Tensor field networks: Rotation- and translation-equivariant neural networks for 3D point clouds (2018)

Thomas et. al

Presentation by Jacob Lyons

Overview

- Tensor field networks introduced, locally equivariant to 3D rotations, translations, and permutations of points at every layer
- Removes need for data augmentation
- Each layer accept as input (and guarantees as output) scalars, vectors, and higher-order tensors. This is possible because the network uses filters built from spherical harmonics

Motivation

- CNNs already translation-equivariant
- Tensor field networks (TFNs) enjoy symmetries of 3D Euclidean space
- TFNs differ from CNNs in three ways
  - Operate on point clouds using continuous convolutions. Layers act on 3D coordinates and features at these points
  - Filters constrained to be product of learnable radial function and spherical harmonic
  - Filter choice requires structure of network to be compatible with algebra of geometric tensors
- Initial motivation: universal architecture for deep learning on atomic systems

Related Work

- D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow. Harmonic networks: Deep translation and rotation equivariance. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017. *Achieves 2D rotation equiariance, uses circular harmonics*
- K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko. Quantum-chemical insights from deep tensor neural networks. 8:13890 EP –, Jan 2017. *Rotation equivariant network using continuous convolutions*
- *G-CNNs guarantee equivariance to finite symmetry groups, not continuous ones like this paper*

Related Work

- T. Cohen, M. Geiger, J. Köhler, and M. Welling. Spherical CNNs. International Conference on Learning Representations, 2018. *use spherical harmonics and Wigner D-matrices but only address spherical signals*
- T. S. Cohen and M. Welling. Steerable CNNs. In International Conference on Learning Representations (ICLR), 2017 *Introduction to steerability and equivariance in the context of neural networks, also focuses on discrete symmetries*

Theory

Group Representations and Equivariance in 3D

Representation: maps elements of group to square matrices such that:

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

Equivariance:

$$\mathcal{L} \circ D^{\mathcal{X}}(g) = D^{\mathcal{Y}}(g) \circ \mathcal{L}$$

$$\mathcal{L} : \mathcal{X} \to \mathcal{Y} \text{ (for vector spaces } \mathcal{X} \text{ and } \mathcal{Y})$$

Invariance: $D^{\mathcal{Y}}(g)$ is the identity for all g

The following diagrams are commutative:

$$\begin{array}{ccc} \mathcal{X} & \xrightarrow{\mathcal{L}} & \mathcal{Y} \\ \downarrow{\scriptstyle D^{\mathcal{X}}(g)} & & \downarrow{\scriptstyle D^{\mathcal{Y}}(g)} \\ \mathcal{X} & \xrightarrow{\mathcal{L}} & \mathcal{Y} \end{array}$$

$$\begin{array}{ccc} \mathcal{X} & \xrightarrow{\mathcal{L}} & \mathcal{Y} \\ \downarrow{\scriptstyle D^{\mathcal{X}}(g)} & & \downarrow{\scriptstyle D^{\mathcal{Y}}(g)} \\ \mathcal{X} & \xrightarrow{\mathcal{L}} & \mathcal{Y} \\ \downarrow{\scriptstyle D^{\mathcal{X}}(h)} & & \downarrow{\scriptstyle D^{\mathcal{Y}}(h)} \\ \mathcal{X} & \xrightarrow{\mathcal{L}} & \mathcal{Y} \end{array}$$

$$\begin{array}{ccccc} \mathcal{X} & \xrightarrow{\mathcal{L}_1} & \mathcal{Y} & \xrightarrow{\mathcal{L}_2} & \mathcal{Z} \\ \downarrow{\scriptstyle D^{\mathcal{X}}(g)} & & \downarrow{\scriptstyle D^{\mathcal{Y}}(g)} & & \downarrow{\scriptstyle D^{\mathcal{Z}}(g)} \\ \mathcal{X} & \xrightarrow{\mathcal{L}_1} & \mathcal{Y} & \xrightarrow{\mathcal{L}_2} & \mathcal{Z} \end{array}$$

Group Representations and Equivariance in 3D (cont.)

- Sufficient to demonstrate permutation, translation, and rotation equivariance individually. (permutation/translation eq are manifest in layer definitions)
- TFNs act on points with associated features. A layer L takes a finite set S of vectors in R3 and a vector in X at each point in S and outputs a vector in Y at each point in S, where X and Y are vector spaces

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

where r_a ∈ R3 are the point coordinates and x_a ∈ X , y_a ∈ Y are the feature vectors (a is index over points in S)

$$\mathbb{R}^3 \oplus \mathcal{X}$$

Type of Equivariance

- Permutation: layers manifestly equivariant to permutation of points (no order imposed)
- Translation: manifestly equivariant since they all only involve differences between points $\vec{r}_i - \vec{r}_j$

Type of Equivariance (cont.)

- Rotation invariance:

Condition:

$$\mathcal{L} \circ \left[\mathcal{R}(g) \oplus D^{\mathcal{X}}(g)\right] = \left[\mathcal{R}(g) \oplus D^{\mathcal{Y}}(g)\right] \circ \mathcal{L}$$

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

$g \in SO(3)$

$\vec{r} \in \mathbb{R}^3$

$x \in \mathcal{X}$

$D^{\mathcal{X}}$ <-representation of SO(3) on vector space X

$\mathcal{R}(g)\vec{r}$ <- group action on vector r

$D^{\mathcal{X}}(g)x$ ^group action on the vector x

Rotation Equivariance (cont.)

- Decompose our representation into irrreducible representations of SO(3), which have dimension 2l+1 and are unitary.
- l=0->scalars, l=1->vectors in 3-space, l=2->symmetric traceless matrices

$D^{(l)}$ <-Wigner D-matrices, map elements of SO(3) to (2l+1) X (2l+1) matrices

$$D^{(0)}(g) = 1 \qquad \text{and} \qquad D^{(1)}(g) = \mathcal{R}(g).$$

Tensor Field Network Layers

- Input and output of each layer in TFN is **a finite set S of points in R3 and a vector in a representation of SO(3) associated with each point**
- Decompose this rep into irreps
- Multiple instance of each l-rotation-order irrep, analogous to "depth" of a convolution in CNN, referred to as **channels**

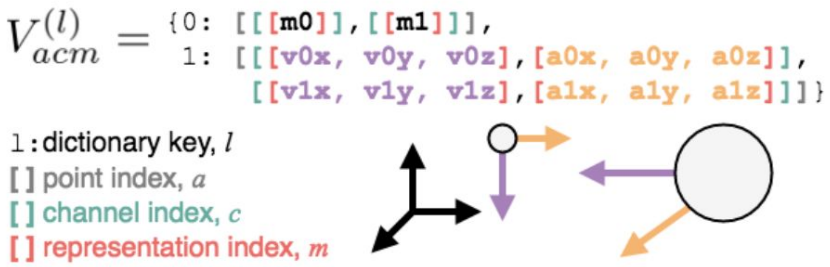


Figure 1: Example of $V^{(l)}_{acm}$ representing two point masses with velocities and accelerations. Colored brackets indicate the a (point), c (channel), and m (representation) indices. (l) is the key for the $V^{(l)}_{acm}$ dictionary of feature tensors.

Point Convolution

- At each layer, have input structure $V_{acm}^{(l)}$
- Point convolution performs same operation at each point a, taking all other points (both their relative location and values of V) as input
- Standard CNN can be viewed as a point cloud of a grid of points with regular spacing

Spherical Harmonics and Filters

- $Y_m^{(l)}$ Are functions from points on the sphere to real or complex numbers. (l is non-negative integer and m runs from -l to l is also an integer)
- Examples: $Y^{(0)}(\hat{r}) \propto 1$ and $Y^{(1)}(\hat{r}) \propto \hat{r}.$
- These are equivariant to SO(3): $Y_m^{(l)}(\mathcal{R}(g)\hat{r}) = \sum_{m'} D_{mm'}^{(l)}(g) Y_{m'}^{(l)}(\hat{r}).$
- Define our filters to be of the form:

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

- Where l_f and and l_i are rotation order of input and filter
- $R_c^{(l_f,l_i)} : \mathbb{R}_{\geq 0} \to \mathbb{R}$ are the learned functions, contain most of the parameters of the TFN

Combining representations with Tensor Products

- Inputs to each layer inhabit reps of SO(3), need to combine outputs in a way that they also inhabit reps of SO(3) so they can be fed downstream
- Tensor product of representations: $D^{\mathcal{X}} \otimes D^{\mathcal{Y}} = D^{\mathcal{X} \otimes \mathcal{Y}}$
- For irreps with $u^{(l_1)} \in l_1$ and $v^{(l_2)} \in l_2$, we can calculate:

$u^{(l_1)} \otimes v^{(l_2)} \in l_1 \otimes l_2$ using Clebsch-Gordan coefficients:

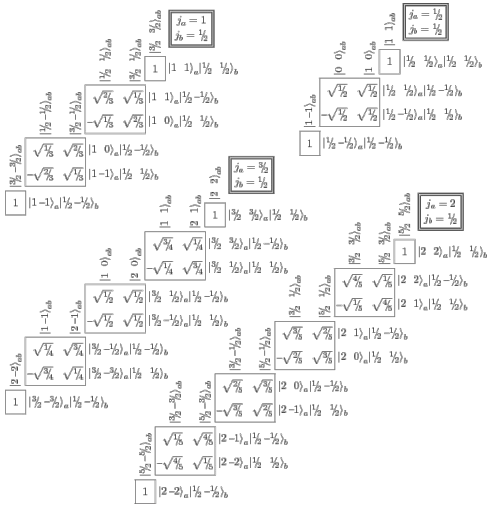
$$(u \otimes v)^{(l)}_m = \sum_{m_1=-l_1}^{l_1} \sum_{m_2=-l_2}^{l_2} C^{(l,m)}_{(l_1,m_1)(l_2,m_2)} u^{(l_1)}_{m_1} v^{(l_2)}_{m_2}$$

$1 \otimes 1 \to 0$:

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

$1 \otimes 1 \to 1$,

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



Layer Definition

- Input inhabits one rep, filter in another, together produce outputs at possibly many rotation orders. Put everything together into pointwise convolution layer definition:

$$\mathcal{L}^{(l_o)}_{acm_o}\left(\vec{r}_a, V^{(l_i)}_{acm_i}\right) := \sum_{m_f, m_i} C^{(l_o,m_o)}_{(l_f,m_f)(l_i,m_i)} \sum_{b \in S} F^{(l_f,l_i)}_{cm_f}(\vec{r}_{ab}) V^{(l_i)}_{bcm_i} \tag{3}$$

- Equivariance of filter F and CG coefficients imply point convolutions are equivariant

Equivariance of Layers

- Red: representation index
- Black: point index
- Green: channel index

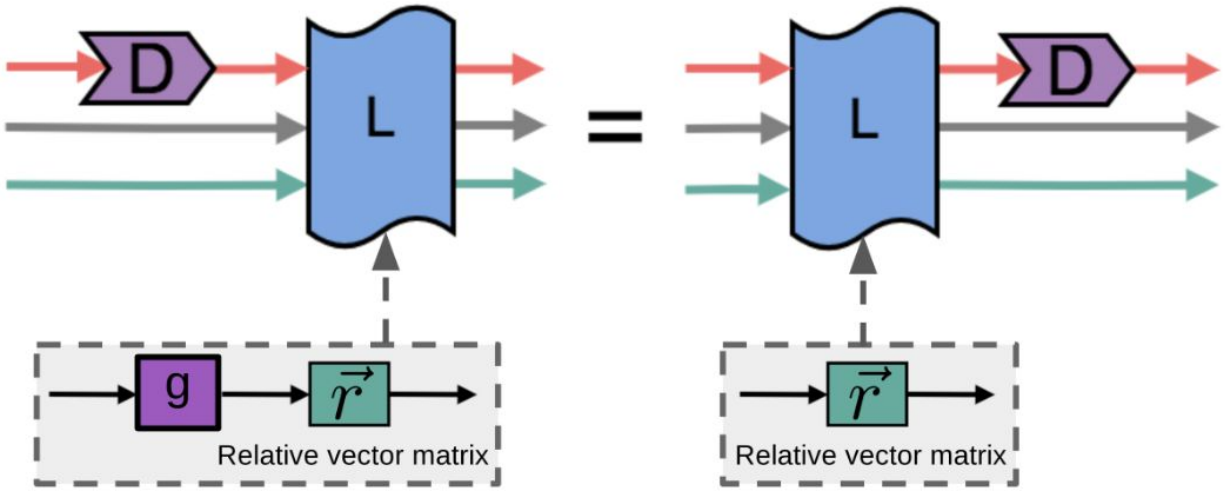


Figure S1: Condition for layer rotation equivariance

Equivariance of Layers

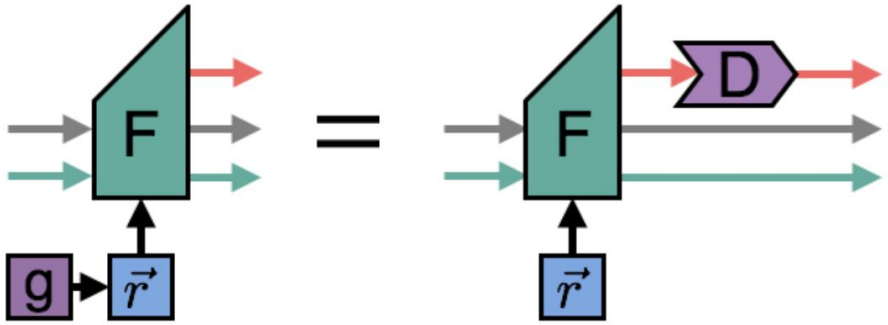


Figure S2: Filter equivariance equation

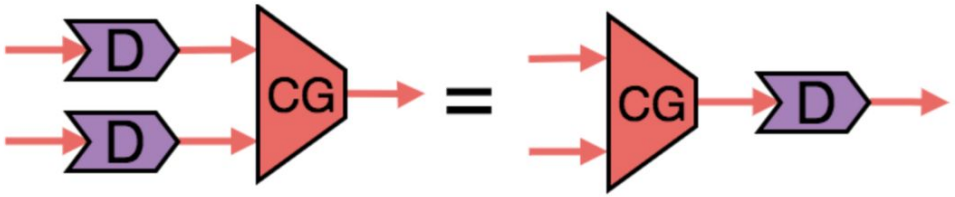


Figure S3: Equivariance of Clebsch-Gordan coefficients. Note that each D may refer to a different irreducible representation.

Equivariance of Layers

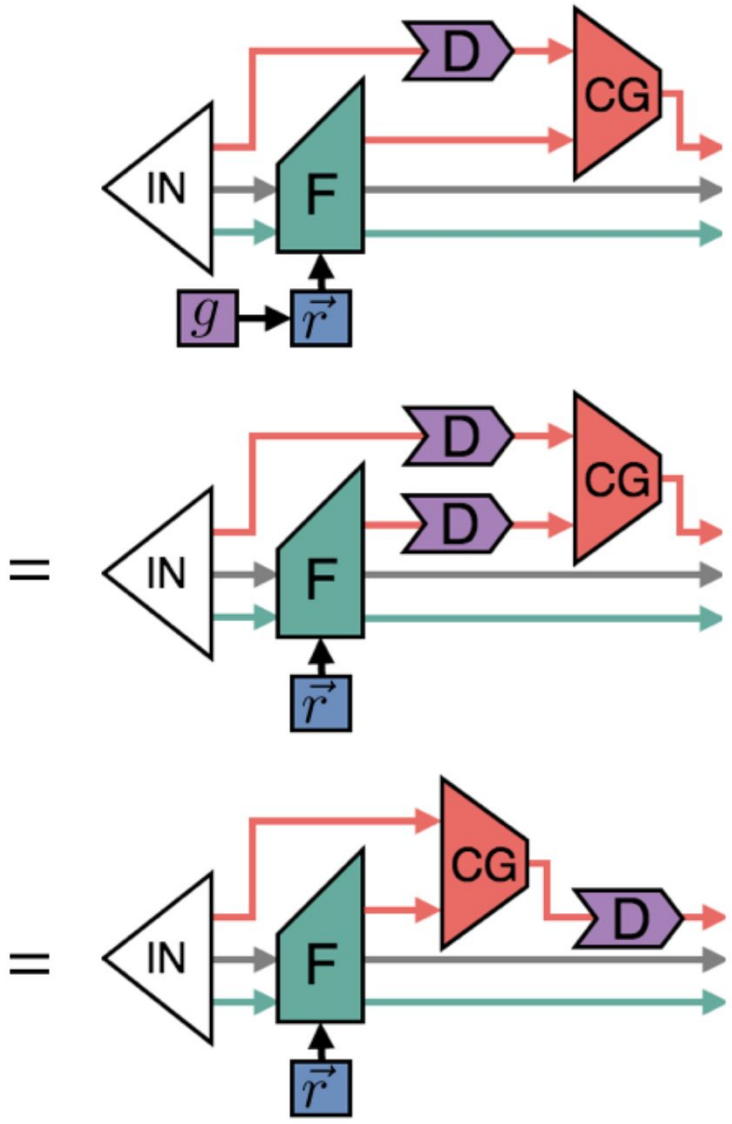


Figure S4: Diagrammatic proof of point convolution rotation equivariance.

Self Interaction

- Point convolutions scale feature vectors element wise and self-interaction layers used to mix together components of the feature vectors at each point.
- Act like l=0 (scalar) filters: $\sum_{c'} W_{cc'}^{(l)} V_{ac'm}^{(l)}$
- Each rotation order has different weights, but same weights used for every m of a given order (essential for equivariance.) For l=0 may also use biases.
- D-matrices commute with weight matrix W (W has no m index) so this layer is equivariant for l>0. (trivially equivariant for l=0) $D^{(0)} = 1.$

Nonlinearity

- Act as scalar transform in the l spaces (along m dimension)

$$\eta^{(0)}\left(V_{ac}^{(0)} + b_c^{(0)}\right) \qquad \eta^{(l)}\left(\|V\|_{ac}^{(l)} + b_c^{(l)}\right) V_{acm}^{(l)}$$

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

- Note that: $\|D(g)V\| = \|V\|$ since D is unitary, therefore this layer is a scalar transformation in rep index m, so it is equivariant

Demonstrations

Demonstrations

- Each of these tasks either unnatural or impossible in existing models
- Largest task described uses dozens of points, but TFNs can scale to over a thousand points on standard GPU.
- Use radial functions and nonlinearities identical to: K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko. Quantum-chemical insights from deep tensor neural networks. 8:13890 EP –, Jan 2017.
- Use radial basis functions composed of Gaussians, and two fully connected layers are applied to this basis vector

Geometry: Shape Classification (0 -> 0)

- During training, input to the network a dataset of shapes in a single orientation, it learns to classify the shapes. Then test the network with shapes that have been rotated/translated randomly. Network automatically performed as well on test dataset as it did on training dataset

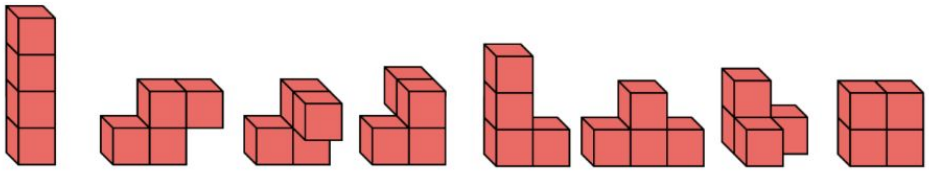


Figure 2: 3D Tetris shapes. Blocks correspond to single points. The third and fourth shapes from the left are mirrored versions of each other.

Geometry: Shape Classification (0 -> 0)

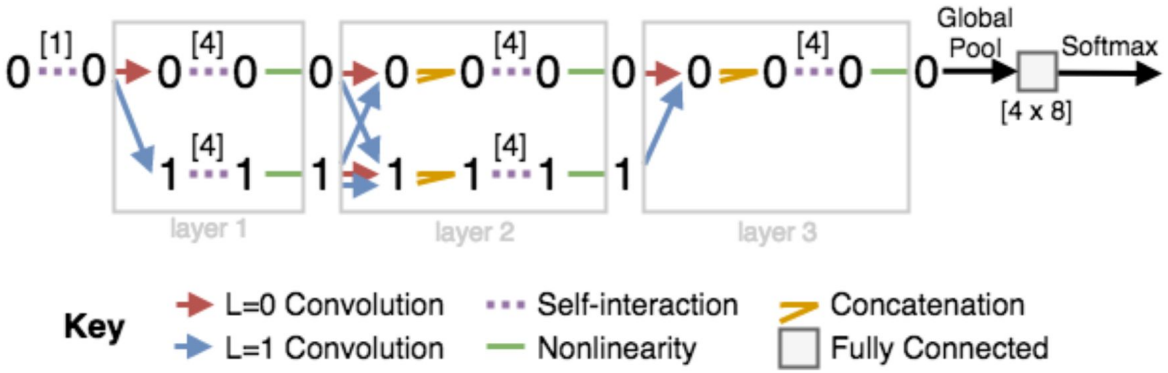


Figure 3: Network diagrams for shape classification task showing how information flows between tensors of different order. Clebsch-Gordan tensors are implied in the arrows indicating convolutions. The numbers above the self-interactions indicate the number of channels. Individual convolutions, indicated by arrows, each produce a separate tensor, and concatenation is performed after the convolutions.

- Only use the l = 0 output since shape classes are invariant under rotation and hence scalars
- To get the classification from the l=0 output, sum over the feature vectors of all points. (global pooling is equivariance because D matrices act linearly
- TFN can distinguish the shapes which are mirrors of each other, other networks can't

Physics: vectors and tensors in classical mechanics

(0->1 and 0->0+2)

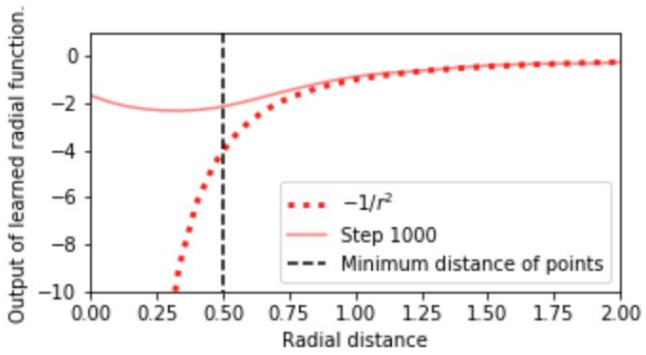
- Train networks to calculate acceleration vectors of point masses under newtonian gravity and moment of inertia tensors at a specific point of a collection of point masses
- Only require a single layer of point convolution to demonstrate (l=1 for Newtonian gravity and l=0 and l=2 for moment of inertia.)
- Input: set of random points with associated random masses, for moment of inertia also designate a different special point about which to calculate the moment of inertia tensor

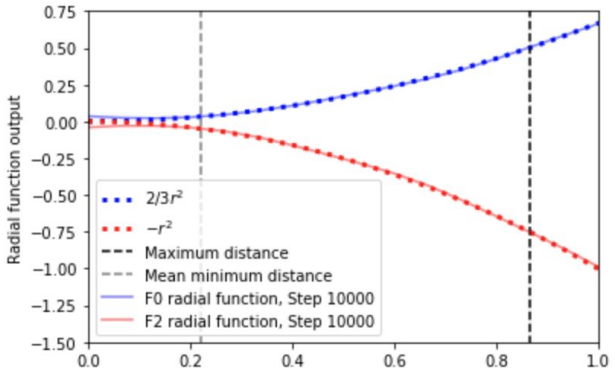
Physics: vectors and tensors in classical mechanics (cont.)

(0->1 and 0->0+2)

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

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





Chemistry: toward geometrically generating molecular structures

(0->0+1)

- Randomly remove a point from a point cloud and ask the network to replace that point.
- First step toward general isometry-equivariant generative models for 3D point clouds
- Train on molecular structures since precise positions important for chemical behaviour

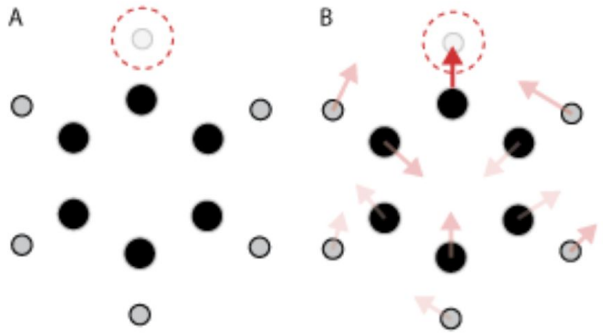


Figure 4: A hypothetical example input and output of the missing point network. (A) A benzene molecule with hydrogen removed (B) The relative output vectors produced by the network, with arrows shaded by the associated probabilities.

Chemistry: toward geometrically generating molecular structures (cont.)

(0->0+1)

- Output consists of a vector at each point (indicating where the missing point should be relative to the starting point,) an array of scalars (indicating the point type,) anda special scalar (used as a probability, measures confidence in that points vote.)
- Aggregate votes for a location using the weighted sum:

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

Chemistry: toward geometrically generating molecular structures (cont.)

(0->0+1)

- Trained on structures in QM9 dataset (consisting of 134,000 molecules with up to 9 heavy atoms.)
- In a single epoch, train on each molecule in the training set with one randomly selected atom deleted. Then test on a random selection of other molecules, making a prediction for every possible atom that could be removed.
- After training fro 225 epochs on a dataset with only 1000 molecules (each with 5-18 atoms) see excellent generalization to test sets containing larger molecules:

Table 1: Performance on missing point task

Atoms	Number of predictions	Accuracy (%) ($\leq$ 0.5 Å and atom type)	Distance MAE in Å
5-18	15863	91.3 (train)	0.16
19	19000	93.9 (test)	0.14
23	23000	96.5 (test)	0.13
25-29	25356	97.3 (test)	0.16

Future Work

Future Work

- Atomic systems: intend to train networks to predict properties of larger heterogeneous systems, learn molecular dynamics, calculate electron densities, and hypothesize new stable structures to ultimately design new useful materials, drugs, and chemicals
- Potential applications in modeling complex fluid flows, analyzing detector events in particle physics experiments, and studying configurations of stars and galaxies
- Other applications in 3D perception, robotics, computational geometry, and bioimaging

Thank you! Questions?

UC SANTA BARBARA