Tensor Field Networks: Rotation- and Translation-Equivariant Neural Networks for 3D Point Clouds

Nathaniel Thomas [*] [1] **Tess Smidt** [*] [2] [3] [4] **Steven Kearnes** [4] **Lusann Yang** [4] **Li Li** [4] **Kai Kohlhoff** [4] **Patrick Riley** [4]

Abstract

We introduce tensor field networks, which are locally equivariant to 3D rotations and translations (and invariant to permutations of points) at every layer. 3D rotation equivariance removes the need for data augmentation to identify features in arbitrary orientations. Our network uses filters built from spherical harmonics; due to the mathematical consequences of this filter choice, each layer accepts as input (and guarantees as output) scalars, vectors, and higher-order tensors, in the geometric sense of these terms. We demonstrate how tensor field networks learn to model simple physics (Newtonian gravitation and moment of inertia), classify simple 3D shapes (trained on one orientation and tested on shapes in arbitrary orientations), and, given a small organic molecule with an atom removed, replace the correct element at the correct location in space.

1. Motivation

Convolutional neural networks are translation-equivariant, which means that features can be identified anywhere in a given input. This capability has contributed significantly to their widespread success.

In this paper, we present a family of networks that enjoy much richer equivariance: the symmetries of 3D Euclidean space. This includes 3D rotation equivariance (the ability to identify a feature in any 3D rotation and its orientation) and 3D translation equivariance.

There are three main benefits. First, this is more efficient than data augmentation to obtain 3D rotation-invariant output, making computation and training less expensive. This is significantly more important in 3D than 2D. Without equivariant filters like those in our design, achieving an angular resolution of δ would require a factor of $\mathcal{O}(\delta^{-1})$ more filters in 2D but $\mathcal{O}(\delta^{-3})$ more filters in 3D.[1] Second, a 3D rotation- and translation-equivariant network can identify local features in different orientations and locations with the same filters, which is helpful for interpretability. Finally, the network naturally encodes geometric tensors (such as scalars, vectors, and higher-rank geometric objects), mathematical objects that transform predictably under geometric transformations such as rotation. Here, and in the rest of this paper, the word "tensor" refers to *geometric* tensors, not generic multidimensional arrays.

Our network differs from a traditional CNN in three ways:

- We operate on point clouds using continuous convolutions. Our layers act on 3D coordinates of points and features at those points.
- We constrain our filters to be the product of a learnable radial function $R(r)$ and a spherical harmonic $Y_m^{(l)}(\hat{r})$.
- Our filter choice requires the structure of our network to be compatible with the algebra of geometric tensors.

We call these *tensor field networks* because every layer of our network inputs and outputs tensor objects: scalars, vectors, and higher-order tensors at every geometric point in the network. Tensor fields are ubiquitous in geometry, physics, and chemistry.

Our motivation was to design a universal architecture for deep learning on atomic systems (such as molecules or materials), but our network design is quite general. We expect tensor field networks to be useful in many tasks involving 3D geometry. For example, tensor field networks could be used to process 3D images in a rotation- and translation-equivariant way. We mention other potential applications in Section 6.

In this paper, we explain the mathematical conditions such a 3D rotation- and translation-equivariant network must

[*]Equal contribution [1]Stanford University, Stanford, California, USA [2]UC Berkeley, Berkeley, California, USA [3]Lawrence Berkeley National Laboratory, Berkeley, California, USA [4]Google, Mountain View, California, USA. Correspondence to: Nathaniel Thomas <ncthomas@stanford.edu>, Tess Smidt <tsmidt@berkeley.edu>.

[1]This is because the manifold of orthonormal frames at a point in 2D (the group $O(2)$) has dimension 1 and in 3D ($O(3)$) has dimension 3.

satisfy, provide several examples of equivariant compatible network components, and give examples of tasks that this family of networks can uniquely perform.

2. Related work

Our work builds upon Harmonic Networks (Worrall et al., 2017), which uses discrete convolutions and filters composed of circular harmonics to achieve 2D rotation equivariance, and SchNet (Schütt et al., 2017), which presents a rotation-invariant network using continuous convolutions. The mathematics of rotation equivariance in 3D is much more complicated than in 2D because rotations in 3D do not commute; that is, for 3D rotation matrices A and B, $AB \neq BA$ in general (see Reisert & Burkhardt (2009) for more about the mathematics of tensors under 3D rotations). Cohen et al. (2018) introduce Spherical CNNs, which are equivariant models on spherical spaces. Our network differs from this work because it is equivariant to all the symmetries of 3D Euclidean space, both 3D rotation and translation, and because it directly operates on more general data types, such as points. The networks presented in each of these three papers can each be emulated by a tensor field network.

Many other authors have investigated the problems of rotation equivariance in 2D, such as Zhou et al. (2017); Gonzalez et al. (2016); Li et al. (2017). Typically these work by looking at rotations of a filter and differ in exactly which rotations and how that orientation information is preserved (or not) higher in the network.

Other authors have dealt with similar issues of invariance or equivariance under particular input transformations. G-CNNs (Cohen & Welling, 2016) create invariance with *finite* symmetry groups (unlike the continuous groups in this work). Cohen et al. (2018) use spherical harmonics and Wigner D-matrices but only address spherical signals (two dimensional data on the surface of a sphere). Kondor et al. (2018) use tensor algebra to create neural network layers that extend Message Passing Neural Networks (Gilmer et al., 2017), but they are permutation group tensors (operating under permutation of the indices of the nodes) not geometric tensors.

The networks presented in Qi et al. (2016; 2017) operate on point clouds and use symmetric functions to encode permutation invariance. Qi et al. (2017) employ a randomized sampling algorithm to include pooling in the network. These networks do not include rotation equivariance.

Other neural networks have been designed and evaluated on atomic systems using nuclei centered calculations. Many of these capture 3D geometry just by including the pairwise distance between atoms (e.g. SchNet (Schütt et al., 2017) and the graph convolutional model from Faber et al.). Our work notably draws from SchNet (Schütt et al., 2017) for the point convolutions, radial functions, and self-interaction layers. All models that rely only on pairwise distances or angles between points have the weakness (unlike this work) that they can not distinguish between chiral inputs (*e.g.* left hand and right hand), which have identical pairwise distances but are different shapes in 3D.

The other major approach to modeling 3D atomic systems is to voxelize the space (Wallach et al., 2015; Boomsma & Frellsen, 2017; Torng & Altman, 2017). In general, these are subject to significant expense, no guarantees of smooth transformation under rotation, and edge effects from the voxelization step.

An introduction to the concepts of steerability and equivariance in the context of neural networks can be found in Cohen & Welling (2017), which focuses on discrete symmetries.

3. Representations and equivariance

A *representation* D of a group G is a function from G to square matrices such that for all $g, h \in G$,

$$D(g)D(h) = D(gh)$$

A function $\mathcal{L} : \mathcal{X} \to \mathcal{Y}$ (for vector spaces $\mathcal{X}$ and $\mathcal{Y}$) is *equivariant* with respect to a group G and group representations $D^{\mathcal{X}}$ and $D^{\mathcal{Y}}$ if for all $g \in G$ and $x \in \mathcal{X}$,

$$\mathcal{L}\left(D^{\mathcal{X}}(g)x\right) = D^{\mathcal{Y}}(g)\mathcal{L}(x)$$

Invariance is a type of equivariance where $D^{\mathcal{Y}}(g)$ is the identity for all g. We are concerned with the group of symmetry operations that includes isometries of 3D space and permutations of the points.

Composing equivariant networks $\mathcal{L}_1$ and $\mathcal{L}_2$ yields an equivariant network $\mathcal{L}_2 \circ \mathcal{L}_1$ (proof in supplementary material). Therefore, proving equivariance for each layer of a network is sufficient to prove (up to numerical accuracy) that a whole network is equivariant.

Furthermore, if a network is equivariant with respect to two transformations g and h, then it is equivariant to the composition of those transformations gh (by the definition of a representation). This implies that demonstrating permutation, translation, and rotation equivariance individually is sufficient to prove equivariance of a network to the group (and corresponding representations) containing all combinations of those transformations. As we describe shortly, translation and permutation equivariance are manifest in our core layers, so we will focus on demonstrating rotation equivariance.

Tensor field networks act on points with associated features. A layer $\mathcal{L}$ takes a finite set S of vectors in $\mathbb{R}^3$ and a vector in $\mathcal{X}$ at each point in S and outputs a vector in $\mathcal{Y}$ at each point in S, where $\mathcal{X}$ and $\mathcal{Y}$ are vector spaces. We write this as

$$\mathcal{L}(\vec{r}_a, x_a) = (\vec{r}_a, y_a)$$

where $\vec{r}_a \in \mathbb{R}^3$ are the point coordinates and $x_a \in \mathcal{X}$, $y_a \in \mathcal{Y}$ are the feature vectors (a being an indexing scheme over the points in S). (To simplify the formulas here, we are assuming that inputs and outputs are at the same points in $\mathbb{R}^3$, but this is not necessary.) This combination of $\mathbb{R}^3$ with another vector space can be written as $\mathbb{R}^3 \oplus \mathcal{X}$, where $\oplus$ refers to the direct sum operation, where vectors in each space are concatenated and matrices acting on each space are combined into a larger block diagonal matrix.

We now describe the conditions on $\mathcal{L}$ for equivariance and invariance with respect to different input transformations.

3.1. Permutation invariance

$$\textit{Condition:} \quad \mathcal{L}(\vec{r}_{\sigma(a)}, x_{\sigma(a)}) = \mathcal{L}(\vec{r}_a, x_a)$$

where σ permutes the points to which the indices refer.

All of the layers that we will introduce in Section 4 are manifestly permutation-invariant because we only treat point clouds as a *set* of points, never requiring an imposed order like in a list. (In our implementation, points have an array index associated with them, but this index only ever used in a symmetric way.) We will only consider permutation *invariance* (as opposed to a more general equivariance condition) in this paper.

3.2. Translation equivariance

$$\textit{Condition:} \quad \mathcal{L}(\vec{r}_a + \vec{t}, x_a) = \mathcal{T}_{\vec{t}}\mathcal{L}(\vec{r}_a, x_a)$$

where $\mathcal{T}_{\vec{t}}(\vec{r}_a, y_a) := (\vec{r}_a + \vec{t}, y_a)$. This condition is analogous to the translation equivariance condition for CNNs. It is possible to use a more general translation equivariance condition, where the operator on the right-hand side of the equation also acts upon the $\mathcal{Y}$ representation subspace, but we will not consider that in this paper.

All of the layers in Section 4 are manifestly translation-equivariant because we only ever use differences between two points $\vec{r}_i - \vec{r}_j$ (for indices i and j).

3.3. Rotation equivariance

The group of proper 3D rotations is called $SO(3)$, which is a manifold that can be parametrized by 3 numbers (see Goodman & Wallach (1998)). Let $D^{\mathcal{X}}$ be a representation of $SO(3)$ on a vector space $\mathcal{X}$ (and $D^{\mathcal{Y}}$ on $\mathcal{Y}$). Acting with $g \in SO(3)$ on $\vec{r} \in \mathbb{R}^3$ we write as $\mathcal{R}(g)\vec{r}$, and acting on $x \in \mathcal{X}$ gives $D^{\mathcal{X}}(g)x$.

Condition:

$$\mathcal{L}\left(\mathcal{R}(g)\vec{r}_a, D^{\mathcal{X}}(g)x_a\right) = \left[\mathcal{R}(g) \oplus D^{\mathcal{Y}}(g)\right] \mathcal{L}\left(\vec{r}_a, x_a\right), \tag{1}$$

(For layers in this paper, only the action of $D^{\mathcal{Y}}(g)$ on $\mathcal{Y}$ that will be nontrivial, so we will use a convention of omitting the $\mathbb{R}^3$ layer output in our equations.)

The key to attaining local rotation equivariance is to restrict our convolution filters to have a particular form. The features will have different types corresponding to whether they transform as scalars, vectors, or higher tensors.

To make our analysis and implementation easier, we decompose representations into irreducible representations. The irreducible representations of $SO(3)$ have dimensions $2l+1$ for $l \in \mathbb{N}$ (including $l = 0$) and can be chosen to be unitary. We will use the term "rotation order" to refer to l in this expression. The rotation orders $l = 0, 1, 2$ correspond to scalars, vectors, and symmetric traceless matrices, respectively. The group elements are represented by $D^{(l)}$, which are called *Wigner D-matrices* (see Gilmore (2008)); they map elements of $SO(3)$ to $(2l + 1) \times (2l + 1)$-dimensional complex matrices.

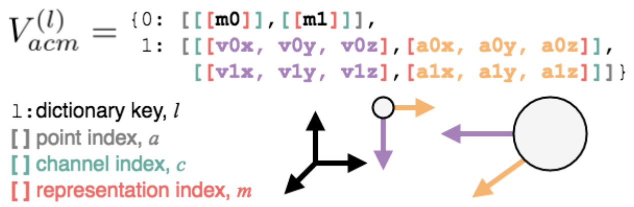
We could additionally have equivariance with respect to $O(3)$, the group of 3D rotations and mirror operations (a discrete symmetry). We would augment our convolution using the method for obtaining equivariance with respect to discrete symmetries described in Cohen & Welling (2017). We discuss this a bit more in Section 5.2.1.

4. Tensor field network layers

At each level of a tensor field network, we have a finite set S of points in $\mathbb{R}^3$ with a vector in a representation of $SO(3)$ (that has been decomposed into a direct sum of irreducible representations) associated with each point. This is essentially a finite version of a tensor field, which was our inspiration for this network design.

At each point, we can have multiple instances of l-rotation-order representations corresponding to different features of that point. We will refer to these different instances as *channels*. We implement this object $V^{(l)}_{acm}$ as a dictionary with key l of multidimensional arrays each with shapes $[|S|, n_l, 2l + 1]$ (where n_l is the number of channels with representation l) corresponding to `[point, channel, representation index]`. See Figure 1 for an example of how to encode a simple system in this notation.

We will describe tensor field network layers and prove that they are equivariant. To prove that a layer is equivariant, we only have to prove rotation equivariance for a given



$V_{acm}^{(l)} =$
```
{0: [[[m0]],[[m1]]],
 1: [[[v0x, v0y, v0z],[a0x, a0y, a0z]],
     [[v1x, v1y, v1z],[a1x, a1y, a1z]]]}
```

Figure 1. Example of $V_{acm}^{(l)}$ representing two point masses with velocities and accelerations. Colored brackets indicate the a (point), c (channel), and m (representation) indices. (l) is the key for the $V_{acm}^{(l)}$ dictionary of feature tensors.

rotation order. This requires showing that when the point cloud rotates and the input features are transformed, the output features transform accordingly. (All of these layers will be manifestly permutation-invariant and translation-equivariant.)

4.1. Point convolution

This layer is the core of our network and builds upon the pointwise convolutions in (Schütt et al., 2017). We start by considering the action of a general pointwise filter transformation

$$\sum_{b \in S} F_c(\vec{r}_a - \vec{r}_b) V_{bc}$$

Note that our design is strictly more general than standard convolutional neural networks, which can be treated as a point cloud S of a grid of points with regular spacing.

4.1.1. Spherical harmonics and filters

To design a rotation-equivariant point convolution, we want rotation-equivariant filters. The spherical harmonics, $Y_m^{(l)}$: $S^2 \to \mathbb{C}$ (where m can be any integer between $-l$ and l), are an orthonormal basis for functions on the sphere. These functions are equivariant to $SO(3)$; that is, they have the property that for $g \in SO(3)$ and $\hat{r} \in S^2$,

$$Y_m^{(l)}(\mathcal{R}(g)\hat{r}) = \sum_{m'} D_{mm'}^{(l)}(g) Y_{m'}^{(l)}(\hat{r}).$$

For our filters to also be rotation-equivariant, we restrict them to the following form:

$$F_{cm}^{(l_f,l_i)}(\vec{r}) = R_c^{(l_f,l_i)}(r) Y_m^{(l_f)}(\hat{r})$$

where l_i and l_f are non-negative integers corresponding to the rotation order of the input and the filter, respectively; $R_c^{(l_f,l_i)} : \mathbb{R}_{\geq 0} \to \mathbb{R}$ are learned functions; and $\hat{r}$ and r are the direction unit vector and magnitude of $\vec{r}$, respectively. Filters of this form inherit the transformation property of spherical harmonics under rotations because $R(r)$ is a scalar in m. This choice of filter restriction is analogous to the use of circular harmonics in Worrall et al. (2017) (though we do not have an analog to the phase offset because of the non-commutativity of $SO(3)$).

4.1.2. Combining representations using tensor products

Our filters and layer input each inhabit representations of $SO(3)$ (that is, they both carry l and m indices). In order to produce output that we can feed into downstream layers, we need to combine the layer input and filters in such a way that the output also transforms appropriately (by inhabiting a representation of $SO(3)$).

A *tensor product* of representations is a prescription for combining two representations l_1 and l_2 to get another representation $l_1 \otimes l_2$. If $u^{(l_1)} \in l_1$ and $v^{(l_2)} \in l_2$, then $u^{(l_1)} \otimes v^{(l_2)} \in l_1 \otimes l_2$ and

$$(u \otimes v)_m^{(l)} = \sum_{m_1=-l_1}^{l_1} \sum_{m_2=-l_2}^{l_2} C_{(l_1,m_1)(l_2,m_2)}^{(l,m)} u_{m_1}^{(l_1)} v_{m_2}^{(l_2)}$$

where C are called *Clebsch-Gordan coefficients* (see Griffiths (2017)). Note that l is any integer between $|l_1 - l_2|$ and $(l_1 + l_2)$ inclusive, and m is any integer between $-l$ and l inclusive.

The crucial property of the Clebsch-Gordan coefficients that we need to prove equivariance of this layer is

$$\begin{aligned}\sum_{m_1',m_2'} & C_{(l_1,m_1')(l_2,m_2')}^{(l_0,m_0)} D_{m_1'm_1}^{(l_1)}(g) D_{m_2'm_2}^{(l_2)}(g) \\ &= \sum_{m_0'} D_{m_0m_0'}^{(l)}(g) C_{(l_1,m_1)(l_2,m_2)}^{(l_0,m_0')}\end{aligned} \tag{2}$$

(see, for example, Reisert & Burkhardt (2009)).

4.1.3. Layer definition

Filters inhabit one representation, inputs another, and outputs yet another. We need the Clebsch-Gordan coefficients to combine inputs and filters in such a way that they yield the desired output representation:

$$\begin{aligned}&\mathcal{L}_{acm_O}^{(l_O)}(\vec{r}_a, V_{acm_I}^{(l_I)}) \\ &\quad := \sum_{m_F,m_I} C_{(l_F,m_F)(l_I,m_I)}^{(l_O,m_O)} \sum_{b \in S} F_{cm_F}^{(l_F,l_I)}(\vec{r}_{ab}) V_{bcm_I}^{(l_I)}\end{aligned} \tag{3}$$

where $\vec{r}_{ab} := \vec{r}_a - \vec{r}_b$ and the subscripts I, F, and O denote the representations of the input, filter, and output, respectively.

Using the equivariance of the filter F and the property of the Clebsch-Gordan coefficients Equation 2, we can show the rotation equivariance of point convolutions, Equation 1 (detailed proof in supplementary material, Section C):

$$\begin{aligned}\mathcal{L}^{(l_O)}_{acm_O}\Big(\mathcal{R}(g)\vec{r}_a, \sum_{m'} D^{(l_I)}_{m_I m'_I}(g) V^{(l_I)}_{acm'_I}\Big) \\ = \sum_{m'_O} D^{(l_O)}_{m_O m'_O}(g) \mathcal{L}^{(l_O)}_{acm'_O}\big(\vec{r}_a, V^{(l_I)}_{acm_I}\big)\end{aligned} \quad (4)$$

This aspect of the design, and this proof of its correctness, is the core of our result.

4.1.4. Scalars ($l = 0$) and vectors ($l = 1$)

In this section, we attempt to demystify Wigner D-matrices, spherical harmonics, and Clebsch-Gordan coefficients by stating their values for a real representation of $l = 0$ (scalars) and $l = 1$ (vectors). (It can sometimes be simpler to use a complex representation, but we can always use a basis change to convert between real and complex versions.) The corresponding objects for $l > 1$ are more complicated.

The Wigner D-matrices are

$$D^{(0)}(g) = 1 \qquad D^{(1)}(g) = \mathcal{R}(g),$$

where $\mathcal{R}$ are the usual rotation matrices on $\mathbb{R}^3$. The spherical harmonics are

$$Y^{(0)}(\hat{r}) \propto 1 \qquad Y^{(1)}(\hat{r}) \propto \hat{r}$$

The Clebsch-Gordan coefficients are just the familiar ways to combine scalars and vectors: For $1 \otimes 1 \to 0$,

$$C^{(0,0)}_{(1,i)(1,j)} \propto \delta_{ij}$$

which is the dot product for 3D vectors. For $1 \otimes 1 \to 1$,

$$C^{(1,i)}_{(1,j)(1,k)} \propto \epsilon_{ijk}$$

which is the cross product for 3D vectors. The $0 \otimes 0 \to 0$ case is just regular multiplication of two scalars, and $0 \otimes 1 \to 1$ and $1 \otimes 0 \to 1$ corresponds to scalar multiplication of a vector.

4.2. Self-interaction

We follow Schütt et al. (2017) in using point convolutions to scale feature vectors elementwise and using self-interaction layers to mix together the components of the feature vectors at each point. Self-interaction layers are analogous to 1x1 convolutions, and they implicitly act like $l = 0$ (scalar) types of filters:

$$\sum_{c'} W^{(l)}_{cc'} V^{(l)}_{ac'm}$$

In general, each rotation order has different weights because there may be different numbers of channels n_l corresponding to that rotation order. For $l = 0$, we may also use biases. Putting a self-interaction layer after a one-hot encoding input is the same as the embedding layer described in Schütt et al. (2017).

The weight matrix W is a scalar transform with respect to the representation index m, so the D-matrices commute with W, straightforwardly implying that this layer is equivariant. Equivariance for $l = 0$ is straightforward because $D^{(0)} = 1$.

4.3. Concatenation

A point convolution of an l_F filter on an l_I input yields outputs at $2\max(l_I, l_F) + 1$ different rotation orders (one for each integer between $|l_I - l_F|$ and $(l_I + l_F)$, inclusive), though in designing a particular network, we may choose not to calculate or use some of those outputs. The most general way to combine incoming channels of the same rotation order before the next point convolution layer is to concatenate along the channel axis (increasing the number of input channels), then apply a self-interaction layer. This concatenation operation is equivariant because concatenation does not affect the representation index.

4.4. Nonlinearity

Our nonlinearity layer acts as a scalar transform in the l spaces (that is, along the m dimension). For $l = 0$ channels, we can use

$$\eta^{(0)}\big(V^{(0)}_{ac} + b^{(0)}_c\big)$$

for some function $\eta^{(0)}: \mathbb{R} \to \mathbb{R}$ and biases $b^{(0)}_c$. For $l > 0$ channels, our nonlinearity has the form

$$\eta^{(l)}\big(\|V\|^{(l)}_{ac} + b^{(l)}_c\big) V^{(l)}_{acm}$$

where $\eta^{(l)}: \mathbb{R} \to \mathbb{R}$ can be different for each l and

$$\|V\|^{(l)}_{ac} := \sqrt{\sum_m |V^{(l)}_{acm}|^2}$$

(It may be necessary to regularize this norm near the origin for this layer to have desirable differentiability properties.)

Note that

$$\|D(g)V\| = \|V\|$$

because D is a unitary representation. Therefore, this layer is a scalar transform in the representation index m, so it is equivariant.

4.5. Global pooling

A global pooling operation that we use in our demonstrations is summing the feature vectors over each point. This operation is equivariant because D-matrices act linearly.

5. Demonstrations and experiments

We chose experiments that are simple enough to be easily understandable yet suggestive of richer problems. They each highlight a different aspect of our framework. Each of these tasks is either unnatural or impossible in existing frameworks.

We have focused on understanding how to construct and train these models in controlled tasks because of the complexity of the architecture. In future work, we will apply our network to more complicated and useful tasks.

In these tasks, we used radial functions identical to those used in Schütt et al. (2017): We used radial basis functions composed of Gaussians, and two dense layers are applied to this basis vector. The number, spacing, and standard deviation of the Gaussians are hyperparameters. We used a shifted softplus activation function $\log(0.5e^x + 0.5)$ for our radial functions and nonlinearity layers. We implemented our models in TensorFlow (Abadi et al., 2015). Our code is available at `https://github.com/tensorfieldnetworks/tensorfieldnetworks`

5.1. Tensor outputs: acceleration vectors in Newtonian gravity and moment of inertia tensor

Network types: $0 \to 1$ and $0 \to 0 \oplus 2$

To demonstrate the simplest non-trivial tensor field networks, we train networks to calculate acceleration vectors of point masses under Newtonian gravity and the moment of inertia tensor at a specific point of a collection of point masses. These tasks only require a single layer of point convolutions to demonstrate. Furthermore, we can check the learned radial functions against analytical solutions.

The acceleration of a point mass in the presence of other point masses according to Newtonian gravity is given by

$$\vec{a}_p = -G_N \sum_{n \neq p} \frac{m_n}{r_{np}^2} \hat{r}_{np}$$

where we define $\vec{r}_{np} := \vec{r}_n - \vec{r}_p$. (We choose units where $G_N = 1$, for simplicity.)

The moment of inertia tensor is used in classical mechanics to calculate angular momentum. Objects generally have different moments of inertia depending on which axis they rotate about, and this is captured by the moment of inertia tensor (see Landau & Lifshitz (1976)):

$$I_{ij} = \sum_p m_p \Big[(\vec{r}_p \cdot \vec{r}_p)\delta_{ij} - (\vec{r}_p)_i (\vec{r}_p)_j \Big]$$

The moment of inertia tensor is a symmetric tensor, so it can be encoded using a $0 \oplus 2$ representation. (The supplementary material contains a more detailed explanation of how to go from irreducible representations to matrix tensor representation.)

For both of these tasks, we input to the network a set of random points with associated random masses. (Details of about how these points are generated and about the hyperparameters for our radial functions are given in the supplementary material.) For the moment of inertia task, we also designate a different special point at which we want to calculate the moment of inertia tensor.

Our networks are simple: for learning Newtonian gravity, we use a single $l = 1$ convolution with 1 channel; for learning the moment of inertia tensor, we use a single layer comprised of $l = 0$ and $l = 2$ convolutions with 1 channel each. See Figure 2 for a diagram of these networks. We use elementwise summed $L2$ losses for the difference of the acceleration vectors and for the difference of the moment of inertia tensor. We get excellent agreement with the Newtonian gravity inverse square law after a 1,000 training steps and with the moment of inertia tensor radial functions after 10,000 steps.

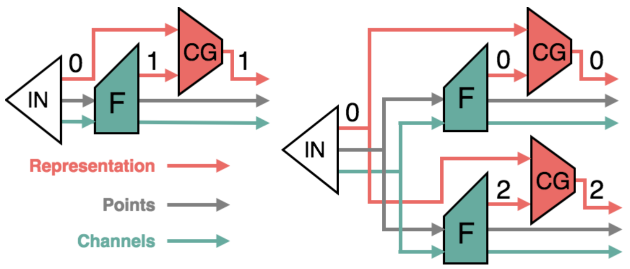


Figure 2. Network diagram for learning gravitational accelerations and moment of inertia tensor. The input tensor has indices `[channels, points, representation]`. The input for the gravitational acceleration and moment of inertia tasks are point mass scalars, therefore the representation index is 1-dimensional. Convolution filters are indicated by blocks marked F and Clebsch-Gordan tensors are indicated by CG. The numbers along the arrows for the representation index indicate the l of that arrow, with CG blocks combining the input and filter ls to generate output of a specific l.

In Figure 3, we show that the single radial filter is able to learn the $1/r^2$ law for gravitational accelerations. (Related figures for the moment of inertia task can be found in the supplementary material.)

We could have obtained equivariant accelerations by using a pure $l = 0$ network to predict the gravitational potential $\phi : \mathbb{R}^3 \to \mathbb{R}$, a scalar field, and then taking derivatives of ϕ with respect to the point coordinates to obtain the forces and accelerations (as in, *e.g.*, Schütt et al. (2017)). However, many important vector quantities, such as magnetic fields in electromagnetism, cannot be derived from scalar fields.

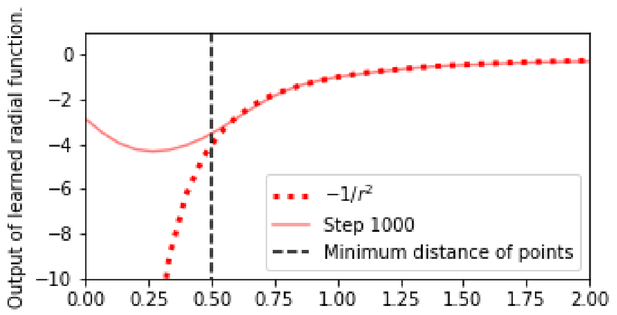


Figure 3. Radial function learned by $l = 1$ filter for gravity toy dataset. Minimum radial cutoff distance of 0.5 is chosen for randomly sampled points such that there are enough samples generated near the minimum distance.

5.2. Shape classification

Network type: $0 \to 0$

In three dimensions, there are 8 unique shapes made of 4 adjacent cubes (up to translations and rotations); see Figure 4. We call these shapes *3D Tetris*, and represent them using points at the center of each cube.

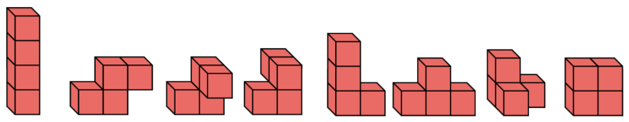


Figure 4. 3D Tetris shapes. Blocks correspond to single points. The third and fourth shapes from the left are mirrored versions of each other.

We demonstrate the rotation equivariance of our network by classifying 3D Tetris pieces in the following way: During training, we input to the network a dataset of shapes in a single orientation, and it outputs a classification of which shape it has seen. To demonstrate the equivariance of the network, we test the network with shapes from the same dataset that have been rotated and translated randomly. Our network performs perfectly on this task.

We use a 3-layer network that includes the following for every layer: all possible paths with $l = 0$ and $l = 1$ convolutions, a concatenation (if necessary), a self-interaction layer, and a rotation-equivariant nonlinearity. We only use the $l = 0$ output of the network since the shape classes are invariant under rotation and hence scalars. To get a classification from the $l = 0$ output of the network, we sum over the output of all points (global pooling).

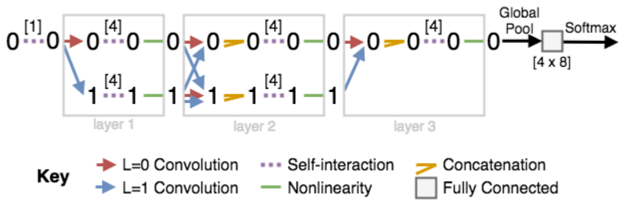


Figure 5. Network diagrams for shape classification task showing how information flows between tensors of different order. Clebsch-Gordan tensors are implied in the arrows indicating convolutions. The numbers above the self-interactions indicate the number of channels. Individual convolutions, indicated by arrows, each produce a separate tensor, and concatenation is performed after the convolutions.

5.2.1. Chirality

There are two shapes in 3D Tetris that are mirrors of each other. Any network that relies solely upon distances (such as SchNet) or angles between points (such as ANI-1 (Smith et al., 2017)) cannot distinguish these shapes. Our network can.

A tensor field network that contains a path with an odd number of odd-rotation-order filters has the possibility of detecting local changes in handedness. This is because

$$Y^{(l)}(-\vec{r}) = (-1)^l Y^{(l)}(\vec{r}).$$

A tensor field network that does not include such paths will be invariant to mirror symmetry. As an example, a $0 \to 0$ network can correctly classify all 3D Tetris shapes except for the chiral shapes. The smallest network that can classify chiral shapes must include the path $0 \xrightarrow{1} 1 \xrightarrow{1} 1 \xrightarrow{1} 0$ (superscripts correspond to filter rotation order), which yields the vector triple product $\vec{F}_1 \cdot (\vec{F}_2 \times \vec{F}_3)$ of the filter vectors $\vec{F}_i$ for each layer. This combination picks up a minus sign under mirror operations.

5.3. Missing point

Network type: $0 \to 0 \oplus 1$

In this task, we randomly remove a point from a shape and ask the network to replace that point. This is a first step toward general isometry-equivariant generative models for 3D point clouds.

We output an array of scalars, a special scalar, and one vector at each point. The vector, $\vec{\delta}_a$, indicates where the missing point should be relative to the starting point $\vec{r}_a$ (both direction and distance) and the array of scalars indicates the point type. The special scalar is put through softmax and used as a probability p_a measuring the confidence in that point's vote. We aggregate votes for location using a weighted sum:

$$\sum_a p_a(\vec{r}_a + \vec{\delta}_a)$$

(See the supplementary material for a proof that this is translation-equivariant.) This scheme is illustrated in Figure 6.

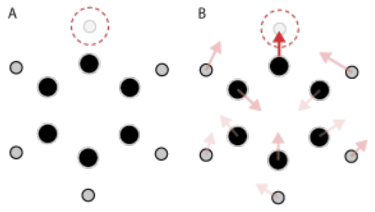


Figure 6. A hypothetical example input and output of the missing point network. (A) A benzene molecule with hydrogen removed (B) The relative output vectors produced by the network, with arrows shaded by the associated probabilities.

We train on molecular structures for this task because molecules can be comprised of a small number of atoms (making the task computationally easy to demonstrate) and because precise positions are important for chemical behavior.

We trained on structures in the QM9 dataset, a collection of 134,000 molecules with up to nine heavy atoms (C, N, O, F); including hydrogens, there are up to 29 atoms per molecule (Ramakrishnan et al., 2014).

Our network is 3 layers deep, with 15 channels per rotation order at each layer. The input features to the network are a one-hot encoding of atom types at each point. We use 2-layer radial functions with 4 Gaussians with centers evenly spaced from 0 to 2.5 angstroms and variance that is half of the spacing. Most covalent bonds are 1-2 angstroms in length, so our radial functions are smaller than the diameter of the molecules of the QM9 dataset.

We train on a 1,000 molecule subset of molecules that have 5 to 18 atoms. In a single epoch, we train on each molecule in the training set (1,000 molecules with 5-18 atoms) with one randomly selected atom deleted (1,000 examples total per epoch). We then test the network on a random selection of 1,000 molecules with 19 atoms, 1,000 molecules with 23 atoms, and 1,000 molecules with 25-29 atoms.

For each epoch during training, we use each molecule once and randomly remove one atom. During evaluation, for each molecule, we make a prediction for every possible atom that could be removed. For example, for a molecule with 10 atoms, we will generate 10 predictions, each with a different atom missing. This is why the number of predictions is larger than the number of molecules.

Table 1. Performance on missing point task

Atoms	Number of predictions	Accuracy (%) (≤ 0.5 Å and atom type)	Distance MAE in Å
5-18 (train)	15 863	91.3	0.16
19	19 000	93.9	0.14
23	23 000	96.5	0.13
25-29	25 356	97.3	0.16

We define an accurate placement of the missing atom position to be within 0.5 angstroms of the removed atom position with the correct atom type according to `argmax` of the returned atom type array. We choose the 0.5 angstrom cutoff because it is smaller than the smallest covalent bond, a hydrogen-hydrogen bond which has a length of 0.7 angstroms. We present the accuracy of our predictions in Table 1. We train for 225 epochs and show comparable prediction accuracy on the test sets of 19, 23, and 25-29 atoms as the training set of 5-18 atoms. We include a breakdown of the accuracy and distance mean absolute error (MAE) in the supplementary material.

Our network is most accurate for carbon and hydrogen missing atoms (over 90%), moderately accurate for oxygen and nitrogen (about 70-80%), and not accurate for fluorine atoms (since there are only 50 examples in our training set, they might not be seen during training due to random selection of the missing atom).

6. Future work

We have explained the theory of tensor field networks and demonstrated how they work on simple examples.

We hope that tensor field networks can be applied to a wide range of phenomena: In the context of atomic systems, we hope to train networks to predict properties of large and heterogeneous systems, learn molecular dynamics, calculate electron densities (as inputs to density functional theory algorithms), and hypothesize new stable structures. Ultimately, we hope to design new useful materials, drugs, and chemicals.

For more general physics, we see potential applications in modeling complex fluid flows, analyzing detector events in particle physics experiments, and studying configurations of stars and galaxies. We see other applications in 3D perception, robotics, computational geometry, and bioimaging.

Acknowledgements

Thanks to Alex Alemi, Patrick Hayden, Sean Lubner, and Cody Wild for feedback on drafts of our paper. NT was supported by a US Department of Defense Multidisciplinary

University Research Initiative grant. TS was supported by an internship at Google.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Boomsma, W. and Frellsen, J. Spherical convolutions and their application in molecular modelling. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 3436–3446. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6935-spherical-convolutions-and-their-application-in-molecular-modelling.pdf.

Cohen, T., Geiger, M., Köhler, J., and Welling, M. Spherical CNNs. *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Hkbd5xZRb.

Cohen, T.S. and Welling, M. Group equivariant convolutional networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

Cohen, T.S. and Welling, M. Steerable CNNs. In *International Conference on Learning Representations (ICLR)*, 2017.

Faber, F. A., Hutchison, L., Huang, B., Gilmer, J., Schoenholz, S. S., Dahl, G. E., Vinyals, O., Kearnes, S., Riley, P. F., and von Lilienfeld, O. A. Prediction errors of molecular machine learning models lower than hybrid DFT error. *Journal of Chemical Theory and Computation*.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In Precup, Doina and Teh, Yee Whye (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1263–1272, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/gilmer17a.html.

Gilmore, R. *Lie groups, physics, and geometry: An introduction for physicists, engineers and chemists*. Cambridge University Press, 2008.

Gonzalez, D. M., Volpi, M., Komodakis, N., and Tuia, D. Rotation equivariant vector field networks. *CoRR*, abs/1612.09346, 2016. URL http://arxiv.org/abs/1612.09346.

Goodman, R. and Wallach, N. R. *Representations and invariants of the classical groups*, volume 68. Cambridge University Press, 1998.

Griffiths, D. J. *Introduction to quantum mechanics*. Cambridge University Press, 2017.

Kondor, R., Truong Son, H., Pan, H., Anderson, B., and Trivedi, S. Covariant compositional networks for learning graphs. *ArXiv e-prints*, January 2018.

Landau, L. D. and Lifshitz, E. M. *Mechanics*, volume 1 of *Course of Theoretical Physics*. Elsevier, 1976.

Li, J., Yang, Z., Liu, H., and Cai, D. Deep Rotation Equivariant Network. *ArXiv e-prints*, May 2017.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. PointNet: Deep learning on point sets for 3D classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. PointNet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 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, 2014.

Reisert, M. and Burkhardt, H. Spherical tensor calculus for local adaptive filtering. *Tensors in Image Processing and Computer Vision*, pp. 153–178, 2009.

Schütt, K. T., Arbabzadah, F., Chmiela, S., Müller, K. R., and Tkatchenko, A. Quantum-chemical insights from deep tensor neural networks. 8:13890 EP –, Jan 2017. URL http://dx.doi.org/10.1038/ncomms13890. Article.

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 e-prints*, June 2017.

Smith, J. S., Isayev, O., and Roitberg, A. E. ANI-1: An extensible neural network potential with DFT accuracy at force field computational cost. *Chem. Sci.*, 8:3192–3203, 2017. doi: 10.1039/C6SC05720A. URL http://dx.doi.org/10.1039/C6SC05720A.

Torng, W. and Altman, R. B. 3D deep convolutional neural networks for amino acid environment similarity analysis. *BMC Bioinformatics*, 18(1):302, Jun 2017. ISSN 1471-2105. doi: 10.1186/s12859-017-1702-0. URL https://doi.org/10.1186/s12859-017-1702-0.

Wallach, I., Dzamba, M., and Heifets, A. AtomNet: A deep convolutional neural network for bioactivity prediction in structure-based drug discovery. *CoRR*, abs/1510.02855, 2015. URL http://arxiv.org/abs/1510.02855.

Worrall, D. E., Garbin, S. J., Turmukhambetov, D., and Brostow, G. J. Harmonic networks: Deep translation and rotation equivariance. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Zhou, Y., Ye, Q., Qiu, Q., and Jiao, J. Oriented Response Networks. *ArXiv e-prints*, January 2017.

A. Proofs of general equivariance propositions

For equivariant $\mathcal{L}$, the following diagram is commutative for all $g \in G$:

$$\begin{tikzcd}
\mathcal{X} \arrow[r, "\mathcal{L}"] \arrow[d, "D^{\mathcal{X}}(g)"] & \mathcal{Y} \arrow[d, "D^{\mathcal{Y}}(g)"] \\
\mathcal{X} \arrow[r, "\mathcal{L}"] & \mathcal{Y}
\end{tikzcd}$$

If a function is equivariant with respect to two transformations g and h, then it is equivariant to the composition of those transformations:

$$\begin{aligned}
\mathcal{L}(D^{\mathcal{X}}(gh)x) &= \mathcal{L}\big(D^{\mathcal{X}}(g)D^{\mathcal{X}}(h)x\big) \\
&= D^{\mathcal{Y}}(g)\mathcal{L}\big(D^{\mathcal{X}}(h)x\big) \\
&= D^{\mathcal{Y}}(gh)\mathcal{L}(x)
\end{aligned}$$

for all $g, h \in G$ and $x \in \mathcal{X}$; that is, the following diagram is commutative:

$$\begin{tikzcd}
\mathcal{X} \arrow[r, "\mathcal{L}"] \arrow[d, "D^{\mathcal{X}}(g)"] & \mathcal{Y} \arrow[d, "D^{\mathcal{Y}}(g)"] \\
\mathcal{X} \arrow[r, "\mathcal{L}"] \arrow[d, "D^{\mathcal{X}}(h)"] & \mathcal{Y} \arrow[d, "D^{\mathcal{Y}}(h)"] \\
\mathcal{X} \arrow[r, "\mathcal{L}"] & \mathcal{Y}
\end{tikzcd}$$

Composing equivariant functions $\mathcal{L}_1 : \mathcal{X} \to \mathcal{Y}$ and $\mathcal{L}_2 : \mathcal{Y} \to \mathcal{Z}$ yields an equivariant function $\mathcal{L}_2 \circ \mathcal{L}_1$:

$$\begin{aligned}
\mathcal{L}_2(\mathcal{L}_1(D^{\mathcal{X}}(g)x)) &= \mathcal{L}_2(D^{\mathcal{Y}}(g)\mathcal{L}_1(x)) \\
&= D^{\mathcal{Z}}(g)\mathcal{L}_2(\mathcal{L}_1(x))
\end{aligned}$$

That is, the following is commutative:

$$\begin{tikzcd}
\mathcal{X} \arrow[r, "\mathcal{L}_1"] \arrow[d, "D^{\mathcal{X}}(g)"] & \mathcal{Y} \arrow[r, "\mathcal{L}_2"] \arrow[d, "D^{\mathcal{Y}}(g)"] & \mathcal{Z} \arrow[d, "D^{\mathcal{Z}}(g)"] \\
\mathcal{X} \arrow[r, "\mathcal{L}_1"] & \mathcal{Y} \arrow[r, "\mathcal{L}_2"] & \mathcal{Z}
\end{tikzcd}$$

B. Motivating point convolutions

We can represent input as a continuous function that is non-zero at a finite set of points (using Dirac δ functions):

$$V(\vec{t}) = \sum_{a \in S} V_a \delta(\vec{t} - \vec{r}_a)$$

A point convolution is then equivalent to applying an integral transform with kernel

$$F(\vec{t} - \vec{s}) \sum_{a \in S} \delta(\vec{t} - \vec{r}_a)$$

for some function F. This transform yields

$$\begin{aligned}
\mathcal{L}(\vec{t}) &= \int d^3\vec{s} F(\vec{t} - \vec{s}) \sum_{a \in S} \delta(\vec{t} - \vec{r}_a) \sum_{b \in S} V_b \delta(\vec{s} - \vec{r}_b) \\
&= \sum_{a \in S} \delta(\vec{t} - \vec{r}_a) \sum_{b \in S} F(\vec{r}_a - \vec{r}_b) V_b \\
&= \sum_{a \in S} \delta(\vec{t} - \vec{r}_a) \mathcal{L}_a
\end{aligned}$$

where we define

$$\mathcal{L}_a := \sum_{b \in S} F(\vec{r}_a - \vec{r}_b) V_b$$

as in the main text.

C. Proof of equivariance of point convolution layer

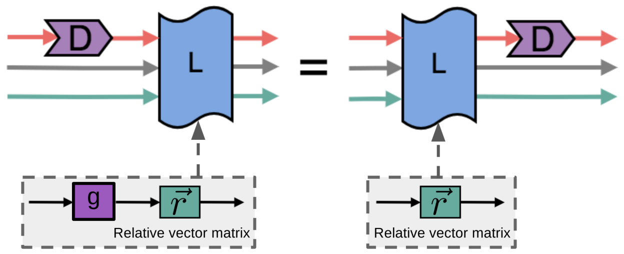


Figure S1. Condition for layer rotation equivariance

Under a rotation $\vec{r}_a \mapsto \mathcal{R}(g)\vec{r}_a$, we know that $\vec{r}_{ab} \mapsto \mathcal{R}(g)\vec{r}_{ab}$ and

$$F^{(l_f,l_i)}_{cm}(\mathcal{R}(g)\vec{r}_{ab}) = \sum_{m'} D^{(l_f)}_{mm'}(g) F^{(l_f,l_i)}_{cm'}(\vec{r}_{ab}) \quad (5)$$

because of the transformation properties of the spherical harmonics $Y^{(l)}_m$.

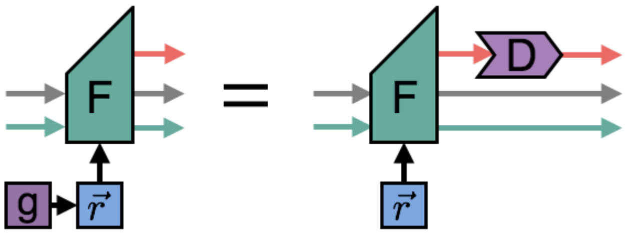


Figure S2. Filter equivariance equation

Then

$$
\begin{aligned}
&\mathcal{L}^{(l_O)}_{acm_O}\left(\mathcal{R}(g)\vec{r}_a, \sum_{m'} D^{(l_I)}_{m_I m'_I}(g) V^{(l_I)}_{acm'_I}\right)\\
&= \sum_{m_F, m_I} C^{(l_O, m_O)}_{(l_F, m_F)(l_I, m_I)}\\
&\qquad \times \sum_{b\in S} F^{(l_F, l_I)}_{cm_F}(\mathcal{R}(g)\vec{r}_{ab}) \sum_{m'} D^{(l_I)}_{m_I m'_I}(g) V^{(l_I)}_{bcm'_I}\\
&= \sum_{m_F, m_I} C^{(l_O, m_O)}_{(l_F, m_F)(l_I, m_I)}\\
&\qquad \times \sum_{b\in S}\left(\sum_{m'_F} D^{(l_F)}_{m_F m'_F}(g) F^{(l_F, l_I)}_{cm'_F}(\vec{r}_{ab})\right)\\
&\qquad\qquad \times \left(\sum_{m'_I} D^{(l_I)}_{m_I m'_I}(g) V^{(l_I)}_{bcm'_I}\right)\\
&= \sum_{m'_O} D^{(l_O)}_{m_O m'_O}(g) C^{(l_O, m'_O)}_{(l_F, m_F)(l_I, m_I)} \sum_{b\in S} F^{(l_F, l_I)}_{cm_F}(\vec{r}_{ab}) V^{(l_I)}_{bcm_I}\\
&= \sum_{m'_O} D^{(l_O)}_{m_O m'_O}(g) \mathcal{L}^{(l_O)}_{acm'_O}\left(\vec{r}_a, V^{(l_I)}_{acm_I}\right).
\end{aligned}
$$

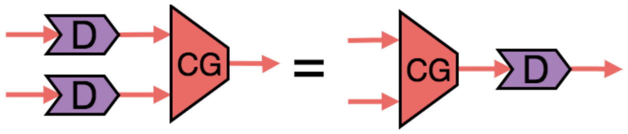


Figure S3. Equivariance of Clebsch-Gordan coefficients. Note that each D may refer to a different irreducible representation.

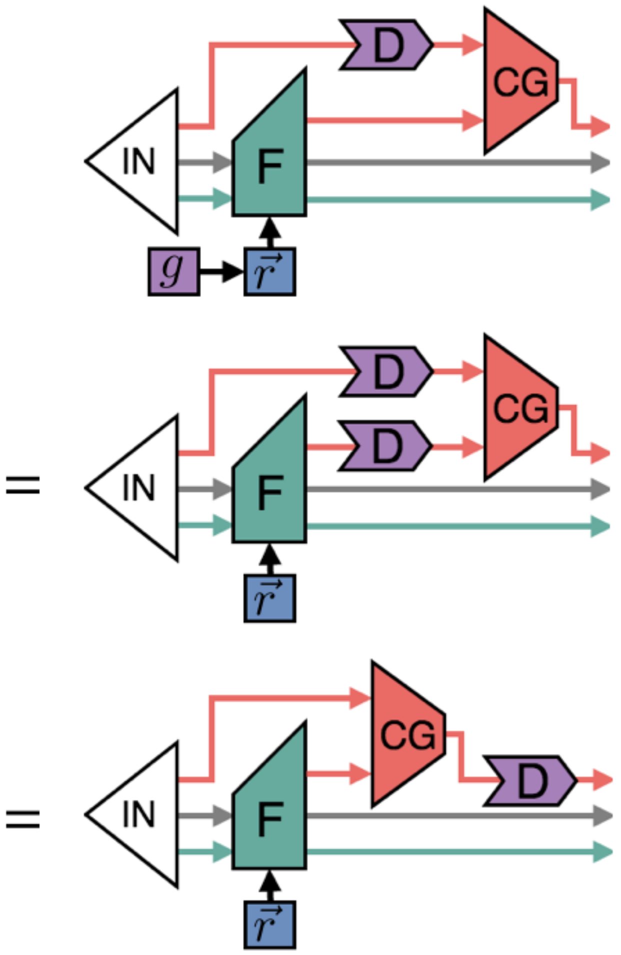


Figure S4. Diagrammatic proof of point convolution rotation equivariance.

D. Details for gravitational accelerations and moment of inertia tasks

D.1. Moment of inertia radial functions

We can write the moment of inertia tensor as

$$I_{ij} := \sum_{a\in S} m_a T_{ij}(\vec{r}_a)$$

where

$$T_{ij}(\vec{r}) := R^{(0)}(r)\delta_{ij} + R^{(2)}(r)\left(\hat{r}_i\hat{r}_j - \frac{\delta_{ij}}{3}\right)$$

The expression that $R^{(2)}$ is multiplying is the 3D symmetric traceless tensor, which can be constructed from the $l = 2$ spherical harmonic. To get agreement with the moment of inertia tensor as defined in the main text, we must have

$$R^{(0)}(r) = \frac{2}{3}r^2 \qquad R^{(2)}(r) = -r^2$$

Figure S5 shows the excellent agreement of our learned radial functions for filters $l = 0$ and $l = 2$ to the analytical solution.

D.2. Point generation details and radial hyperparameters

The number of points is uniformly randomly selected from 2 through 10. The masses are scalar values that are randomly chosen from a uniform distribution from 0.5 to 2.0. The coordinates of the points are randomly generated from a uniform distribution to be inside a cube with sides of length 4 for gravity and 1 for moment of inertia.

We use 30 Gaussian basis functions whose centers are evenly spaced between 0 and 2. We use a Gaussian variance that is one half the distance between the centers. We use a batch size of 1. For the test set, we simply use more randomly generated points from the same distribution.

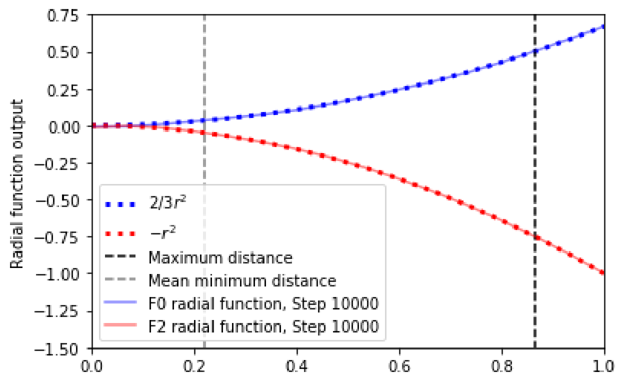


Figure S5. Radial function learned by $l = 0$ and $l = 2$ filters for moment of inertia toy dataset. The filters learn the analytical radial functions. For a collection of randomly generated point sets, the mean minimum distance is the average of the minimum distance between points in each point set. Distances smaller than the mean minimum distance might not have been seen by the network enough times to correct the radial filter.

We note that $-1/r^2$ diverges as $r \to 0$. We choose a cutoff minimum distance at 0.5 distance because it is easy to generate sufficient examples at that distance with a few number of points per example. If we wanted to properly sample for closer distances, we would need to change how we generate the random points or use close to 1000 points per example.

E. Proof of weighted point-averaging layer equivariance

Let S be the set of points (not including the missing point at $\vec{M}$) with locations $\vec{r}_a$. Suppose that the output of the network is a scalar and a vector $\vec{\delta}_a$ at each point in S. We take the softmax of the scalars over S to get a probability p_a at each point. Define the votes as $\vec{v}_a := \vec{r}_a + \vec{\delta}_a$, so the guessed point is

$$\vec{u} := \sum_{a \in S} p_a \vec{v}_a$$

This is the first operation that we have introduced that lacks manifest translation equivariance because it uses $\vec{r}_a$ by itself instead of only using $\vec{r}_a - \vec{r}_b$ combinations. We can show that $\vec{r}_a \mapsto \vec{r}_a + \vec{t}$ implies

$$\vec{u} \mapsto \sum_{a \in S} \left[p_a(\vec{r}_a + \vec{t}) + p_a \delta_a \right] = \vec{u} + \vec{t}$$

because the p_a sum to 1. This voting scheme is also rotation-equivariant because it is a sum of 3D vectors. The loss function

$$\text{loss} = (\vec{u} - \vec{M})^2$$

is translation-invariant because it is a function of the difference of vectors in 3D space and rotation-invariant because it is a dot product of vectors.

Table S1. Performance on missing point task by atom type

Atoms	Number of atoms with given type in set	Accuracy (%) (≤ 0.5 Å and atom type)	Distance MAE in Å
Hydrogen			
5-18 (train)	7207	94.6	0.16
19	10 088	93.2	0.16
23	14 005	96.7	0.14
25-29	16 362	97.7	0.15
Carbon			
5-18 (train)	5663	94.3	0.16
19	6751	99.9	0.10
23	7901	100.0	0.11
25-29	8251	99.7	0.17
Nitrogen			
5-18 (train)	1407	74.8	0.16
19	616	74.7	0.18
23	37	81.1	0.19
25-29	16	93.8	0.26
Oxygen			
5-18 (train)	1536	83.3	0.17
19	1539	80.2	0.21
23	1057	68.0	0.20
25-29	727	60.1	0.21
Fluorine			
5-18 (train)	50	0.0	0.18
19	6	0.0	0.07
23	0		
25-29	0		

F. Missing point task accuracies and MAE by epoch

In Table S1, we give the prediction accuracy and MAE on distance for the missing point task broken down by atom type. There are 1,000 molecules in each of the train and test sets; however, when comparing results by atom type, the relevant number to compare is the number of examples where a specific atom type is removed. In Figure S6 and Figure S7, we give the accuracy and distance MAE for the missing point task as a function of the number of training epochs (Tables 1 and S1 contain the results after 225 epochs).

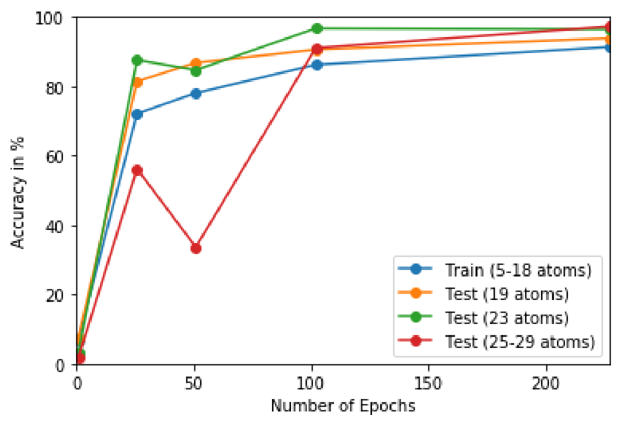


Figure S6. Accuracy of missing point task by epoch of training

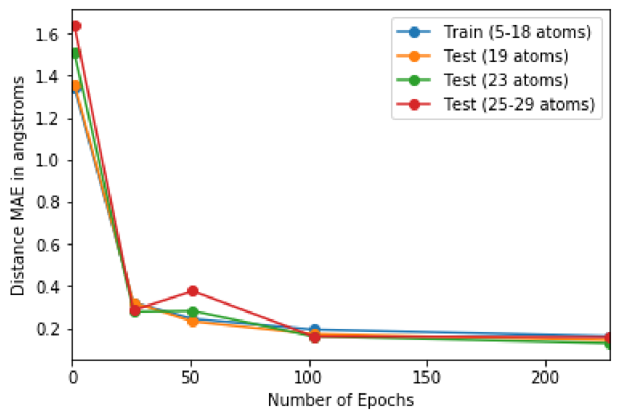


Figure S7. Distance MAE of missing point task by epoch of training