow of information.

Similarly to (Serviansky et al., 2020; Kipf et al., 2018) we present a simple solution to infer the relations/edges of the graph in our model. We can re-write the aggregation operation from our model (eq. 5) in the following way:

$$\mathbf{m}_i = \sum_{j\in\mathcal{N}(i)} \mathbf{m}_{ij} = \sum_{j\neq i} e_{ij}\mathbf{m}_{ij} \tag{8}$$

Where e_{ij} takes value 1 if there is an edge between nodes (i, j) and 0 otherwise. Notice this doesn't modify yet the original equation used in our model, it is just a change in notation. Now we can choose to approximate the relations e_{ij} with the following function $e_{ij} = \phi_{inf}(\mathbf{m}_{ij})$, where $\phi_{inf} : \mathbb{R}^{nf} \to [0, 1]^1$ resembles a linear layer followed by a sigmoid function that takes as input the current edge embedding and outputs a soft estimation of its edge value. This modification doesn't change the $\text{E}(n)$ properties of the

	GNN	Radial Field	TFN	Schnet	EGNN
Edge	$\mathbf{m}_{ij} = \phi_e(\mathbf{h}_i^l, \mathbf{h}_j^l, a_{ij})$	$\mathbf{m}_{ij} = \phi_{\text{rf}}(\|\mathbf{r}_{ij}^l\|)\mathbf{r}_{ij}^l$	$\mathbf{m}_{ij} = \sum_k \mathbf{W}^{lk}\mathbf{r}_{ji}^l\mathbf{h}_i^{lk}$	$\mathbf{m}_{ij} = \phi_{\text{cf}}(\|\mathbf{r}_{ij}^l\|)\phi_{\text{s}}(\mathbf{h}_j^l)$	$\mathbf{m}_{ij} = \phi_e(\mathbf{h}_i^l, \mathbf{h}_j^l, \|\mathbf{r}_{ij}^l\|^2, a_{ij})$ $\hat{\mathbf{m}}_{ij} = \mathbf{r}_{ij}^l\phi_x(\mathbf{m}_{ij})$
Agg.	$\mathbf{m}_i = \sum_{j\in\mathcal{N}(i)} \mathbf{m}_{ij}$	$\mathbf{m}_i = \sum_{j\neq i} \mathbf{m}_{ij}$	$\mathbf{m}_i = \sum_{j\neq i} \mathbf{m}_{ij}$	$\mathbf{m}_i = \sum_{j\neq i} \mathbf{m}_{ij}$	$\mathbf{m}_i = \sum_{j\in\mathcal{N}(i)} \mathbf{m}_{ij}$ $\hat{\mathbf{m}}_i = C\sum_{j\neq i} \hat{\mathbf{m}}_{ij}$
Node	$\mathbf{h}_i^{l+1} = \phi_h(\mathbf{h}_i^l, \mathbf{m}_i)$	$\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \mathbf{m}_i$	$\mathbf{h}_i^{l+1} = w^{ll}\mathbf{h}_i^l + \mathbf{m}_i$	$\mathbf{h}_i^{l+1} = \phi_h(\mathbf{h}_i^l, \mathbf{m}_i)$	$\mathbf{h}_i^{l+1} = \phi_h\left(\mathbf{h}_i^l, \mathbf{m}_i\right)$ $\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \hat{\mathbf{m}}_i$
	Non-equivariant	$\mathrm{E}(n)$-Equivariant	$\mathrm{SE}(3)$-Equivariant	$\mathrm{E}(n)$-Invariant	$\mathrm{E}(n)$-Equivariant

Table 1. Comparison over different works from the literature under the message passing framework notation. We created this table with the aim to provide a clear and simple way to compare over these different methods. The names from left to right are: Graph Neural Networks (Gilmer et al., 2017); Radial Field from Equivariant Flows (Köhler et al., 2019); Tensor Field Networks (Thomas et al., 2018); Schnet (Schütt et al., 2017b); and our Equivariant Graph Neural Network. The difference between two points is written $\mathbf{r}_{ij} = (\mathbf{x}_i - \mathbf{x}_j)$.

model since we are only operating on the messages $\mathbf{m}_{ij}$ which are already $\mathrm{E}(n)$ invariant.

4. Related Work

Group equivariant neural networks have demonstrated their effectiveness in a wide variety of tasks (Cohen & Welling, 2016; 2017; Weiler & Cesa, 2019; Rezende et al., 2019; Romero & Cordonnier, 2021). Recently, various forms and methods to achieve E(3) or SE(3) equivariance have been proposed. Thomas et al. (2018); Fuchs et al. (2020) utilize the spherical harmonics to compute a basis for the transformations, which allows transformations between higher-order representations. A downside to this method is that the spherical harmonics need to be recomputed which can be expensive. Currently, an extension of this method to arbitrary dimensions is unknown. Finzi et al. (2020) parametrize transformations by mapping kernels on the Lie Algebra. For this method the neural network outputs are in certain situations stochastic, which may be undesirable. Köhler et al. (2019); Köhler et al. (2020) propose an $\mathrm{E}(n)$ equivariant network to model 3D point clouds, but the method is only defined for positional data on the nodes without any feature dimensions.

Another related line of research concerns message passing algorithms on molecular data. (Gilmer et al., 2017) presented a message passing setting (or Graph Neural Network) for quantum chemistry, this method is permutation equivariant but not translation or rotation equivariant. (Kondor et al., 2018) extends the equivariance of GNNs such that each neuron transforms in a specific way under permutations, but this extension only affects its permutation group and not translations or rotations in a geometric space. Further works (Schütt et al., 2017b;a) build $\mathrm{E}(n)$ invariant message passing networks by inputting the relative distances between points. Klicpera et al. (2020b;a) in addition to relative distances it includes a modified message passing scheme analogous to Belief Propagation that considers angles and directional information equivariant to rotations. It also uses Bessel functions and spherical harmonics to construct and orthogonal basis. Anderson et al. (2019); Miller et al. (2020) include $SO(3)$ equivariance in its intermediate layers for modelling the behavior and properties of molecular data. Our method is also framed as a message passing framework but in contrast to these methods it achieves $\mathrm{E}(n)$ equivariance.

Relationship with existing methods

In Table 1 the EGNN equations are detailed together with some of its closest methods from the literature under the message passing notation from (Gilmer et al., 2017). This table aims to provide a simple way to compare these different algorithms. It is structured in three main rows that describe i) the edge ii) aggregation and iii) node update operations. The GNN algorithm is the same as the previously introduced in Section 2.1. Our EGNN algorithm is also equivalent to the description in Section 3 but notation has been modified to match the (edge, aggregation, node) format. In all equations $\mathbf{r}_{ij}^l = (\mathbf{x}_i - \mathbf{x}_j)^l$. Notice that except the EGNN, all algorithms have the same aggregation operation and the main differences arise from the edge operation. The algorithm that we call "Radial Field" is the $\mathrm{E}(n)$ equivariant update from (Köhler et al., 2019). This method is $\mathrm{E}(n)$ equivariant, however its main limitation is that it only operates on $\mathbf{x}$ and it doesn't propagate node features $\mathbf{h}$ among nodes. In the method ϕ_{rf} is modelled as an MLP. Tensor Field Networks (TFN) (Thomas et al., 2018) instead propagate the node embeddings $\mathbf{h}$ but it uses spherical harmonics to compute its learnable weight kernel $\mathbf{W}^{\ell k} : \mathbb{R}^3 \to \mathbb{R}^{(2\ell+1)\times(2k+1)}$ which preserves $SE(3)$ equivariance but is expensive to compute an limited to the 3 dimensional space. The SE(3) Transformer (Fuchs et al., 2020) (not included in this table), can be interpreted as an extension of TFN with attention. Schnet (Schütt et al., 2017b) can be interpreted as an $\mathrm{E}(n)$ invariant Graph Neural Network where ϕ_{cf} receives as input relative distances and outputs a continuous filter convolution that multiplies the neighbor embeddings $\mathbf{h}$. Our EGNN differs from these other methods in terms that it performs two different updates in each of the table rows, one related

to the embeddings $\mathbf{h}$ and another related to the coordinates $\mathbf{x}$, these two variables exchange information in the edge operation. In summary the EGNN can retain the flexibility of GNNs while remaining $\mathrm{E}(n)$ equivariant as the Radial Field algorithm and without the need to compute expensive operations (i.e. spherical harmonics).

5. Experiments

5.1. Modelling a dynamical system — N-body system

In a dynamical system a function defines the time dependence of a point or set of points in a geometrical space. Modelling these complex dynamics is crucial in a variety of applications such as control systems (Chua et al., 2018), model based dynamics in reinforcement learning (Nagabandi et al., 2018), and physical systems simulations (Grzeszczuk et al., 1998; Watters et al., 2017). In this experiment we forecast the positions for a set of particles which are modelled by simple interaction rules, yet can exhibit complex dynamics.

Similarly to (Fuchs et al., 2020), we extended the Charged Particles N-body experiment from (Kipf et al., 2018) to a 3 dimensional space. The system consists of 5 particles that carry a positive or negative charge and have a position and a velocity associated in 3-dimensional space. The system is controlled by physic rules: particles are attracted or repelled depending on their charges. This is an equivariant task since rotations and translations on the input set of particles result in the same transformations throughout the entire trajectory.

Dataset: We sampled 3.000 trajectories for training, 2.000 for validation and 2.000 for testing. Each trajectory has a duration of 1.000 timesteps. For each trajectory we are provided with the initial particle positions $\mathbf{p}^{(0)} = \{\mathbf{p}_1^{(0)}, \dots \mathbf{p}_5^{(0)}\} \in \mathbb{R}^{5\times 3}$, their initial velocities $\mathbf{v}^{(0)} = \{\mathbf{v}_1^{(0)}, \dots \mathbf{v}_5^{(0)}\} \in \mathbb{R}^{5\times 3}$ and their respective charges $\mathbf{c} = \{c_1, \dots c_5\} \in \{-1, 1\}^5$. The task is to estimate the positions of the five particles after 1.000 timesteps. We optimize the averaged Mean Squared Error of the estimated position with the ground truth one.

Implementation details: In this experiment we used the extension of our model that includes velocity from section 3.2. We input the position $\mathbf{p}^{(0)}$ as the first layer coordinates $\mathbf{x}^0$ of our model and the velocity $\mathbf{v}^{(0)}$ as the initial velocity in Equation 7, the norms $\|\mathbf{v}_i^{(0)}\|$ are also provided as features to $\mathbf{h}_i^0$ through a linear mapping. The charges are input as edge attributes $a_{ij} = c_i c_j$. The model outputs the last layer coordinates $\mathbf{x}^L$ as the estimated positions. We compare our method to its non equivariant Graph Neural Network (GNN) cousin, and the equivariant methods: Radial Field (Köhler et al., 2019), Tensor Field Networks and the SE(3) Transformer. All algorithms are composed of 4 layers and have been trained under the same conditions, batch size 100, 10.000 epochs, Adam optimizer, the learning rate was tuned independently for each model. We used 64 features for the hidden layers in the Radial Field, the GNN and our EGNN. As non-linearity we used the Swish activation function (Ramachandran et al., 2017). For TFN and the SE(3) Transformer we swept over different number of vector types and features and chose those that provided the best performance. Further implementation details are provided in Appendix C.1. A Linear model that simply considers the motion equation $\mathbf{p}^{(t)} = \mathbf{p}^{(0)} + \mathbf{v}^{(0)}t$ is also included as a baseline. We also provide the average forward pass time in seconds for each of the models for a batch of 100 samples in a GTX 1080 Ti GPU.

Method	MSE	Forward time (s)
Linear	0.0819	.0001
SE(3) Transformer	0.0244	.1346
Tensor Field Network	0.0155	.0343
Graph Neural Network	0.0107	.0032
Radial Field	0.0104	.0039
EGNN	**0.0071**	.0062

Table 2. Mean Squared Error for the future position estimation in the N-body system experiment, and forward time in seconds for a batch size of 100 samples running in a GTX 1080Ti GPU.

Results As shown in Table 2 our model significantly outperforms the other equivariant and non-equivariant alternatives while still being efficient in terms of running time. It reduces the error with respect to the second best performing method by a 32%. In addition it doesn't require the computation of spherical harmonics which makes it more time efficient than Tensor Field Networks and the SE(3) Transformer.

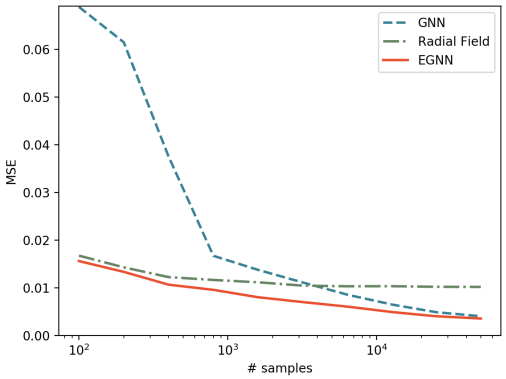

Figure 2. Mean Squared Error in the N-body experiment for the Radial Field, GNN and EGNN methods when sweeping over different amounts of training data.

Analysis for different number of training samples: We want to analyze the performance of our EGNN in the small and large data regime. In the following, we report on a similar experiment as above, but instead of using 3.000

training samples we generated a new training partition of 50.000 samples and we sweep over different amounts of data from 100 to 50.000 samples. We compare the performances of our EGNN vs its non-equivariant GNN counterpart and the Radial Field algorithm. Results are presented in Figure 2. Our method outperforms both Radial Field and GNNs in the small and large data regimes. This shows the EGNN is more data efficient than GNNs since it doesn't require to generalize over rotations and translations of the data while it ensembles the flexibility of GNNs in the larger data regime. Due to its high model bias, the Radial Field algorithm performs well when data is scarce but it is unable to learn the subtleties of the dataset as we increase the training size. In summary, our EGNN benefits from both the high bias of $\mathrm{E}(n)$ methods and the flexibility of GNNs.

5.2. Graph Autoencoder

A Graph Autoencoder can learn unsupervised representations of graphs in a continuous latent space (Kipf & Welling, 2016b; Simonovsky & Komodakis, 2018). In this experiment section we use our EGNN to build an Equivariant Graph Autoencoder. We will explain how Graph Autoencoders can benefit from equivariance and we will show how our method outperforms standard GNN autoencoders in the provided datasets. This problem is particularly interesting since the embedding space can be scaled to larger dimensions and is not limited to a 3 dimensional Euclidean space.

Similarly to the work of (Kipf & Welling, 2016b) further extended by section 3.3 in (Liu et al., 2019), our graph autoencoder $\mathbf{z} = q(\mathcal{G})$ embeds a graph $\mathcal{G}$ into a set of latent nodes $\mathbf{z} = \{\mathbf{z}_1, \dots \mathbf{z}_M\} \in \mathbb{R}^{M \times n}$, where M is the number of nodes and n the embedding size per node. Notice this may reduce the memory complexity to store the graphs from $O(M^2)$ to $O(Mn)$ where n may depend on M for a certain approximation error tolerance. This differs from the variational autoencoder proposed in (Simonovsky & Komodakis, 2018) which embeds the graph in a single vector $\mathbf{z} \in \mathbb{R}^K$, which causes the reconstruction to be computationally very expensive since the nodes of the decoded graph have to be matched again to the ground truth. In addition to the introduced graph generation and representation learning methods, it is worth it mentioning that in the context of graph compression other methods (Candès & Recht, 2009) can be used.

More specifically, we will compare our Equivariant Graph Auto-Encoder in the task presented in (Liu et al., 2019) where a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with node features $H \in \mathbb{R}^{M \times \mathrm{nf}}$ and adjacency matrix $A \in \{0, 1\}^{M \times M}$ is embedded into a latent space $\mathbf{z} = q(H, A) \in \mathbb{R}^{M \times n}$. Following (Kipf & Welling, 2016b; Liu et al., 2019), we are only interested in reconstructing the adjacency matrix A since the datasets we will work with do not contain node features. The decoder $g(\cdot)$ proposed by (Liu et al., 2019) takes as input the embedding space $\mathbf{z}$ and outputs the reconstructed adjacency matrix $\hat{A} = g(\mathbf{z})$, this decoder function is defined as follows:

$$\hat{A}_{ij} = g_e(\mathbf{z}_i, \mathbf{z}_j) = \frac{1}{1 + \exp(w \left\|\mathbf{z}_i - \mathbf{z}_j\right\|^2 + b)} \tag{9}$$

Where w and b are its only learnable parameters and $g_e(\cdot)$ is the decoder edge function applied to every pair of node embeddings. It reflects that edge probabilities will depend on the relative distances among node embeddings. The training loss is defined as the binary cross entropy between the estimated and the ground truth edges $\mathcal{L} = \sum_{ij} \mathrm{BCE}(\hat{A}_{ij}, A_{ij})$.

The symmetry problem: The above stated autoencoder may seem straightforward to implement at first sight but in some cases there is a strong limitation regarding the symmetry of the graph. Graph Neural Networks are convolutions on the edges and nodes of a graph, i.e. the same function is applied to all edges and to all nodes. In some graphs (e.g. those defined only by its adjacency matrix) we may not have input features in the nodes, and for that reason the difference among nodes relies only on their edges or neighborhood topology. Therefore, if the neighborhood of two nodes is exactly the same, their encoded embeddings will be the same too. A clear example of this is a cycle graph (an example of a 4 nodes cycle graph is provided in Figure 3). When running a Graph Neural Network encoder on a node featureless cycle graph, we will obtain the exact same embedding for each of the nodes, which makes it impossible to reconstruct the edges of the original graph from the node embeddings. The cycle graph is a severe example where all nodes have the exact same neighborhood topology but these symmetries can be present in different ways for other graphs with different edge distributions or even when including node features if these are not unique.

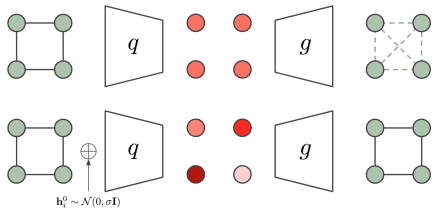

Figure 3. Visual representation of a Graph Autoencoder for a 4 nodes cycle graph. The bottom row illustrates that adding noise at the input graph breaks the symmetry of the embedding allowing the reconstruction of the adjacency matrix.

An ad-hoc method to break the symmetry of the graph is introduced by (Liu et al., 2019). This method introduces noise sampled from a Gaussian distribution into the input node features of the graph $\mathbf{h}_i^0 \sim \mathcal{N}(\mathbf{0}, \sigma\mathbf{I})$. This noise allows different representations for all node embeddings and

Encoder	Community Small BCE	% Error	F1	Erdos&Renyi BCE	% Error	F1
Baseline	-	31.79	.0000	-	25.13	0.000
GNN	6.75	1.29	0.980	14.15	4.62	0.907
Noise-GNN	3.32	0.44	0.993	4.56	1.25	0.975
Radial Field	9.22	1.19	0.981	6.78	1.63	0.968
EGNN	**2.14**	**0.06**	**0.999**	**1.65**	**0.11**	**0.998**

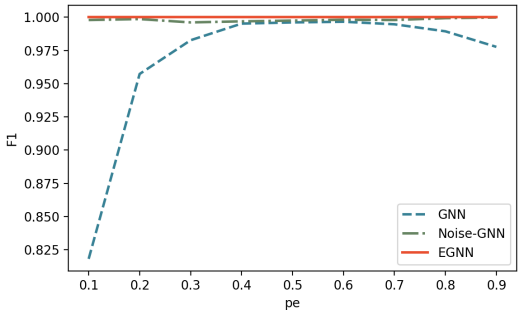

Figure 5. In the Table at the left we report the Binary Cross Entropy, % Error and F1 scores for the test partition on the Graph Autoencoding experiment in the Community Small and Erdos&Renyi datasets. In the Figure at the right, we report the F1 score when overfitting a training partition of 100 samples in the Erdos&Renyi dataset for different values of sparsity p_e. The GNN is not able to successfully auto-encode sparse graphs (small p_e values) for the Erdos&Renyi dataset even when training and testing on the same small subset.

as a result the graph can be decoded again, but it comes with a drawback, the network has to generalize over the new introduced noise distribution. Our Equivariant Graph Autoencoder will remain translation and rotation equivariant to this sampled noise which we find makes the generalization much easier. Another way of looking at this is considering the sampled noise makes the node representations go from structural to positional (Srinivasan & Ribeiro, 2019) where $\mathrm{E}(n)$ equivariance may be beneficial. In our case we will simply input this noise as the input coordinates $\mathbf{x}^0 \sim \mathcal{N}(\mathbf{0}, \sigma\mathbf{I}) \in \mathbb{R}^{M\times n}$ of our EGNN which will output an equivariant transformation of them $\mathbf{x}^L$, this output will be used as the embedding of the graph (i.e. $\mathbf{z} = \mathbf{x}^L$) which is the input to the decoder from Equation 9.

Dataset: We generated community-small graphs (You et al., 2018; Liu et al., 2019) by running the original code from (You et al., 2018). These graphs contain $12 \leq M \leq 20$ nodes. We also generated a second dataset using the Erdos&Renyi generative model (Bollobás & Béla, 2001) sampling random graphs with an initial number of $7 \leq M \leq 16$ nodes and edge probability $p_e = 0.25$. We sampled 5.000 graphs for training, 500 for validation and 500 for test for both datasets. Each graph is defined as and adjacency matrix $A \in \{0, 1\}^{M\times M}$.

Implementation details: Our Equivariant Graph Auto-Encoder is composed of an EGNN encoder followed by the decoder from Equation 9. The graph edges A_{ij} are input as edge attributes a_{ij} in Equation 3. The noise used to break the symmetry is input as the coordinates $\mathbf{x}^0 \sim \mathcal{N}(\mathbf{0}, \sigma\mathbf{I}) \in \mathbb{R}^{M\times n}$ in the first layer and $\mathbf{h}^0$ is initialized as ones since we are working with featureless graphs. As mentioned before, the encoder outputs an equivariant transformation on the coordinates which is the graph embedding and input to the decoder $\mathbf{z} = \mathbf{x}^L \in \mathbb{R}^{M\times n}$. We use $n = 8$ dimensions for the embedding space. We compare the EGNN to its GNN cousin, we also compare to Noise-GNN which is an adaptation of our GNN to match the setting from (Liu et al., 2019), and we also include the Radial Field algorithm as an additional baseline. All four models have 4 layers, 64 features for the hidden layers, the Swish activation function as a non-linearity and they were all trained for 100 epochs using the Adam optimizer and learning rate 10^{-4}. More details are provided in Appendix C.2. Since the number of nodes is larger than the number of layers, the receptive field of a GNN may not comprise the whole graph which can make the comparison unfair with our EGNN. To avoid this limitation, all models exchange messages among all nodes and the edge information is provided as edge attributes $a_{ij} = A_{ij}$ in all of them.

Results: In the table from Figure 5 we report the Binary Cross Entropy loss between the estimated and ground truth edges, the % Error which is defined as the percentage of wrong predicted edges with respect to the total amount of potential edges, and the F1 score of the edge classification, all numbers refer to the test partition. We also include a "Baseline" that predicts all edges as missing $\hat{A}_{ij} = 0$. The standard GNN seems to suffer from the symmetry problem and provides the worst performance. When introducing noise (Noise-GNN), both the loss and the error decrease showing that it is actually useful to add noise to the input nodes. Finally, our EGNN remains $\mathrm{E}(n)$ equivariant to this noise distribution and provides the best reconstruction with a 0.11% error in the Erdos&Renyi dataset and close to optimal 0.06% in the Community Small dataset. A further analysis of the reconstruction error for different n embedding sizes is reported in Appendix D.1.

Overfitting the training set: We explained the symmetry problem and we showed the EGNN outperforms other methods in the given datasets. Although we observed that adding noise to the GNN improves the results, it is difficult to exactly measure the impact of the symmetry limitation in these results independent from other factors such as generalization from the training to the test set. In this section we conduct an experiment where we train the different models in a subset of 100 Erdos&Renyi graphs and embedding size $n = 16$ with the aim to overfit the data. We evaluate the

Task Units	α bohr^3	$\Delta\varepsilon$ meV	$\varepsilon_{\text{HOMO}}$ meV	$\varepsilon_{\text{LUMO}}$ meV	μ D	C_ν cal/mol K	G meV	H meV	R^2 bohr^3	U meV	U_0 meV	ZPVE meV
NMP	.092	69	43	38	.030	.040	19	17	.180	20	20	1.50
Schnet	.235	63	41	34	.033	.033	14	14	.073	19	14	1.70
Cormorant	.085	61	34	38	.038	.026	20	21	.961	21	22	2.03
L1Net	.088	68	46	35	.043	.031	14	14	.354	14	13	1.56
LieConv	.084	49	30	25	.032	.038	22	24	.800	19	19	2.28
DimeNet++*	.044	33	25	20	.030	.023	8	7	.331	6	6	1.21
TFN	.223	58	40	38	.064	.101	-	-	-	-	-	-
SE(3)-Tr.	.142	53	35	33	.051	.054	-	-	-	-	-	-
EGNN	.071	48	29	25	.029	.031	12	12	.106	12	11	1.55

Table 3. Mean Absolute Error for the molecular property prediction benchmark in QM9 dataset. *DimeNet++ uses slightly different train/val/test partitions than the other papers listed here.

methods on the training data. In this experiment the GNN is unable to fit the training data properly while the EGNN can achieve perfect reconstruction and Noise-GNN close to perfect. We sweep over different p_e sparsity values from 0.1 to 0.9 since the symmetry limitation is more present in very sparse or very dense graphs. We report the F1 scores of this experiment in the right plot of Figure 5.

In this experiment we showed that $\mathrm{E}(n)$ equivariance improves performance when embedding graphs in a continuous space as a set of nodes in dimension n. Even though this is a simple reconstruction task, we think this can be a useful step towards generating graphs or molecules where often graphs (i.e. edges) are decoded as pairwise distances or similarities between nodes e.g. (Kipf & Welling, 2016b; Liu et al., 2019; Grover et al., 2019), and these metrics (e.g. eq. 9) are $\mathrm{E}(n)$ invariant. Additionally this experiment also showed that our method can successfully perform in a $\mathrm{E}(n)$ equivariant task for higher dimensional spaces where $n > 3$.

5.3. Molecular data — QM9

The QM9 dataset (Ramakrishnan et al., 2014) has become a standard in machine learning as a chemical property prediction task. The QM9 dataset consists of small molecules represented as a set of atoms (up to 29 atoms per molecule), each atom having a 3D position associated and a five dimensional one-hot node embedding that describe the atom type (H, C, N, O, F). The dataset labels are a variety of chemical properties for each of the molecules which are estimated through regression. These properties are invariant to translations, rotations and reflections on the atom positions. Therefore those models that are E(3) invariant are highly suitable for this task.

We imported the dataset partitions from (Anderson et al., 2019), 100K molecules for training, 18K for validation and 13K for testing. A variety of 12 chemical properties were estimated per molecule. We optimized and report the Mean Absolute Error between predictions and ground truth.

Implementation details: Our EGNN receives as input the 3D coordinate locations of each atom which are provided as $\mathbf{x}_i^0$ in Equation 3 and an embedding of the atom properties which is provided as input node features $\mathbf{h}_i^0$. Since this is an invariant task and also $\mathbf{x}^0$ positions are static, there is no need to update the particle's position $\mathbf{x}$ by running Equation 4 as we did in previous experiments. Consequently, we tried both manners and we didn't notice any improvement by updating $\mathbf{x}$. When not updating the particle's position (i.e. skipping Equation 4), our model becomes $\mathrm{E}(n)$ invariant, which is analogous to a standard GNN where all relative squared norms between pairs of points $\|\mathbf{x}_i - \mathbf{x}_j\|^2$ are inputted to the edge operation (eq. 3). Additionally, since we are not provided with an adjacency matrix and molecules can scale up to 29 nodes, we use the extension of our model from Section 3.3 that infers a soft estimation of the edges. Our EGNN network consists of 7 layers, 128 features per hidden layer and the Swish activation function as a non-linearity. A sum-pooling operation preceded and followed by two layers MLPs maps all the node embeddings $\mathbf{h}^L$ from the output of the EGNN to the estimated property value. Further implementation details are reported in Appendix. C. We compare to NMP (Gilmer et al., 2017), Schnet (Schütt et al., 2017b), Cormorant (Anderson et al., 2019), L1Net (Miller et al., 2020), LieConv (Finzi et al., 2020), DimeNet++ (Klicpera et al., 2020a), TFN (Thomas et al., 2018) and SE(3)-Tr. (Fuchs et al., 2020).

Results are presented in Table 3. Our method reports very competitive results in all property prediction tasks while remaining relatively simple, i.e. not introducing higher order representations, angles or spherical harmonics. Perhaps, surprisingly, we outperform other equivariant networks that consider higher order representations while in this task, we are only using type-0 representations (i.e. relative distances) to define the geometry of the molecules. In Appendix E we

prove that when only positional information is given (i.e. no velocity or higher order type features), then the geometry is completely defined by the norms in-between points up to $\mathrm{E}(n)$-transformations, in other words, if two collections of points separated by $\mathrm{E}(n)$ transformations are considered to be identical, then the relative norms between points is a unique identifier of the collection.

6. Conclusions

Equivariant graph neural networks are receiving increasing interest from the natural and medical sciences as they represent a new tool for analyzing molecules and their properties. In this work, we presented a new $\mathrm{E}(n)$ equivariant deep architecture for graphs that is computationally efficient, easy to implement, and significantly improves over the current state-of-the-art on a wide range of tasks. We believe these properties make it ideally suited to make a direct impact on topics such as drug discovery, protein folding and the design of new materials, as well as applications in 3D computer vision.

Acknowledgements

We would like to thank Patrick Forré for his support to formalize the invariance features identification proof.

References

Anderson, B., Hy, T.-S., and Kondor, R. Cormorant: Covariant molecular neural networks. *arXiv preprint arXiv:1906.04015*, 2019.

Bollobás, B. and Béla, B. *Random graphs*. Number 73. Cambridge university press, 2001.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Candès, E. J. and Recht, B. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717–772, 2009.

Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018.

Cohen, T. and Welling, M. Group equivariant convolutional networks. In Balcan, M. and Weinberger, K. Q. (eds.), *Proceedings of the 33nd International Conference on Machine Learning, ICML*, 2016.

Cohen, T. S. and Welling, M. Steerable cnns. In *5th International Conference on Learning Representations, ICLR*, 2017.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pp. 3844–3852, 2016.

Finzi, M., Stanton, S., Izmailov, P., and Wilson, A. G. Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data. *arXiv preprint arXiv:2002.12880*, 2020.

Fuchs, F., Worrall, D., Fischer, V., and Welling, M. Se (3)-transformers: 3d roto-translation equivariant attention networks. *Advances in Neural Information Processing Systems*, 33, 2020.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

Grover, A., Zweig, A., and Ermon, S. Graphite: Iterative generative modeling of graphs. In *International conference on machine learning*, pp. 2434–2444. PMLR, 2019.

Grzeszczuk, R., Terzopoulos, D., and Hinton, G. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 9–20, 1998.

Kipf, T., Fetaya, E., Wang, K.-C., Welling, M., and Zemel, R. Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687*, 2018.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016a.

Kipf, T. N. and Welling, M. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016b.

Klicpera, J., Giri, S., Margraf, J. T., and Günnemann, S. Fast and uncertainty-aware directional message passing for non-equilibrium molecules. *arXiv preprint arXiv:2011.14115*, 2020a.

Klicpera, J., Groß, J., and Günnemann, S. Directional message passing for molecular graphs. *arXiv preprint arXiv:2003.03123*, 2020b.

Köhler, J., Klein, L., and Noé, F. Equivariant flows: sampling configurations for multi-body systems with symmetric energies. *CoRR*, abs/1910.00753, 2019.

Köhler, J., Klein, L., and Noé, F. Equivariant flows: exact likelihood generative learning for symmetric densities. *arXiv preprint arXiv:2006.02425*, 2020.

Kondor, R., Son, H. T., Pan, H., Anderson, B., and Trivedi, S. Covariant compositional networks for learning graphs. *arXiv preprint arXiv:1801.02144*, 2018.

Liu, J., Kumar, A., Ba, J., Kiros, J., and Swersky, K. Graph normalizing flows. In *Advances in Neural Information Processing Systems*, pp. 13578–13588, 2019.

Miller, B. K., Geiger, M., Smidt, T. E., and Noé, F. Relevance of rotationally equivariant convolutions for predicting molecular properties. *arXiv preprint arXiv:2008.08461*, 2020.

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7559–7566. IEEE, 2018.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Ramakrishnan, R., Dral, P. O., Rupp, M., and Von Lilienfeld, O. A. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

Rezende, D. J., Racanière, S., Higgins, I., and Toth, P. Equivariant hamiltonian flows. *CoRR*, abs/1909.13739, 2019.

Romero, D. W. and Cordonnier, J.-B. Group equivariant stand-alone self-attention for vision. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=JkfYjnOEo6M.

Schütt, K. T., Arbabzadah, F., Chmiela, S., Müller, K. R., and Tkatchenko, A. Quantum-chemical insights from deep tensor neural networks. *Nature communications*, 8 (1):1–8, 2017a.

Schütt, K. T., Kindermans, P.-J., Sauceda, H. E., Chmiela, S., Tkatchenko, A., and Müller, K.-R. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. *arXiv preprint arXiv:1706.08566*, 2017b.

Serviansky, H., Segol, N., Shlomi, J., Cranmer, K., Gross, E., Maron, H., and Lipman, Y. Set2graph: Learning graphs from sets. *Advances in Neural Information Processing Systems*, 33, 2020.

Simonovsky, M. and Komodakis, N. Graphvae: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*, pp. 412–422. Springer, 2018.

Srinivasan, B. and Ribeiro, B. On the equivalence between positional node embeddings and structural graph representations. *arXiv preprint arXiv:1910.00452*, 2019.

Thomas, N., Smidt, T., Kearnes, S., Yang, L., Li, L., Kohlhoff, K., and Riley, P. Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*, 2018.

Uy, M. A., Pham, Q.-H., Hua, B.-S., Nguyen, T., and Yeung, S.-K. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1588–1597, 2019.

Watters, N., Zoran, D., Weber, T., Battaglia, P., Pascanu, R., and Tacchetti, A. Visual interaction networks: Learning a physics simulator from video. *Advances in neural information processing systems*, 30:4539–4547, 2017.

Weiler, M. and Cesa, G. General e(2)-equivariant steerable cnns. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems*, 2019.

You, J., Ying, R., Ren, X., Hamilton, W. L., and Leskovec, J. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*, 2018.

A. Equivariance Proof

In this section we prove that our model is translation equivariant on $\mathbf{x}$ for any translation vector $g \in \mathbb{R}^n$ and it is rotation and reflection equivariant on $\mathbf{x}$ for any orthogonal matrix $Q \in \mathbb{R}^{n \times n}$. More formally, we will prove the model satisfies:

$$Q\mathbf{x}^{l+1} + g, \mathbf{h}^{l+1} = \text{EGCL}(Q\mathbf{x}^l + g, \mathbf{h}^l)$$

We will analyze how a translation and rotation of the input coordinates propagates through our model. We start assuming $\mathbf{h}^0$ is invariant to $\text{E}(n)$ transformations on $\mathbf{x}$, in other words, we do not encode any information about the absolute position or orientation of $\mathbf{x}^0$ into $\mathbf{h}^0$. Then, the output $\mathbf{m}_{ij}$ of Equation 3 will be invariant too since the distance between two particles is invariant to translations $\|\mathbf{x}_i^l + g - [\mathbf{x}_j^l + g]\|^2 = \|\mathbf{x}_i^l - \mathbf{x}_j^l\|^2$, and it is invariant to rotations and reflections $\|Q\mathbf{x}_i^l - Q\mathbf{x}_j^l\|^2 = (\mathbf{x}_i^l - \mathbf{x}_j^l)^\top Q^\top Q(\mathbf{x}_i^l - \mathbf{x}_j^l) = (\mathbf{x}_i^l - \mathbf{x}_j^l)^\top \mathbf{I}(\mathbf{x}_i^l - \mathbf{x}_j^l) = \|\mathbf{x}_i^l - \mathbf{x}_j^l\|^2$ such that the edge operation becomes invariant:

$$\mathbf{m}_{i,j} = \phi_e\left(\mathbf{h}_i^l, \mathbf{h}_j^l, \left\|Q\mathbf{x}_i^l + g - [Q\mathbf{x}_j^l + g]\right\|^2, a_{ij}\right) = \phi_e\left(\mathbf{h}_i^l, \mathbf{h}_j^l, \left\|\mathbf{x}_i^l - \mathbf{x}_j^l\right\|^2, a_{ij}\right)$$

The second equation of our model (eq. 4) that updates the coordinates $\mathbf{x}$ is $\text{E}(n)$ equivariant. Following, we prove its equivariance by showing that an $\text{E}(n)$ transformation of the input leads to the same transformation of the output. Notice $\mathbf{m}_{ij}$ is already invariant as proven above. We want to show:

$$Q\mathbf{x}_i^{l+1} + g = Q\mathbf{x}_i^l + g + C\sum_{j \neq i}\left(Q\mathbf{x}_i^l + g - [Q\mathbf{x}_j^l + g]\right)\phi_x\left(\mathbf{m}_{i,j}\right)$$

Derivation.

$$\begin{aligned} Q\mathbf{x}_i^l + g + C\sum_{j \neq i}\left(Q\mathbf{x}_i^l + g - Q\mathbf{x}_j^l - g\right)\phi_x\left(\mathbf{m}_{i,j}\right) &= Q\mathbf{x}_i^l + g + QC\sum_{j \neq i}\left(\mathbf{x}_i^l - \mathbf{x}_j^l\right)\phi_x\left(\mathbf{m}_{i,j}\right) \\ &= Q\left(\mathbf{x}_i^l + C\sum_{j \neq i}\left(\mathbf{x}_i^l - \mathbf{x}_j^l\right)\phi_x\left(\mathbf{m}_{i,j}\right)\right) + g \\ &= Q\mathbf{x}_i^{l+1} + g \end{aligned}$$

Therefore, we have proven that rotating and translating $\mathbf{x}^l$ results in the same rotation and translation on $\mathbf{x}^{l+1}$ at the output of Equation 4.

Furthermore equations 5 and 6 only depend on $\mathbf{m}_{ij}$ and $\mathbf{h}^l$ which as saw at the beginning of this proof, are $\text{E}(n)$ invariant, therefore the output of Equation 6 $\mathbf{h}^{l+1}$ will be invariant too. Thus concluding that a transformation $Q\mathbf{x}^l + g$ on $\mathbf{x}^l$ will result in the same transformation on $\mathbf{x}^{l+1}$ while $\mathbf{h}^{l+1}$ will remain invariant to it such that $Q\mathbf{x}^{l+1} + g, \mathbf{h}^{l+1} = \text{EGCL}(Q\mathbf{x}^l + g, \mathbf{h}^l)$ is satisfied.

B. Re-formulation for velocity type inputs

In this section we write down the EGNN transformation layer $\mathbf{h}^{l+1}, \mathbf{x}^{l+1}, \mathbf{v}^{l+1} = \text{EGCL}[\mathbf{h}^l, \mathbf{x}^l, \mathbf{v}^l, \mathcal{E}]$ that can take in velocity input and output channels. We also prove it remains $\text{E}(n)$ equivariant.

$$\begin{aligned} \mathbf{m}_{ij} &= \phi_e\left(\mathbf{h}_i^l, \mathbf{h}_j^l, \left\|\mathbf{x}_i^l - \mathbf{x}_j^l\right\|^2, a_{ij}\right) \\ \mathbf{v}_i^{l+1} &= \phi_v\left(\mathbf{h}_i^l\right)\mathbf{v}_i^l + C\sum_{j \neq i}\left(\mathbf{x}_i^l - \mathbf{x}_j^l\right)\phi_x\left(\mathbf{m}_{ij}\right) \\ \mathbf{x}_i^{l+1} &= \mathbf{x}_i^l + \mathbf{v}_i^{l+1} \\ \mathbf{m}_i &= \sum_{j \in \mathcal{N}(i)}\mathbf{m}_{ij} \\ \mathbf{h}_i^{l+1} &= \phi_h\left(\mathbf{h}_i^l, \mathbf{m}_i\right) \end{aligned}$$

B.1. Equivariance proof for velocity type inputs

In this subsection we prove that the velocity types input formulation of our model is also $\mathrm{E}(n)$ equivariant on $\mathbf{x}$. More formally, for any translation vector $g \in \mathbb{R}^n$ and for any orthogonal matrix $Q \in \mathbb{R}^{n \times n}$, the model should satisfy:

$$\mathbf{h}^{l+1}, Q\mathbf{x}^{l+1} + g, Q\mathbf{v}^{l+1} = \text{EGCL}[\mathbf{h}^l, Q\mathbf{x}^l + g, Q\mathbf{v}^l, \mathcal{E}]$$

In Appendix A we already proved the equivariance of our EGNN (Section 3) when not including vector type inputs. In its velocity type inputs variant we only replaced its coordinate updates (eq. 4) by Equation 7 that includes velocity. Since this is the only modification we will only prove that Equation 7 re-written below is equivariant.

$$\mathbf{v}_i^{l+1} = \phi_v\left(\mathbf{h}_i^l\right)\mathbf{v}_i^l + C\sum_{j\neq i}\left(\mathbf{x}_i^l - \mathbf{x}_j^l\right)\phi_x\left(\mathbf{m}_{ij}\right)$$
$$\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \mathbf{v}_i^{l+1}$$

First, we prove the first line preserves equivariance, that is we want to show:

$$Q\mathbf{v}_i^{l+1} = \phi_v\left(\mathbf{h}_i^l\right)Q\mathbf{v}_i^l + C\sum_{j\neq i}\left(Q\mathbf{x}_i^l + g - [Q\mathbf{x}_j^l + g]\right)\phi_x\left(\mathbf{m}_{ij}\right)$$

Derivation.

$$\phi_v\left(\mathbf{h}_i^l\right)Q\mathbf{v}_i^l + C\sum_{j\neq i}\left(Q\mathbf{x}_i^l + g - [Q\mathbf{x}_j^l + g]\right)\phi_x\left(\mathbf{m}_{ij}\right) = Q\phi_v\left(\mathbf{h}_i^l\right)\mathbf{v}_i^l + QC\sum_{j\neq i}\left(\mathbf{x}_i^l - \mathbf{x}_j^l\right)\phi_x\left(\mathbf{m}_{ij}\right) \tag{10}$$

$$= Q\left(\phi_v\left(\mathbf{h}_i^l\right)\mathbf{v}_i^l + C\sum_{j\neq i}\left(\mathbf{x}_i^l - \mathbf{x}_j^l\right)\phi_x\left(\mathbf{m}_{ij}\right)\right) \tag{11}$$

$$= Q\mathbf{v}_i^{l+1} \tag{12}$$

Finally, it is straightforward to show the second equation is also equivariant, that is we want to show $Q\mathbf{x}_i^{l+1} + g = Q\mathbf{x}_i^l + g + Q\mathbf{v}_i^{l+1}$

Derivation.

$$Q\mathbf{x}_i^l + g + Q\mathbf{v}_i^{l+1} = Q\left(\mathbf{x}_i^l + \mathbf{v}_i^{l+1}\right) + g$$
$$= Q\mathbf{x}_i^{l+1} + g$$

Concluding we showed that an $\mathrm{E}(n)$ transformation on the input set of points results in the same transformation on the output set of points such that $\mathbf{h}^{l+1}, Q\mathbf{x}^{l+1} + g, Q\mathbf{v}^{l+1} = \text{EGCL}[\mathbf{h}^l, Q\mathbf{x}^l + g, Q\mathbf{v}^l, \mathcal{E}]$ is satisfied.

C. Implementation details

In this Appendix section we describe the implementation details of the experiments. First, we describe those parts of our model that are the same across all experiments. Our EGNN model from Section 3 contains the following three main learnable functions.

- **The edge function** ϕ_e (eq. 3) is a two layers MLP with two Swish non-linearities: Input $\rightarrow$ {LinearLayer() $\rightarrow$ Swish() $\rightarrow$ LinearLayer() $\rightarrow$ Swish() } $\rightarrow$ Output.
- **The coordinate function** ϕ_x (eq. 4) consists of a two layers MLP with one non-linearity: $\mathbf{m}_{ij} \rightarrow$ {LinearLayer() $\rightarrow$ Swish() $\rightarrow$ LinearLayer() } $\rightarrow$ Output
- **The node function** ϕ_h (eq. 6) consists of a two layers MLP with one non-linearity and a residual connection: $[\mathbf{h}_i^l, \mathbf{m}_i] \rightarrow$ {LinearLayer() $\rightarrow$ Swish() $\rightarrow$ LinearLayer() $\rightarrow$ Addition($\mathbf{h}_i^l$) } $\rightarrow \mathbf{h}_i^{l+1}$

These functions are used in our EGNN across all experiments. Notice the GNN (eq. 2) also contains and edge operation and a node operation ϕ_e and ϕ_h respectively. We use the same functions described above for both the GNN and the EGNN such that comparisons are as fair as possible.

C.1. Implementation details for Dynamical Systems

Dataset

In the dynamical systems experiment we used a modification of the Charged Particle's N-body (N=5) system from (Kipf et al., 2018). Similarly to (Fuchs et al., 2020), we extended it from 2 to 3 dimensions customizing the original code from (https://github.com/ethanfetaya/NRI) and we removed the virtual boxes that bound the particle's positions. The sampled dataset consists of 3.000 training trajectories, 2.000 for validation and 2.000 for testing. Each trajectory has a duration of 1.000 timesteps. To move away from the transient phase, we actually generated trajectories of 5.000 time steps and sliced them from timestep 3.000 to timestep 4.000 (1.000 time steps into the future) such that the initial conditions are more realistic than the Gaussian Noise initialization from which they are initialized.

In our second experiment, we sweep from 100 to 50.000 training samples, for this we just created a new training partition following the same procedure as before but now generating 50.000 trajectories instead. The validation and test partition remain the same from last experiment.

Models

All models are composed of 4 layers, the details for each model are the following.

- **EGNN**: For the EGNN we use its variation that considers vector type inputs from Section 3.2. This variation adds the function ϕ_v to the model which is composed of two linear layers with one non-linearity: Input $\rightarrow$ {LinearLayer() $\rightarrow$ Swish() $\rightarrow$ LinearLayer() } $\rightarrow$ Output. Functions ϕ_e, ϕ_x and ϕ_h that define our EGNN are the same than for all experiments and are described at the beginning of this Appendix C.
- **GNN**: The GNN is also composed of 4 layers, its learnable functions edge operation ϕ_e and node operation ϕ_h from Equation 2 are exactly the same as ϕ_e and ϕ_h from our EGNN introduced in Appendix C. We chose the same functions for both models to ensure a fair comparison. In the GNN case, the initial position $\mathbf{p}^0$ and velocity $\mathbf{v}^0$ from the particles is passed through a linear layer and inputted into the GNN first layer $\mathbf{h}^0$. The particle's charges are inputted as edge attributes $a_{ij} = c_i c_j$. The output of the GNN $\mathbf{h}^L$ is passed through a two layers MLP that maps it to the estimated position.
- **Radial Field**: The Radial Field algorithm is described in the Related Work 4, its only parameters are contained in its edge operation $\phi_{\text{rf}}()$ which in our case is a two layers MLP with two non linearities Input $\rightarrow$ {LinearLayer() $\rightarrow$ Swish() $\rightarrow$ LinearLayer() $\rightarrow$ Tanh } $\rightarrow$ Output. Notice we introduced a Tanh at the end of the MLP which fixes some instability issues that were causing this model to diverge in the dynamical system experiment. We also augmented the Radial Field algorithm with the vector type inputs modifications introduced in Section 3.2. In addition to the norms between pairs of points, $\phi_{\text{rf}}()$ also takes as input the particle charges $c_i c_j$.
- **Tensor Field Network:** We used the Pytorch implementation from https://github.com/FabianFuchsML/se3-transformer-public. We swept over different hyper paramters, degree $\in \{2, 3, 4\}$, number of features $\in \{12, 24, 32, 64, 128\}$. We got the best performance in our dataset for degree 2 and number of features 32. We used the Relu activation layer instead of the Swish for this model since it provided better performance.
- **SE(3) Transformers:** For the SE(3)-Transformer we used code from https://github.com/FabianFuchsML/se3-transformer-public. Notice this implementation has only been validated in the QM9 dataset but it is the only available implementation of this model. We swept over different hyperparamters degree $\in \{1, 2, 3, 4\}$, number of features $\in$ 16, 32, 64 and divergence $\in \{1, 2\}$, along with the learning rate. We obtained the best performance for degree 3, number of features 64 and divergence 1. As in Tensor Field Networks we obtained better results by using the Relu activation layer instead of the Swish.

Other implementation details

In Table 2 all models were trained for 10.000 epochs, batch size 100, Adam optimizer, the learning rate was fixed and independently chosen for each model. All models are 4 layers deep and the number of training samples was set to 3.000.

C.2. Implementation details for Graph Autoneoders

Dataset

In this experiment we worked with Community Small (You et al., 2018) and Erdos&Renyi (Bollobás & Béla, 2001) generated datasets.

- Community Small: We used the original code from (You et al., 2018) (https://github.com/JiaxuanYou/graph-generation) to generate a Community Small dataset. We sampled 5.000 training graphs, 500 for validation and 500 for testing.
- Erdos&Renyi is one of the most famous graph generative algorithms. We used the "gnp_random_graph(M, p)" function from (https://networkx.org/) that generates random graphs when povided with the number of nodes M and the edge probability p following the Erdos&Renyi model. Again we generated 5.000 graphs for training, 500 for validation and 500 for testing. We set the edge probability (or sparsity value) to $p = 0.25$ and the number of nodes M ranging from 7 to 16 deterministically uniformly distributed. Notice that edges are generated stochastically with probability p, therefore, there is a chance that some nodes are left disconnected from the graph, "gnp_random_graph(M, p)" function discards these disconnected nodes such that even if we generate graphs setting parameters to $7 \leq M \leq 16$ and $p = 0.25$ the generated graphs may have less number of nodes.

Finally, in the graph autoencoding experiment we also overfitted in a small partition of 100 samples (Figure 5) for the Erdos&Renyi graphs described above. We reported results for different p values ranging from 0.1 to 0.9. For each p value we generated a partition of 100 graphs with initial number of nodes between $7 \leq M \leq 16$ using the Erdos&Renyi generative model.

Models

All models consist of 4 layers, 64 features for the hidden layers and the Swish activation function as a non linearity. The EGNN is defined as explained in Section 3 without any additional modules (i.e. no velocity type features or inferring edges). The functions ϕ_e, ϕ_x and ϕ_h are defined at the beginning of this Appendix C. The GNN (eq. 2) mimics the EGNN in terms that it uses the same ϕ_h and ϕ_e than the EGNN for its edge and node updates. The Noise-GNN is exactly the same as the GNN but inputting noise into the $\mathbf{h}_0$ features. Finally the Radial Field was defined in the Related Related work Section 4 which edge's operation ϕ_{rf} consists of a two layers MLP: Input $\rightarrow$ { Linear() $\rightarrow$ Swish() $\rightarrow$ Linear() } $\rightarrow$ Output.

Other implementation details

All experiments have been trained with learning rate 10^{-4}, batch size 1, Adam optimizer, weight decay 10^{-16}, 100 training epochs for the 5.000 samples sized datasets performing early stopping for the minimum Binary Cross Entropy loss in the validation partition. The overfitting experiments were trained for 10.000 epochs on the 100 samples subsets.

C.3. Implementation details for QM9

For QM9 (Ramakrishnan et al., 2014) we used the dataset partitions from (Anderson et al., 2019). We imported the dataloader from his code repository (https://github.com/risilab/cormorant) which includes his data-preprocessing. Additionally all properties have been normalized by substracting the mean and dividing by the Mean Absolute Deviation.

Our EGNN consists of 7 layers. Functions ϕ_e and ϕ_h are defined at the beginning of this Appendix C. Additionally, we use the module ϕ_{inf} presented in Section 3.3 that infers the edges . This function ϕ_{inf} is defined as a linear layer followed by a sigmoid: Input $\rightarrow$ {Linear() $\rightarrow$ sigmoid()} $\rightarrow$ Output. Finally, the output of our EGNN $\mathbf{h}^L$ is forwarded through a two layers MLP that acts node-wise, a sum pooling operation and another two layers MLP that maps the averaged embedding to the predicted property value, more formally: $\mathbf{h}^L \rightarrow$ {Linear() $\rightarrow$ Swish() $\rightarrow$ Linear() $\rightarrow$ Sum-Pooling() $\rightarrow$ Linear() $\rightarrow$ Swish() $\rightarrow$ Linear} $\rightarrow$ Property. The number of hidden features for all model hidden layers is 128.

We trained each property individually for a total of 1.000 epochs, we used Adam optimizer, batch size 96, weight decay 10^{-16}, and cosine decay for the learning rate starting at at a lr=$5 \cdot 10^{-4}$ except for the Homo, Lumo and Gap properties where its initial value was set to 10^{-3}.

D. Further experiments

D.1. Graph Autoencoder

In this section we present an extension of the Graph Autoencoder experiment 5.2. In Table 4 we report the approximation error of the reconstructed graphs as the embedding dimensionality n is reduced $n \in \{4, 6, 8\}$ in the Community Small

	Community Small						Erdos&Renyi					
	n=4		n=6		n=8		n=4		n=6		n=8	
	% Err.	F1	% Err.	F1	% Err.	F1	% Err.	F1	% Err.	F1	% Err.	F1
GNN	**1.45**	**0.977**	1.29	0.9800	1.29	0.980	7.92	0.844	5.22	0.894	4.62	0.907
Noise-GNN	1.94	0.970	0.44	0.9931	0.44	0.993	3.80	0.925	2.66	0.947	1.25	0.975
EGNN	2.19	0.966	**0.42**	**0.9934**	**0.06**	**0.999**	**3.09**	**0.939**	**0.58**	**0.988**	**0.11**	**0.998**

Table 4. Analysis of the % of wrong edges and F1 score for different n embedding sizes $\{2, 4, 8\}$ for the GNN, Noise-GNN and EGNN in Community Small and Erdos&Renyi datasets.

and Erdos&Renyi datasets for the GNN, Noise-GNN and EGNN models. For small embedding sizes ($n = 4$) all methods perform poorly, but as the embedding size grows our EGNN significantly outperforms the others.

E. Sometimes invariant features are all you need.

Perhaps surprisingly we find our EGNNs outperform other equivariant networks that consider higher-order representations. In this section we prove that when only positional information is given (i.e. no velocity-type features) then the geometry is completely defined by the invariant distance norms in-between points, without loss of relevant information. As a consequence, it is not necessary to consider higher-order representation types of the relative distances, not even the relative differences as vectors. To be precise, note that these invariant features still need to be *permutation* equivariant, they are only $\mathrm{E}(n)$ invariant.

To be specific, we want to show that for a collection of points $\{\mathbf{x}_i\}_{i=1}^M$ the norm of in-between distances $\ell_2(\mathbf{x}_i, \mathbf{x}_j)$ are a *unique* identifier of the geometry, where collections separated by an $\mathrm{E}(n)$ transformations are considered to be identical. We want to show *invariance* of the norms under $\mathrm{E}(n)$ transformations and *uniqueness*: two point collections are identical (up to $\mathrm{E}(n)$ transform) when they have the same distance norms.

Invariance. Let $\{\mathbf{x}_i\}$ be a collection of M points where $\mathbf{x}_i \in \mathbb{R}^n$ and the ℓ_2 distances are $\ell_2(\mathbf{x}_i, \mathbf{x}_j)$. We want to show that all $\ell_2(\mathbf{x}_i, \mathbf{x}_j)$ are unaffected by $\mathrm{E}(n)$ transformations.

Proof. Consider an arbitrary $\mathrm{E}(n)$ transformation $\mathbb{R}^n \to \mathbb{R}^n : \mathbf{x} \mapsto Q\mathbf{x} + t$ where Q is orthogonal and $t \in \mathbb{R}^n$ is a translation. Then for all i, j:

$$\begin{aligned}\ell_2(Q\mathbf{x}_i + t, Q\mathbf{x}_j + t) &= \sqrt{(Q\mathbf{x}_i + t - [Q\mathbf{x}_j + t])^T (Q\mathbf{x}_i + t - [Q\mathbf{x}_j + t])} = \sqrt{(Q\mathbf{x}_i - Q\mathbf{x}_j)^T (Q\mathbf{x}_i - Q\mathbf{x}_j)} \\ &= \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T Q^T Q (\mathbf{x}_i - \mathbf{x}_j)} = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T (\mathbf{x}_i - \mathbf{x}_j)} = \ell_2(\mathbf{x}_i, \mathbf{x}_j)\end{aligned}$$

This proves that the ℓ_2 distances are invariant under $\mathrm{E}(n)$ transforms.

Uniqueness. Let $\{\mathbf{x}_i\}$ and $\{\mathbf{y}_i\}$ be two collection of M points each where all in-between distance norms are identical, meaning $\ell_2(\mathbf{x}_i, \mathbf{x}_j) = \ell_2(\mathbf{y}_i, \mathbf{y}_j)$. We want to show that $\mathbf{x}_i = Q\mathbf{y}_i + t$ for some orthogonal Q and translation t, for all i.

Proof. Subtract $\mathbf{x}_0$ from all $\{\mathbf{x}_i\}$ and $\mathbf{y}_0$ from all $\{\mathbf{y}_i\}$, so $\tilde{\mathbf{x}}_i = \mathbf{x}_i - \mathbf{x}_0$ and $\tilde{\mathbf{y}}_i = \mathbf{y}_i - \mathbf{y}_0$. As proven above, since translation is an $\mathrm{E}(n)$ transformation the distance norms are unaffected and:

$$\ell_2(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j) = \ell_2(\mathbf{x}_i, \mathbf{x}_j) = \ell_2(\mathbf{y}_i, \mathbf{y}_j) = \ell_2(\tilde{\mathbf{y}}_i, \tilde{\mathbf{y}}_j).$$

So without loss of generality, we may assume that $\mathbf{x}_0 = \mathbf{y}_0 = \mathbf{0}$. As a direct consequence $||\mathbf{x}_i||_2 = ||\mathbf{y}_i||_2$. Now writing out the square:

$$\mathbf{x}_i^T \mathbf{x}_i - 2\mathbf{x}_i^T \mathbf{x}_j + \mathbf{x}_j^T \mathbf{x}_j = ||\mathbf{x}_i - \mathbf{x}_j||_2^2 = ||\mathbf{y}_i - \mathbf{y}_j||_2^2 = \mathbf{y}_i^T \mathbf{y}_i - 2\mathbf{y}_i^T \mathbf{y}_j + \mathbf{y}_j^T \mathbf{y}_j$$

And since $||\mathbf{x}_i||_2 = ||\mathbf{y}_i||_2$, it follows that $\mathbf{x}_i^T \mathbf{x}_j = \mathbf{y}_i^T \mathbf{y}_j$ or equivalently written as dot product $\langle \mathbf{x}_i, \mathbf{x}_j \rangle = \langle \mathbf{y}_i, \mathbf{y}_j \rangle$. Notice that this already shows that angles between pairs of points are the same.

At this moment, it might already be intuïtive that the collections of points are indeed identical. To finalize the proof formally we will construct a linear map A for which we will show that (1) it maps every $\mathbf{x}_i$ to $\mathbf{y}_i$ and (2) that it is orthogonal. First note that from the angle equality it follows immediately that for every linear combination:

$$||\sum_i c_i \mathbf{x}_i||_2 = ||\sum_i c_i \mathbf{y}_i||_2 \quad (*).$$

Let V_x be the linear span of $\{\mathbf{x}_i\}$ (so V_x is the linear subspace of all linear combinations of $\{\mathbf{x}_i\}$). Let $\{\mathbf{x}_{i_j}\}_{j=1}^d$ be a basis of V_x, where $d \leq n$. Recall that one can define a linear map by choosing a basis, and then define for each basis vector where it maps to. Define a linear map A from V_x to V_y by the transformation from the basis $\mathbf{x}_{i_j}$ to $\mathbf{y}_{i_j}$ for $j = 1, ..., d$. Now pick any point $\mathbf{x}_i$ and write it in its basis $\mathbf{x}_i = \sum_j c_j \mathbf{x}_{i_j} \in V_x$. We want to show $A\mathbf{x}_i = \mathbf{y}_i$ or alternatively $||\mathbf{y}_i - A\mathbf{x}_i||_2 = 0$. Note that $A\mathbf{x}_i = A\sum_j c_j \mathbf{x}_{i_j} = \sum_j c_j A\mathbf{x}_{i_j} = \sum_j c_j \mathbf{y}_{i_j}$. Then:

$$\begin{aligned}||\mathbf{y}_i - \sum_j c_j \mathbf{y}_{i_j}||_2^2 &= \langle \mathbf{y}_i, \mathbf{y}_i \rangle - 2\langle \mathbf{y}_i, \sum_j c_i \mathbf{y}_{i_j} \rangle + \langle \sum_j c_i \mathbf{y}_{i_j}, \sum_j c_i \mathbf{y}_{i_j} \rangle \\ &\overset{(*)}{=} \langle \mathbf{x}_i, \mathbf{x}_i \rangle - 2\langle \mathbf{x}_i, \sum_j c_i \mathbf{x}_{i_j} \rangle + \langle \sum_j c_i \mathbf{x}_{i_j}, \sum_j c_i \mathbf{x}_{i_j} \rangle = \langle \mathbf{x}_i, \mathbf{x}_i \rangle - 2\langle \mathbf{x}_i, \mathbf{x}_i \rangle + \langle \mathbf{x}_i, \mathbf{x}_i \rangle = 0.\end{aligned}$$

Thus showing that $A\mathbf{x}_i = \mathbf{y}_i$ for all $i = 1, \ldots, M$, proving (1). Finally we want to show that A is orthogonal, when restricted to V_x. This follows since:

$$\langle A\mathbf{x}_{i_j}, A\mathbf{x}_{i_j} \rangle = \langle \mathbf{y}_{i_j}, \mathbf{y}_{i_j} \rangle = \langle \mathbf{x}_{i_j}, \mathbf{x}_{i_j} \rangle$$

for the basis elements $\mathbf{x}_{i_1}, ..., \mathbf{x}_{i_d}$. This implies that A is orthogonal (at least when restricted to V_x). Finally A can be extended via an orthogonal complement of V_x to the whole space. This concludes the proof for (2) and shows that A is indeed orthogonal.