Neural networks with Euclidean Symmetry for the Physical Sciences

SLIDES: https://tinyurl.com/e3nn-physics-meets-ml

Tess Smidt
2018 Alvarez Fellow
in Computing Sciences

Physics ∩ ML
2020.11.18

Neural networks with Euclidean Symmetry for the Physical Sciences

SLIDES: https://tinyurl.com/e3nn-physics-meets-ml

All I wanted was 3D rotation equivariance and I got...

....geometric tensors, space groups, point groups, selection rules, normal modes, degeneracy, 2nd order phase transitions, and a much better understanding of physics.

BERKELEY LAB

Tess Smidt
2018 Alvarez Fellow
in Computing Sciences

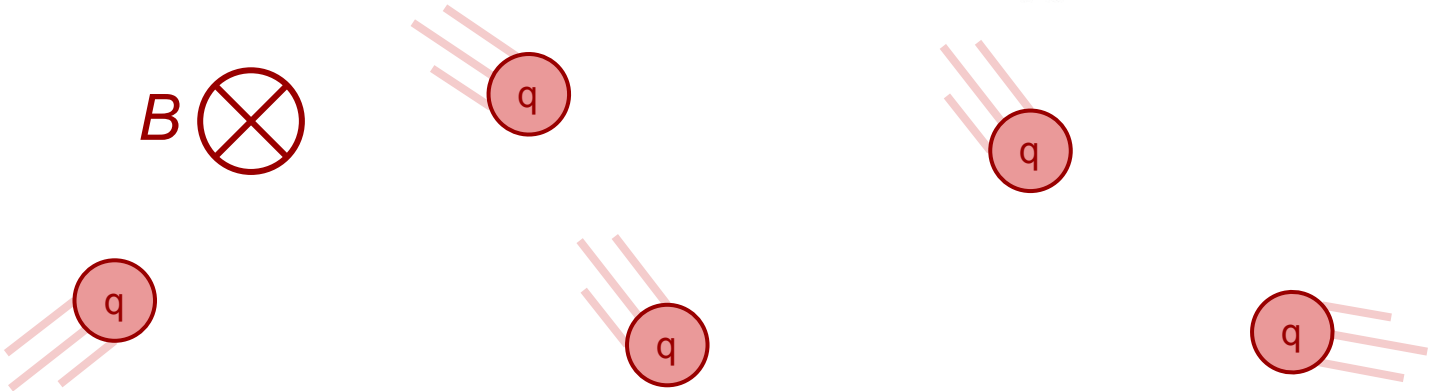
Physics ∩ ML
2020.11.18

The laws of physics have rotational, translational, and (unless you're a particle physicist) parity symmetry.

We want machine learning models that also obey this symmetry.

e.g. a **network** is our model of **"physics"**. The **input** to the network is our **system**.

$$\vec{F}(q, \vec{r}, \vec{v}, \vec{B}) = \vec{F}_i = \sum_i q_i(\vec{v_i} \times \vec{B}) + \sum_{i \neq j} \frac{q_i q_j}{r_{ij}^2} \hat{r_{ij}}$$



Symmetry emerges when different ways of representing something “mean” the same thing.

Symmetry of representation vs. objects

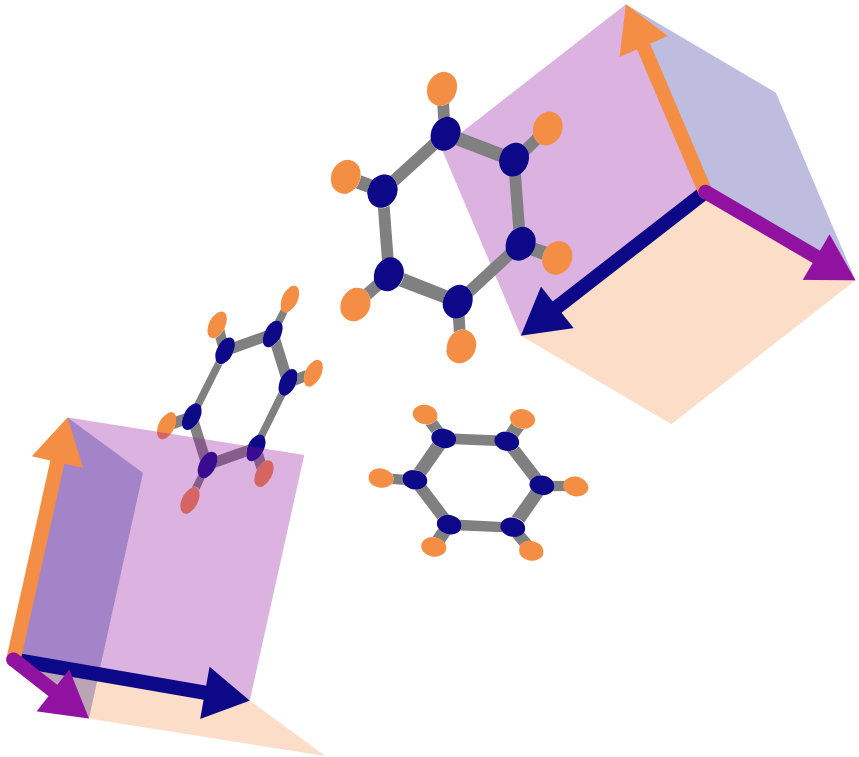
Symmetry emerges when different ways of representing something “mean” the same thing.

Symmetry of representation vs. objects

Euclidean symmetry, E(3):

Symmetry of 3D space

The freedom to choose your coordinate system



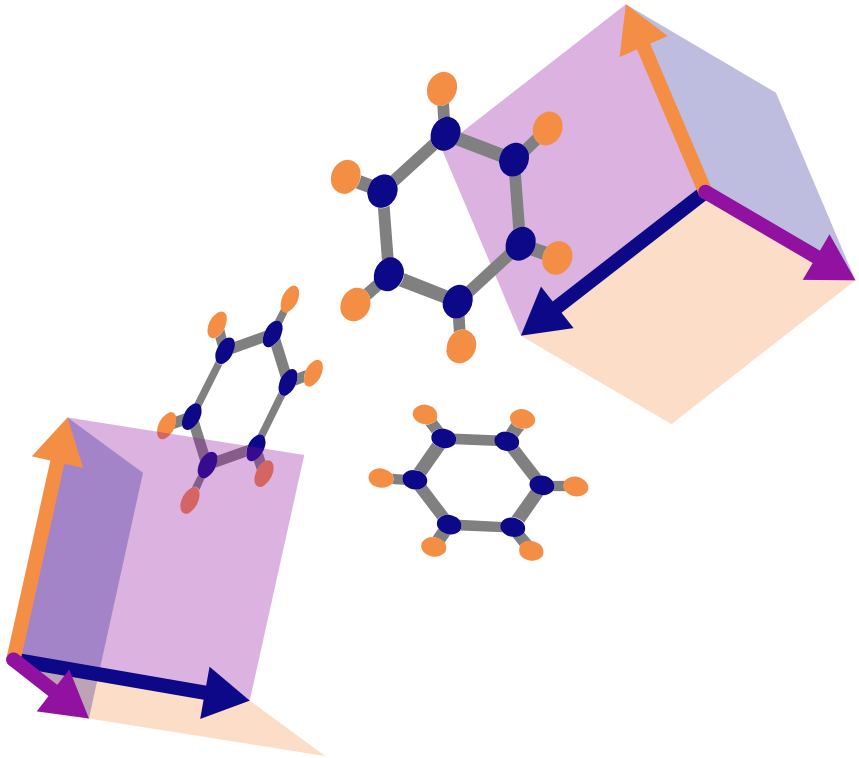
Symmetry emerges when different ways of representing something "mean" the same thing.

Symmetry of representation vs. objects

Euclidean symmetry, E(3):

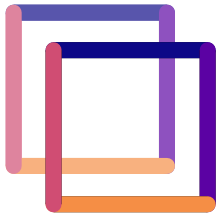
Symmetry of 3D space

The freedom to choose your coordinate system

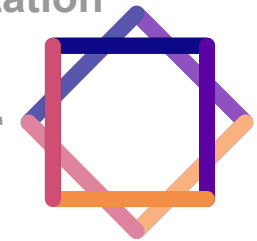


We transform between coordinate systems with...

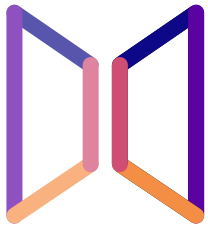
3D Translation



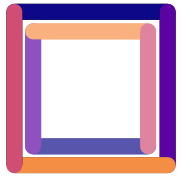
3D Rotation



Mirrors



3D Inversion



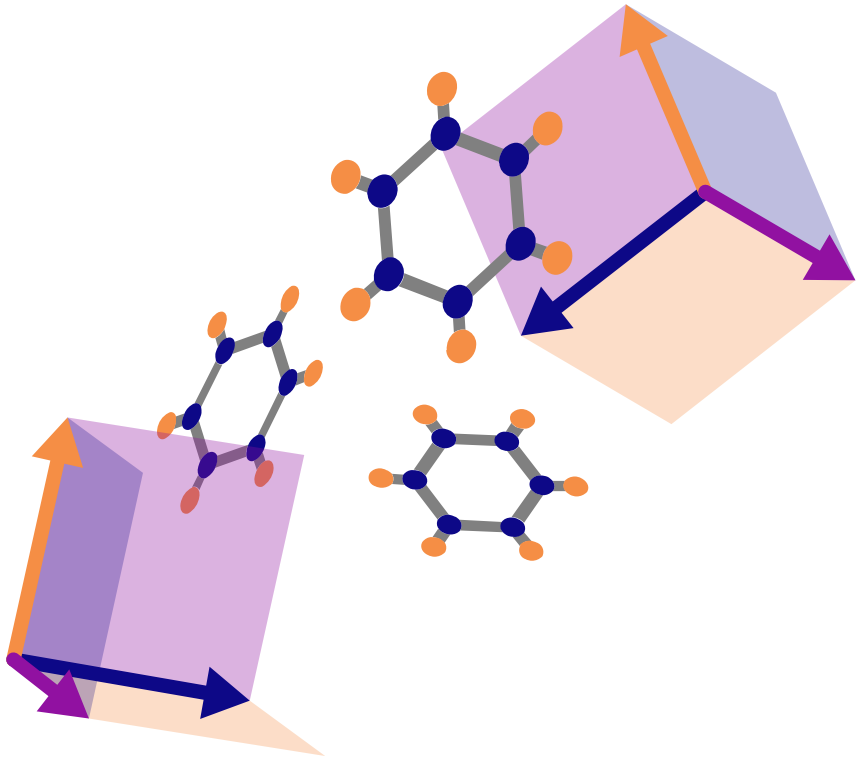
Symmetry emerges when different ways of representing something "mean" the same thing.

Symmetry of representation vs. objects

Euclidean symmetry, E(3):

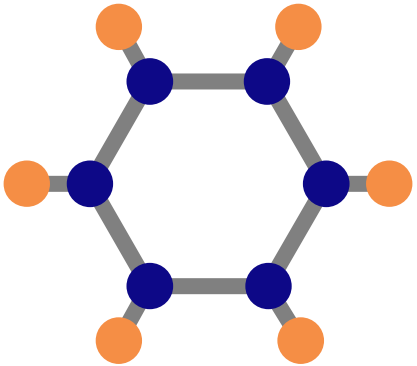
Symmetry of 3D space

The freedom to choose your coordinate system



Symmetry of geometric objects

Looks the same under specific rotations, translations, and inversion (includes mirrors).



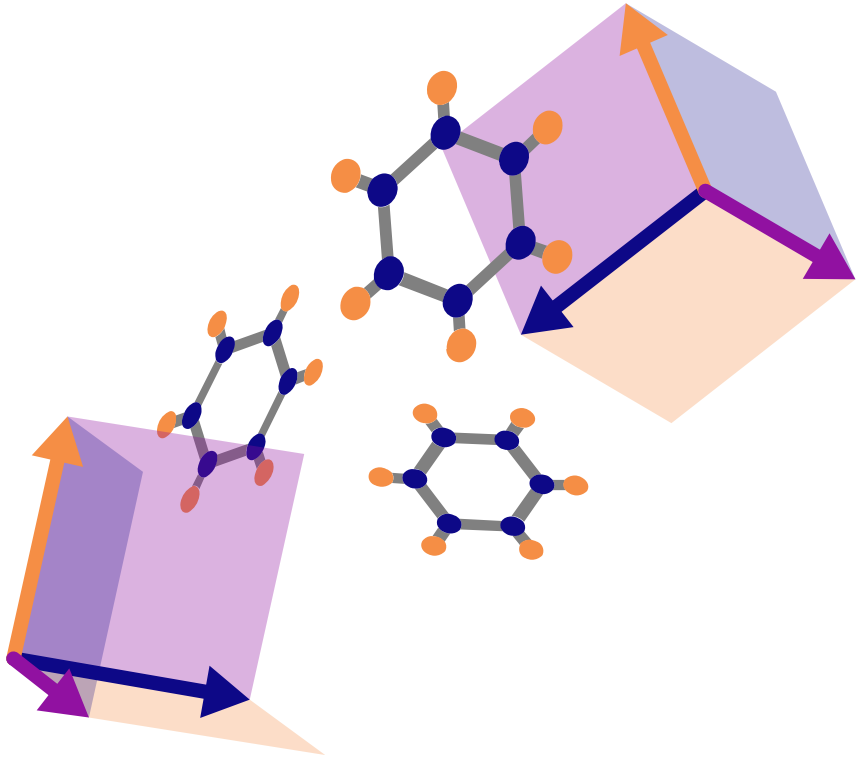
Symmetry emerges when different ways of representing something "mean" the same thing.

Symmetry of representation vs. objects

Euclidean symmetry, E(3):

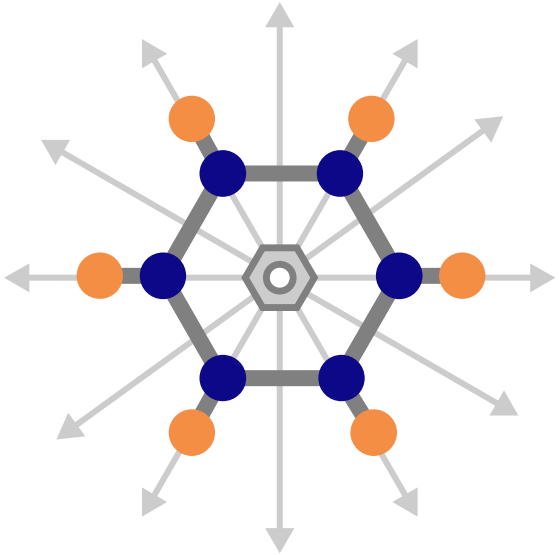
Symmetry of 3D space

The freedom to choose your coordinate system



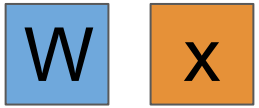
Symmetry of geometric objects

Looks the same under specific rotations, translations, and inversion (includes mirrors).

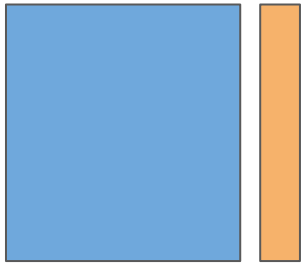


$$D_{6h} = 6/mmm$$

Neural networks are specially designed for different data types.
Assumptions about the data type are built into how the network operates.

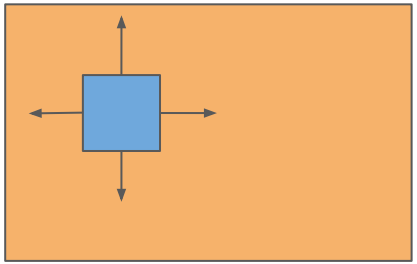


Arrays ⇨ Dense NN



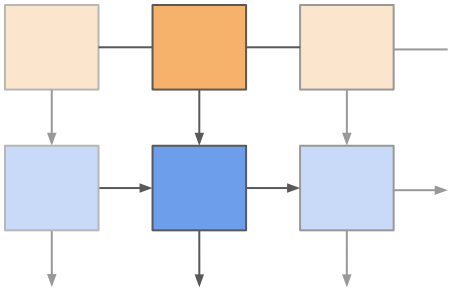
Components are independent.

2D images ⇨ Convolutional NN



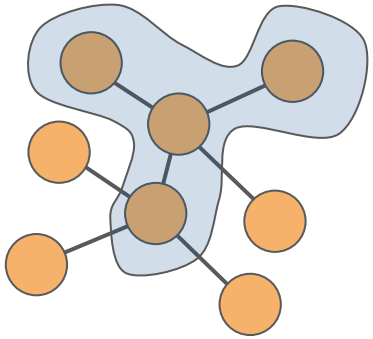
The same features can be found anywhere in an image. Locality.

Text ⇨ Recurrent NN



Sequential data. Next input/output depends on input/output that has come before.

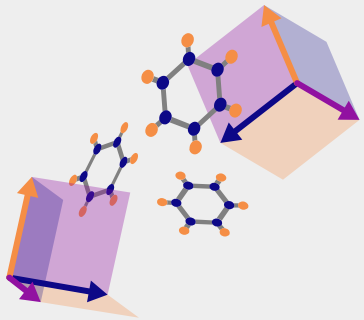
Graph ⇨ Graph (Conv.) NN



Topological data. Nodes have features and network passes messages between nodes connected via edges.

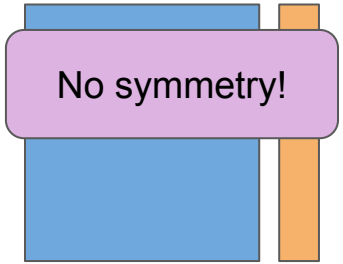
3D physical data ⇨ Euclidean NN

Data in 3D Euclidean space. Freedom to choose coordinate system.



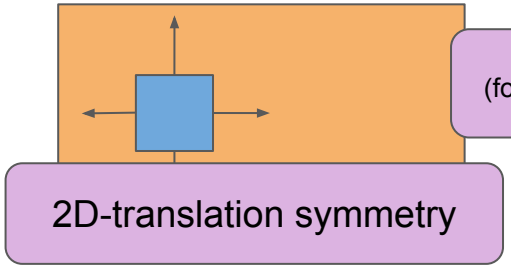
Neural networks are specially designed for different data types.
Assumptions about the data type are built into how the network operates.
Thus, symmetries are encoded by tailoring network operations.

Arrays ⇨ Dense NN



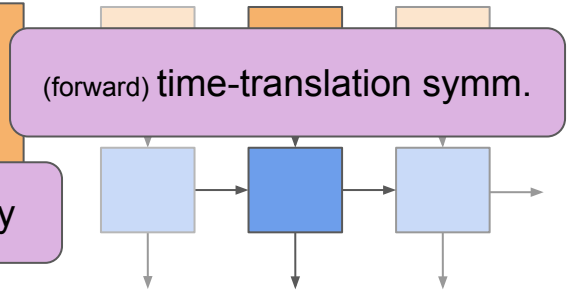
Components are independent.

2D images ⇨ Convolutional NN



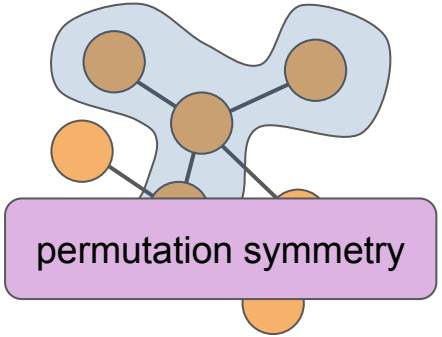
The same features can be found anywhere in an image. Locality.

Text ⇨ Recurrent NN



Sequential data. Next input/output depends on input/output that has come before.

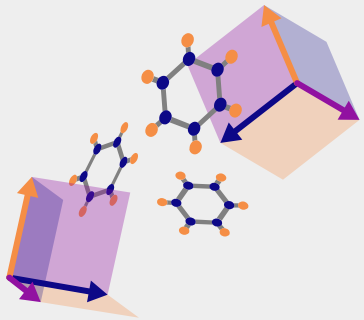
Graph ⇨ Graph (Conv.) NN



Topological data. Nodes have features and network passes messages between nodes connected via edges.

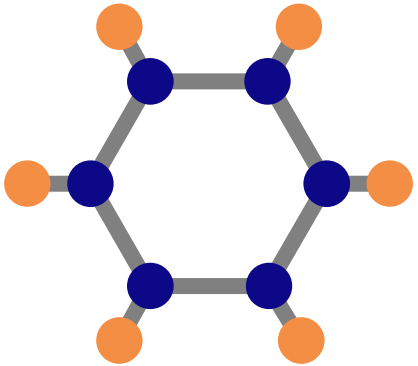
3D physical data ⇨ Euclidean NN

3D Euclidean symmetry **E(3)**: 3D rotations, translations, and inversion



Three ways to make models “symmetry-aware” for 3D data

e.g. How to make a model that “understands” the symmetry of atomic structures?

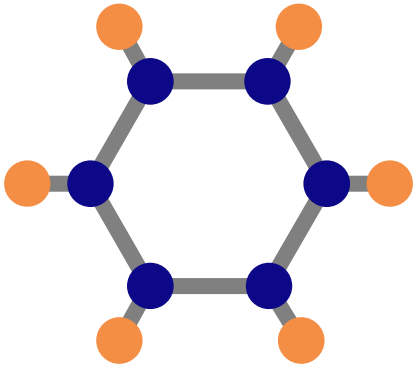


H	-0.21463	0.97837	0.33136
C	-0.38325	0.66317	-0.70334
C	-1.57552	0.03829	-1.05450
H	-2.34514	-0.13834	-0.29630
C	-1.78983	-0.36233	-2.36935
H	-2.72799	-0.85413	-2.64566
C	-0.81200	-0.13809	-3.33310
H	-0.98066	-0.45335	-4.36774
C	0.38026	0.48673	-2.98192
H	1.14976	0.66307	-3.74025
C	0.59460	0.88737	-1.66708
H	1.53276	1.37906	-1.39070

Coordinates are most general, but sensitive to translations, rotations, and inversion.

Three ways to make models “symmetry-aware” for 3D data

e.g. How to make a model that “understands” the symmetry of atomic structures?



H	-0.21463	0.97837	0.33136
C	-0.38325	0.66317	-0.70334
C	-1.57552	0.03829	-1.05450
H	-2.34514	-0.13834	-0.29630
C	-1.78983	-0.36233	-2.36935
H	-2.72799	-0.85413	-2.64566
C	-0.81200	-0.13809	-3.33310
H	-0.98066	-0.45335	-4.36774
C	0.38026	0.48673	-2.98192
H	1.14976	0.66307	-3.74025
C	0.59460	0.88737	-1.66708
H	1.53276	1.37906	-1.39070

Coordinates are most general, but sensitive to translations, rotations, and inversion.

Approach 1:

Data Augmentation

Throw data at the problem and see what you get!

Approach 2:

Invariant Inputs

Convert your data to invariant representations so the neural network can’t possibly mess it up.

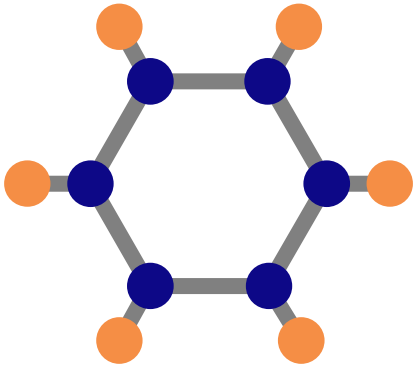
Approach 3:

Invariant models
Equivariant models

If there’s no model that naturally handles coordinates, we will make one.

Three ways to make models “symmetry-aware” for 3D data

e.g. How to make a model that “understands” the symmetry of atomic structures?



H	-0.21463	0.97837	0.33136
C	-0.38325	0.66317	-0.70334
C	-1.57552	0.03829	-1.05450
H	-2.34514	-0.13834	-0.29630
C	-1.78983	-0.36233	-2.36935
H	-2.72799	-0.85413	-2.64566
C	-0.81200	-0.13809	-3.33310
H	-0.98066	-0.45335	-4.36774
C	0.38026	0.48673	-2.98192
H	1.14976	0.66307	-3.74025
C	0.59460	0.88737	-1.66708
H	1.53276	1.37906	-1.39070

Coordinates are most general, but sensitive to translations, rotations, and inversion.

Approach 1:
~~Data Augmentation~~

Throw data at the problem and see what you get!

Approach 2:
~~Invariant Inputs~~

👌

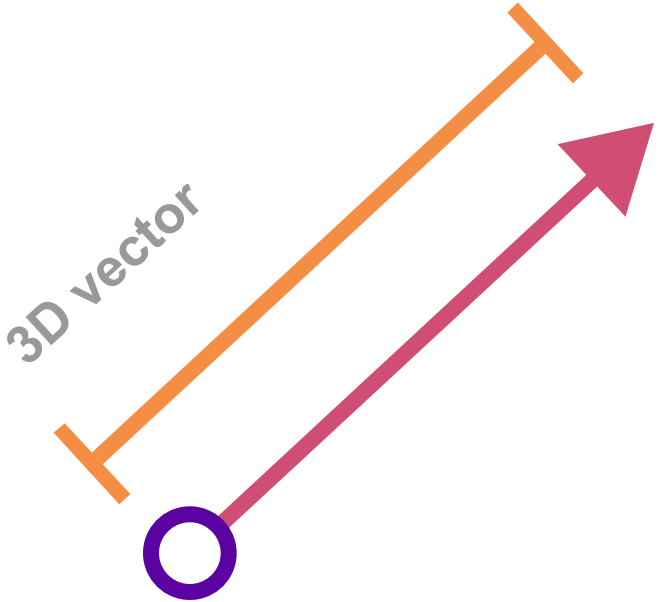
Convert your data to invariant representations so the neural network can’t possibly mess it up.

Approach 3:
~~Invariant models~~
Equivariant models

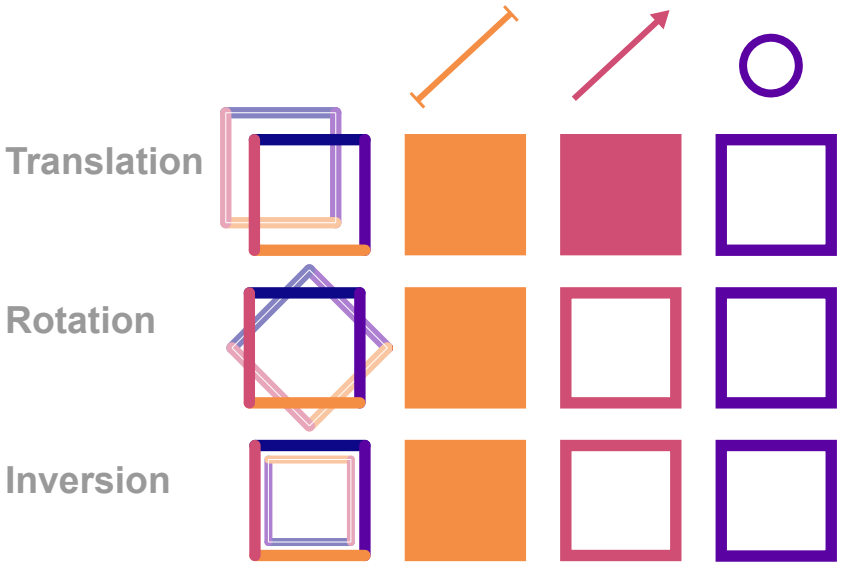
If there’s no model that naturally handles coordinates, we will make one.

Invariance vs. Equivariance (covariance) e.g. in 3D space

Does NOT change ⇨ Invariant
Changes deterministically ⇨ Equivariant

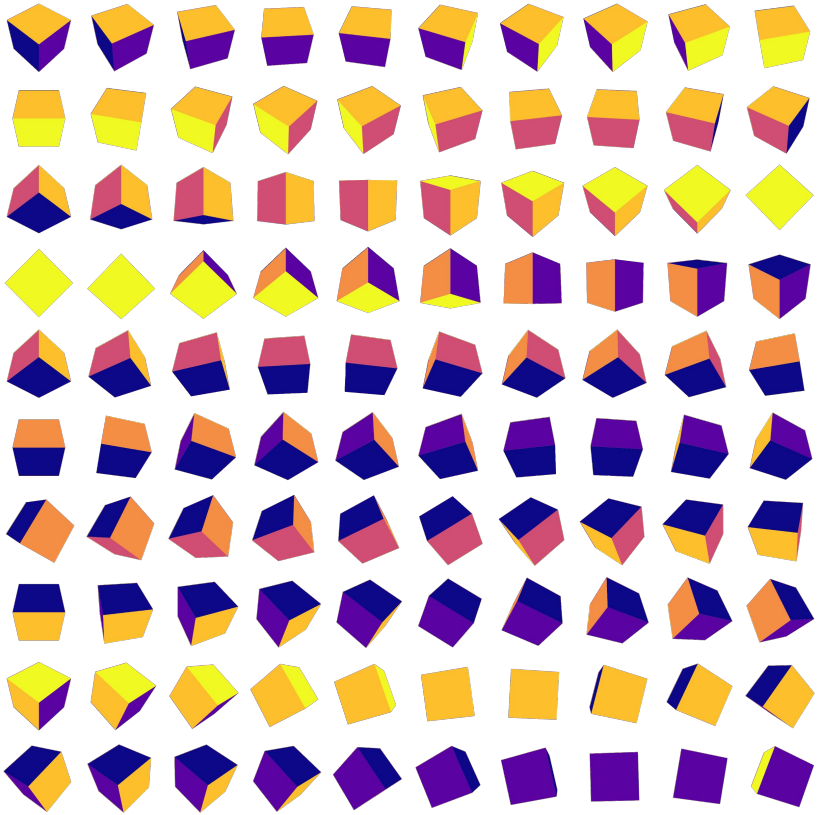


Properties of a vector under E(3)



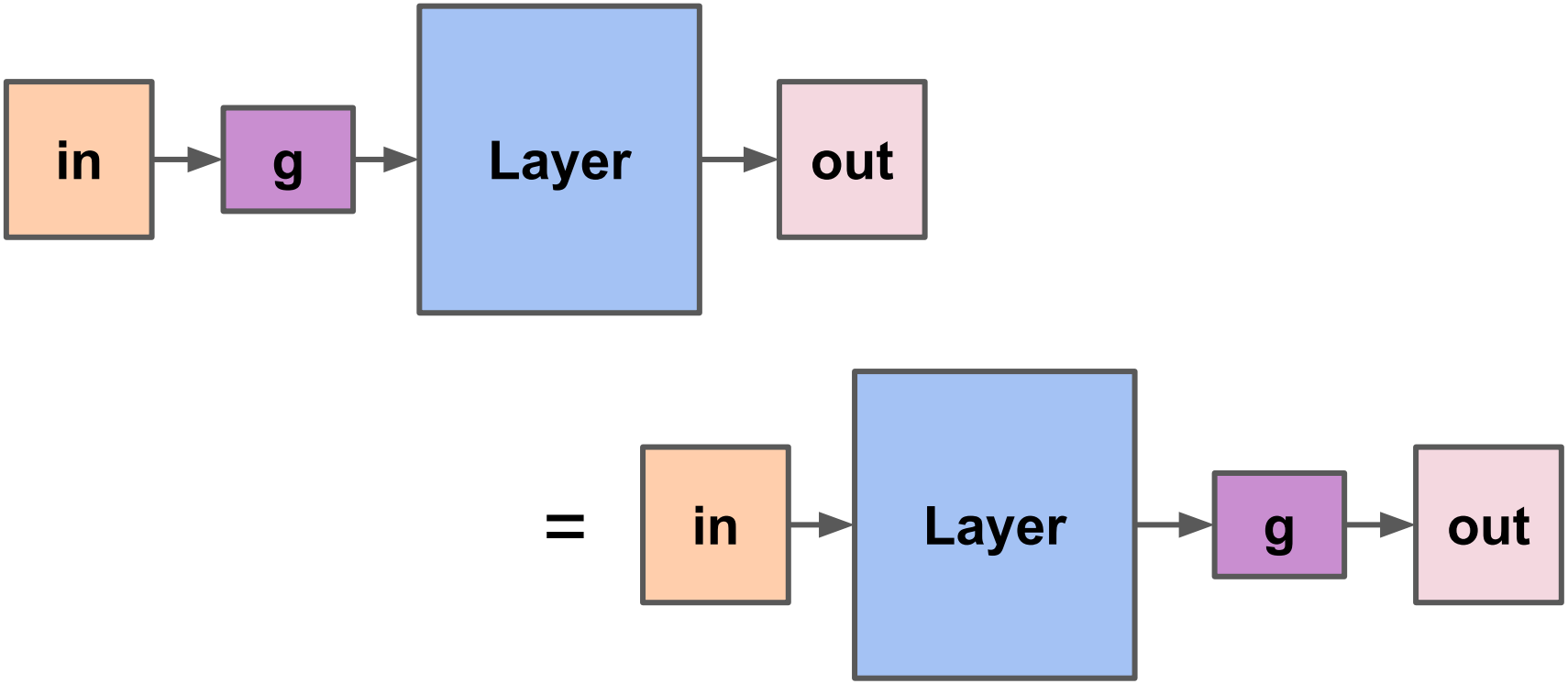
For 3D data, data augmentation is expensive, ~500 fold augmentation and you *still* don't get the guarantee of equivariance (it's only emulated).

training without rotational symmetry

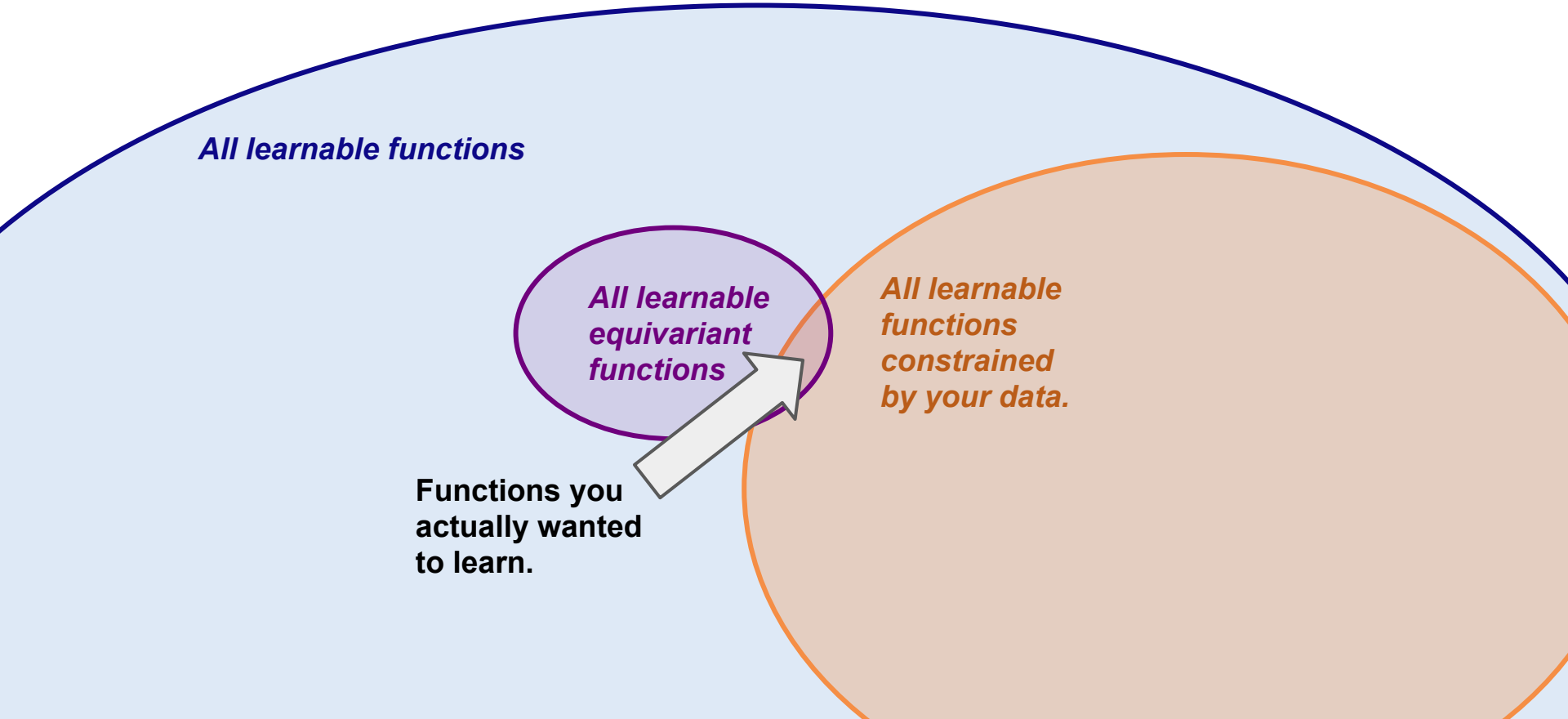


training with symmetry

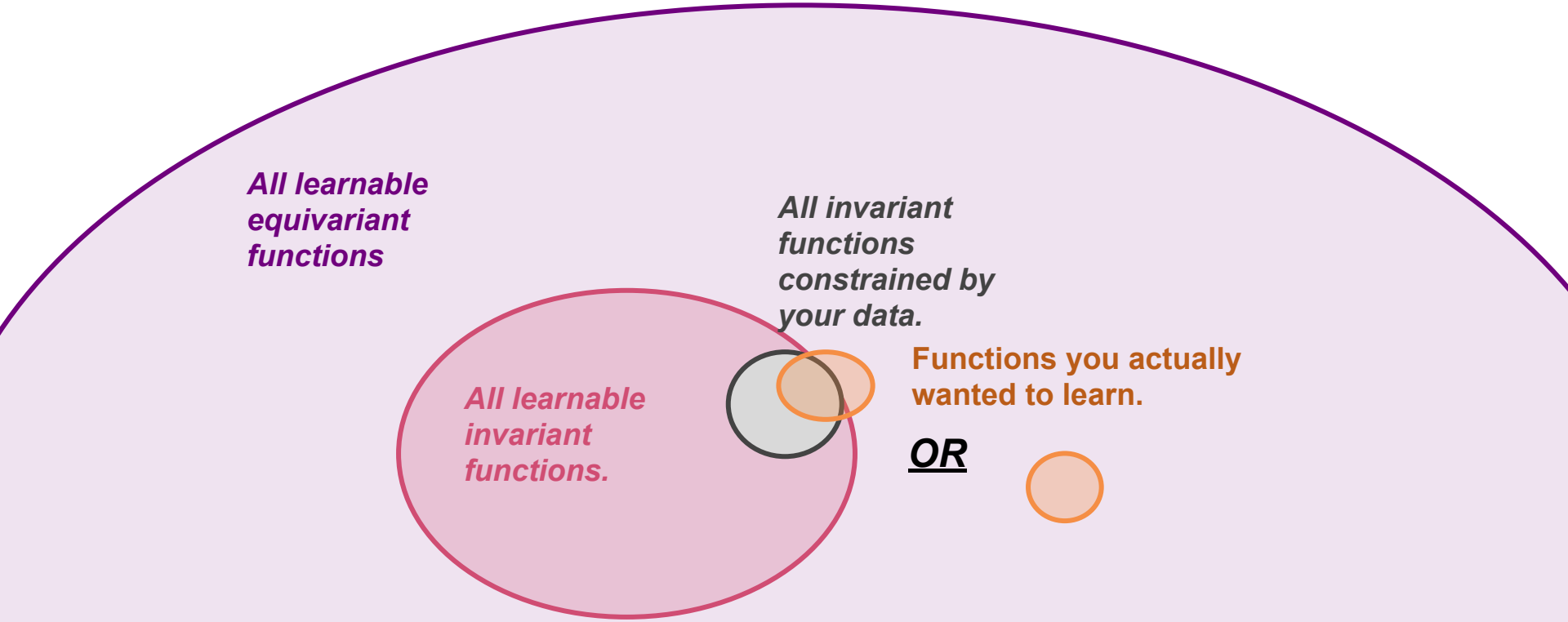
For a function to be *equivariant* means that we can act on our inputs with g OR act our outputs with g and we get the same answer (for every operation).
For a function with *invariant* input (e.g. invariant models) means g is the identity (no change).



Why limit yourself to equivariant functions?
You can *substantially* shrink the space of functions you need to optimize over.
This means you need less data to constrain your function.



Why not limit yourself to invariant functions?
You have to guarantee that your input features already contain any necessary equivariant interactions *(e.g. cross-products)*.



How Euclidean Neural Networks achieve equivariance to Euclidean symmetry (high level)

Euclidean Neural Networks encompass

Tensor Field Networks (arXiv:1802.08219)
Clebsch-Gordon Nets (arXiv:1806.09231)
3D Steerable CNNs (arXiv:1807.02547)
Cormorant (arXiv:1906.04015)
SE(3)-Transformers (arXiv:2006.10503)
e3nn (github.com/e3nn/e3nn)
(Technically, e3nn is the only one that implements inversion)

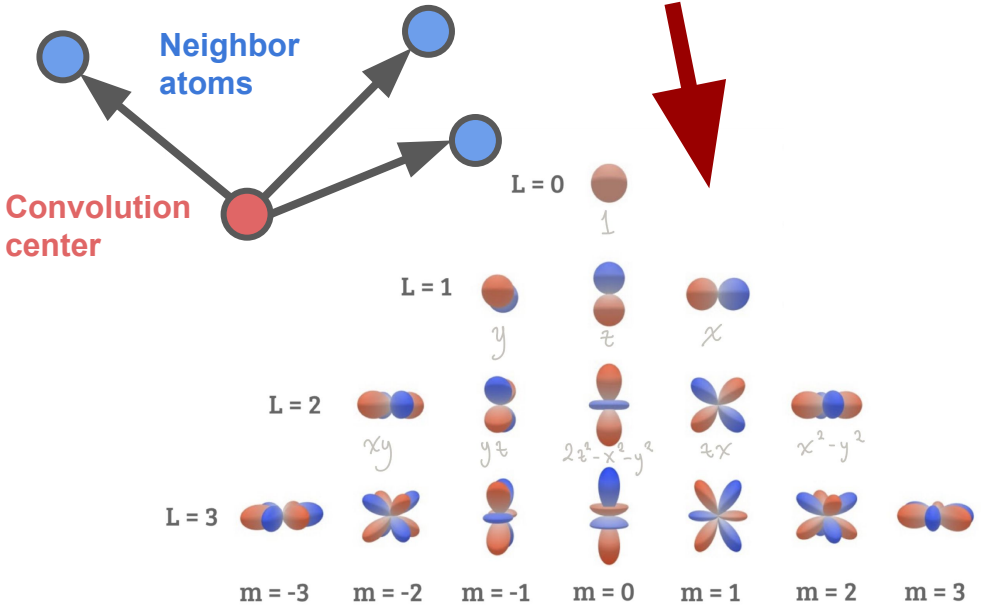
Some relevant folks… Mario Geiger, Ben Miller, Risi Kondor, Taco Cohen, Maurice Weiler, Daniel E. Worrall, Fabian B. Fuchs, Max Welling, Nathaniel Thomas, Shubhendu Trivedi,...

Euclidean Neural Networks are similar to convolutional neural networks...

Euclidean Neural Networks are similar to convolutional neural networks...

Equivariant convolutional filters are based on learned radial functions and spherical harmonics...

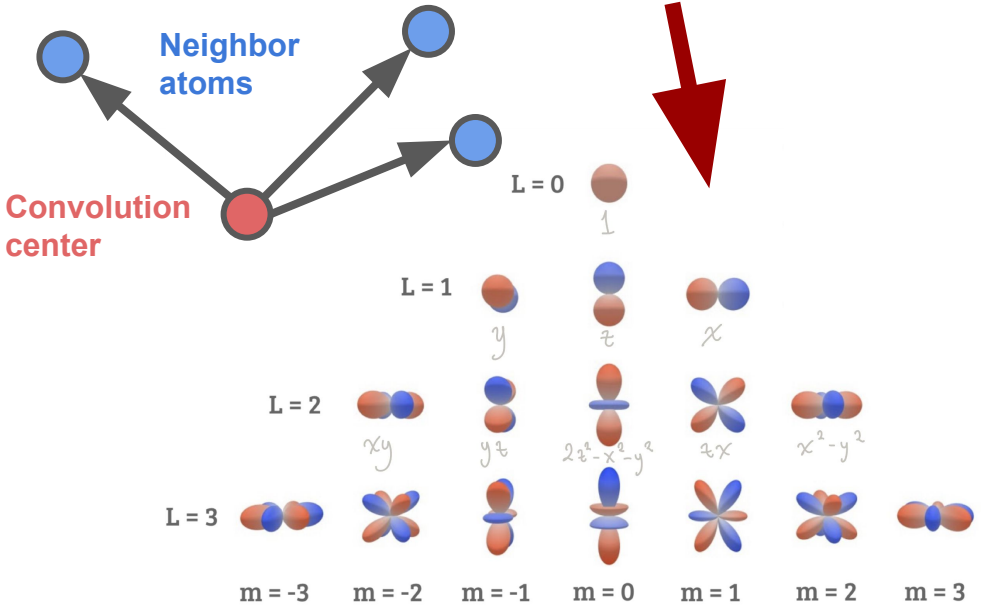
$$W(\vec{r}) = R(r)Y_l^m(\hat{r})$$



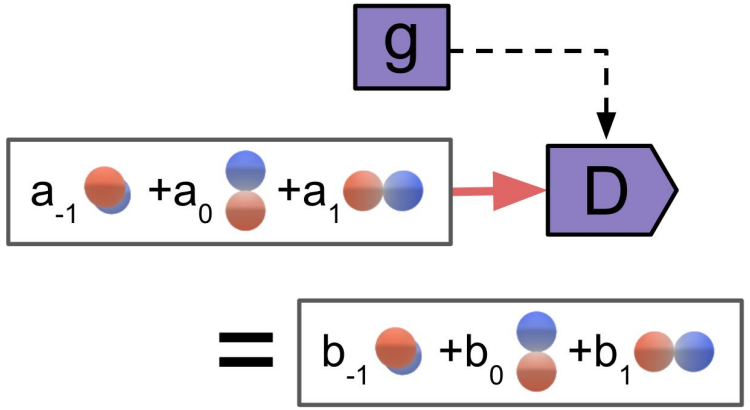
Euclidean Neural Networks are similar to convolutional neural networks...

Equivariant convolutional filters are based on learned radial functions and spherical harmonics...

$$W(\vec{r}) = R(r)Y_l^m(\hat{r})$$



Spherical harmonics of the same L transform together under rotation **g**.

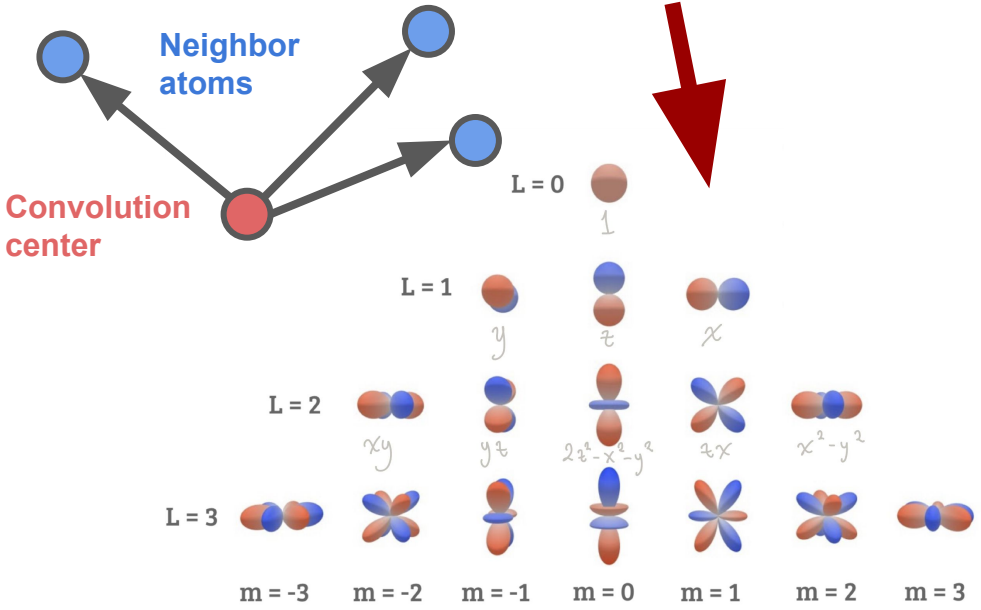


Spherical harmonics transform in the same manner as the irreducible representations of SO(3).

Euclidean Neural Networks are similar to convolutional neural networks...

Equivariant convolutional filters are based on learned radial functions and spherical harmonics...

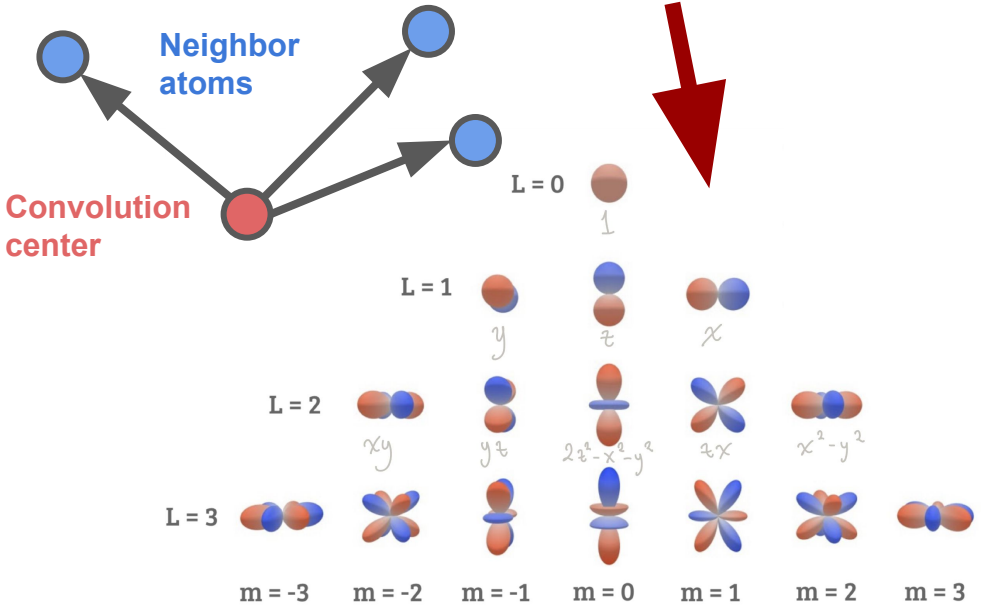
$$W(\vec{r}) = R(r)Y_l^m(\hat{r})$$



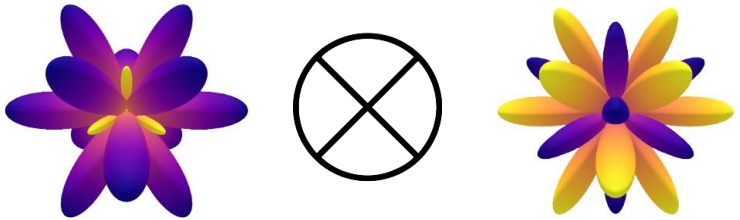
Euclidean Neural Networks are similar to convolutional neural networks...

Equivariant convolutional filters are based on learned radial functions and spherical harmonics...

$$W(\vec{r}) = R(r) Y_l^m(\hat{r})$$



...and geometric tensor algebra allow us to generalize scalar operations to more complex geometric tensors.



e.g. How to multiply two vectors?

$$\vec{a} \cdot \vec{b} = c$$ scalar

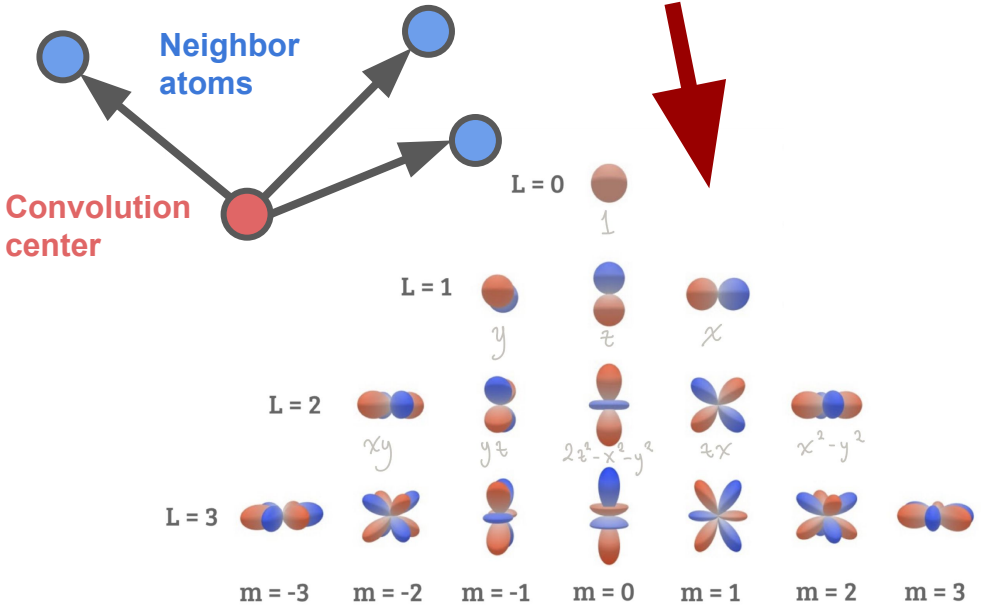
$$\vec{a} \times \vec{b} = \vec{c}$$ vector

$$\vec{a} \otimes \vec{b} = C$$ 3x3 matrix

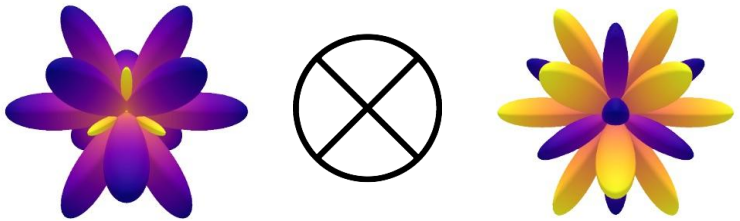
Euclidean Neural Networks are similar to convolutional neural networks...

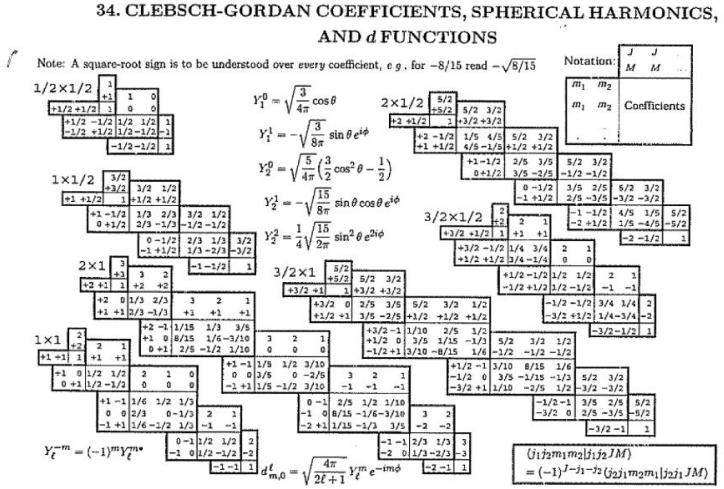
Equivariant convolutional filters are based on learned radial functions and spherical harmonics...

$$W(\vec{r}) = R(r)Y_l^m(\hat{r})$$



...and geometric tensor algebra allow us to generalize scalar operations to more complex geometric tensors.

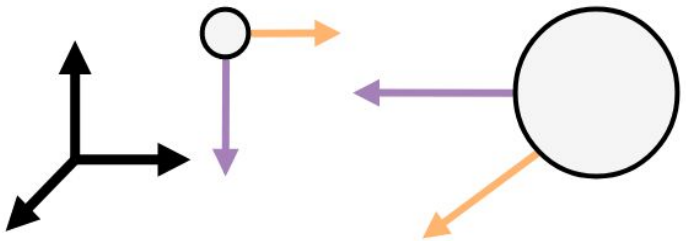



34. CLEBSCH-GORDAN COEFFICIENTS, SPHERICAL HARMONICS, AND d FUNCTIONS

Note: A square-root sign is to be understood over every coefficient, e.g., for $-8/15$ read $-\sqrt{8/15}$.

Notation: $\begin{matrix} J & J \\ M & M \end{matrix}$ Coefficients $\begin{matrix} m_1 & m_2 \\ m_1 & m_2 \end{matrix}$

The input to our network is geometry and features on that geometry.

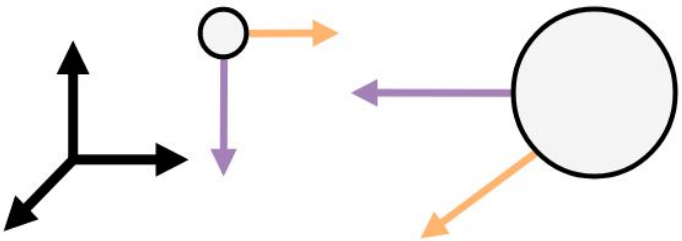


```
geometry = [[x0, y0, z0],[x1, y1, z1]]
features = [
    [m0, v0y, v0z, v0x, a0y, a0z, a0x]
    [m1, v1y, v1z, v1x, a1y, a1z, a1x]
]
...
```

The input to our network is geometry and features on that geometry.

We categorize our features by how they transform under rotation and parity

as *irreducible representations of O(3).*



```
geometry = [[x0, y0, z0],[x1, y1, z1]]
features = [
    [m0, v0y, v0z, v0x, a0y, a0z, a0x]
    [m1, v1y, v1z, v1x, a1y, a1z, a1x]
]
Rs = [(1, 0, 1), (1, 1, -1), (1, 1, -1)]
# OR
Rs = [(1, 0, 1), (2, 1, -1)]
```

"Representation List"
Notation

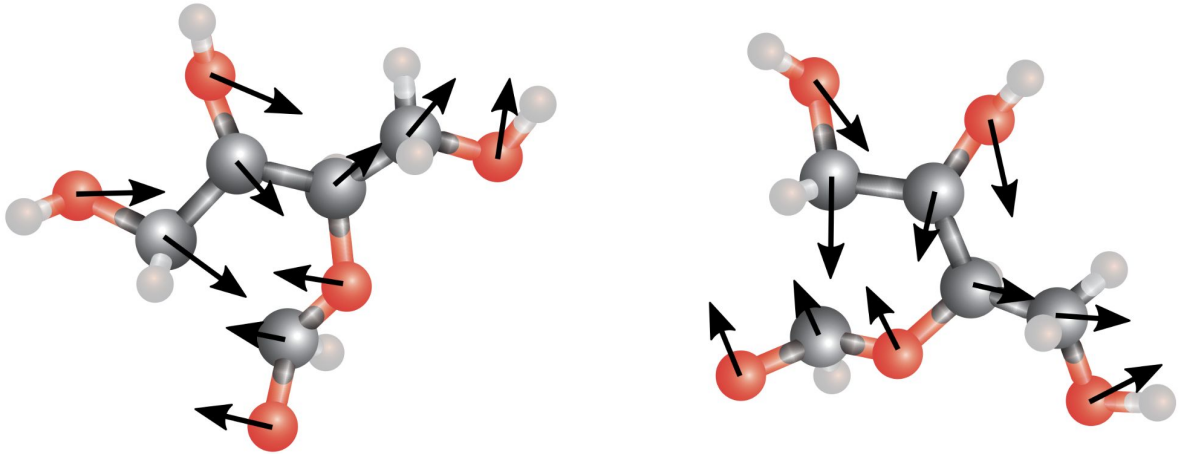
`Rs = [(copies, L, parity),...]`

1 Scalar *(L=0)*
(even parity)

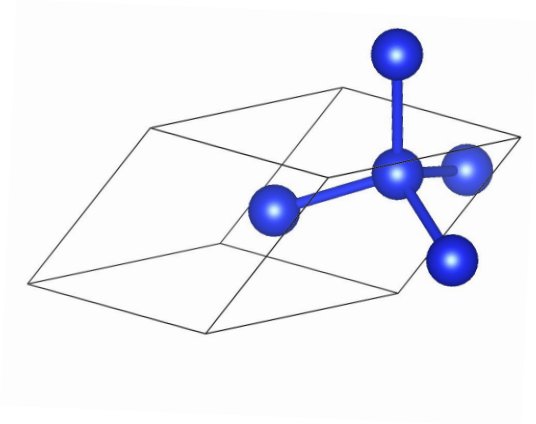
2 Vectors *(L=1)*
(odd parity)

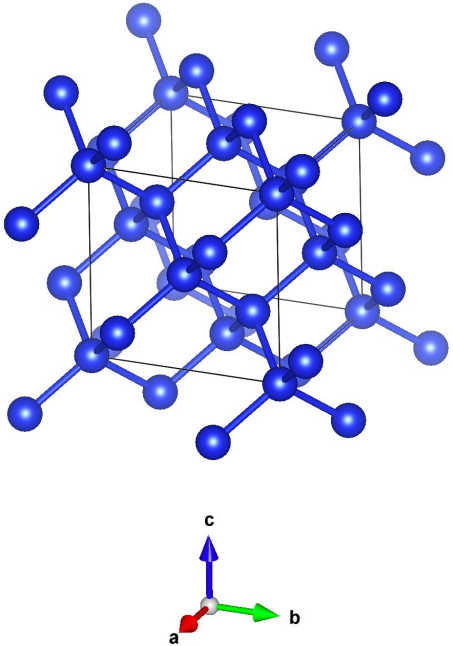
What does equivariance get you?

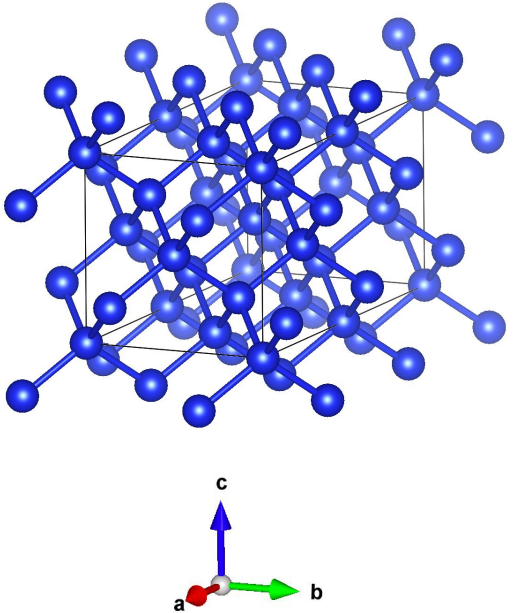
Given a molecule and a rotated copy,
predicted forces are the same up to rotation.
(Predicted forces are equivariant to rotation.)
Additionally, networks generalize to molecules with similar motifs.



Primitive unit cells, conventional unit cells, and supercells of the same crystal produce the same output (assuming periodic boundary conditions).

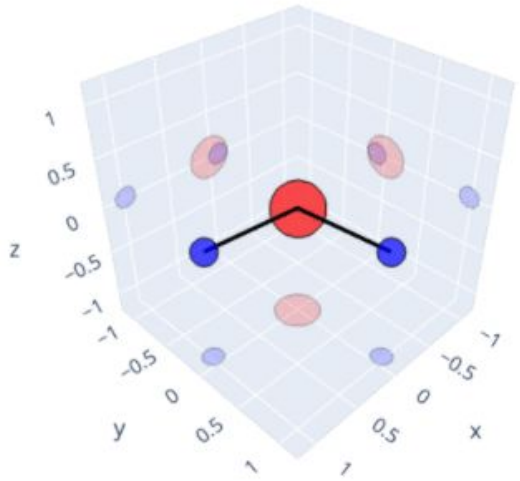




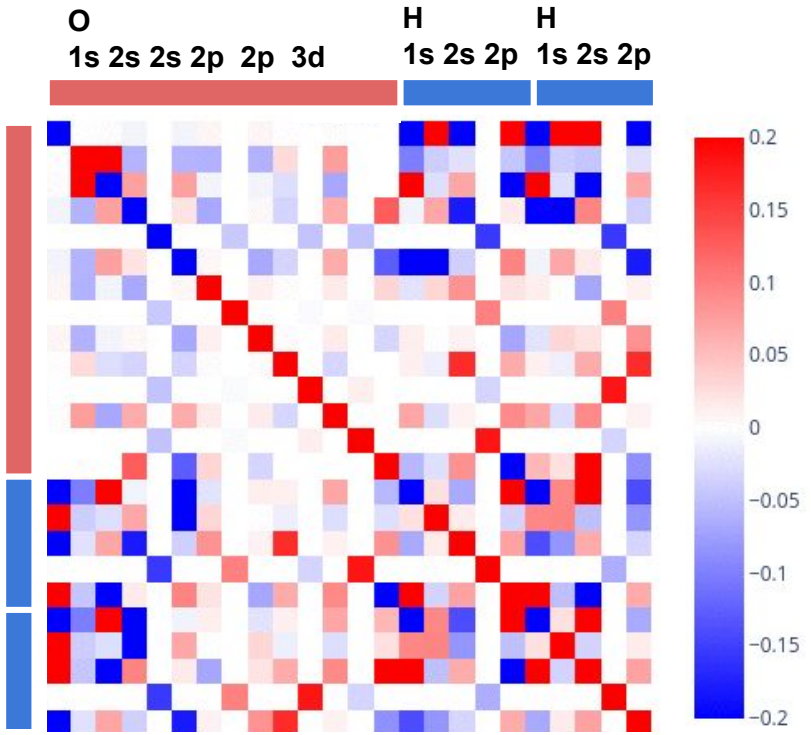


E(3)NNs can express tensors of atomic orbitals and predict molecular Hamiltonians in any orientation from seeing a single example.

$R\vec{v}$

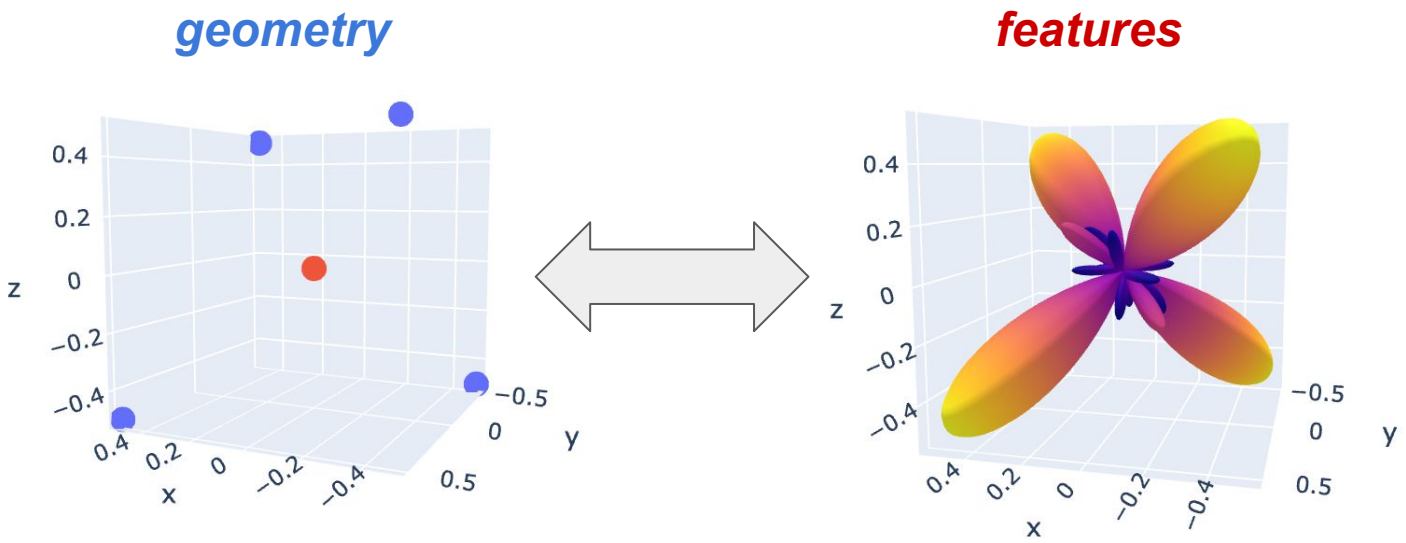


RHR^T



E(3)NNs can manipulate geometry,
which means they can be used for generative models such as autoencoders.

We can convert local *geometry* into *features* and *vice versa*
via spherical harmonic projections.



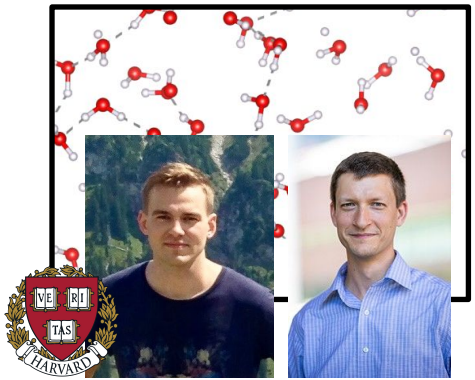
E(3)NNs are extremely data efficient.

E(3)NNs are extremely data efficient.

E(3)NNs for molecular dynamics (*coming soon*)
Water

System	TFMD, 133 data points	DeepMD, 133,500 data points
Liquid Water	**29.7** (meV/Å)	40.4 (meV/Å)
Ice Ih (b)	**26.7**	43.3
Ice Ih (c)	**16.1**	26.8
Ice Ih (d)	**13.3**	25.4

Data from: Zhang, L. et al. E. (2018). *PRL*, *120*(14), 143001.



Simon Batzner

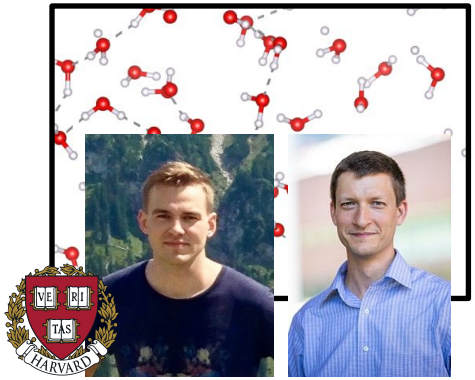
Boris Kozinsky

E(3)NNs are extremely data efficient.

E(3)NNs for molecular dynamics (*coming soon*)
Water

System	TFMD, 133 data points	DeepMD, 133,500 data points
Liquid Water	**29.7** (meV/Å)	40.4 (meV/Å)
Ice Ih (b)	**26.7**	43.3
Ice Ih (c)	**16.1**	26.8
Ice Ih (d)	**13.3**	25.4

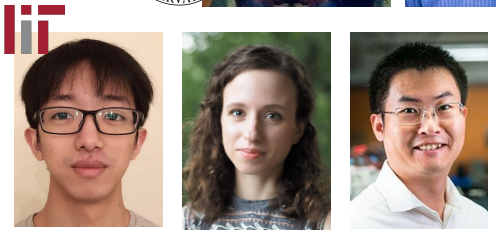
Data from: Zhang, L. et al. E. (2018). *PRL*, *120*(14), 143001.



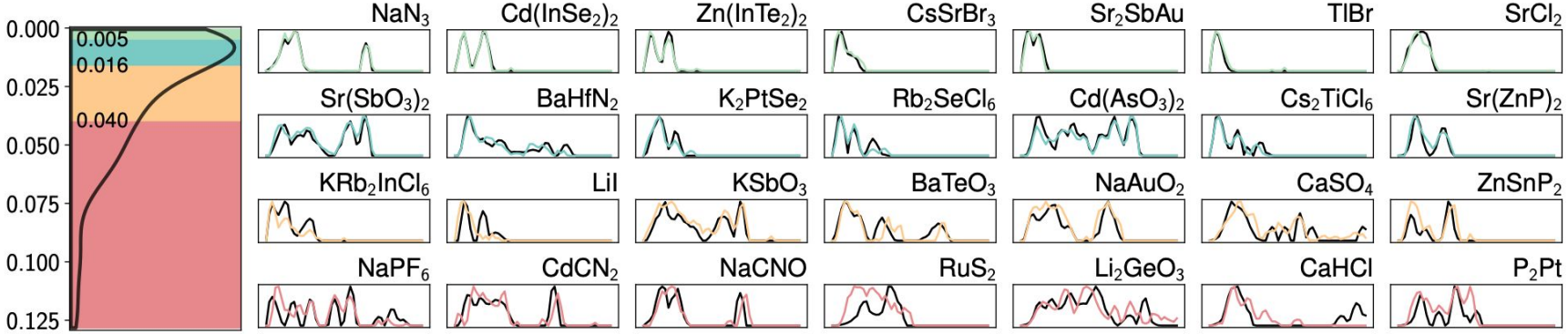
Simon Batzner

Boris Kozinsky

E(3)NNs for phonon density of states (arxiv:2009.05163)
Training set of 1,200 crystal structures with 64 atom types.
Test set includes atom types never seen. Used to find high C_v.
Data from Materials Project

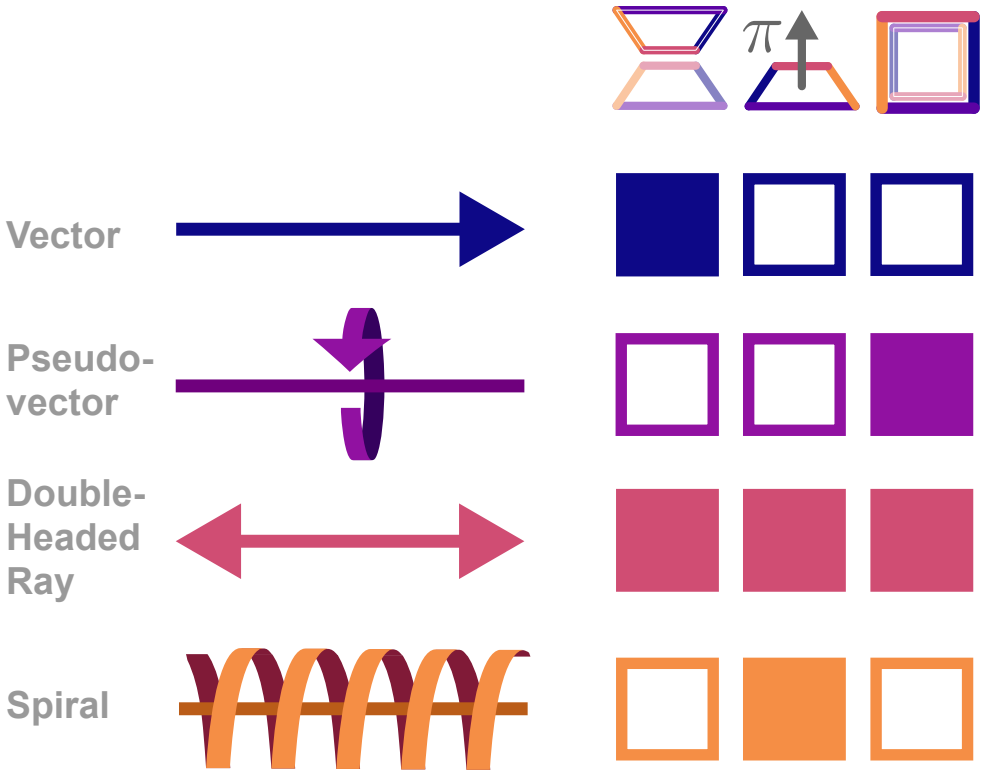


Zhantao Chen
Nina Andrejevic
Mingda Li



Features that are consequences of fully treating Euclidean symmetry...

Feature 1: All data (input, intermediates, output) in E(3)NNs are geometric tensors.
Geometric tensors are the "data types" of 3D space and have many forms.

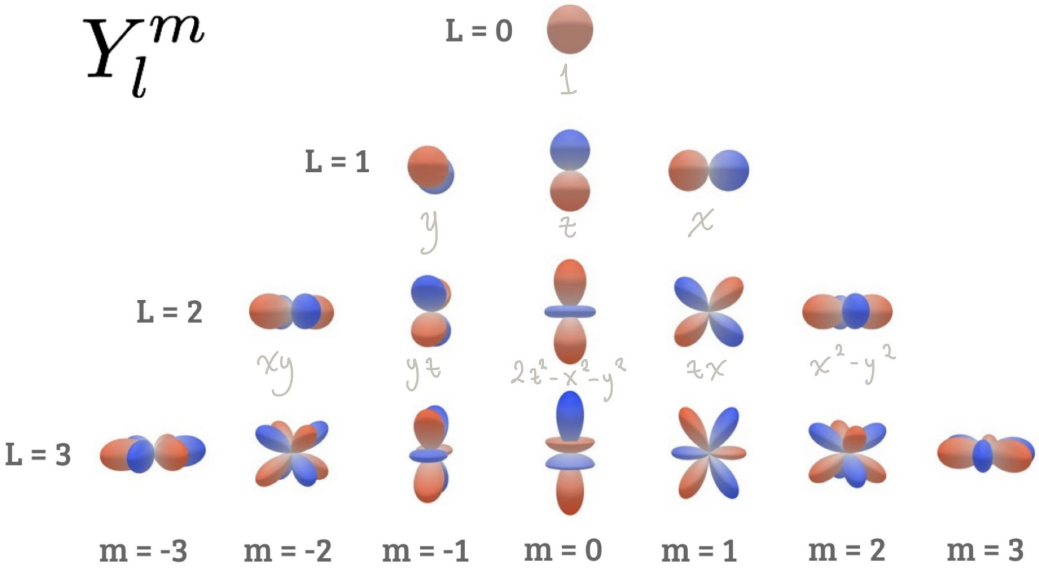



```
Rs_vector = [(1, 1, -1)]

Rs_pseudovector = [(1, 1, 1)]

Rs_doubleray = [(1, 2, 1)]

Rs_spiral = [(1, 2, -1)]
```

Feature 1: All data (input, intermediates, output) in E(3)NNs are geometric tensors.
Geometric tensors are the "data types" of 3D space and have many forms.

Spherical harmonics

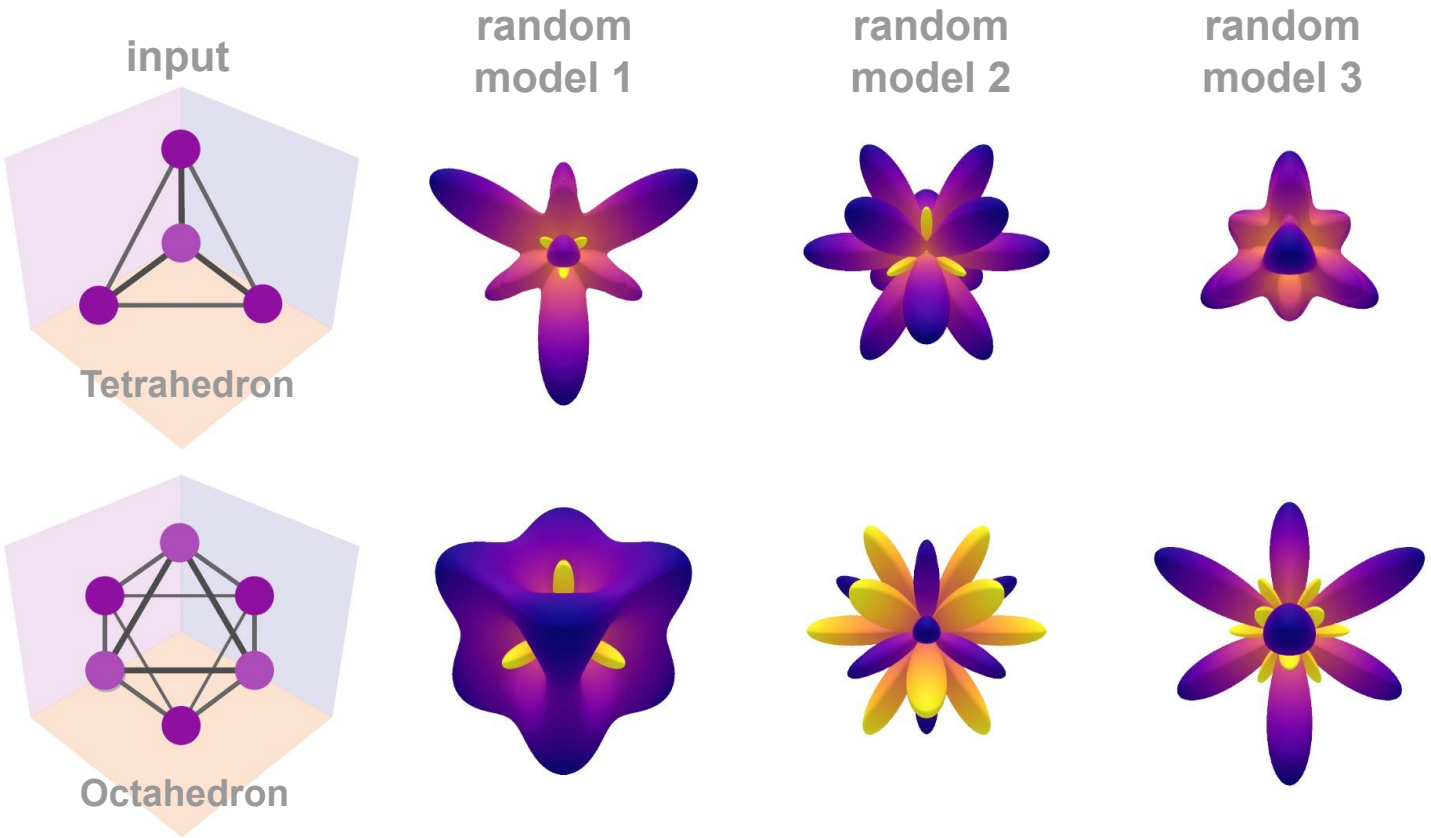
Y_l^m




```
Rs_s_orbital = [(1, 0,  1)]

Rs_p_orbital = [(1, 1, -1)]

Rs_d_orbital = [(1, 2,  1)]

Rs_f_orbital = [(1, 3, -1)]
```

Feature 2: The outputs have equal or higher symmetry than the inputs.

Curie's principle (1894): "When effects show certain asymmetry, this asymmetry must be found in the causes that gave rise to them."



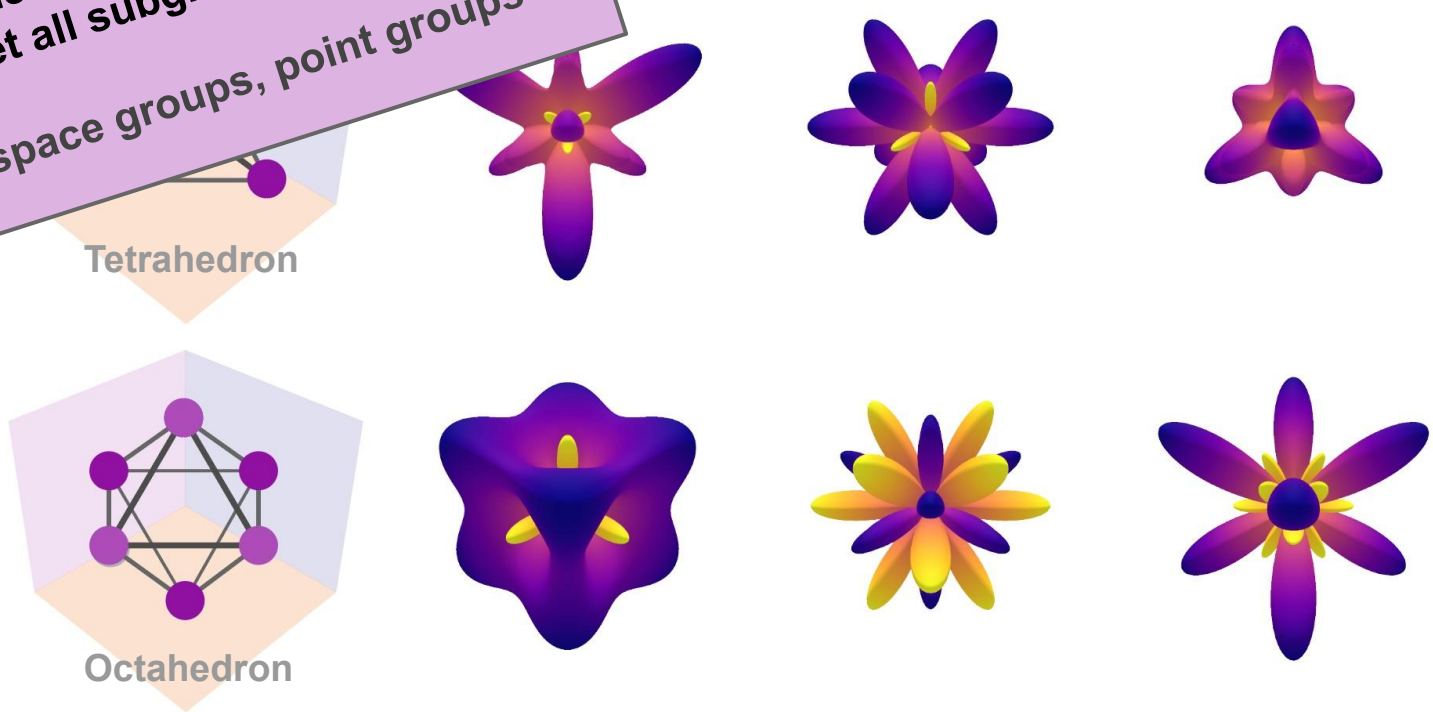
T. E. Smidt, M. Geiger, B. K. Miller. https://arxiv.org/abs/2007.02005 (2020)

Feature 2: The outputs have equal or higher symmetry than the inputs.

Curie's principle (1894): "Wh… s show certain asymmetry, this asymmetry must be found …s that gave rise to them."

Implement group equivariance and get all subgroups for FREE!

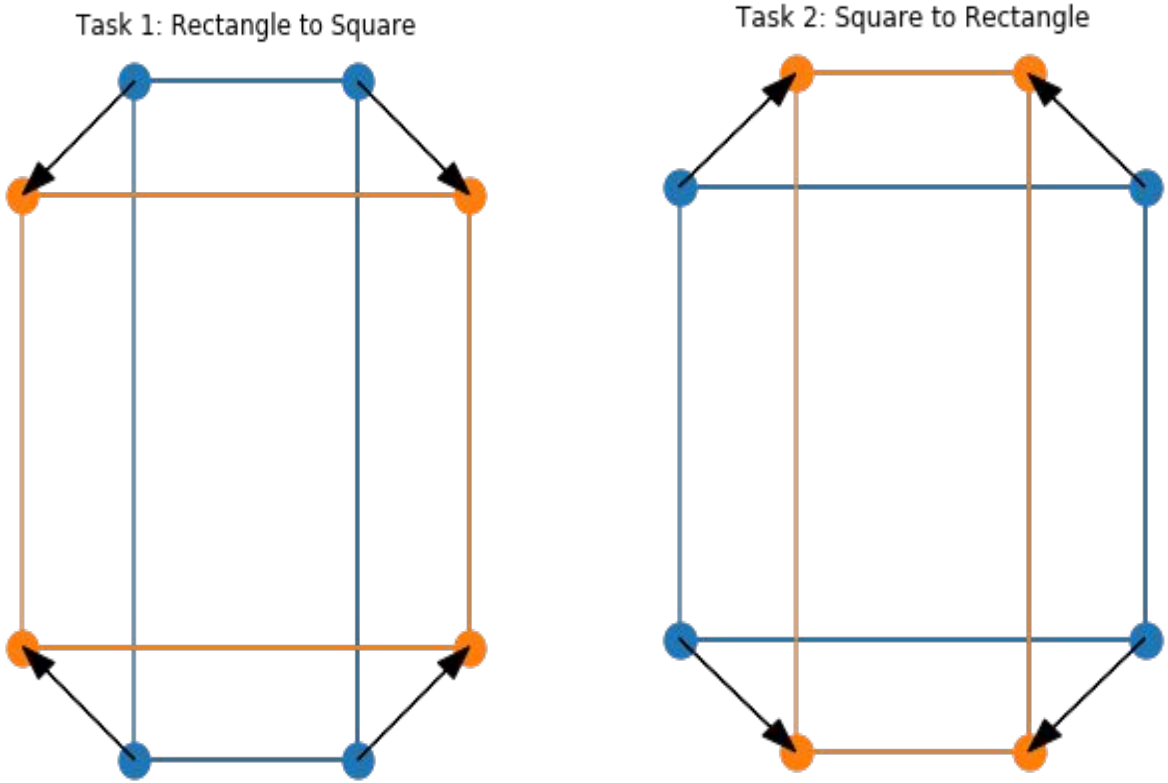
e.g. space groups, point groups



T. E. Smidt, M. Geiger, B. K. Miller. https://arxiv.org/abs/2007.02005 (2020)

Feature 2: The outputs have equal or higher symmetry than the inputs.

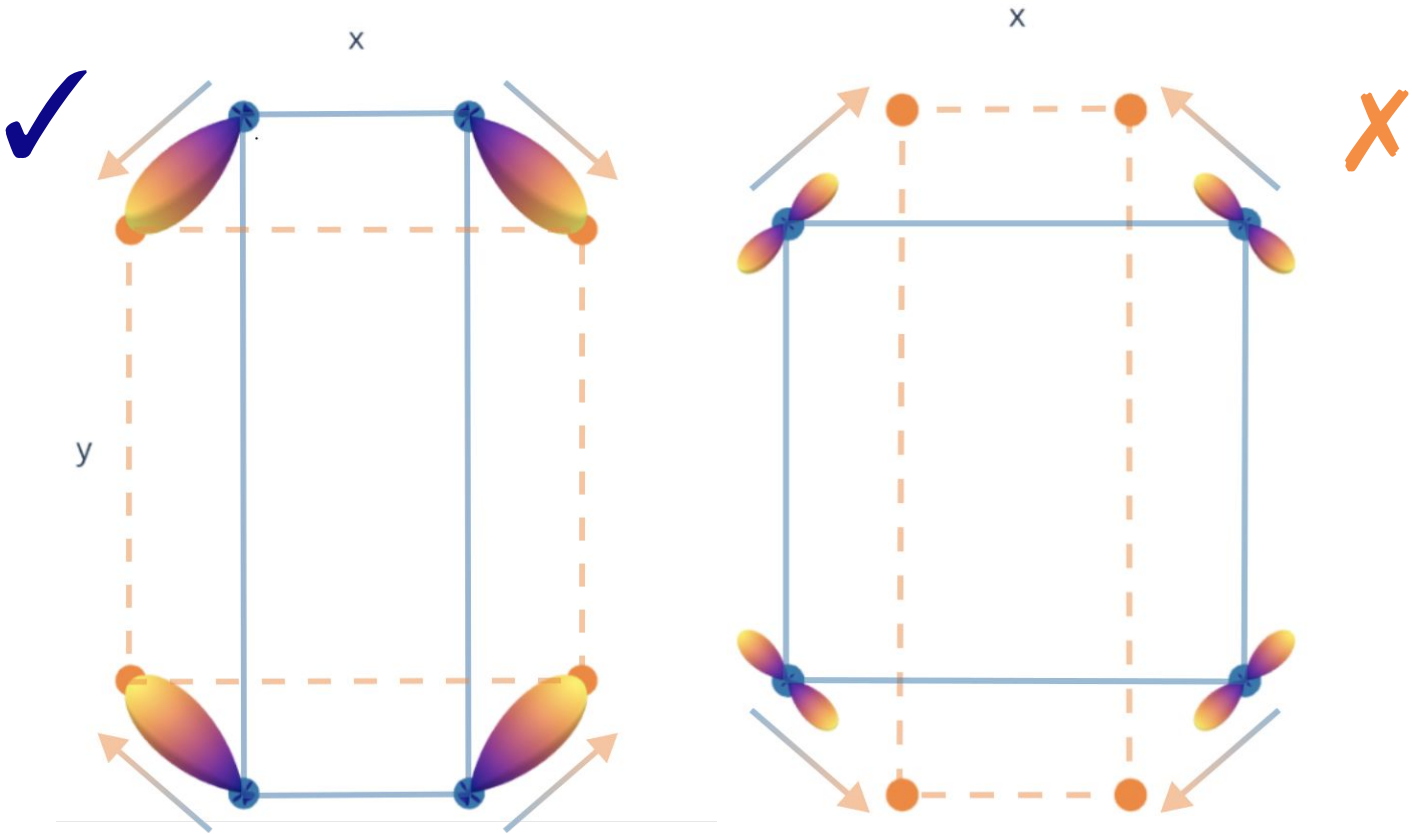
Symmetry compiler -- can't fit a model that does symmetrically make sense



T. E. Smidt, M. Geiger, B. K. Miller. https://arxiv.org/abs/2007.02005 (2020)

<u>Feature 2</u>: The outputs have equal or higher symmetry than the inputs.

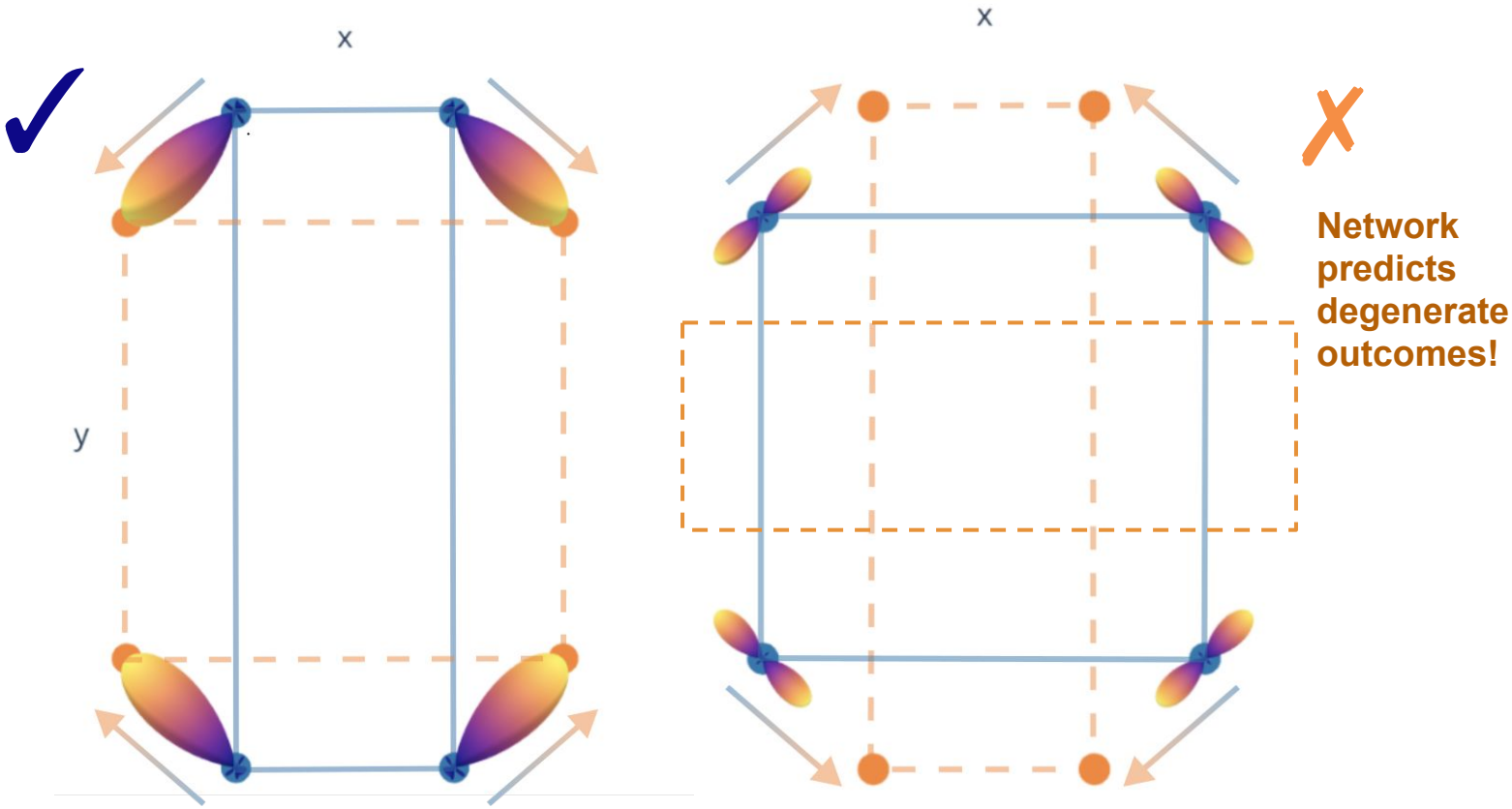
Symmetry compiler -- can't fit a model that does symmetrically make sense



T. E. Smidt, M. Geiger, B. K. Miller. https://arxiv.org/abs/2007.02005 (2020)

Feature 2: **The outputs have equal or higher symmetry than the inputs.**

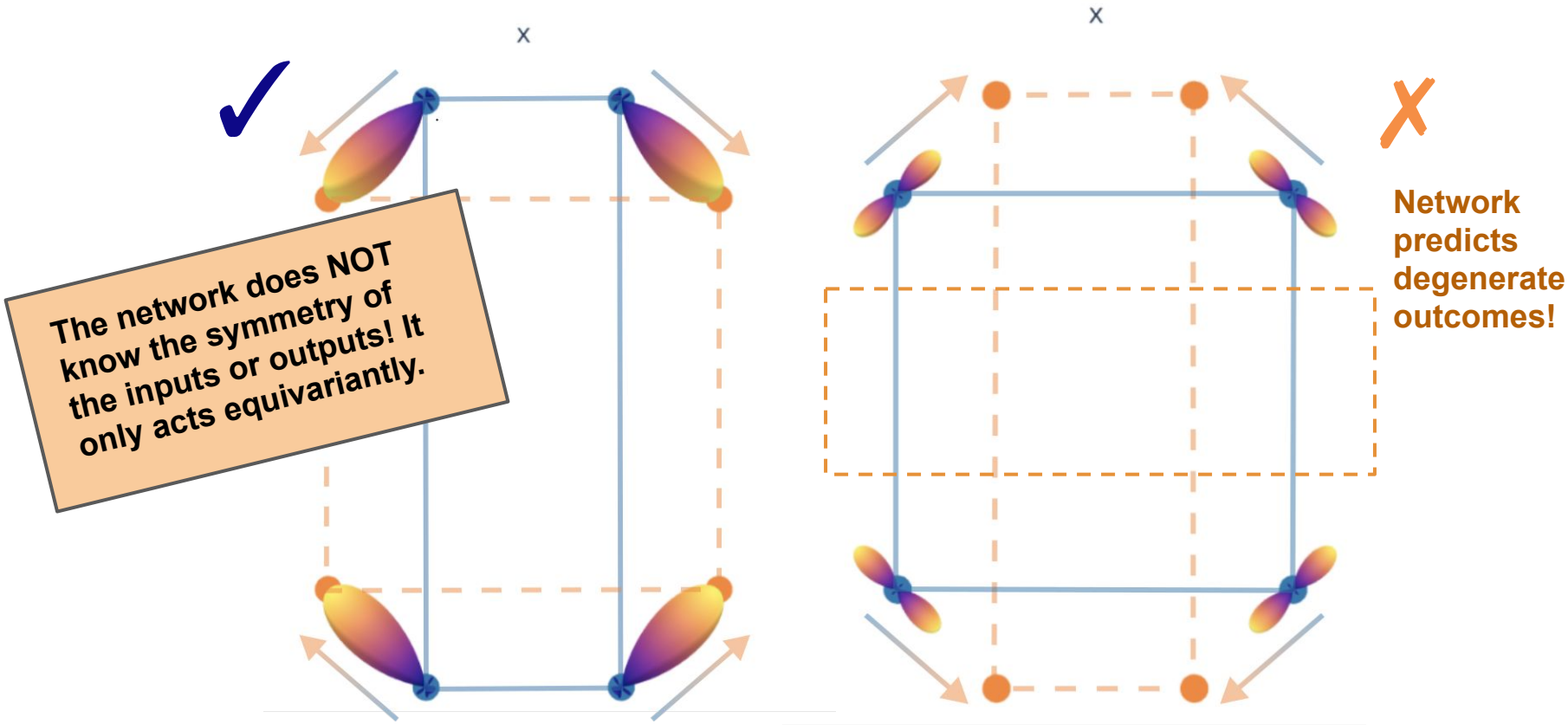
Symmetry compiler -- can't fit a model that does symmetrically make sense



T. E. Smidt, M. Geiger, B. K. Miller. https://arxiv.org/abs/2007.02005 (2020)

Feature 2: **The outputs have equal or higher symmetry than the inputs.**

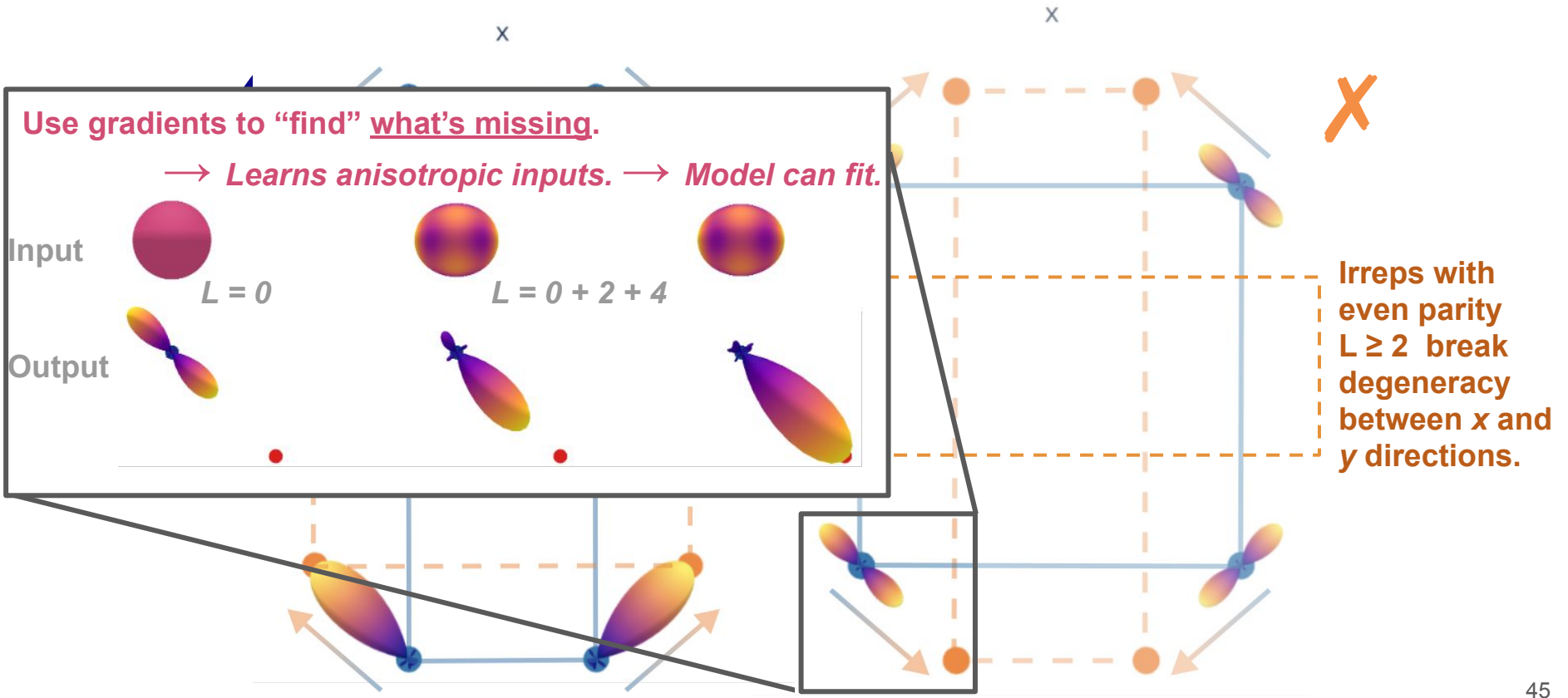
Symmetry compiler -- can't fit a model that does symmetrically make sense



T. E. Smidt, M. Geiger, B. K. Miller. https://arxiv.org/abs/2007.02005 (2020)

Feature 3: **We can find data that is implied by symmetry.**

Using gradients of loss wrt input we can find symmetry breaking "order parameters"



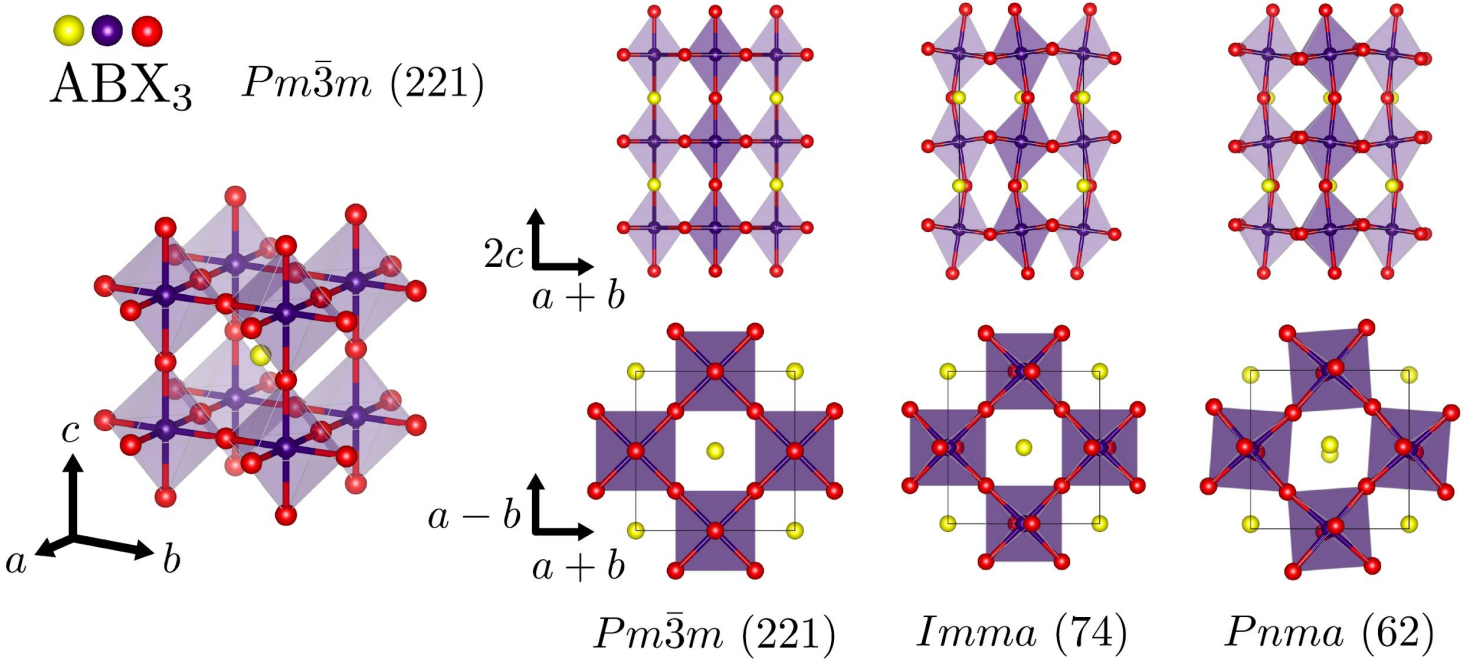
T. E. Smidt, M. Geiger, B. K. Miller. https://arxiv.org/abs/2007.02005 (2020)

Feature 3: We can find data that is implied by symmetry.

Using gradients of loss wrt input we can find symmetry breaking "order parameters"

Octahedral tilting in perovskites ($M^{3+} \oplus R^{4+}$) ⇨

Network learns equal magnitude pseudovector order parameters on B site with proper spatial patterning.



T. E. Smidt, M. Geiger, B. K. Miller. https://arxiv.org/abs/2007.02005 (2020)

e3nn: a modular PyTorch framework for Euclidean neural networks

https://github.com/e3nn/e3nn

Utilities and classes for

- building E(3) equivariant neural networks
- manipulating geometric tensors
- visualizing spherical harmonics

developers of e3nn

Mario Geiger
(EPFL)

Ben Miller
(U of Amsterdam,
formerly FU Berlin)

Kostiantyn
Lapchevskyi

Tess Smidt
(LBNL)

e3nn: a modular PyTorch framework for Euclidean neural networks

https://github.com/e3nn/e3nn

Creating a basic convolution E(3) neural network

```
import torch
from e3nn import rs
from e3nn.networks import GatedConvParityNetwork
torch.set_default_dtype(torch.float64)


N_atom_types = 3  # For example H, C, O
Rs_in = [(N_atom_types, 0, 1)]  # Input are scalars
Rs_out = [(1, 1, -1)]  # Predict vectors

model_kwargs = {
    'Rs_in': Rs_in, 'Rs_out': Rs_out, 'mul': 4, 'lmax': 2,
    'layers': 3, 'max_radius': r_max, 'number_of_basis': 10,
}

model = GatedConvParityNetwork(**model_kwargs)
```

e3nn: a modular PyTorch framework for Euclidean neural networks

https://github.com/e3nn/e3nn

Convert between Cartesian tensors (with symmetric indices) and Irrep tensors and calculate degrees of freedom (e.g. elasticity tensor)

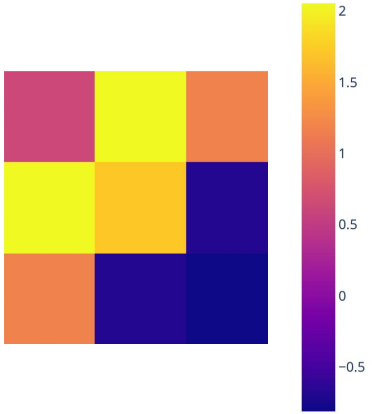
```
import torch
from e3nn import rs
from e3nn.tensor import CartesianTensor
torch.set_default_dtype(torch.float64)


rank4 = torch.zeros(3, 3, 3, 3)  # Placeholder
Rs, Q = CartesianTensor(rank4, 'ijkl=jikl=klij').to_irrep_transformation()
print("Representations: ", Rs)
print("Degrees of freedom: ", rs.dim(Rs))

>> Representations: [(2, 0, 1), (2, 2, 1), (1, 4, 1)]
>> Degrees of freedom: 21
```

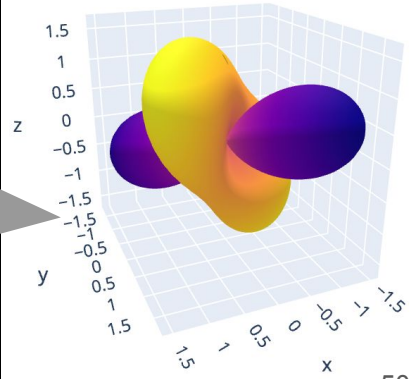
e3nn: a modular PyTorch framework for Euclidean neural networks

https://github.com/e3nn/e3nn

Plot 3x3 matrix as linear combination of spherical harmonics.

```
...

# Symmetric Matrix
M = torch.randn(3,3)
M = M + M.transpose(0, 1)

# Plot matrix
px.imshow(M)

matrix = CartesianTensor(M, formula='ij=ji').to_irrep_tensor()
r, f = SphericalTensor.from_irrep_tensor(matrix).plot()

# Plot SH signal
surface_plot = lambda r, f: go.Surface(
    x=r[..., 0], y=r[..., 1], z=r[..., 2],
    surfacecolor=f, showscale=False)
go.Figure([surface_plot(r, f)])
```





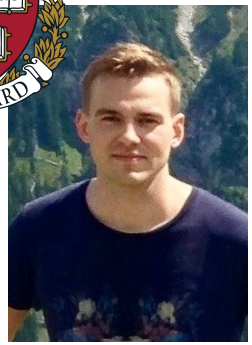
collaborators of e3nn

Sandia National Laboratories

Thomas Hardin

Josh Rackers

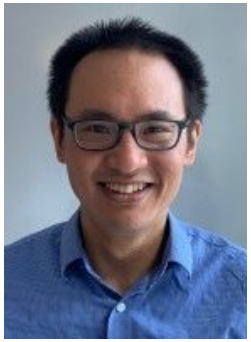
HARVARD



Simon Batzner

Boris Kozinsky

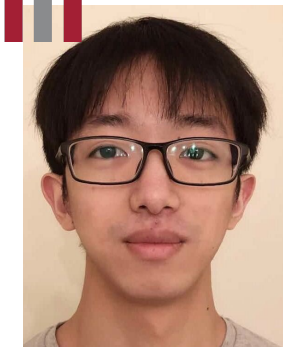
MERCK



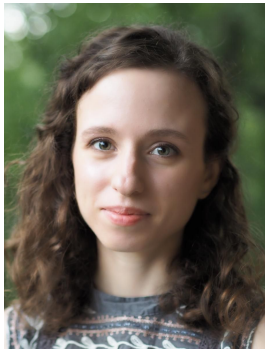
Eugene Kwan

Frank Noé

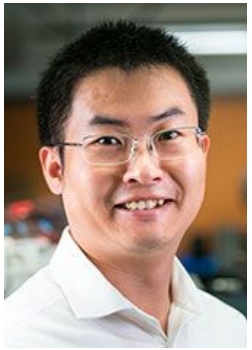
Claire West



Zhantao Chen



Nina Andrejevic



Mingda Li

A Quick Recap!

3D Euclidean symmetry:

- rotations, translation, inversion
- Different coordinate systems
  - ⇨ same physical system

Euclidean Neural Networks are equivariant to E(3)

- Convolutional filters
  - ⇨ learned radial functions and spherical harmonics
- Geometric tensor algebra
- Equivariant nonlinearities (did not discuss)

Equivariance can have unintended features.

1) Symmetry specific data types
2) Output symmetry equal to inputs
   - Implement group equivariance and get all subgroups for FREE!
   - Symmetry compilers
3) Grad loss wrt input can break symmetry

Feel free to reach out if you have any questions!

Tess Smidt
tsmidt@lbl.gov

A Quick Recap!

3D Euclidean symmetry:
- rotations, translation, inversion
- Different coordinate systems
  - ⇨ same physical system

Euclidean Neural Networks are equivariant to E(3)
- Convolutional filters
  - ⇨ learned radial functions and spherical harmonics
- Geometric tensor algebra
- Equivariant nonlinearities (did not discuss)

Equivariance can have unintended features.
1) Symmetry specific data types
2) Output symmetry equal to inputs
   - Implement group equivariance and get all subgroups for FREE!
   - Symmetry compilers
3) Grad loss wrt input can break symmetry

Feel free to reach out if you have any questions!

Tess Smidt
tsmidt@lbl.gov

Resources on Euclidean neural networks:
e3nn.org

`e3nn` Code (PyTorch):
http://github.com/e3nn/e3nn
"quick" tutorial:
https://tinyurl.com/e3nn-quick-tutorial-202011
`e3nn_tutorial`:
http://blondegeek.github.io/e3nn_tutorial/
Papers:

Tensor Field Networks (arXiv:1802.08219)
Clebsch-Gordon Nets (arXiv:1806.09231)
3D Steerable CNNs (arXiv:1807.02547)
Cormorant (arXiv:1906.04015)
SE(3)-Transformers (arXiv:2006.10503)
tfnns on proteins (arXiv:2006.09275)
e3nn on QM9 (arXiv:2008.08461)
e3nn for symm breaking (arXiv:2008.08461)
E(3) and equivariance in ML (chemrxiv.12935198.v1)
e3nn for phonon DOS (arxiv:2009.05163)

My past talks (look for video / slide links):
https://blondegeek.github.io/talks

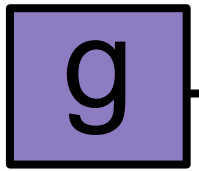
Calling in backup (slides)!

Applications so far...

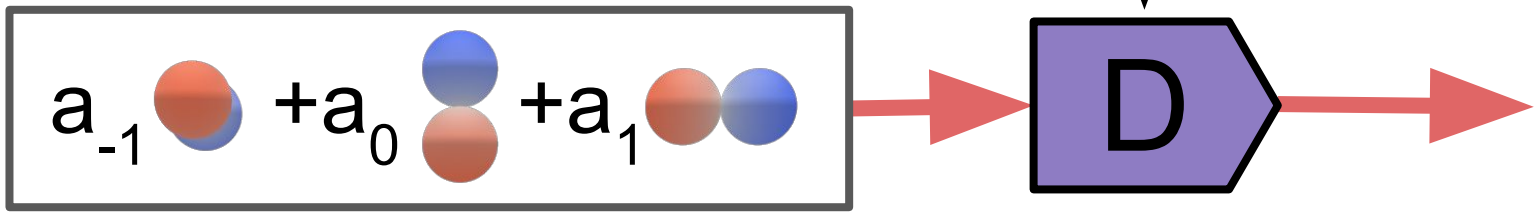
- Finding order parameters of 2nd order structural phase transitions
- Molecular dynamics *(Harvard)*
- Molecule and crystal property prediction *(FU Berlin)*
- Inverting invariant representations of atomic geometries *(Sandia)*
- Autoencoding Geometry
- Predicting molecular Hamiltonians *(TU Berlin)*
- Long range interactions *(FU Berlin, TU Berlin)*
- Electron density prediction for large molecules *(Sandia)*
- Predicting chemical shifts for NMR *(Merck, MIT)*
- Conditional protein design *(UW)*
- Inverse design of optical properties of nanoparticle assemblies *(LBL and UW)*
- Phonon properties of crystal structures *(MIT)*
- Anharmonic elastic properties of crystal structures *(UTEP)*
- ...

Spherical harmonics of a given L transform together under rotation.

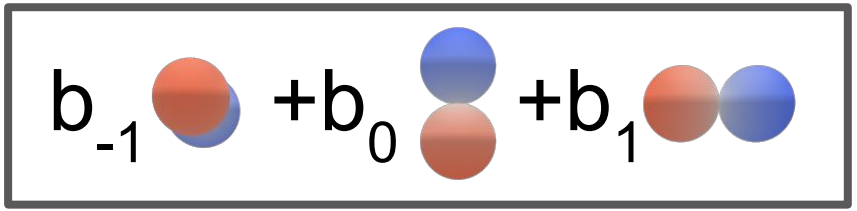
Let **g** be an element of SO(3)



D is the Wigner-D matrix. It has shape $[2l+1, 2l+1]$ and is a function of **g**.



=

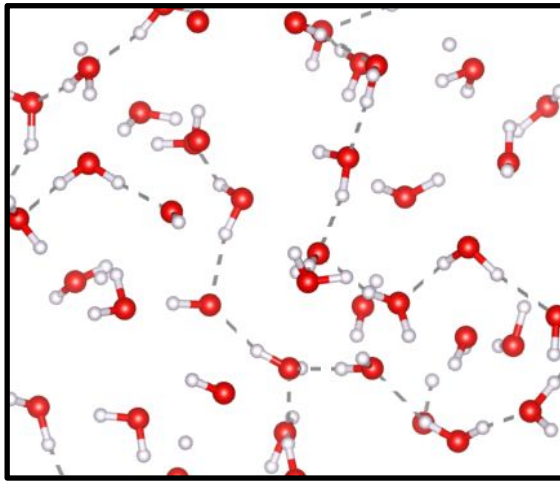


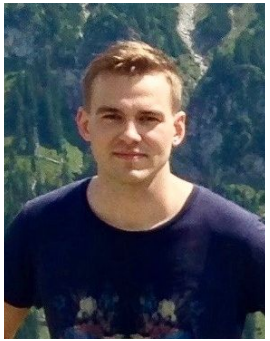
Predict ab initio forces for molecular dynamics

Preliminary results originally presented at APS March Meeting 2019.
Paper in progress.

Testing on liquid water, Euclidean neural networks *(Tensor-Field Molecular Dynamics)* require less data to train than traditional networks to get state of the art results.

	MAE [meV/A]	RMSE [meV/A]
TFMD, 100	27.9	38.20
TFMD, 1000	**11.29**	**14.82**
Deep-MD, 133,500	not reported	40.0





Simon Batzner

Boris Kozinsky

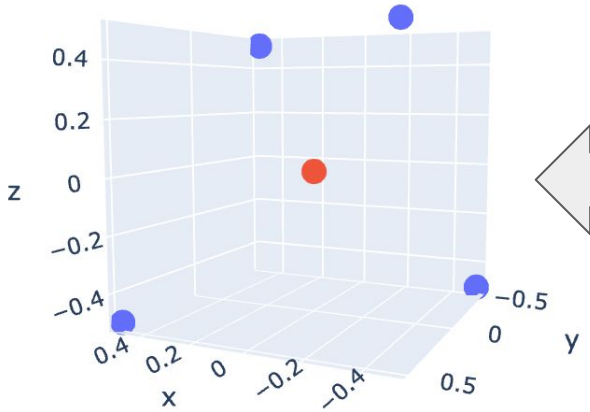
Data set from: [1]
Zhang, L. et al. E. (2018). *PRL*, *120*(14), 143001.

Euclidean neural networks can manipulate geometry,
which means they can be used for generative models such as autoencoders.

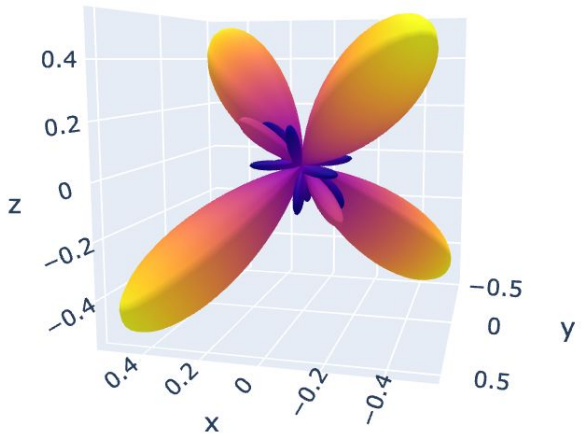
Euclidean neural networks can manipulate geometry,
which means they can be used for generative models such as autoencoders.

To encode/decode, we have to be able to
convert *geometry* into *features* and *vice versa*.
We do this via spherical harmonic projections.

geometry



features

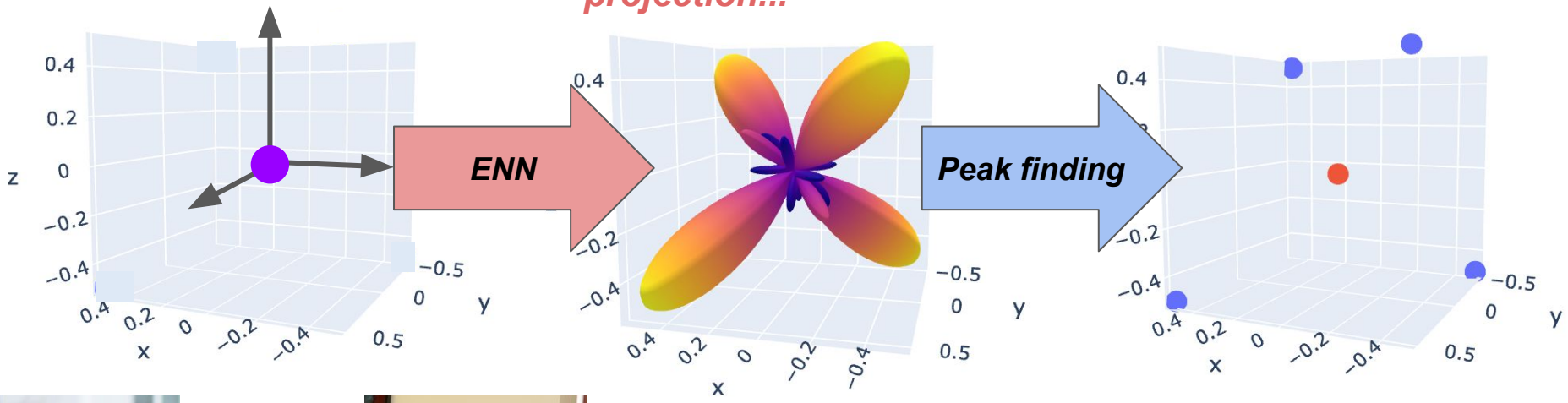


Equivariant neural networks can learn to invert invariant representations.

Invariant features + coordinate frame

Network can predict spherical harmonic projection...

Which can be used to recover geometry.

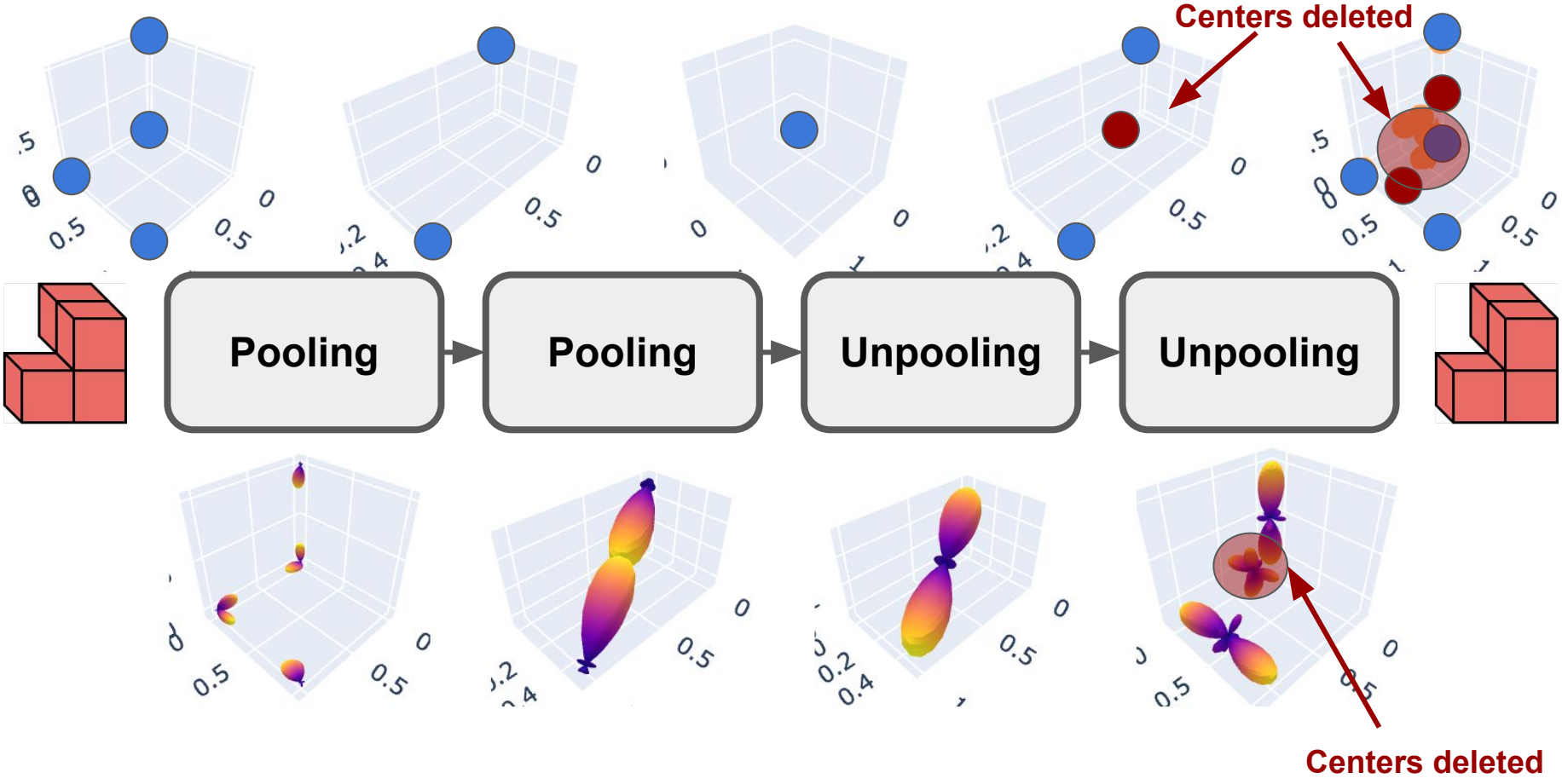


Thomas Hardin

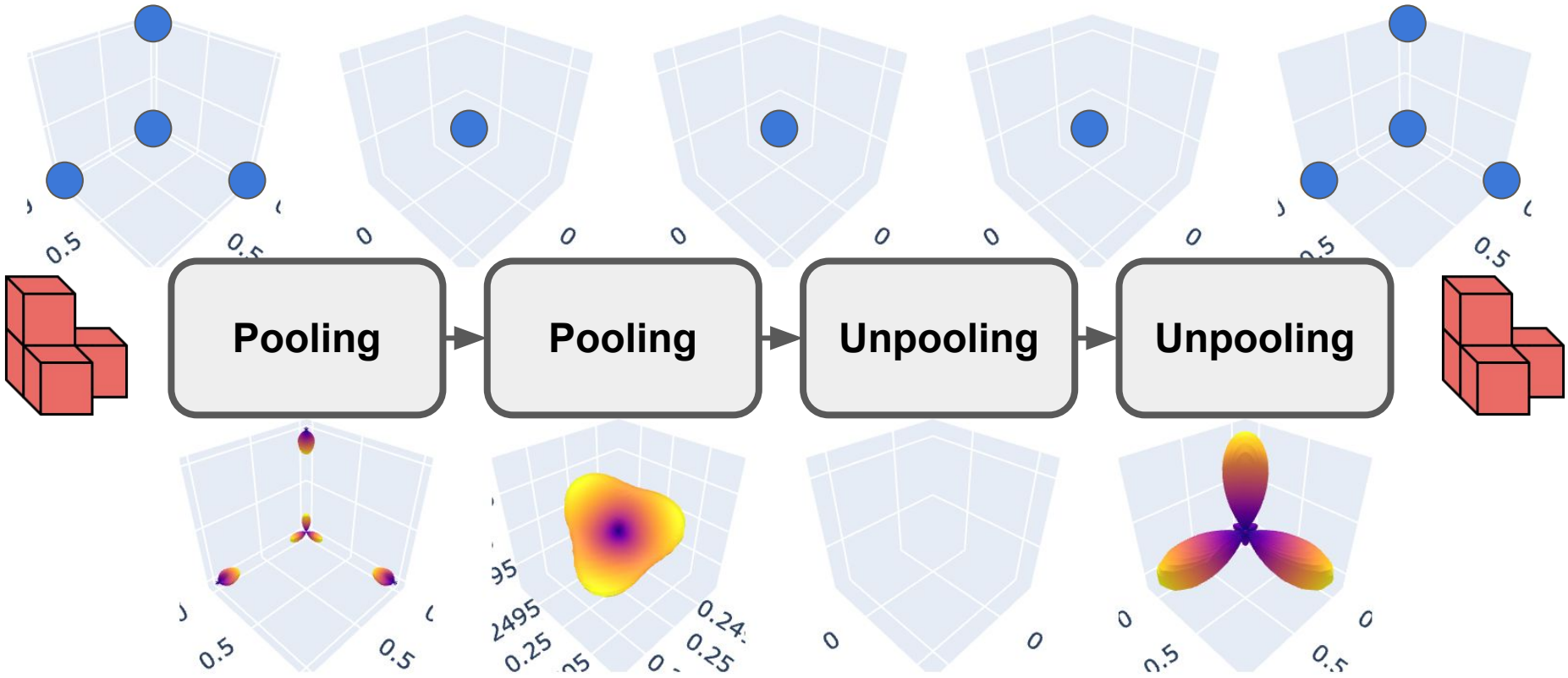
Josh Rackers

Sandia National Laboratories

We can also build an autoencoder for geometry: e.g. Autoencoder on 3D Tetris

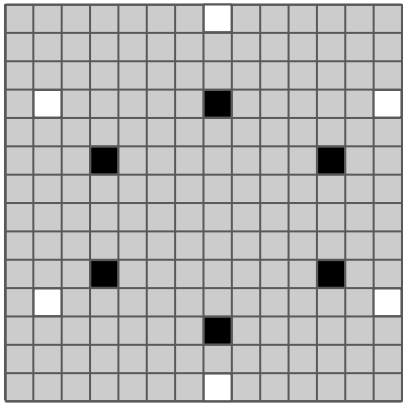


We can also build an autoencoder for geometry: e.g. Autoencoder on 3D Tetris

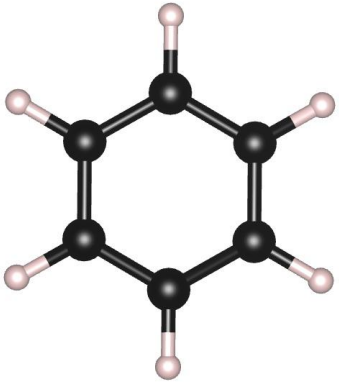


Euclidean Neural Networks are similar to convolutional neural networks...

We use points. Images of atomic systems are sparse and imprecise.



vs.



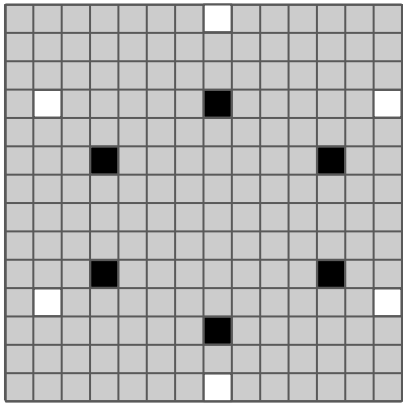
We use continuous convolutions with atoms as convolution centers.

We encode the symmetries of 3D Euclidean space (3D translation- and 3D rotation-equivariance).

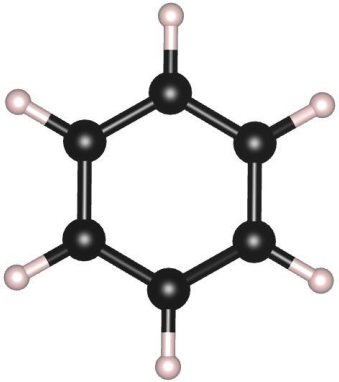
$g \in SE(3)$

Euclidean Neural Networks are similar to convolutional neural networks...

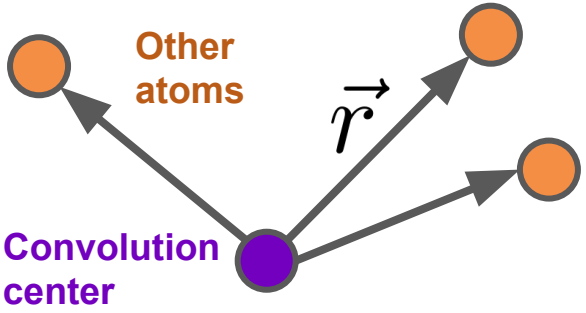
We use points. Images of atomic systems are sparse and imprecise.



vs.



We use continuous convolutions with atoms as convolution centers.

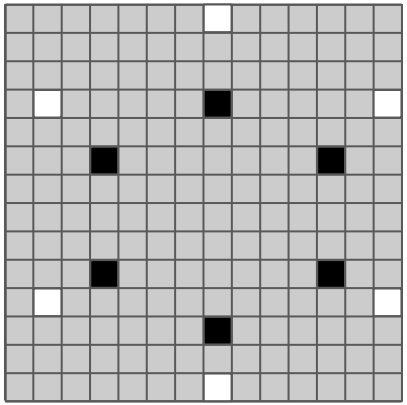


We encode the symmetries of 3D Euclidean space (3D translation- and 3D rotation-equivariance).

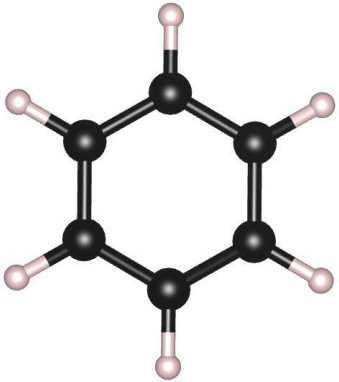
$g \in SE(3)$

Euclidean Neural Networks are similar to convolutional neural networks...

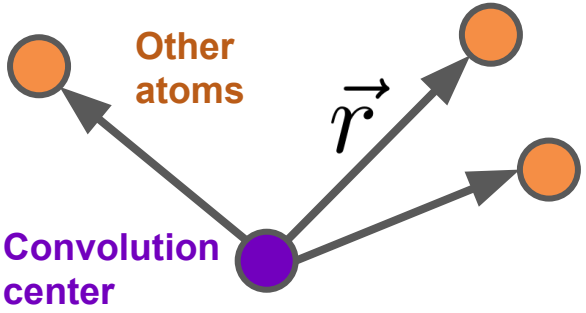
We use points. Images of atomic systems are sparse and imprecise.



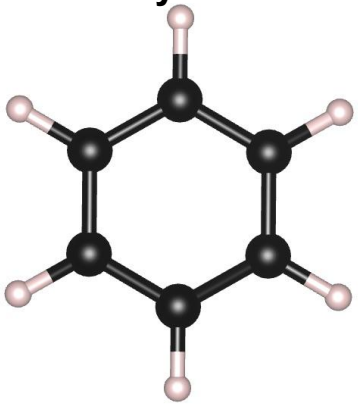
vs.



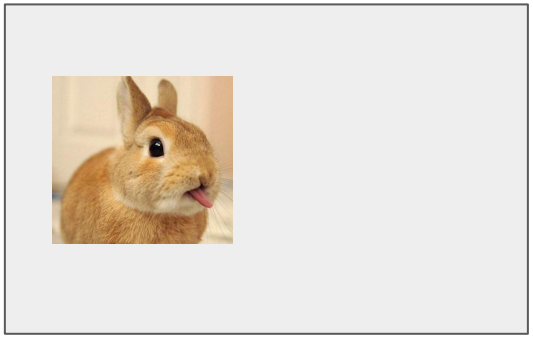
We use continuous convolutions with atoms as convolution centers.

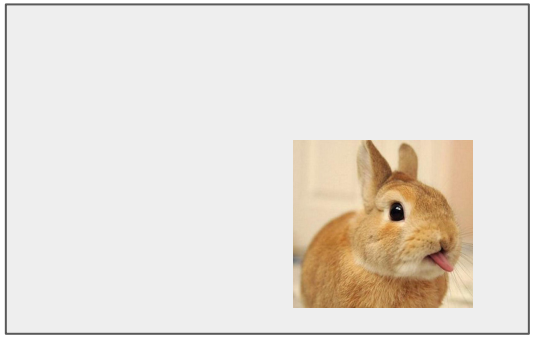


We encode the symmetries of 3D Euclidean space (3D translation- and 3D rotation-equivariance).



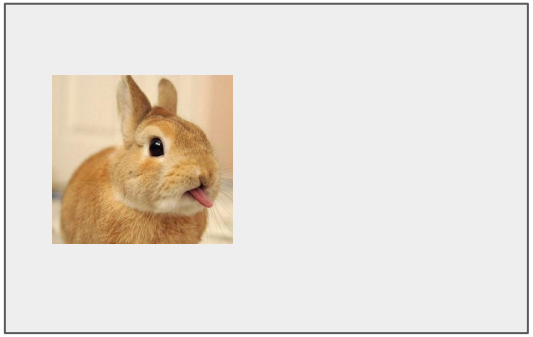
$$g \in SE(3)$$

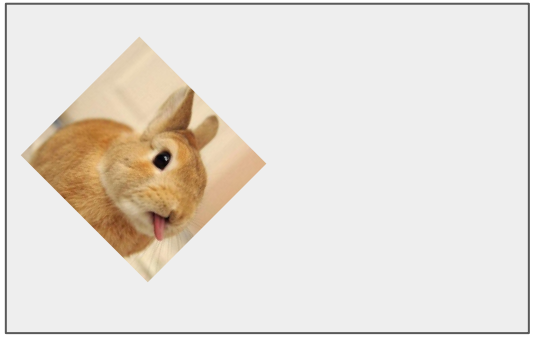




Translation equivariance

Convolutional neural network ✓





Rotation equivariance

~~Data augmentation~~

~~Radial functions (invariant)~~

Want a network that both preserves geometry and exploits symmetry.

Invariant featurizations can be very expressive if well-crafted

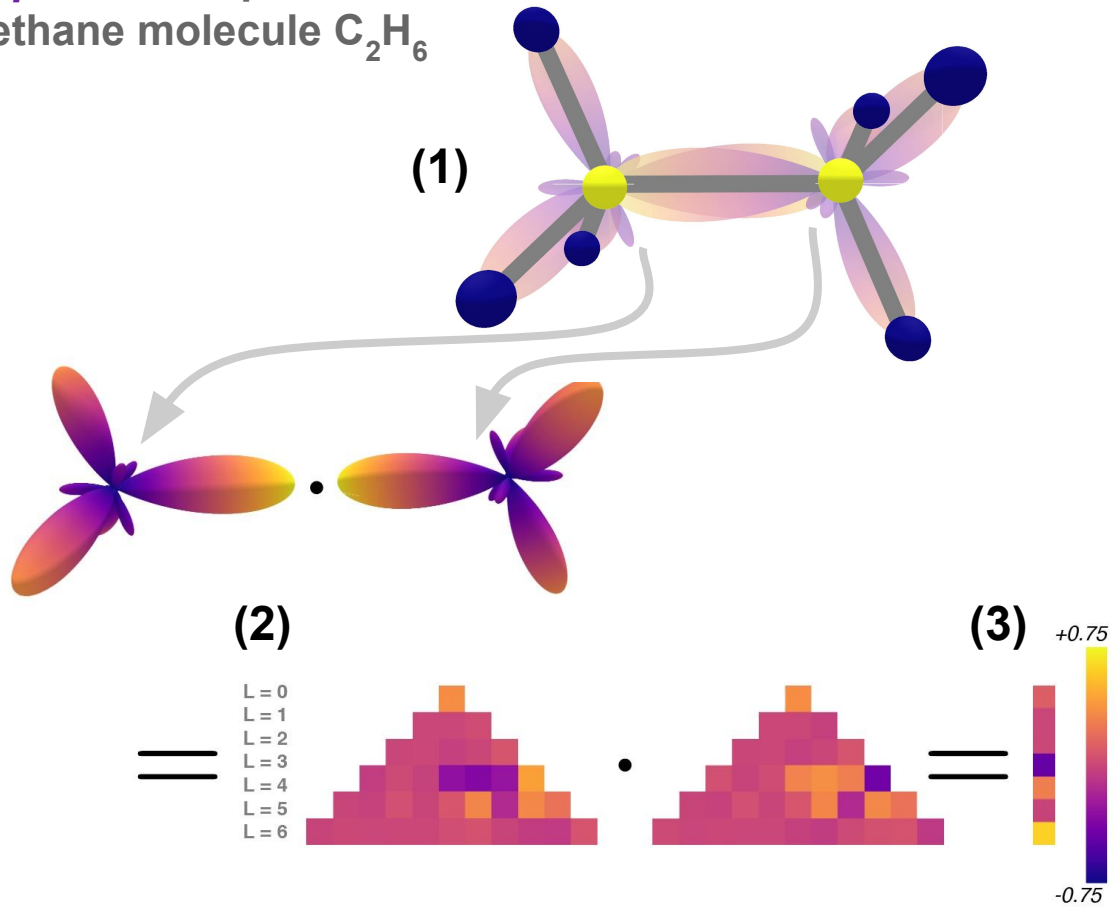
Many *invariant* featurizations use *equivariant* operations

e.g. a (simplified) SOAP kernel for ethane molecule C_2H_6

(1) **Project neighbors of given atom onto spherical harmonics (*equivariant* quantity).**

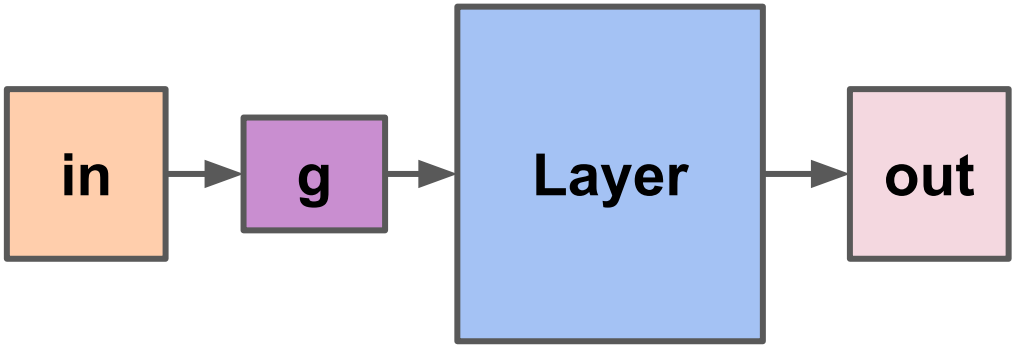
(2) **Interact signals from different atoms via tensor dot product (*equivariant* operation) to produce scalars (*invariant* quantity).**

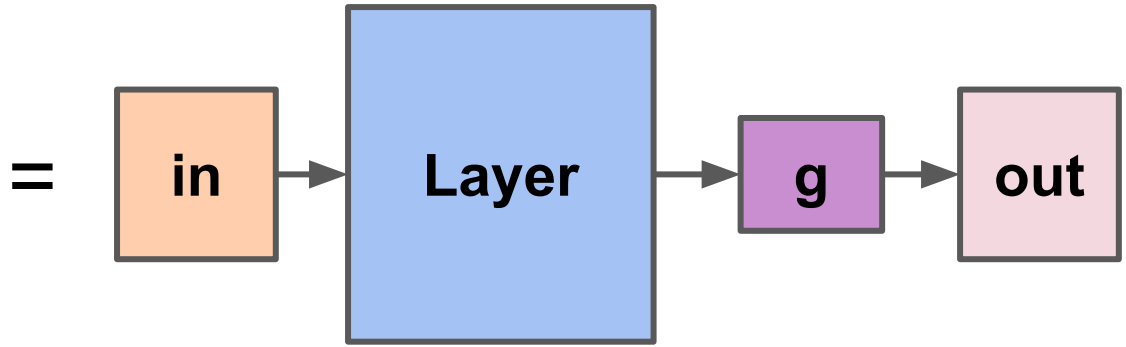
(3) **Give scalars to model.**



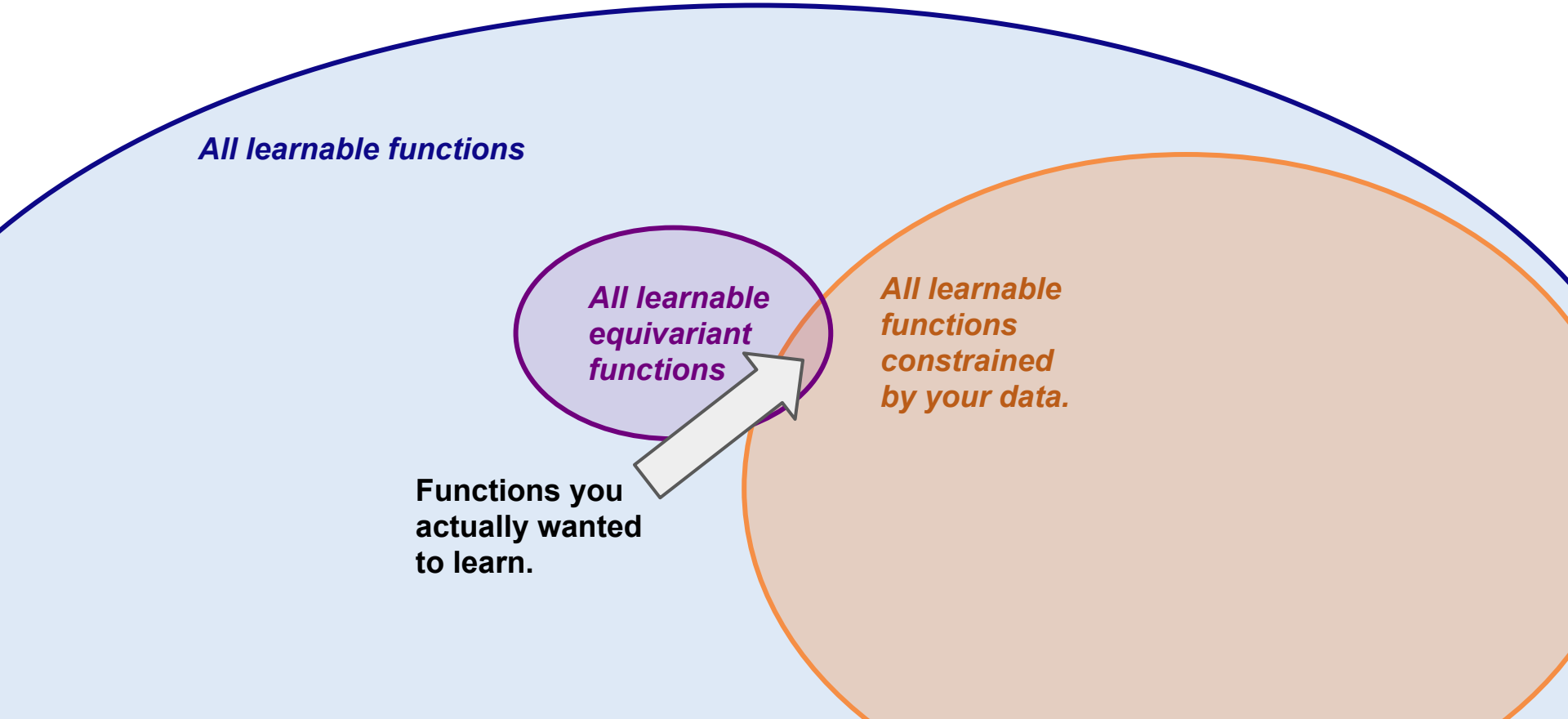
(Favored for kernel methods)

For a function to be *equivariant* means that we can act on our inputs with g OR act our outputs with g and we get the same answer *(for every operation)*.
For a function to be *invariant* means g is the identity (no change).

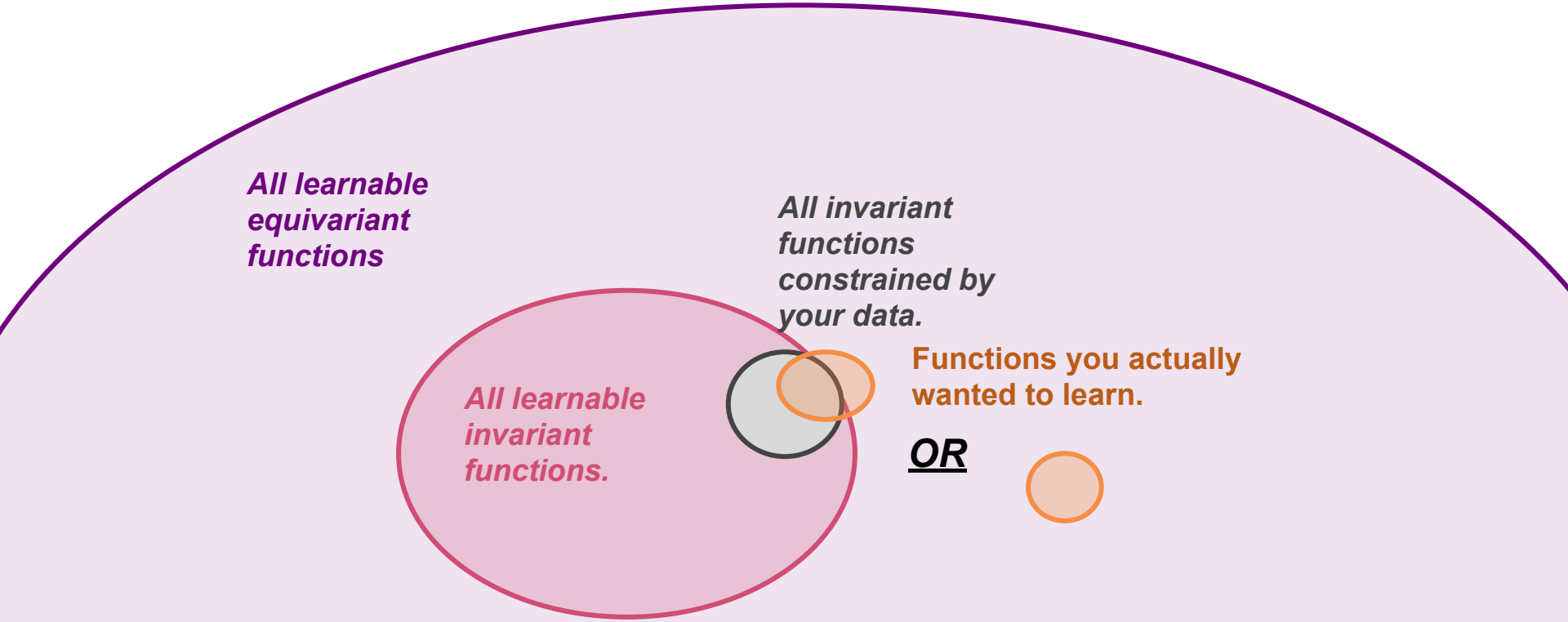




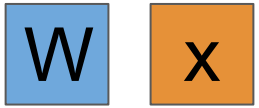
Why limit yourself to equivariant functions?
You can *substantially* shrink the space of functions you need to optimize over.
This means you need less data to constrain your function.



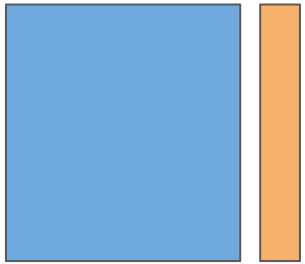
Why not limit yourself to invariant functions?
You have to guarantee that your input features already contain any necessary equivariant interactions *(e.g. cross-products)*.



Neural networks are specially designed for different data types.
Assumptions about the data type are built into how the network operates.

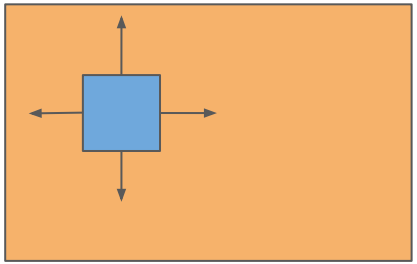


Arrays ⇨ Dense NN



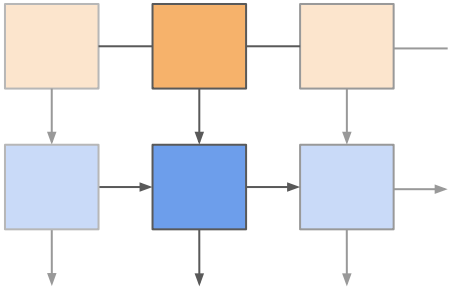
Components are independent.

2D images ⇨ Convolutional NN



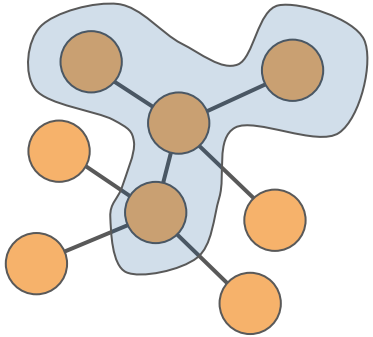
The same features can be found anywhere in an image. Locality.

Text ⇨ Recurrent NN



Sequential data. Next input/output depends on input/output that has come before.

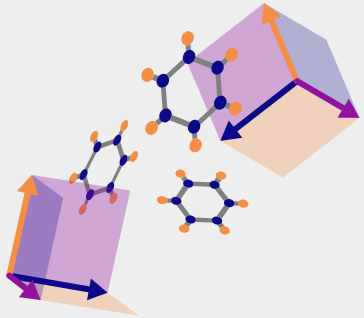
Graph ⇨ Graph (Conv.) NN



Topological data. Nodes have features and network passes messages between nodes connected via edges.

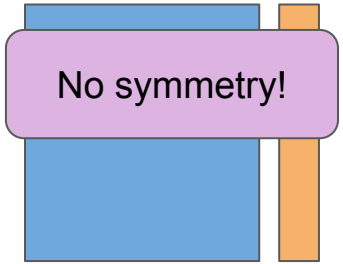
3D physical data ⇨ Euclidean NN

Data in 3D Euclidean space. Freedom to choose coordinate system.



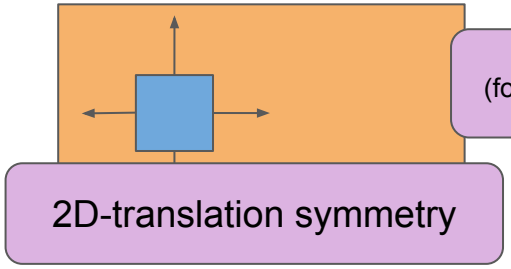
Neural networks are specially designed for different data types.
Assumptions about the data type are built into how the network operates.
Symmetries emerge from these assumptions.

Arrays ⇨ Dense NN



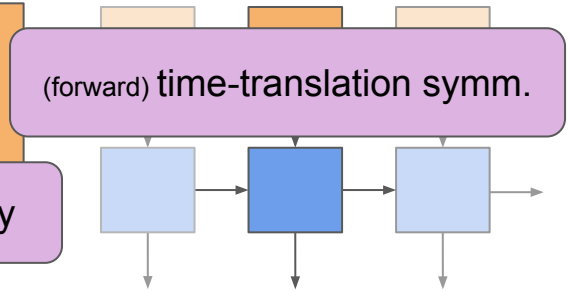
Components are independent.

2D images ⇨ Convolutional NN



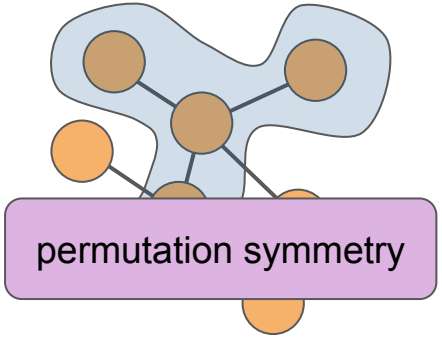
The same features can be found anywhere in an image. Locality.

Text ⇨ Recurrent NN



Sequential data. Next input/output depends on input/output that has come before.

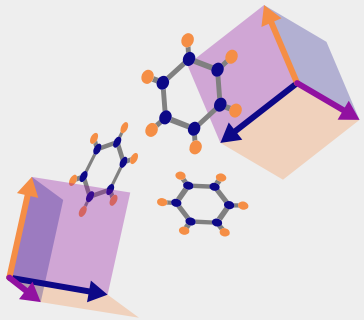
Graph ⇨ Graph (Conv.) NN



Topological data. Nodes have features and network passes messages between nodes connected via edges.

3D physical data ⇨ Euclidean NN

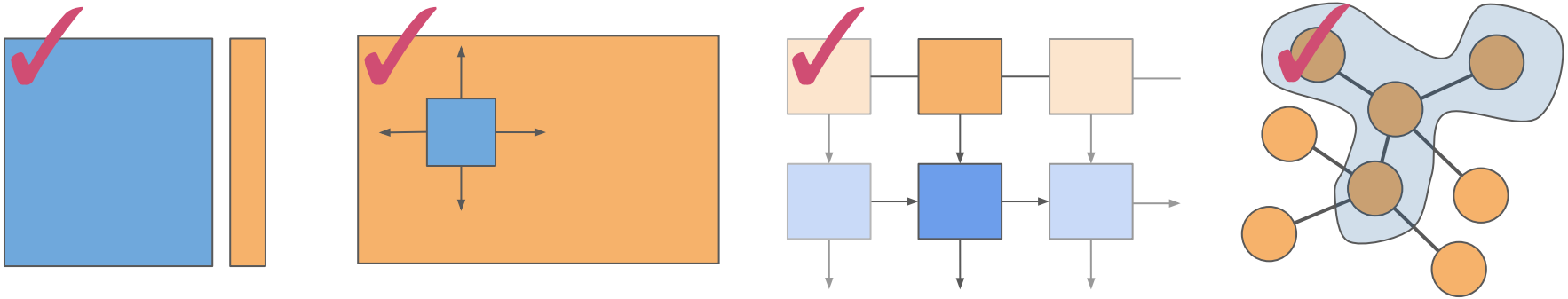
3D Euclidean symmetry **E(3)**: 3D rotations translations and inversion



Neural networks can't mess up invariant representations.

You can use *ANY* neural network with an invariant representation.

Invariant representations can be used for other machine learning algorithms (e.g. kernel methods).



If you can craft a good representation -- great!

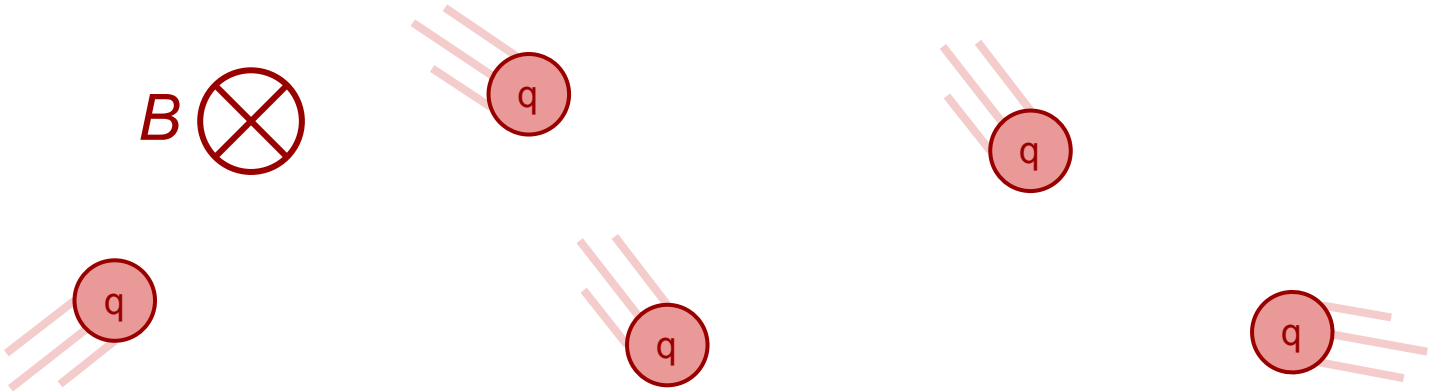
But deep learning's specialty is feature learning.

So, maybe use a different machine learning approach (e.g. kernel methods).

Analogous to... **the laws of** *(non-relativistic)* **physics have Euclidean symmetry,** ***even if systems do not*****.**

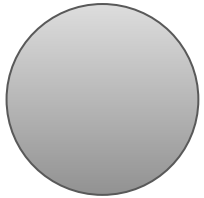
The **network** is our model of **"physics"**. The **input** to the network is our **system**.

$$\vec{F}(q, \vec{r}, \vec{v}, \vec{B}) = \vec{F}_i = \sum_i q_i(\vec{v}_i \times \vec{B}) + \sum_{i \neq j} \frac{q_i q_j}{r_{ij}^2} \hat{r_{ij}}$$



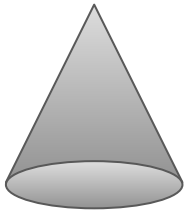
A Euclidean symmetry preserving network produces *outputs* that preserve the subset of symmetries induced by the *input*.

3D rotations and inversions



O(3)

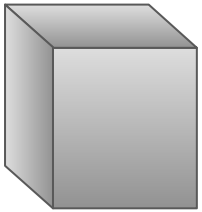
2D rotation and mirrors along cone axis



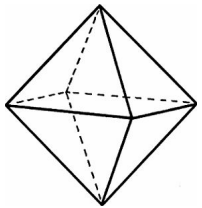
SO(2) + mirrors

($C_{\infty v}$)

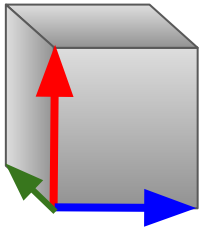
Discrete rotations and mirrors



O_h



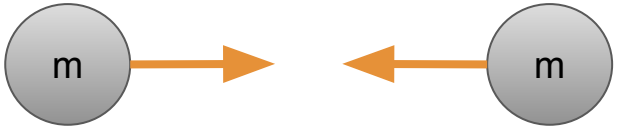
Discrete rotations, mirrors, and translations



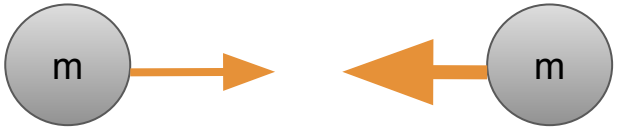
Pm-3m (221)

Properties of a system must be compatible with symmetry.
Which of these situations (inputs / outputs) are symmetrically allowed / forbidden?

a.

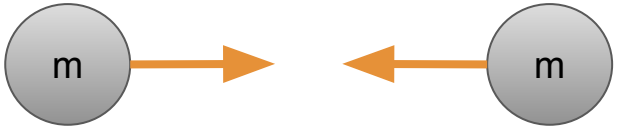


b.

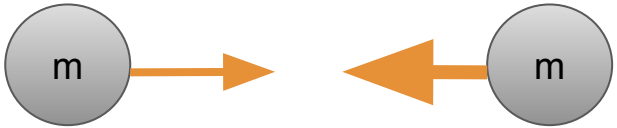


c.

Properties of a system must be compatible with symmetry.
Which of these situations (inputs / outputs) are symmetrically allowed / forbidden?

a. 

✓

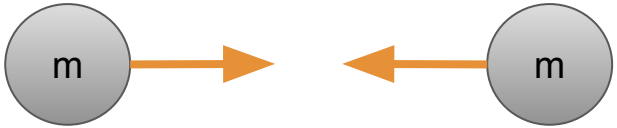
b. 

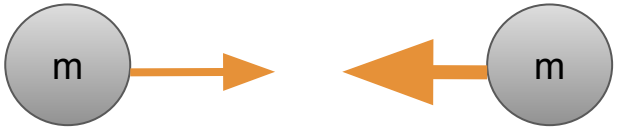
✗

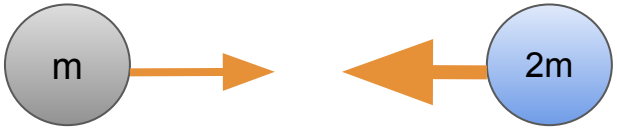
c. 

✗

Properties of a system must be compatible with symmetry.
Which of these situations (inputs / outputs) are symmetrically allowed / forbidden?

a. 

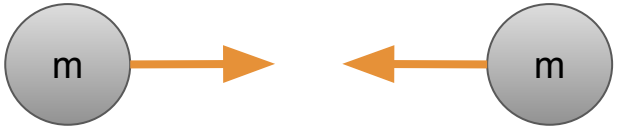
b. 

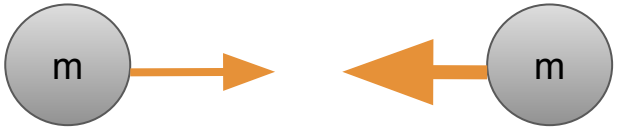
c. 

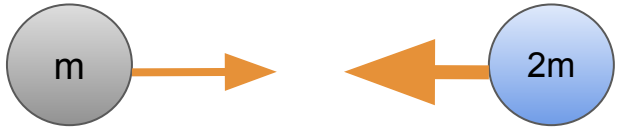
Properties of a system must be compatible with symmetry.
Which of these situations (inputs / outputs) are symmetrically allowed / forbidden?

a.

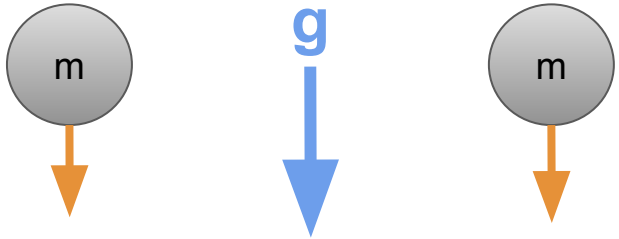


b.





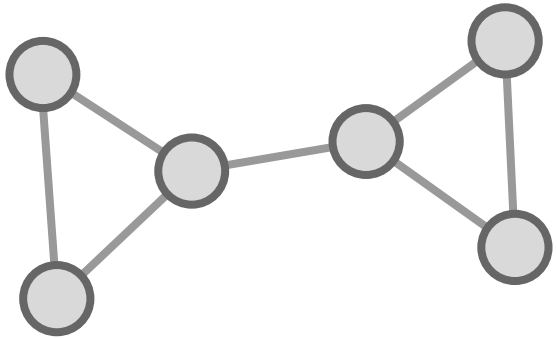
c.



Equivariance can have unintuitive consequences.

Partition graph with permutation equivariant function into two sets using ordered labels.

Predict node labels
[0, 1] vs. [1, 0]

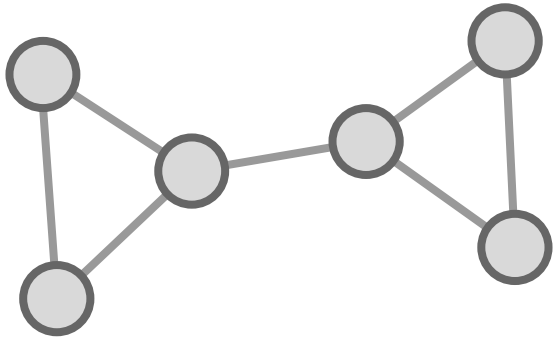


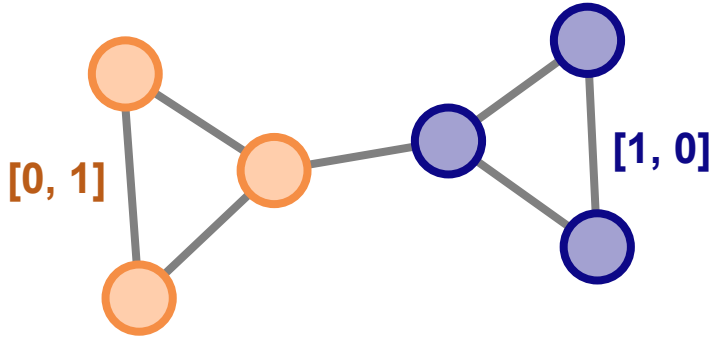
Equivariance can have unintuitive consequences.

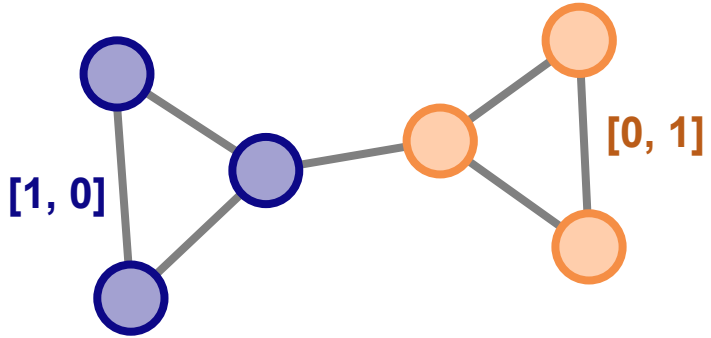
Partition graph with permutation equivariant function into two sets using ordered labels.

You can't *due to degeneracy*.

Predict node labels
[0, 1] vs. [1, 0]







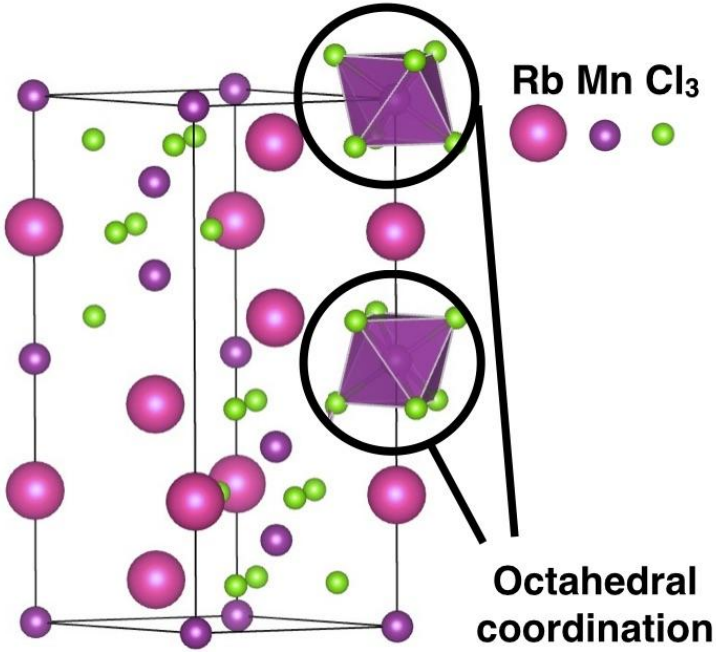
There's nothing to distinguish one partition to be "first" vs. "second".

Convolutions: Local vs. Global Symmetry

Convolutions capture local symmetry. Interaction of features in later layers yields global symmetry.
e.g. Coordination environments in crystals

Atomic systems form geometric motifs that can appear at multiple locations and orientations.

Space group:
Symmetry of unit cell
(Global symmetry)



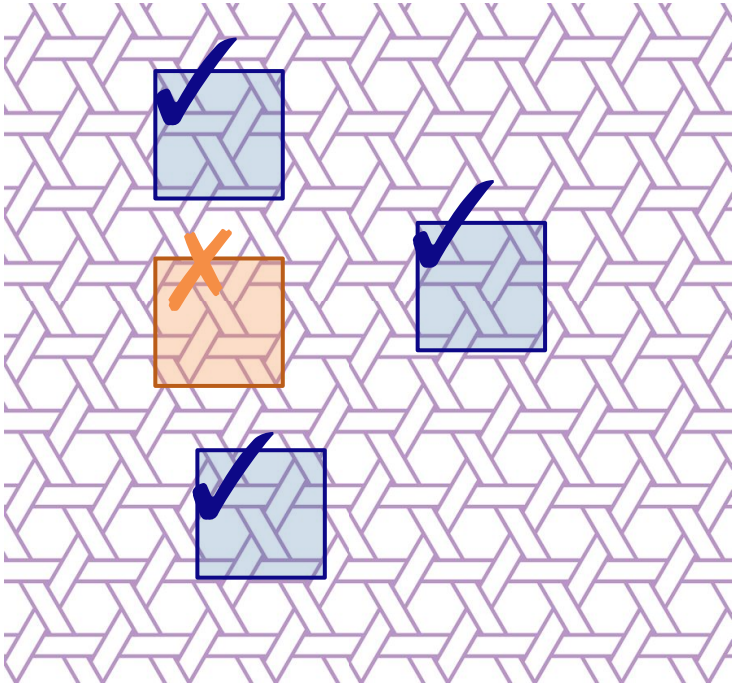
(Local symmetry)

Symmetry emerges when different ways of representing something "mean" the same thing.

Representation can have symmetry, operations can preserve symmetry, and objects can have symmetry.

Translation symmetry in 2D:

Features "mean" the same thing in any location.

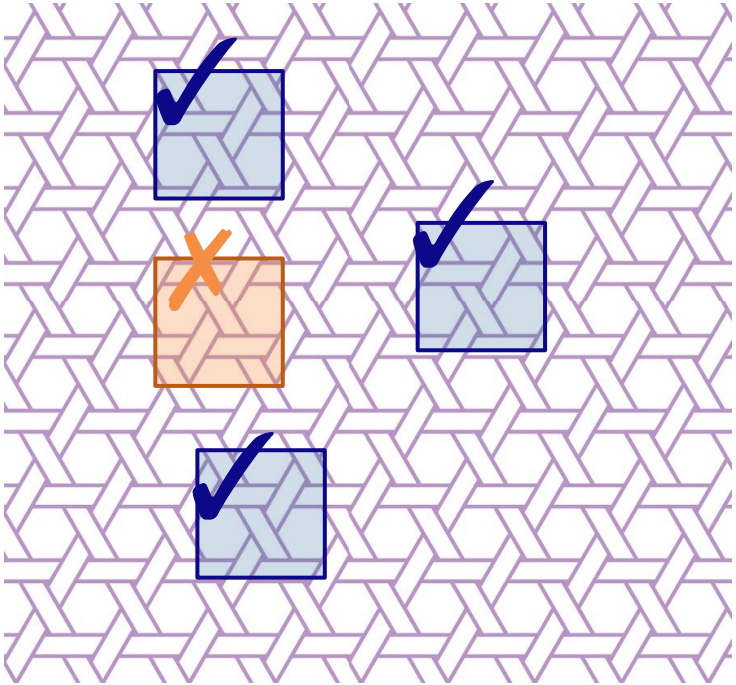


Symmetry emerges when different ways of representing something "mean" the same thing.

Representation can have symmetry, operations can preserve symmetry, and objects can have symmetry.

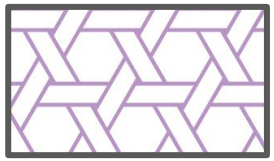
Translation symmetry in 2D:

Features "mean" the same thing in any location.

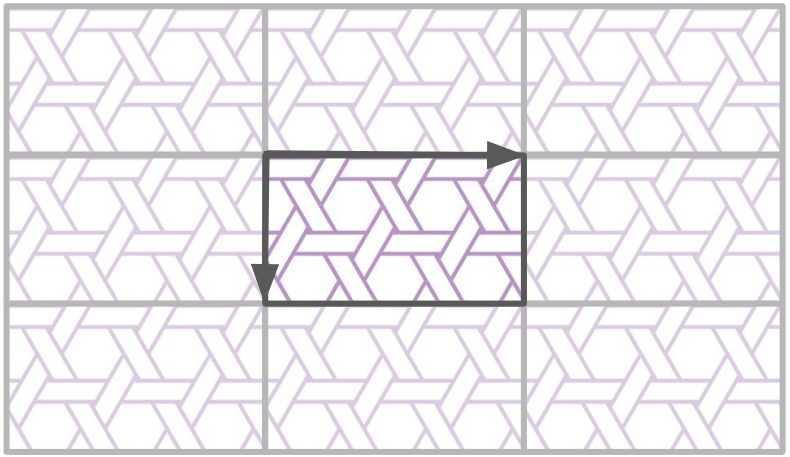


Symmetry of 2D objects

Boundaries "break" global translation symmetry.



Periodic boundary conditions preserve discrete translation symmetry.



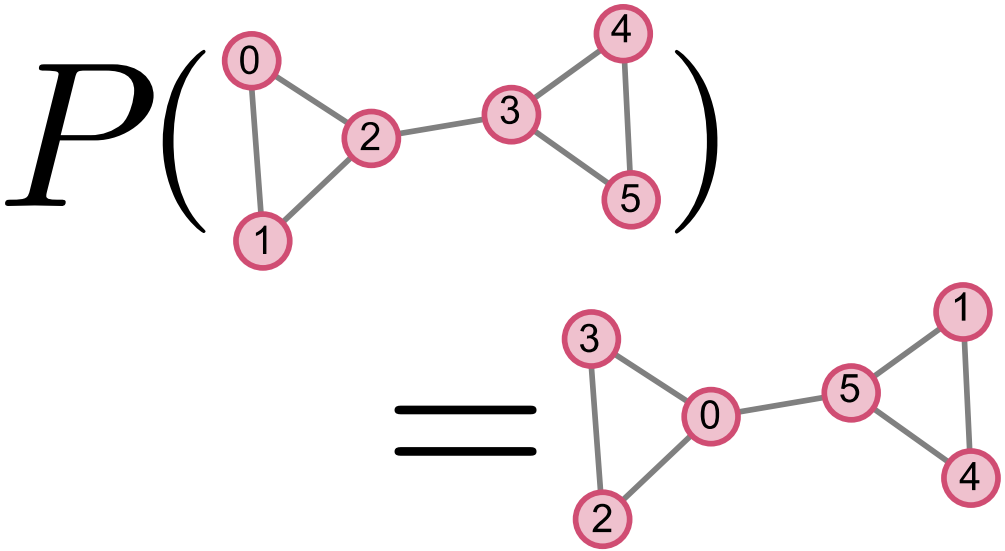
Symmetry emerges when different ways of representing something "mean" the same thing.

Representation can have symmetry, operations can preserve symmetry, and objects can have symmetry.

Permutation symmetry, S_N:

Symmetry of sets

The freedom to list things in any order



Symmetry emerges when different ways of representing something "mean" the same thing.

Representation can have symmetry, operations can preserve symmetry, and objects can have symmetry.

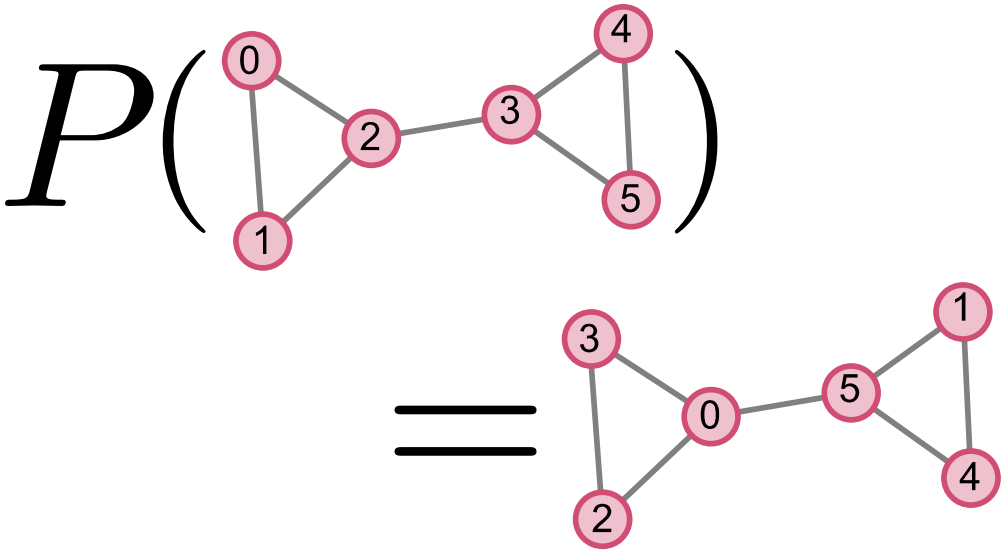
Permutation symmetry, S_N:

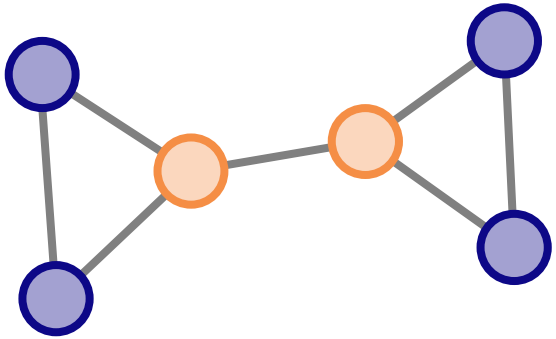
Symmetry of sets

The freedom to list things in any order

Symmetry of elements of a graph

Graph automorphism, specific nodes are indistinguishable (same global connectivity)





A bit of group theory! Don’t worry just a bit!
Formally, what are invariant vs. equivariant functions

vector in vector space

$x \in X$ inputs

$y \in Y$ outputs

$w \in W$ weights

function (neural network)...

$$f(x, w) = y$$

...which is equivalent to writing.

$$f : X, W \rightarrow Y$$

A bit of group theory! Don't worry just a bit!
Formally, what are invariant vs. equivariant functions

vector in vector space

$x \in X$ inputs

$y \in Y$ outputs

$w \in W$ weights

element of group

$g \in G$

function (neural network)...

$$f(x, w) = y$$

...which is equivalent to writing.

$$f : X, W \rightarrow Y$$

representation of g acting on vector space

$$D_x(g)$$

$$D_x(g) : X \rightarrow X$$

$$D_y(g)$$

$$D_y(g) : Y \rightarrow Y$$

A bit of group theory! Don't worry just a bit!
Formally, what are invariant vs. equivariant functions

vector in vector space

$x \in X$ inputs

$y \in Y$ outputs

$w \in W$ weights

element of group

$g \in G$

function (neural network)...

$f(x, w) = y$

...which is equivalent to writing.

$f : X, W \rightarrow Y$

representation of g acting on vector space

$D_x(g)$

$D_y(g)$

$D_x(g) : X \rightarrow X$

$D_y(g) : Y \rightarrow Y$

equivariant to x if

$f(D_x(g)\,x, D_w(g)w)$

A bit of group theory! Don't worry just a bit!
Formally, what are invariant vs. equivariant functions

vector in vector space

$x \in X$ inputs

$y \in Y$ outputs

$w \in W$ weights

element of group

$g \in G$

function (neural network)...

$f(x, w) = y$

...which is equivalent to writing.

$f : X, W \to Y$

representation of g acting on vector space

$D_x(g)$ $\quad D_x(g) : X \to X$

$D_y(g)$ $\quad D_y(g) : Y \to Y$

$D_w(g) = \mathbb{1}$

If we want to be equivariant to x, this has to be the case… weights must be "scalars"

equivariant to x if

$$f(D_x(g)\,x, D_w(g)w) = f(D_x(g)x, w)$$

A bit of group theory! Don't worry just a bit!
Formally, what are invariant vs. equivariant functions

vector in vector space

$x \in X$ inputs

$y \in Y$ outputs

$w \in W$ weights

element of group

$g \in G$

function (neural network)...

$$f(x, w) = y$$

...which is equivalent to writing.

$$f : X, W \rightarrow Y$$

representation of g acting on vector space

$$D_x(g) \qquad D_x(g) : X \rightarrow X$$

$$D_y(g) \qquad D_y(g) : Y \rightarrow Y$$

$$D_w(g) = \mathbb{1}$$

If we want to be equivariant to x, this has to be the case… weights must be "scalars"

equivariant to x if

$$f(D_x(g)\,x, D_w(g)w) = f(D_x(g)x, w)$$

$$= D_y f(x, w) = D_y(g)y$$

A bit of group theory! Don't worry just a bit!
Formally, what are invariant vs. equivariant functions

vector in vector space

$x \in X$ inputs

$y \in Y$ outputs

$w \in W$ weights

element of group

$g \in G$

function (neural network)...

$$f(x, w) = y$$

...which is equivalent to writing.

$$f : X, W \rightarrow Y$$

representation of g acting on vector space

$$D_x(g) \qquad D_x(g) : X \rightarrow X$$

$$D_y(g) \qquad D_y(g) : Y \rightarrow Y$$

$$D_w(g) = \mathbb{1}$$

If we want to be equivariant to x, this has to be the case… weights must be "scalars"

equivariant to x if

$$f(D_x(g)\,x, D_w(g)w) = f(D_x(g)x, w)$$

$$= D_y f(x, w) = D_y(g)y$$

(special case) invariant to x if

$$D_x(g) = D_y(g) = \mathbb{1}$$

Theorem 2 *A function $f(X)$ operating on a set X having elements from a countable universe, is a valid set function,* i.e., ***invariant*** *to the permutation of instances in X, iff it can be decomposed in the form $\rho\left(\sum_{x\in X}\phi(x)\right)$, for suitable transformations ϕ and ρ.*

The extension to case when $\mathfrak{X}$ is uncountable, like $\mathfrak{X} = \mathbb{R}$, we could only prove that $f(X) = \rho\left(\sum_{x\in X}\phi(x)\right)$ holds for sets of fixed size. The proofs and difficulties in handling the uncountable case, are discussed in Appendix A. However, we still conjecture that exact equality holds in general.

Next, we analyze the **equivariant** case when $\mathfrak{X} = \mathcal{Y} = \mathbb{R}$ and $\mathbf{f}$ is restricted to be a neural network layer. The standard neural network layer is represented as $\mathbf{f}_\Theta(\mathbf{x}) = \boldsymbol{\sigma}(\Theta\mathbf{x})$ where $\Theta \in \mathbb{R}^{M\times M}$ is the weight vector and $\sigma : \mathbb{R} \to \mathbb{R}$ is a nonlinearity such as sigmoid function. The following lemma states the necessary and sufficient conditions for permutation-equivariance in this type of function.

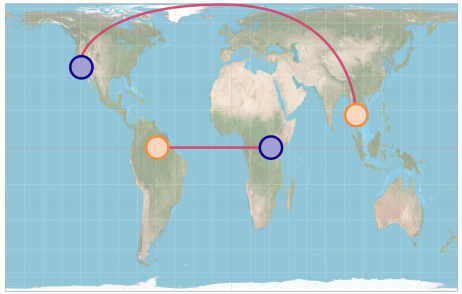
Lemma 3 *The function $\mathbf{f}_\Theta : \mathbb{R}^M \to \mathbb{R}^M$ defined above is permutation* ***equivariant*** *iff all the off-diagonal elements of Θ are tied together and all the diagonal elements are equal as well. That is,*

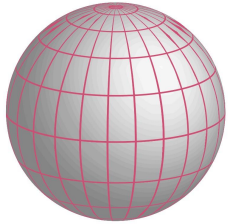
$$\Theta = \lambda\mathbf{I} + \gamma\,(\mathbf{1}\mathbf{1}^\mathsf{T}) \qquad \lambda,\gamma \in \mathbb{R} \quad \mathbf{1} = [1,\ldots,1]^\mathsf{T} \in \mathbb{R}^M \qquad \mathbf{I} \in \mathbb{R}^{M\times M} \textit{ is the identity matrix}$$

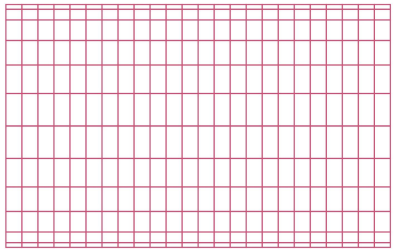
This result can be easily extended to higher dimensions, *i.e.*, $\mathfrak{X} = \mathbb{R}^d$ when λ, γ can be matrices.

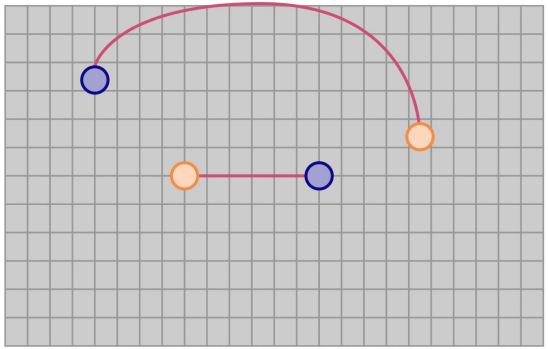
Convolutional neural networks can “cheat” by being sensitive to “boundaries”.

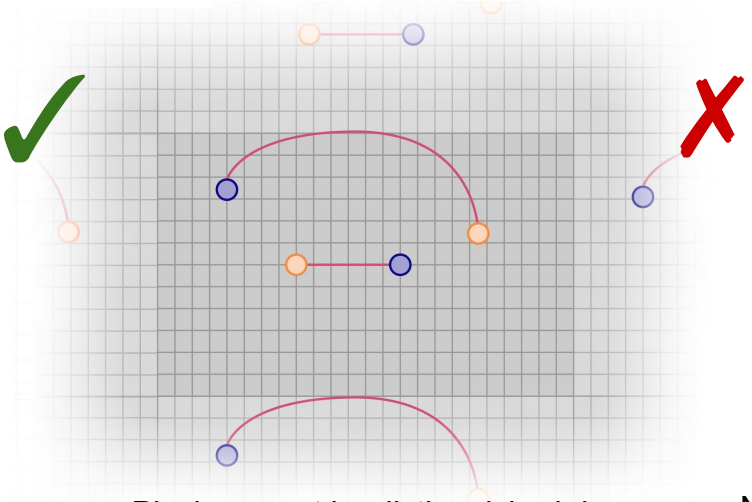
(e.g. Predict geodesics on projected maps with and without periodic boundary conditions)

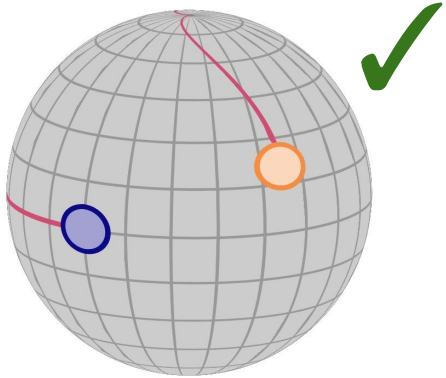












Boundaries break symmetry.

Pixels cannot be distinguished due to translation equivariance.

Nodes can be distinguished due to differing topology by latitude (e.g. poles)!

User: Stebe
https://en.wikipedia.org/wiki/Gall-Peters_projection

In the physical sciences...
What our our data types?
3D geometry and geometric tensors...
...which transform predictably under 3D rotation, translation, and inversion.

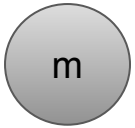
These data types assume Euclidean symmetry.
⇨ Thus, we need neural networks that preserve Euclidean symmetry.

Geometric tensors take many forms. *They are a general data type beyond materials.*

Scalars

- Energy
- Mass
- Isotropic *

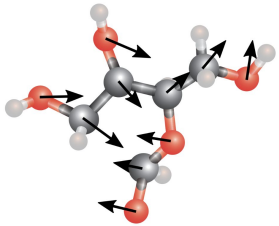
$$E$$



Vectors

- Force
- Velocity
- Acceleration
- Polarization

$$\vec{F}$$



Pseudovectors

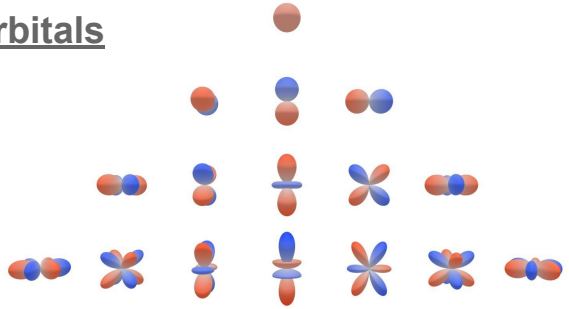
- Angular momentum
- Magnetic fields

Matrices, Tensors, …

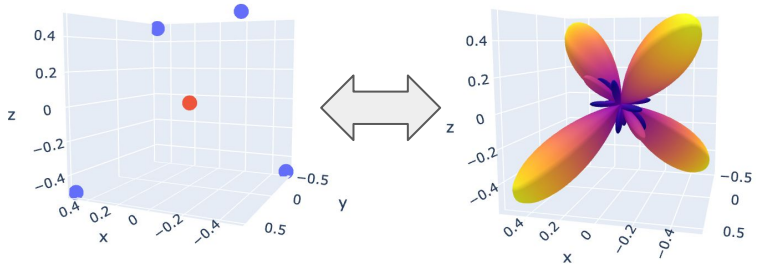
- Moment of Inertia
- Polarizability
- Interaction of multipoles
- Elasticity tensor (rank 4)

$$\alpha = \begin{bmatrix} \alpha_{xx} & \alpha_{xy} & \alpha_{xz} \\ \alpha_{yx} & \alpha_{yy} & \alpha_{yz} \\ \alpha_{zx} & \alpha_{zy} & \alpha_{zz} \end{bmatrix}$$

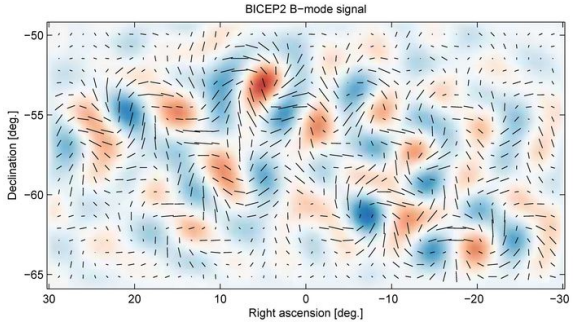
Atomic orbitals



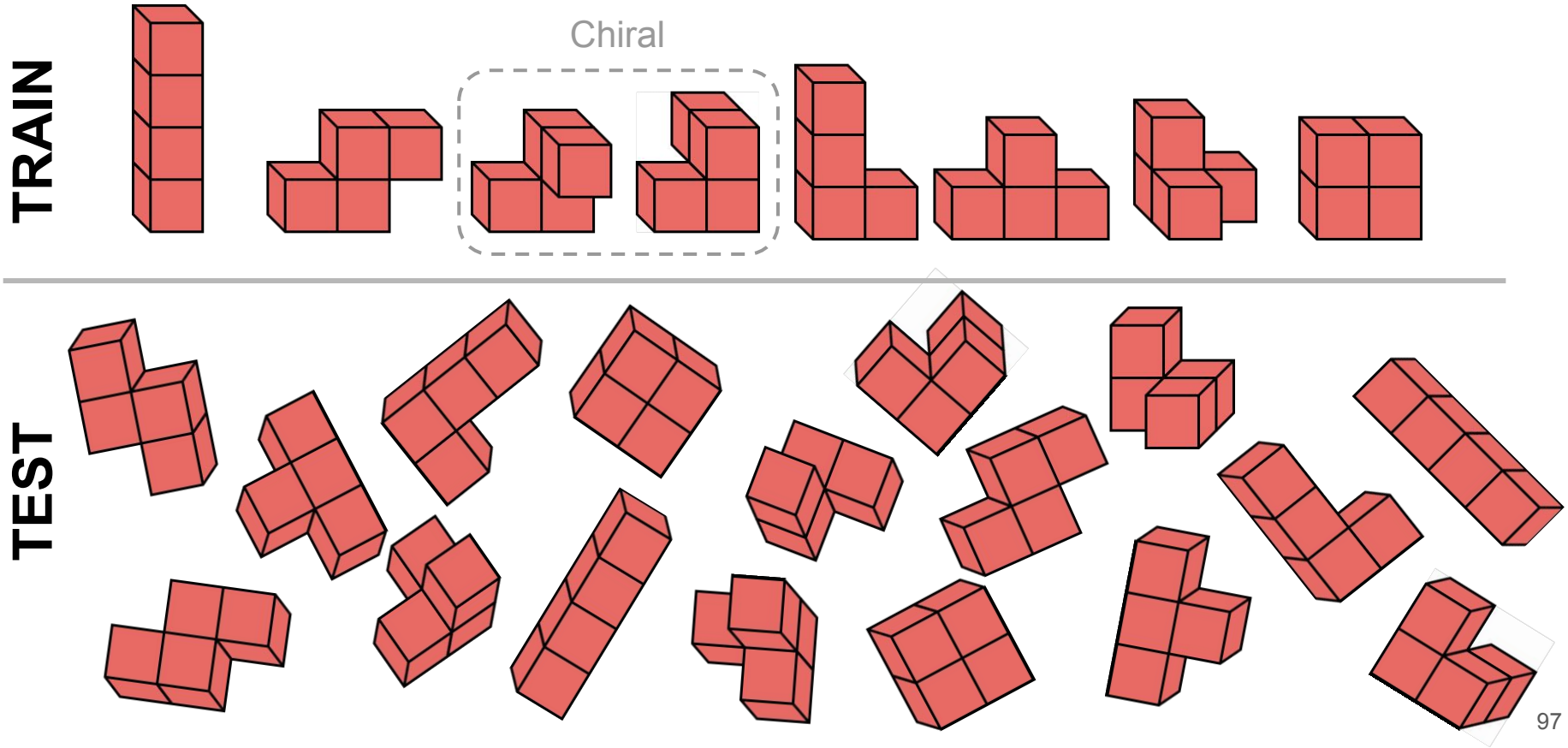
Output of Angular Fourier Transforms



Vector fields on spheres (e.g. B-modes of the Cosmic Microwave Background)



Our unit test: Trained on 3D Tetris shapes in one orientation, these network can perfectly identify these shapes in any orientation.



Several groups converged on similar ideas around the same time.

Tensor field networks: Rotation- and translation-equivariant neural networks for 3D point clouds
(arXiv:1802.08219)
Tess Smidt*, Nathaniel Thomas*, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, Patrick Riley
Points, nonlinearity on norm of tensors

Clebsch-Gordan Nets: a Fully Fourier Space Spherical Convolutional Neural Network
(arXiv:1806.09231)
Risi Kondor, Zhen Lin, Shubhendu Trivedi
Only use tensor product as nonlinearity, no radial function

3D Steerable CNNs: Learning Rotationally Equivariant Features in Volumetric Data
(arXiv:1807.02547)
Mario Geiger*, Maurice Weiler*, Max Welling, Wouter Boomsma, Taco Cohen
Efficient framework for voxels, gated nonlinearity

**denotes equal contribution*

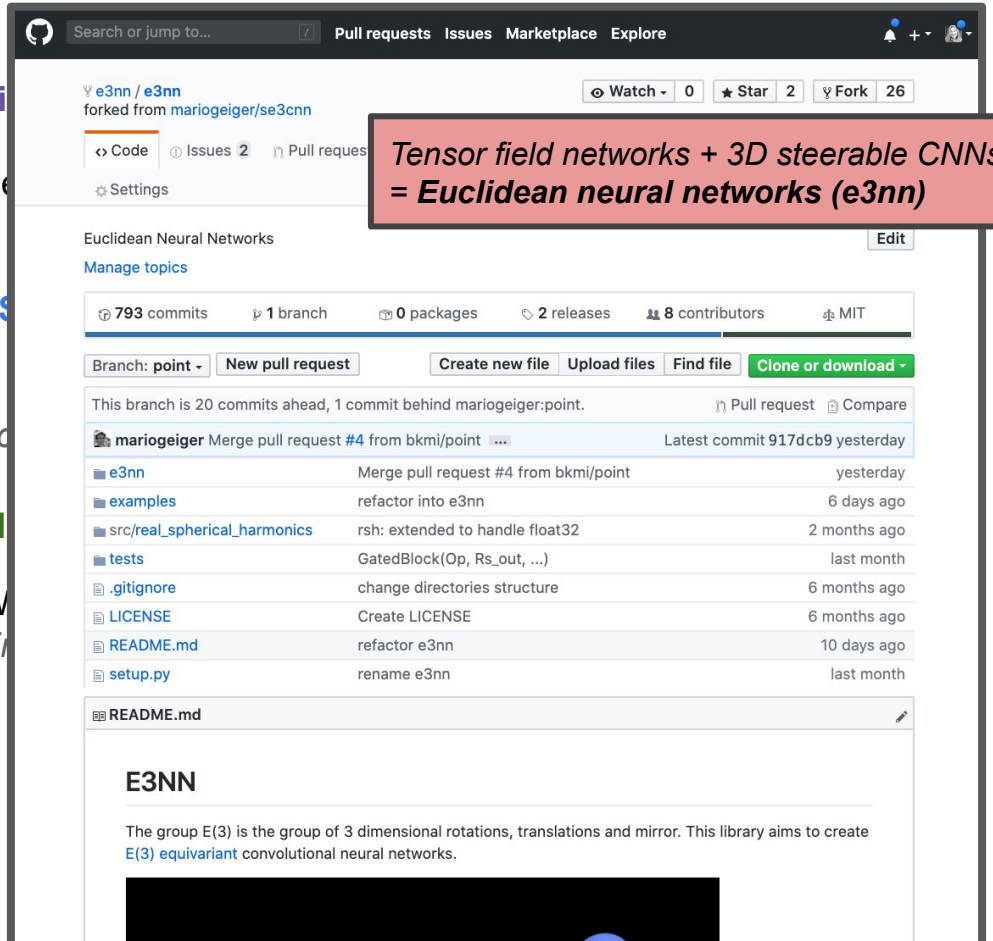
Several groups converged on similar ideas around the same time.

Tensor field networks: Rotation- and translati
(arXiv:1802.08219)
Tess Smidt*, Nathaniel Thomas*, Steven Kearne
Points, nonlinearity on norm of tensors

Clebsch-Gordan Nets: a Fully Fourier Space S
(arXiv:1806.09231)
Risi Kondor, Zhen Lin, Shubhendu Trivedi
Only use tensor product as nonlinearity, nc

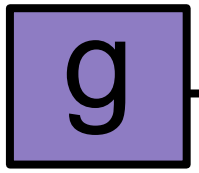
3D Steerable CNNs: Learning Rotationally Eq
(arXiv:1807.02547)
Mario Geiger*, Maurice Weiler*, Max Welling, W
Efficient framework for voxels, gated nonli

**denotes equal contribution*

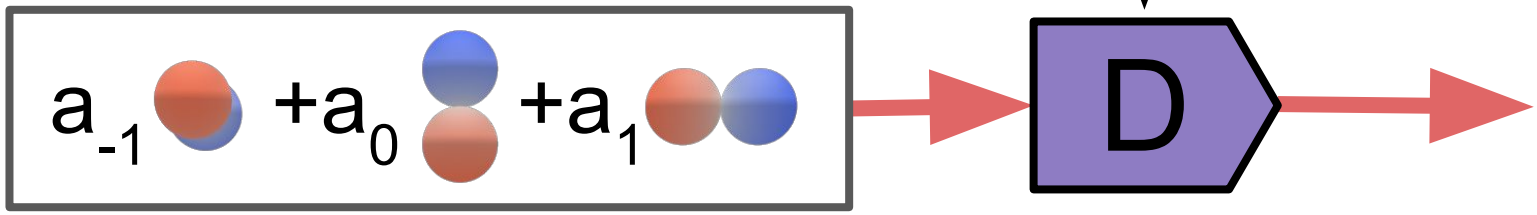


Spherical harmonics of a given L transform together under rotation.

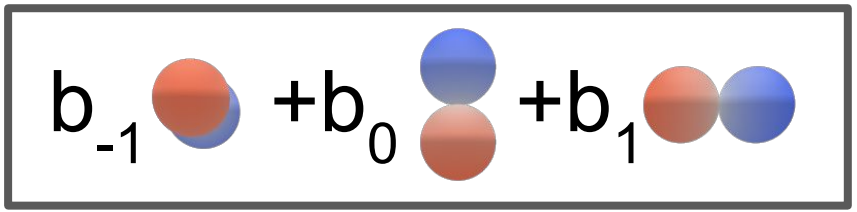
Let **g** be a 3d rotation matrix.



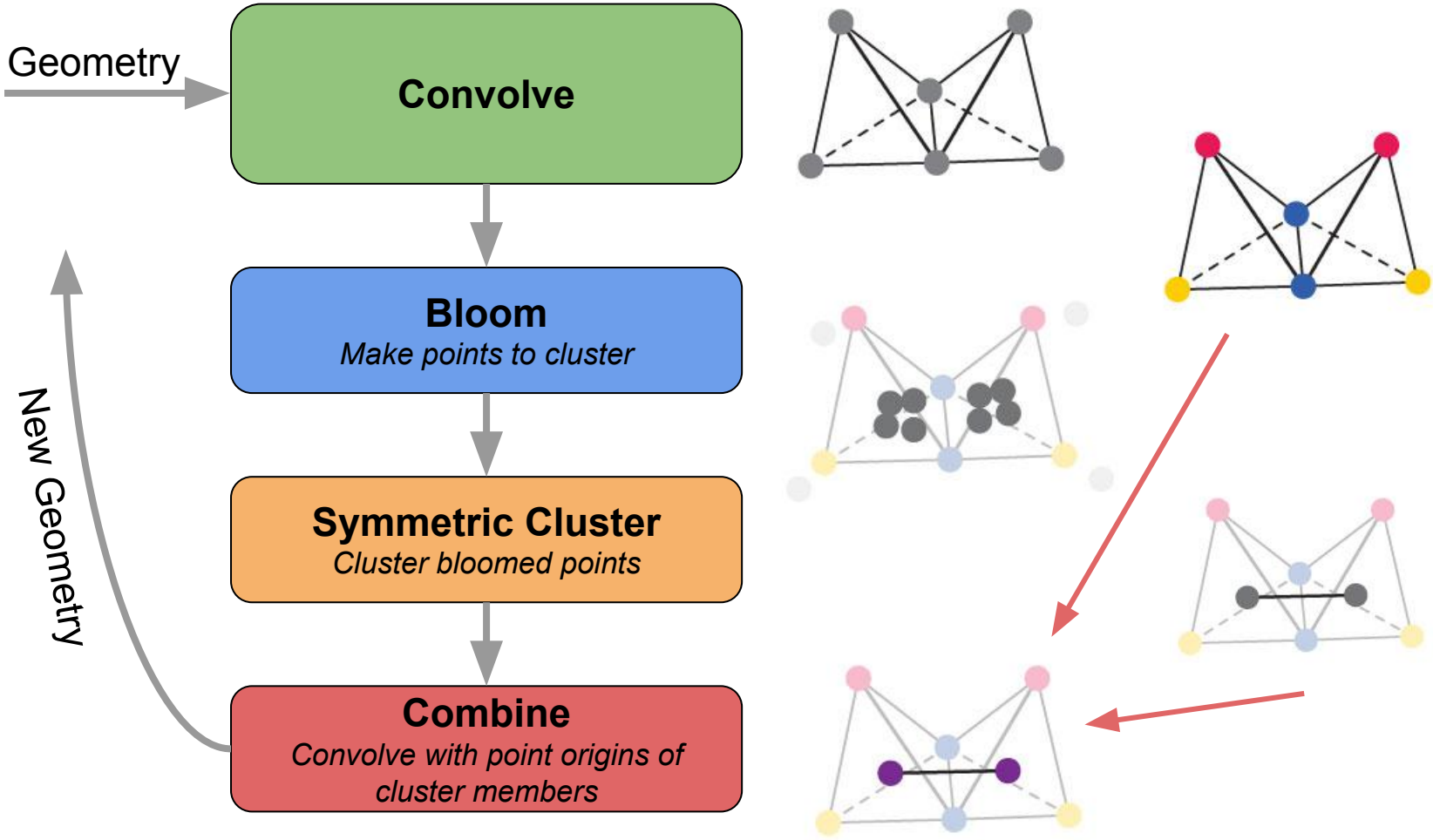
D is the Wigner-D matrix. It has shape $[2l+1, 2l+1]$ and is a function of **g**.



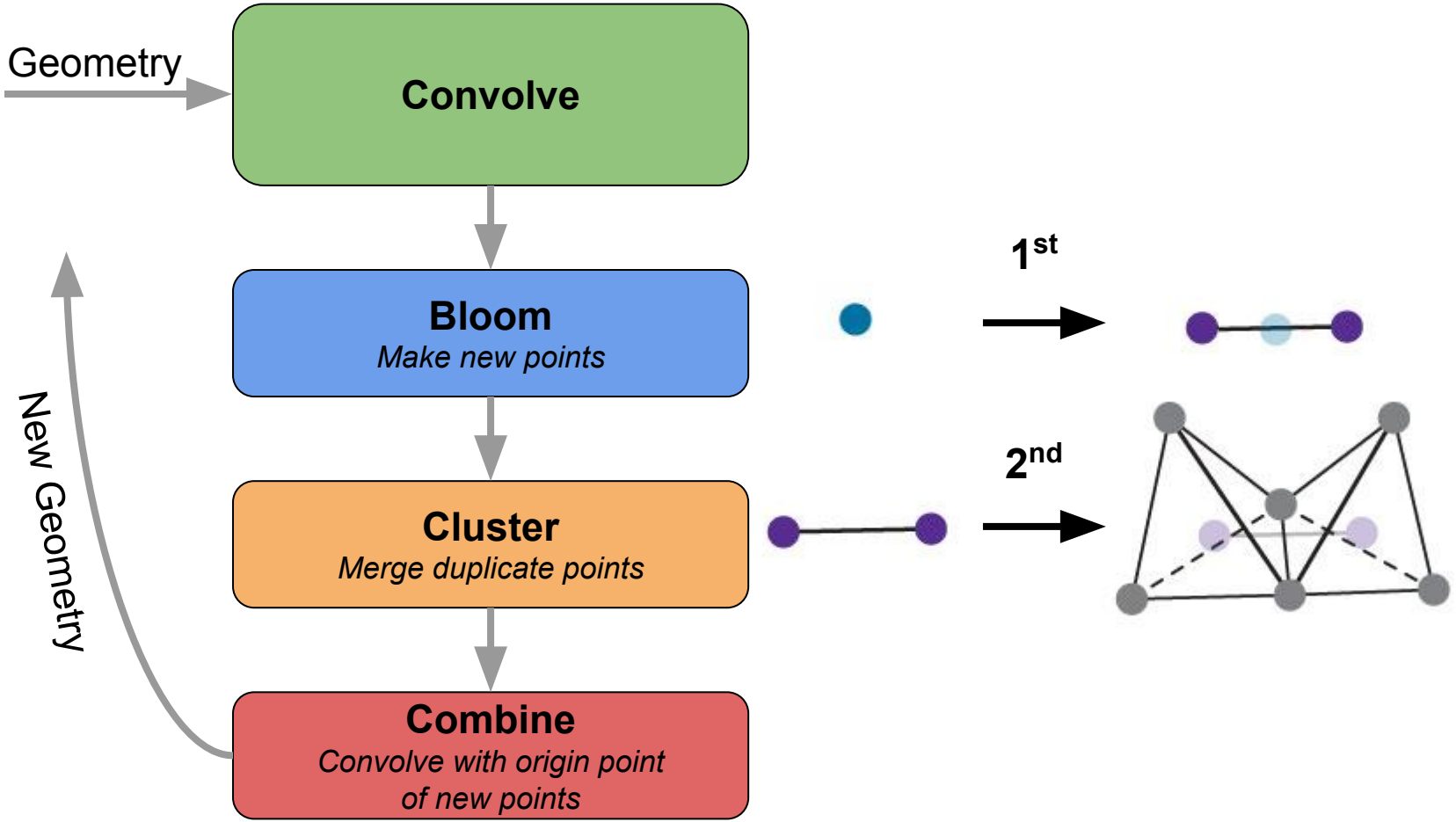
=



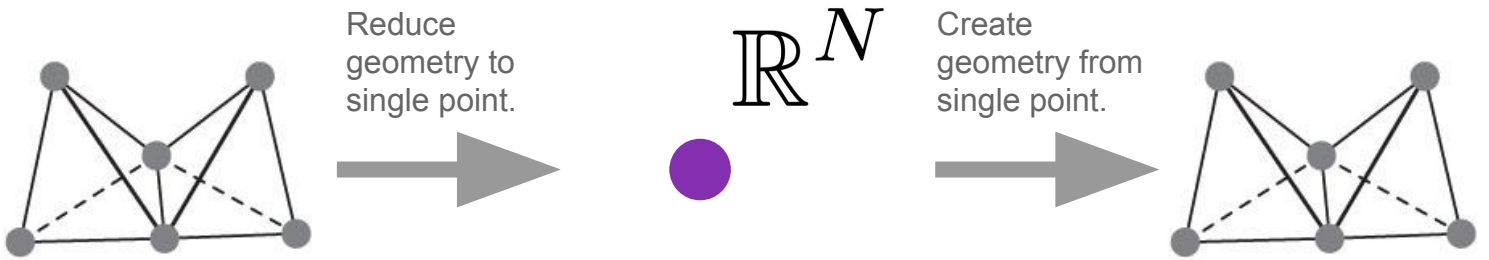
How to encode *(Pooling layer).* Recursively convert geometry to features.



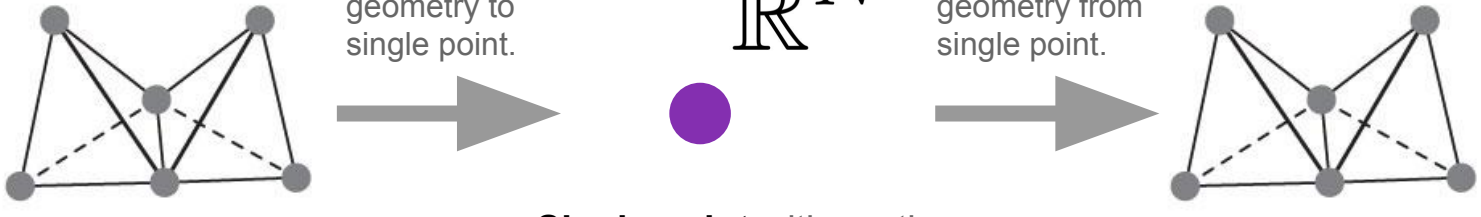
How to decode *(Unpooling layer).* Recursively convert features to geometry.



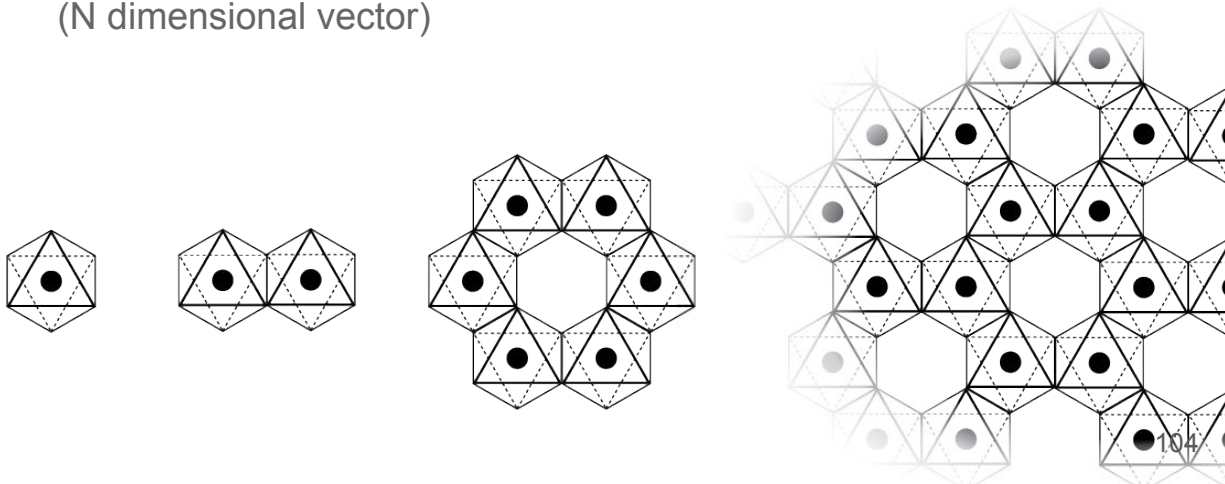
We want to convert geometric information (3D coordinates of atomic positions) **into features on a trivial geometry** (a single point) **and back again.**



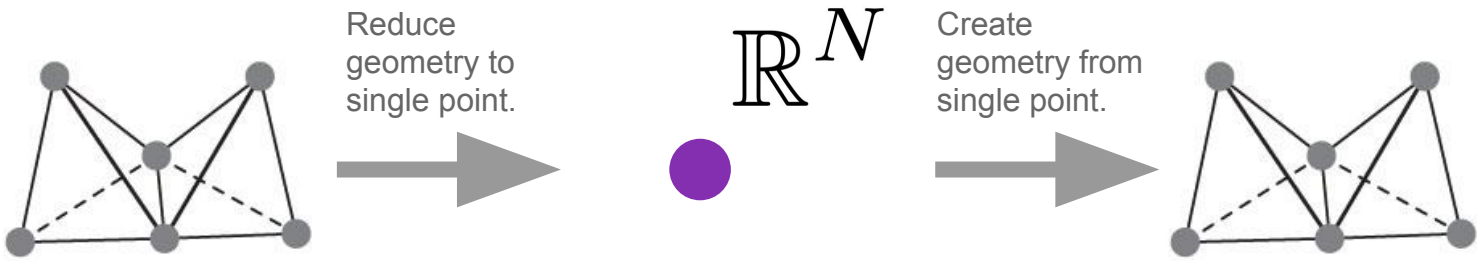
We want to convert geometric information (3D coordinates of atomic positions) **into features on a trivial geometry** (a single point) **and back again.**



Atomic structures are <u>hierarchical</u> and can be constructed from <u>recurring geometric motifs</u>.



We want to convert geometric information (3D coordinates of atomic positions) into features on a trivial geometry (a single point) and back again.



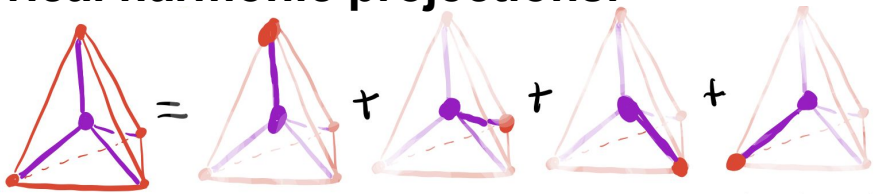
Atomic structures are <u>hierarchical</u> and can be constructed from <u>recurring geometric motifs</u>.

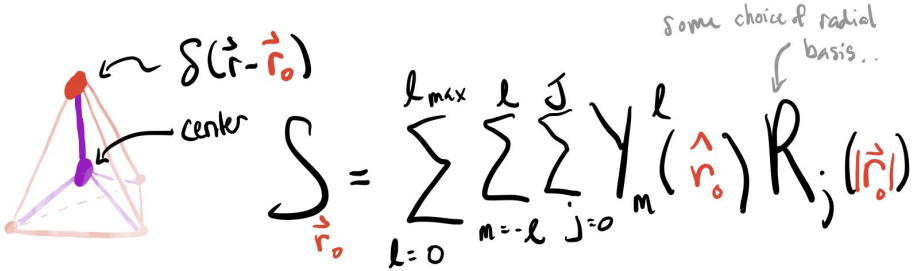
\+ Encode geometry
\+ Encode hierarchy

\+ Decode geometry
\+ Decode hierarchy

(Need to do this in a recursive manner)

To autoencode, we have to be able to convert *geometry* into *features* and *vice versa*.
We do this via spherical harmonic projections.





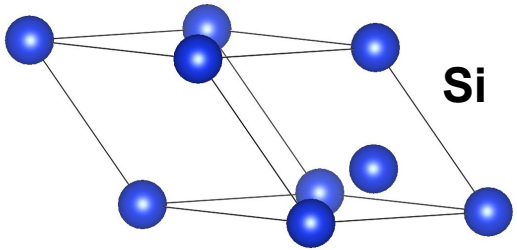
vector size $\left[\sum_{l=0}^{l_{max}} (2l+1) \times J\right]$

Yay! Linearity!

$$S_{\text{tetrahedron}} = S_{\vec{r}_0} + S_{\vec{r}_1} + S_{\vec{r}_2} + S_{\vec{r}_3}$$

What a computational materials physicist does:

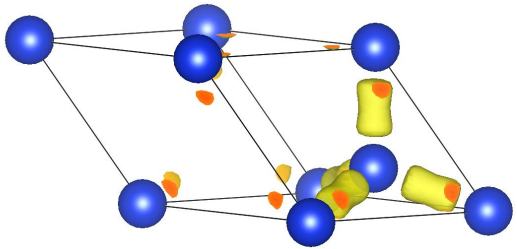
Given an atomic structure,



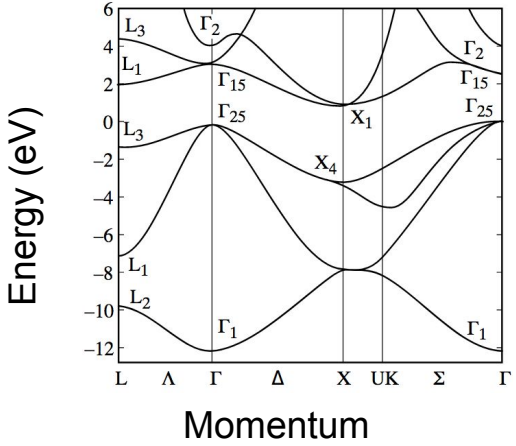
...use quantum theory and supercomputers to determine...

$$\hat{H}\,|\psi\rangle = E\,|\psi\rangle$$

...where the electrons are...



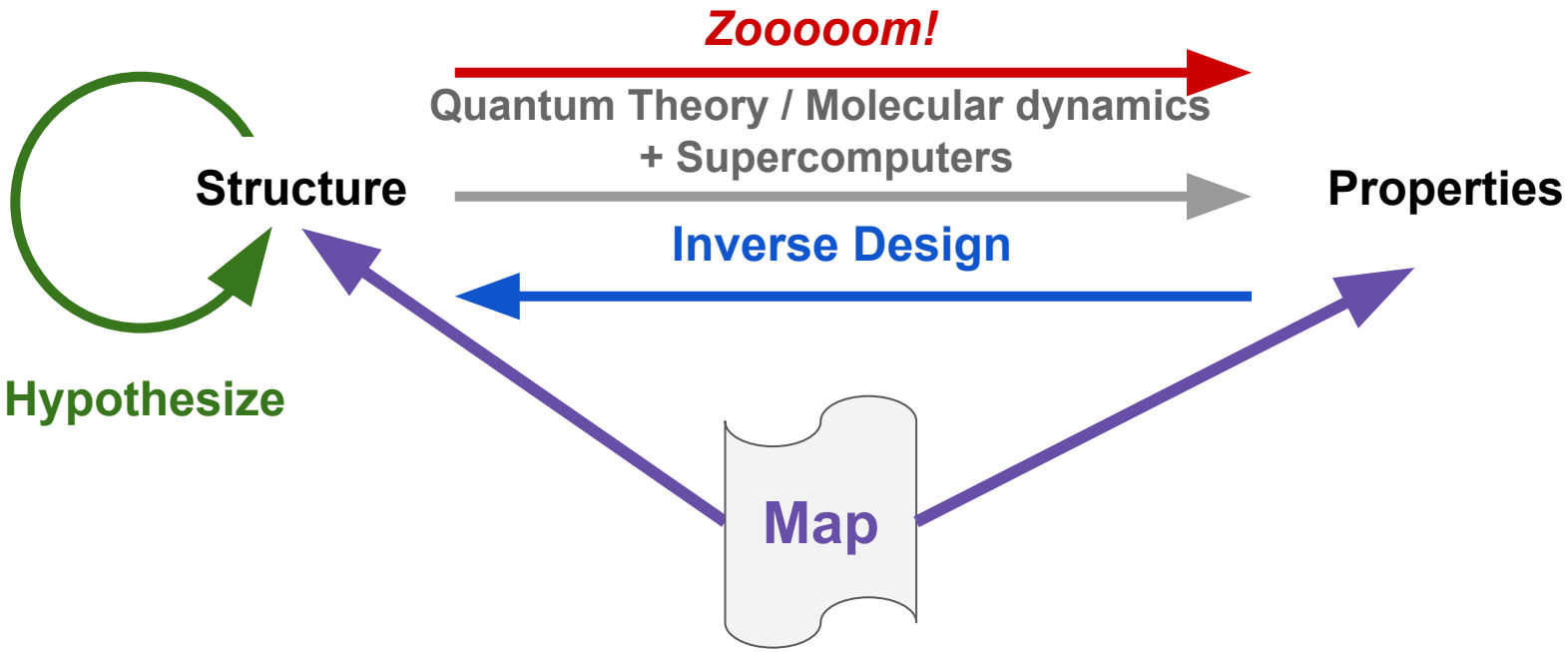
...and what the electrons are doing.



Structure

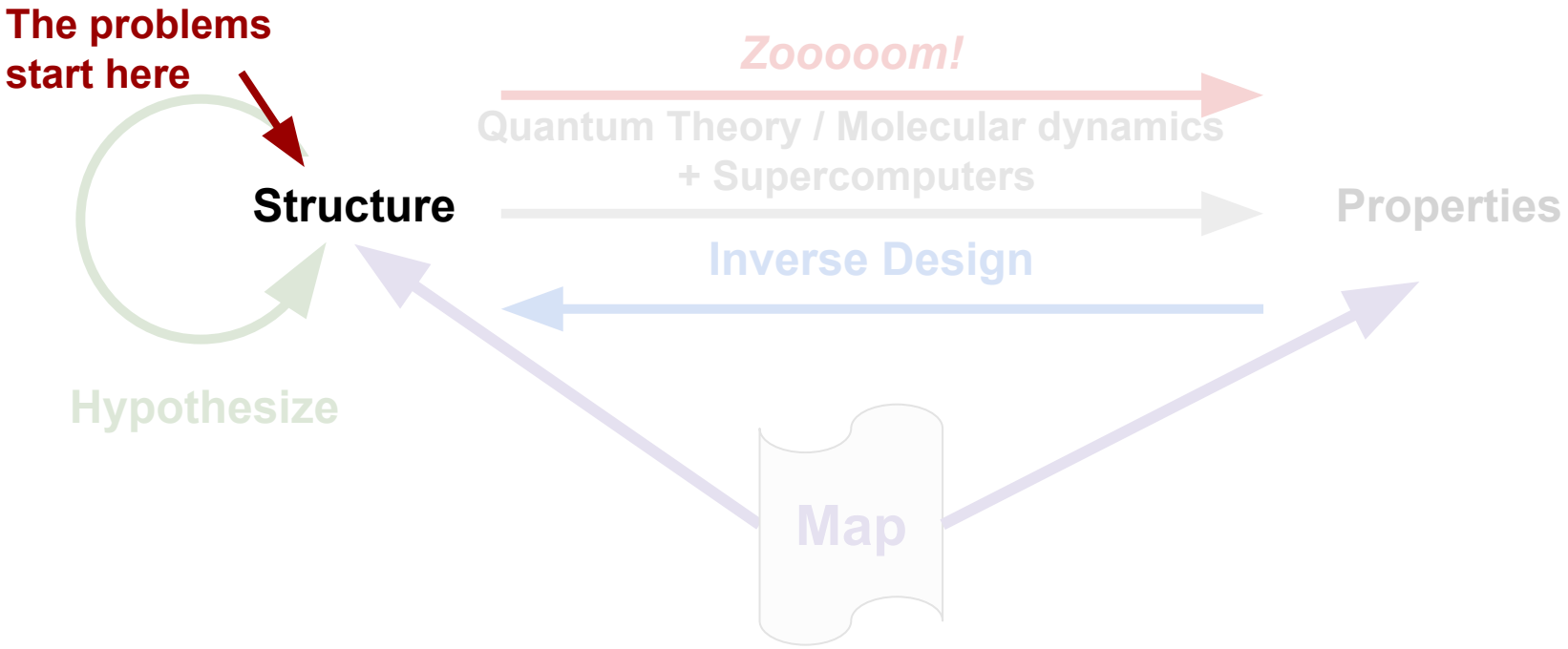
Properties

We want to use deep learning to speed up calculations, hypothesize new structures, perform inverse design, and organize these relations.



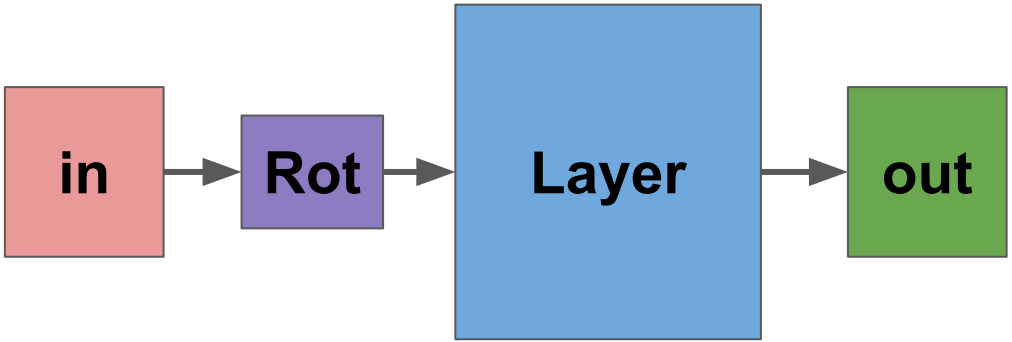
Primer?

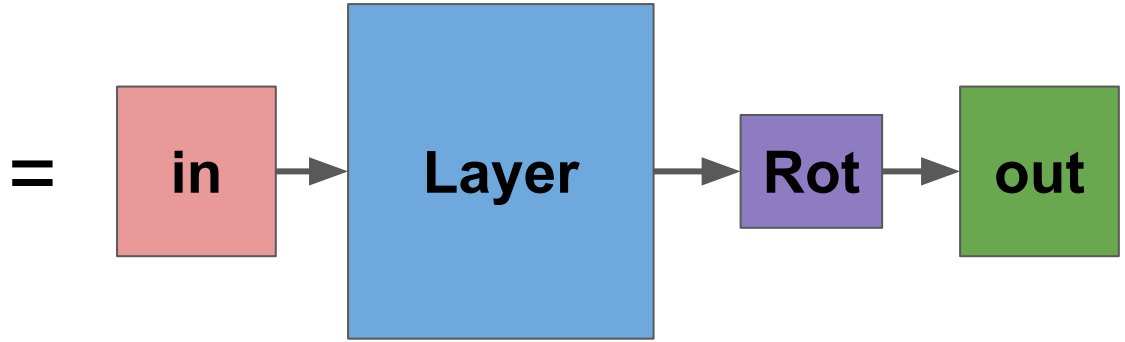
We want to use deep learning to speed up calculations, hypothesize new structures, perform inverse design, and organize these relations.



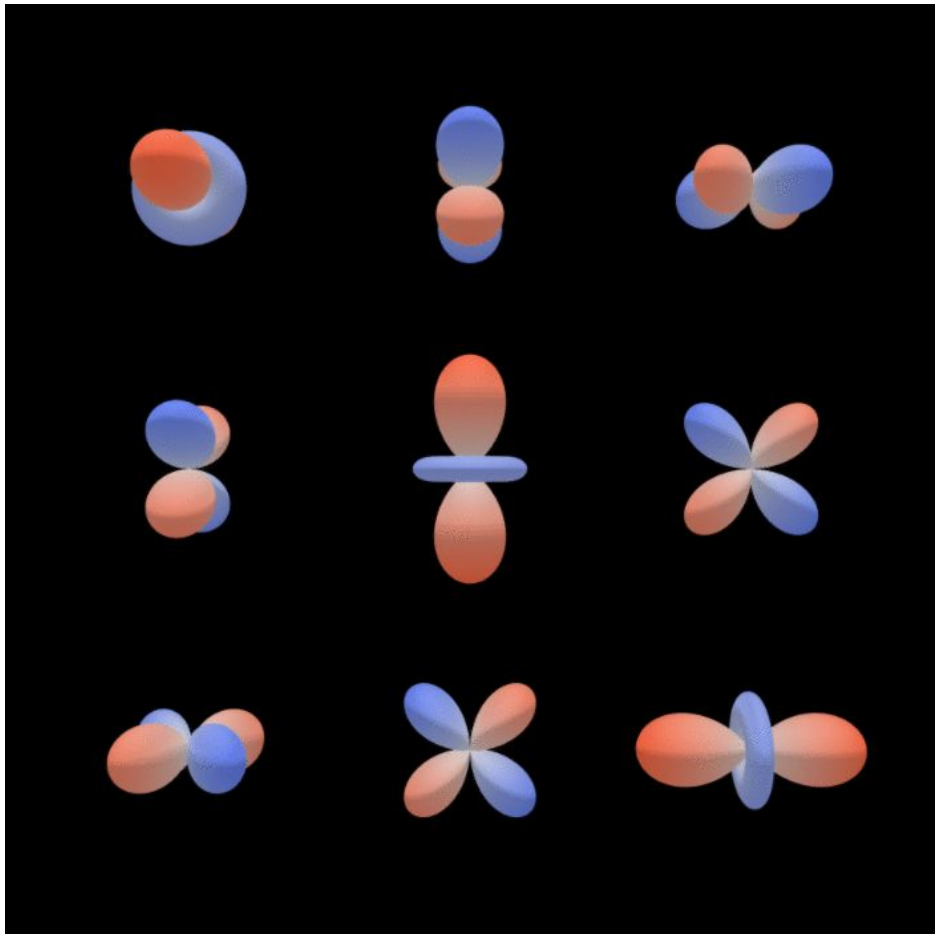
Given a single example of a degenerate solution,
it knows what other solutions are possible by symmetry.
(Useful for ensuring you're not biasing your sampling.)

To be rotation-equivariant means that we can rotate our inputs OR rotate our outputs and we get the same answer ***(for every operation)*****.**





For L=1 ⇨ L=1, the filters will be a learned, radially-dependent linear combinations of the L = 0, 1, and 2 spherical harmonics.



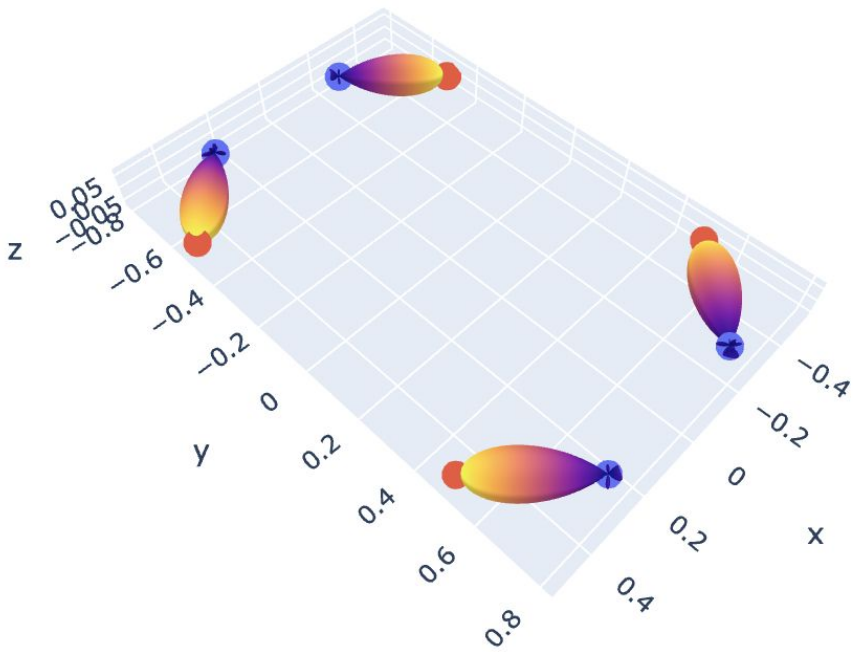
Random filters for L=1 ⇨ L=1…

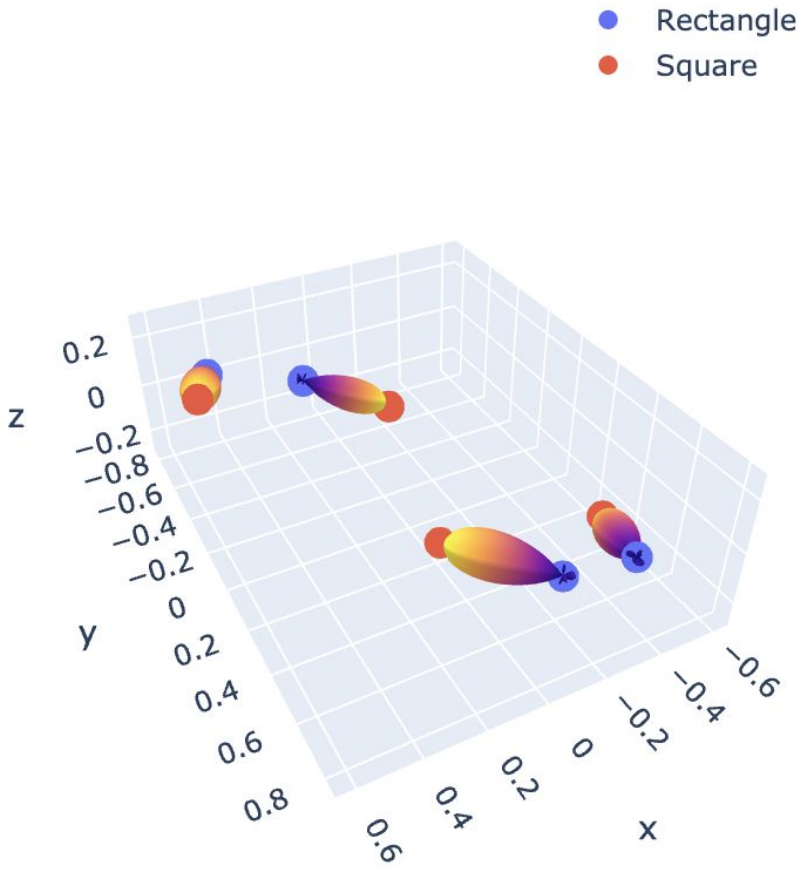
(3 in L=1 channels by 3 out L=1 channels)

… as a function of increasing r.

Time showing filter for varying r, where $0 \leq r \leq r_{max}$.

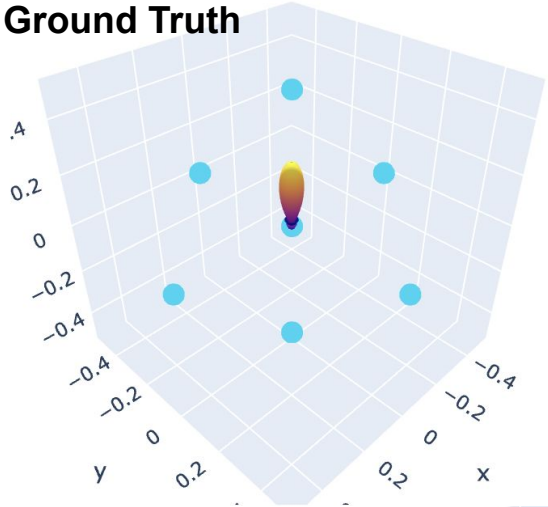
(+ / –) Radial distance is magnitude as a function of angle

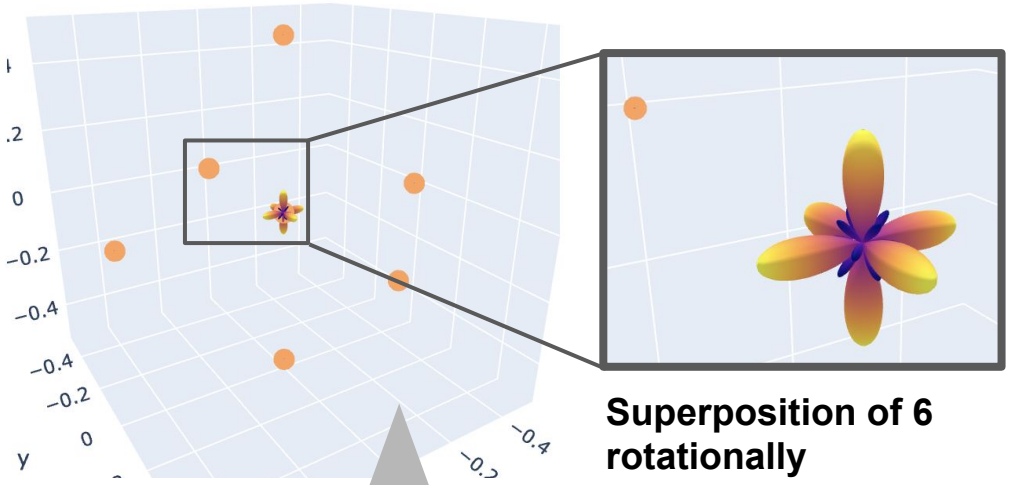

z
0.05
−0.8
−0.6
−0.4
−0.2
y
0
0.2
0.4
0.6
0.8
−0.4
−0.2
0
0.2
0.4
x


Rectangle
Square
z
0.2
0
−0.2
−0.8
−0.6
−0.4
−0.2
y
0
0.2
0.4
0.6
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
x

Predictions for O_h symmetry

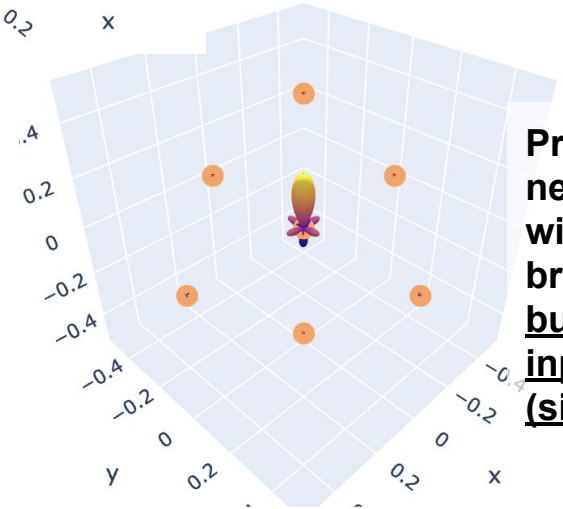
Ground Truth



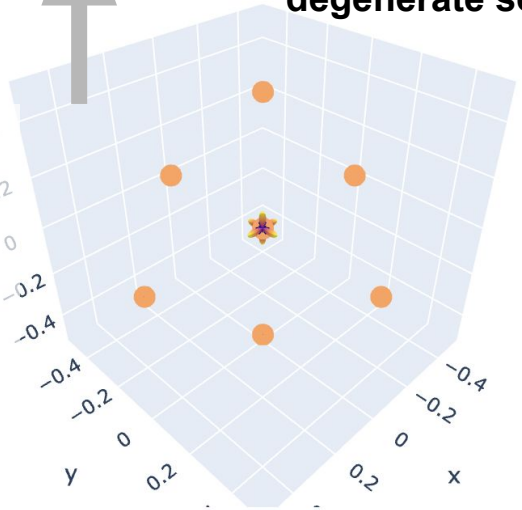


Superposition of 6 rotationally degenerate solutions.

Prediction of network trained with symmetry breaking input <u>and given symmetry breaking input along z.</u>



Prediction of network trained with symmetry breaking input <u>but given trivial input (single scalar)</u>.



A brief primer on deep learning

Skip?

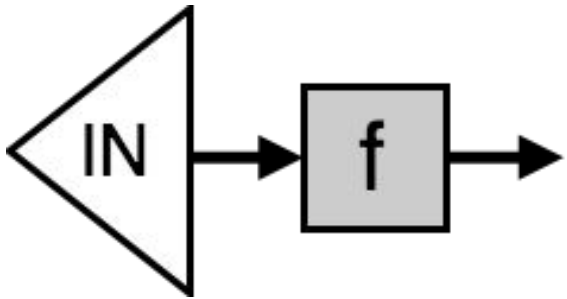
deep learning $\subset$ machine learning $\subset$ artificial intelligence

A brief primer on deep learning

model (“neural network”):

Function with learnable parameters.

$$y = f(x)$$



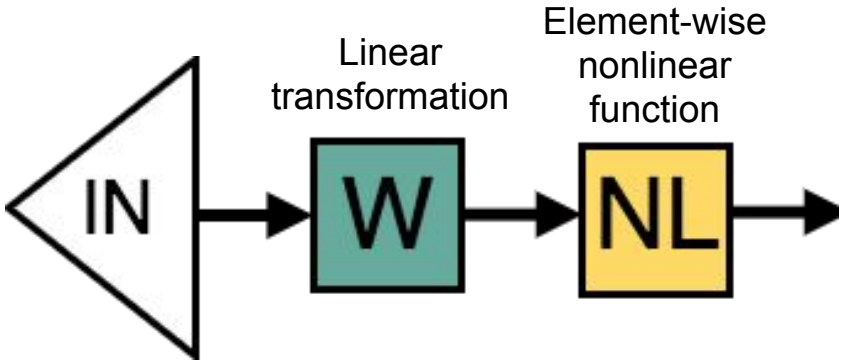
A brief primer on deep learning

model (“neural network”):
Function with learnable parameters.

Ex: "Fully-connected" network

$$y = \tanh(Wx + b)$$

Learned Parameters



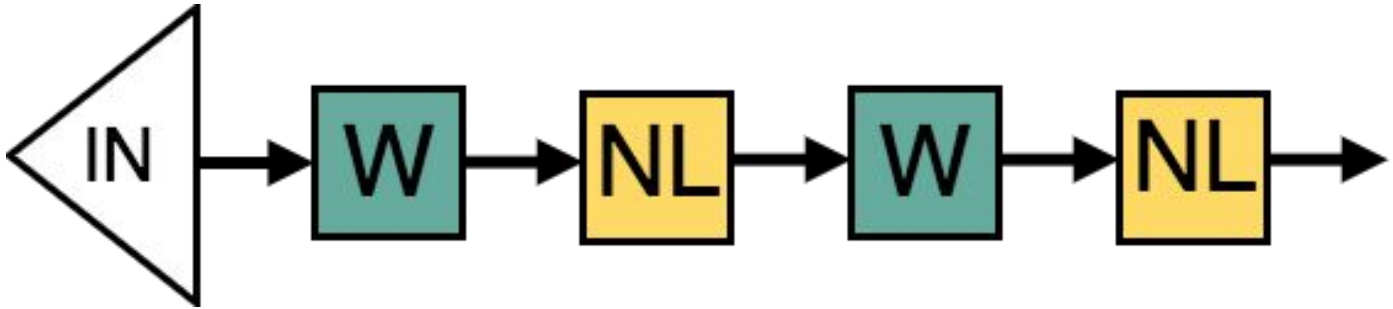
A brief primer on deep learning

model (“neural network”):
Function with learnable parameters.

Ex: "Fully-connected" network

$$y = \tanh(W_2 \tanh(W_1 x + b_1) + b_2)$$

Learned Parameters



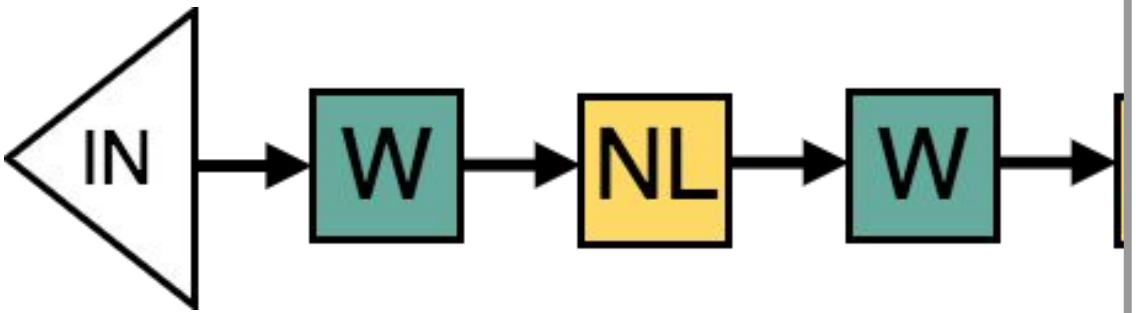
Neural networks with multiple layers can learn more complicated functions.

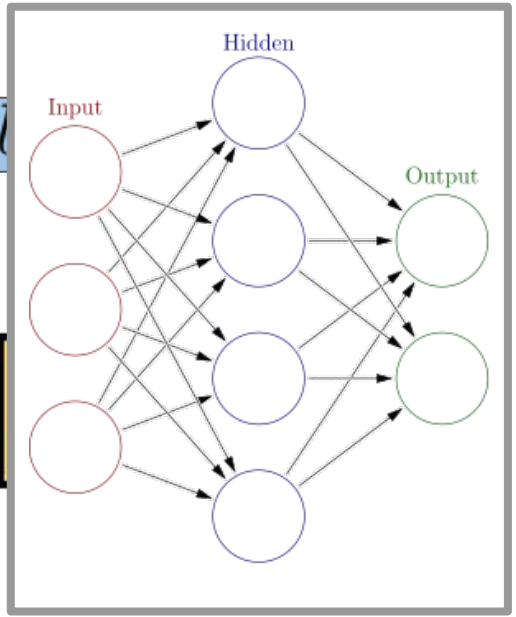
A brief primer on deep learning

model (“neural network”):
Function with learnable parameters.

Ex: "Fully-connected" network

$$y = \tanh(W_2 \tanh(W_1 x + b_1) + b$$



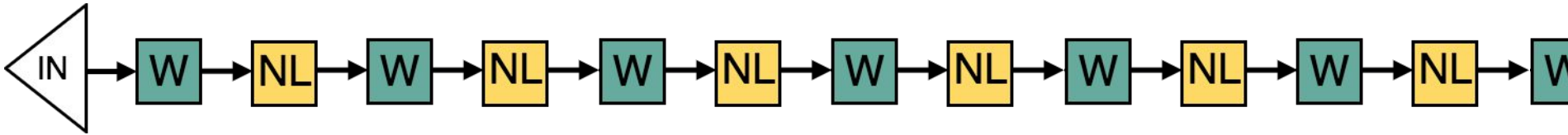


Neural networks with multiple layers can learn more complicated functions.

A brief primer on deep learning

deep learning:

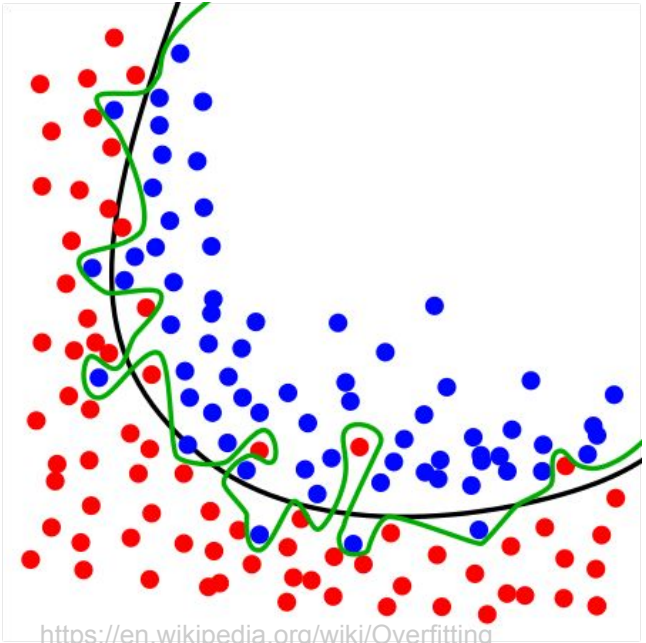
Add more layers.



A brief primer on deep learning

data:

Want lots of it. Model has many parameters. Don't want to easily overfit.



https://en.wikipedia.org/wiki/Overfitting

A brief primer on deep learning

cost function:

A metric to assess how well the model is performing.
The cost function is evaluated on the output of the model.
Also called the **loss** or **error**.

A brief primer on deep learning

way to update parameters:

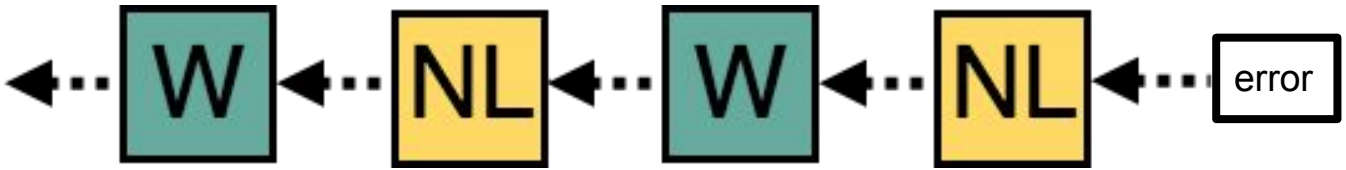
Construct a model that is differentiable

Easiest to do with differentiable programming frameworks: e.g. Torch, TensorFlow, JAX, ...

Take derivatives of the cost function (loss or error) wrt to learnable parameters.

This is called backpropogation (aka the chain rule).

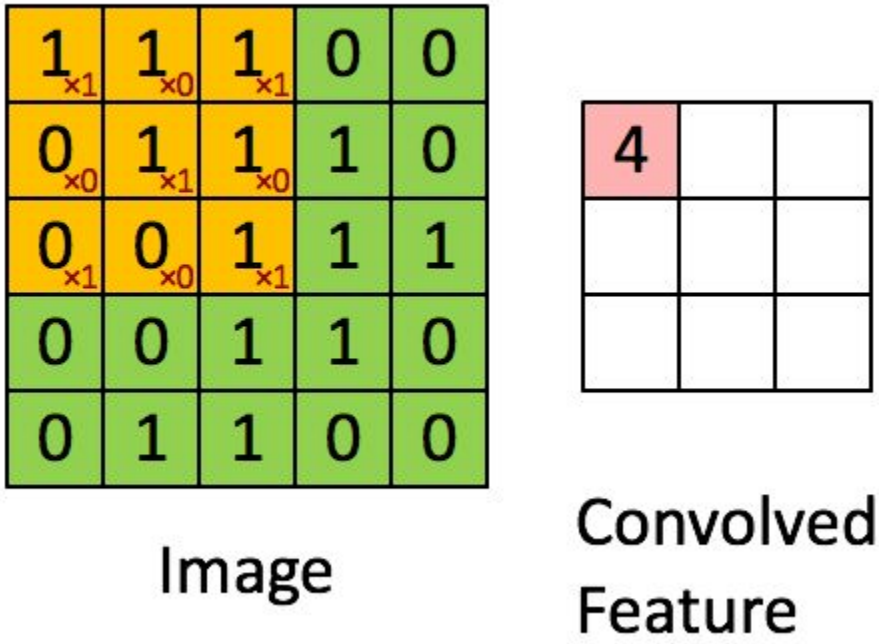
$$\Delta W_{ij} = -\eta \frac{\partial \text{ error}(f(W, x), y)}{\partial W_{ij}}$$



A brief primer on deep learning

convolutional neural networks:

Used for images. In each layer, scan over image with learned filters.



http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

convolutional neural networks:

Used for images. In each layer, scan over image with learned filters.

http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/