



Stony Brook University

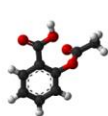
Equivariant GNNs

Wenhan Gao

Department of Applied Mathematics and Statistics

Geometric Graph Neural Networks

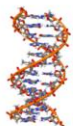
Why we care about geometric GNNs? There are many systems with geometric & relational structures.



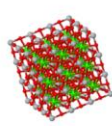
Small
Molecules



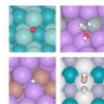
Proteins



DNA/RNA



Inorganic
Crystals



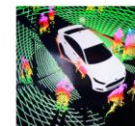
Catalysis
Systems



Transportation &
Logistics

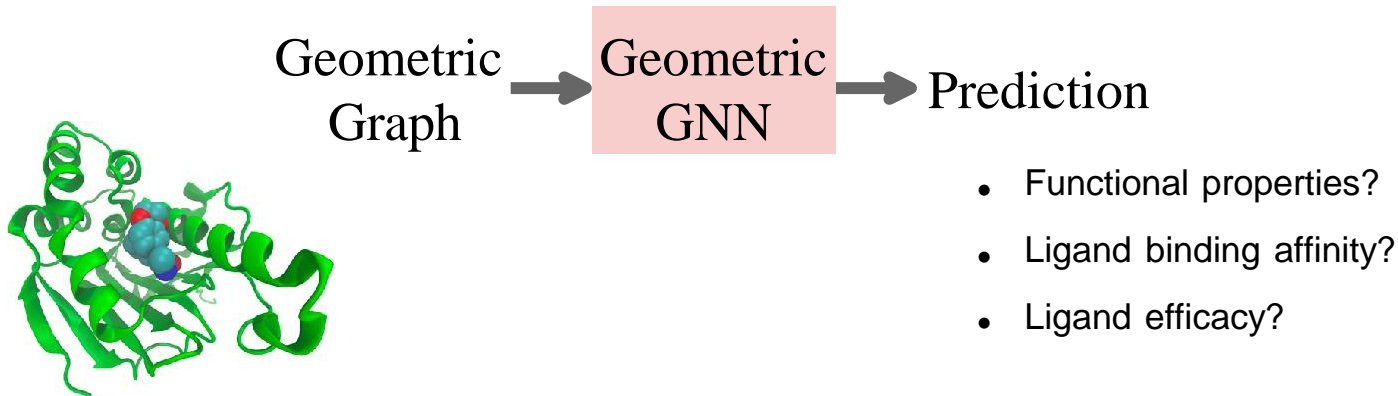


Robotic
Navigation



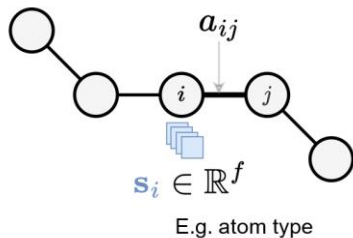
3D Computer
Vision

Geometric GNNs is a fundamental tool for machine learning on geometric (3D) graphs.



Graphs and Geometric Graphs

Graphs are purely topological objects.



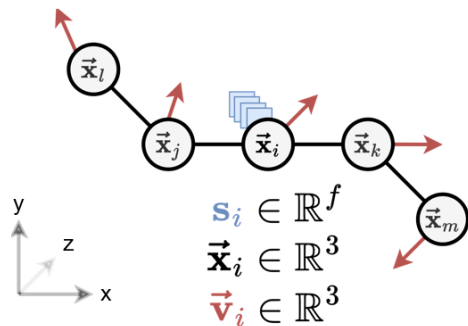
$$G = (A, S)$$

$A \in \mathbb{R}^{n \times n}$: Adjacency matrix

$S \in \mathbb{R}^{n \times a}$: Scalar features

a is the dimension or number of scalar feature channels.

Geometric graphs are a type of graphs where nodes are additionally endowed with geometric information.



$$G = (A, S, X, V)$$

$A \in \mathbb{R}^{n \times n}$: Adjacency matrix

$S \in \mathbb{R}^{n \times a}$: Scalar features

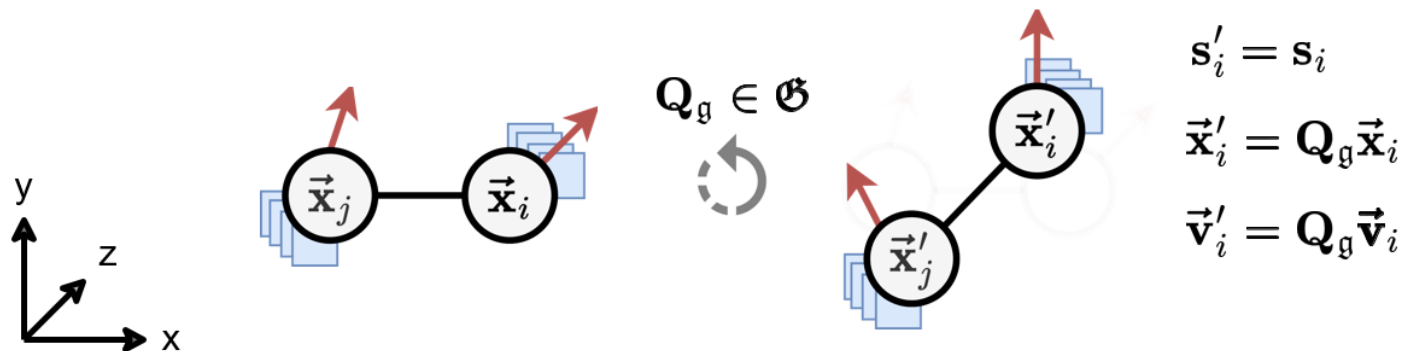
$X \in \mathbb{R}^{n \times d}$: Coordinates

$V \in \mathbb{R}^{n \times b \times d}$: Geometric features

b is the dimension or number of geometric feature channels.

Physical Symmetries

We have two types of features: **Scalar features** and **Geometric features**. Geometric features transform with **Euclidean transformations** of the system (equivariance); Scalar features remain unchanged (invariance).



Review: Equivariance and Invariance

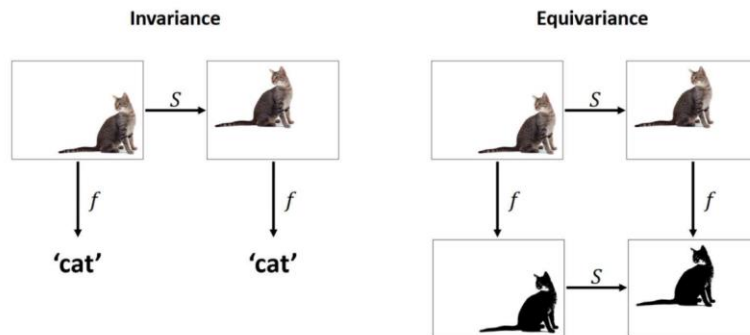
Equivariance is a property of an operator $\Phi : X \rightarrow Y$ (such as a neural network layer) by which it commutes with the group action:

$$\Phi \circ \rho^X(g) = \rho^Y(g) \circ \Phi,$$

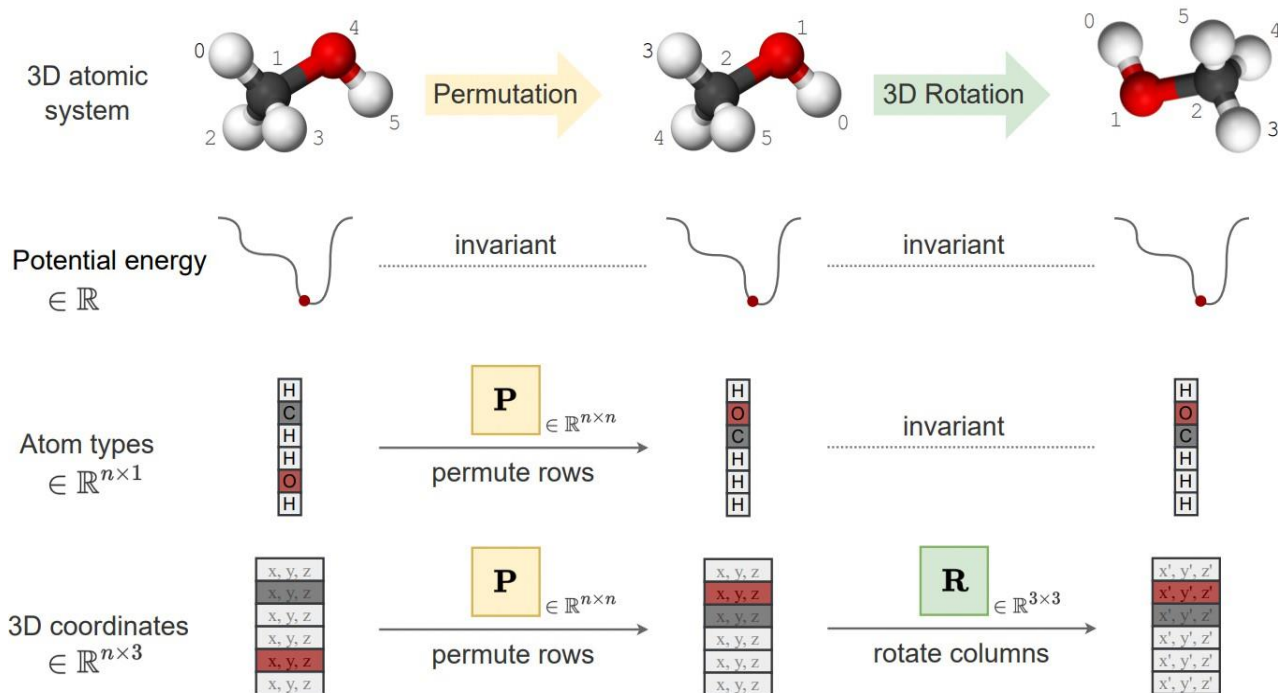
Invariance is a property of an operator $\Phi : X \rightarrow Y$ (such as a neural network layer) by which it remains unchanged after the group action:

$$\Phi \circ \rho^X(g) = \Phi,$$

- $\rho^X(g)$: group representation action on X
- $\rho^Y(g)$: group representation action on Y
- Invariance is a special case of equivariance when $\rho^Y(g)$ is the identity.

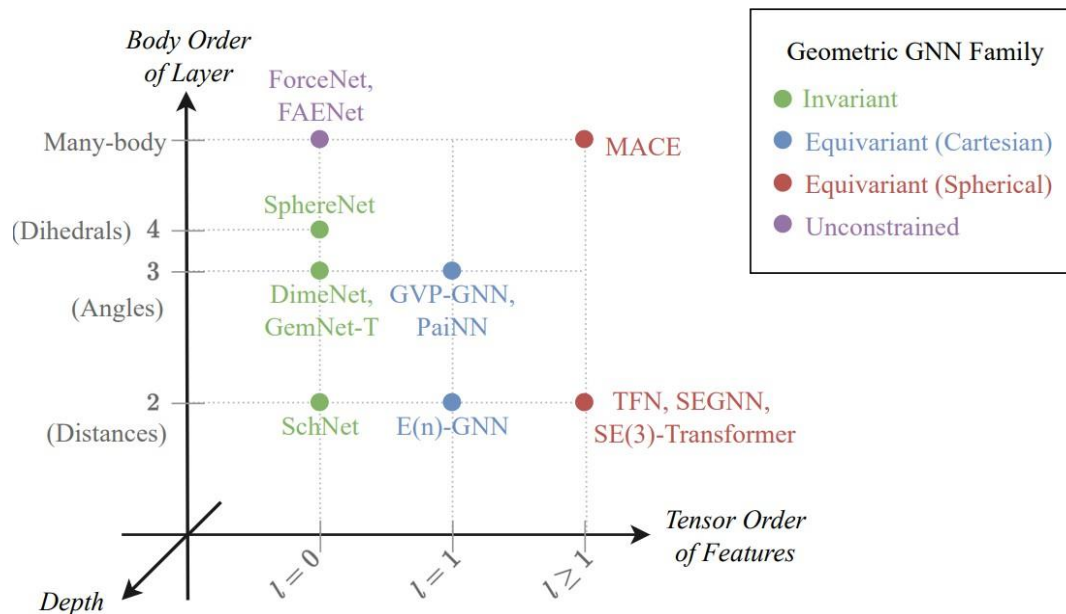


Geometric GNNs should account for physical symmetries

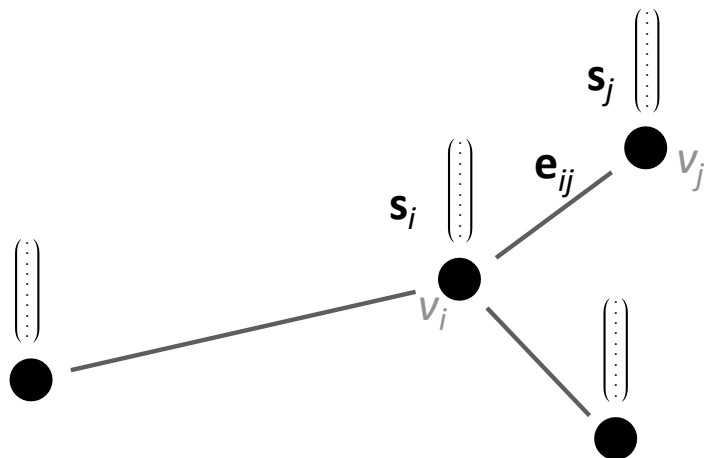


Building blocks of Geometric GNNs

- **Scalar features** must be updated in an invariant manner.
- **Geometric features** must be updated in an equivariant manner.



Review: The Message Passing Framework



$$G = (A, S, E)$$

$A \in \mathbb{R}^{n \times n}$: Adjacency matrix

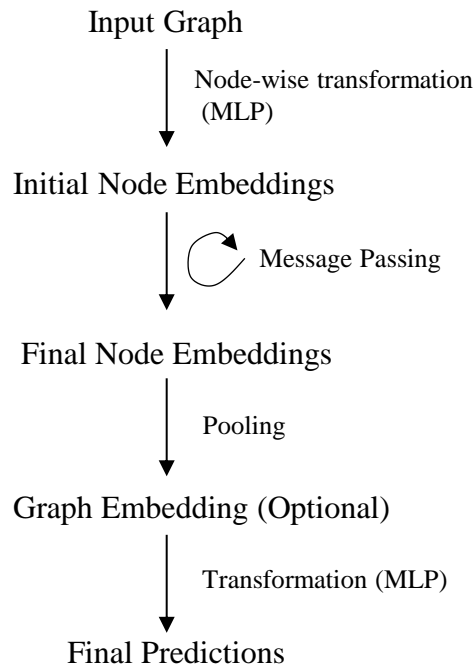
$S \in \mathbb{R}^{n \times a}$: Node features

$E \in \mathbb{R}^{n \times n \times b}$: Edges features

Goal of Message Passing: iteratively update node features to obtain useful hidden representations



Review: The Message Passing Framework



Message passing layer:

- Messages

$$\mathbf{m}_{ij} = f_1(\mathbf{s}_i, \mathbf{s}_j, \mathbf{a}_{ij}),$$

- Aggregate + node updates

$$\mathbf{s}'_i := f_2\left(\mathbf{s}_i, \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij}\right),$$

or equivalently,

$$\mathbf{s}'_i := f_2\left(\mathbf{s}_i, \sum_{j \in \mathcal{N}(i)} f_1(\mathbf{s}_i, \mathbf{s}_j, \mathbf{a}_{ij})\right),$$

where f_1, f_2 are message and updating functions (MLPs).

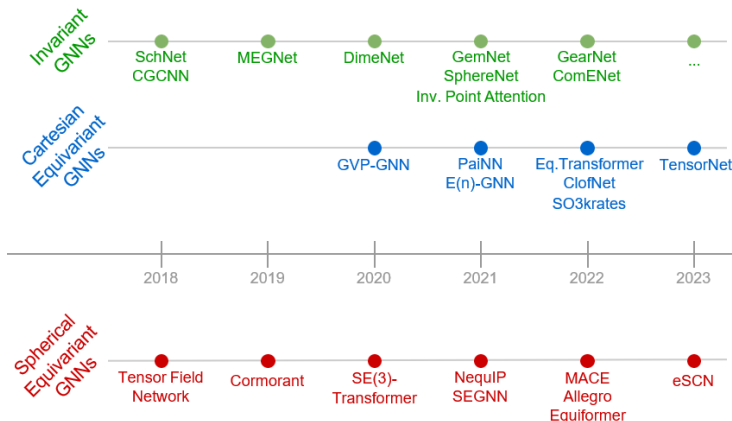
$$\mathbf{s}'_i := f_2\left(\mathbf{s}_i, \sum_{j \in \mathcal{N}(i)} f_1(\mathbf{s}_i, \mathbf{s}_j, \mathbf{a}_{ij})\right),$$

Geometric Message Passing

For geometric message passing, we condition on geometries. As an illustrative example, assume we have the coordinate information and let a_{ij} contain geometric information, we can have the following message passing schemes:

$$\mathbf{m}_{ij} = f_1(\mathbf{s}_i, \mathbf{s}_j, x_j - x_i)$$

To make it equivariant (invariant) to E(3), there are in general two directions: **Scalarization** and **Using Steerable Tensor Features**. We term them as **invariant GNNs** (scalarization) and **equivariant GNNs** (Tensor Operations). Invariant GNNs constraint the geometric information that can be utilized, while the other constraints the model operations.

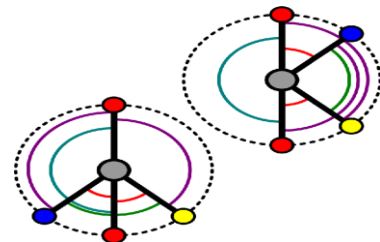


Invariant GNNs

1. Using relative distances (1-hop; body order 2): $\mathbf{m}_{ij} = f_1(s_i, s_j, d_{ij})$, where $d_{ij} = \|x_j - x_i\|$. This is $E(3)$ -equivariant, but we limit the expressivity of the model as we cannot distinguish different local geometries. We cannot distinguish two local neighbourhoods apart using the unordered set of distances only.
2. Using relative distances and bond angles (2-hop; body order 3): $\mathbf{m}_{ij} = f_1\left(s_i, s_j, d_{ij}, \sum_{k \in \mathcal{N}_j \setminus \{i\}} f_3(s_j, s_k, d_{ij}, d_{jk}, \angle ijk)\right)$. This is $E(3)$ -equivariant, but again we limit the expressivity of the model.
3. Using relative distances, bond angles, and torsion angles (3-hop; body order 4):

$$\mathbf{m}_{ij} = f_1\left(s_i, s_j, d_{ij}, \sum_{k \in \mathcal{N}_j \setminus \{i\}, l \in \mathcal{N}_k \setminus \{i, j\}} f_3(s_k, s_l, d_{kl}, d_{ij}, d_{jk}, \angle ijk, \angle jkl, \angle ijl)\right).$$

This is $E(3)$ -equivariant and complete.



Invariant GNNs

In summary, invariant GNNs update latent representations by scalarizing local geometry information. This is efficient, and we can achieve invariance with simple MLP without specific constraints on the operations or activations we can take.

Pros:

- Simple usage of non-linearities on many-body scalars.
- Great performance on some use-cases (e.g. GemNet on OC20).

Cons:

- Scalability of scalar's pre-computation. The accounting of higher-order tuples is expensive.
- Making invariant predictions may still require solving equivariant sub-tasks.
- May lack generalization capabilities (equivariant tasks, multi-domain).

GemNet (Torsion) in theory is complete; however, it requires a complete graph and a certain discretization scheme to be universal.

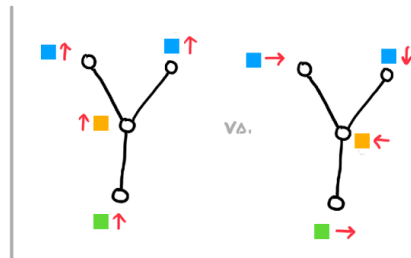
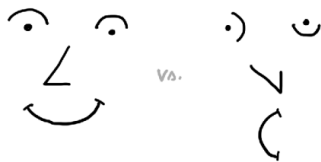
So far, the precise body order of scalars at which all geometric graphs can be uniquely identified remains an open question.

Equivariant GNNs

- ❑ In invariant GNNs, invariants are ‘fixed’ prior to message passing.
- ❑ Equivariant GNNs build up invariants ‘on the go’ during message passing. More layers of message passing can lead to more complex invariants being built up. Furthermore, equivariant quantities remain available.

Intuition: The Picasso Problem

Making invariant predictions may still require solving equivariant sub-tasks



Relative orientation of eyes, nose, mouth is important
(**orientation of sub-graphs w.r.t. one another**),
not just their presence (**invariant quantities**)!



Equivariant GNNs

- In invariant GNNs, we work with only scalars $f(s_1, s_2, \dots, s_n)$.
- In equivariant GNNs, we work with vectors $f(s_1, s_2, \dots, s_n, v_1, \dots, v_m)$.

Example: “Scalar-vector” GNNs

Scalar message:

$$\mathbf{m}_i := f_1(s_i, \|\mathbf{v}_i\|) + \sum_{j \in \mathcal{N}_i} f_2(s_i, s_j, \|\vec{x}_{ij}\|, \|\mathbf{v}_j\|, \vec{x}_{ij} \cdot \mathbf{v}_j, \vec{x}_{ij} \cdot \mathbf{v}_i, \mathbf{v}_i \cdot \mathbf{v}_j)$$

Vector message:

$$\begin{aligned} \vec{\mathbf{m}}_i := & f_3(s_i, \|\mathbf{v}_i\|) \odot \mathbf{v}_i + \sum_{j \in \mathcal{N}_i} f_4(s_i, s_j, \|\vec{x}_{ij}\|, \|\mathbf{v}_j\|, \vec{x}_{ij} \cdot \mathbf{v}_j, \vec{x}_{ij} \cdot \mathbf{v}_i, \mathbf{v}_i \cdot \mathbf{v}_j) \odot \mathbf{v}_j \\ & + \sum_{j \in \mathcal{N}_i} f_5(s_i, s_j, \|\vec{x}_{ij}\|, \|\mathbf{v}_j\|, \vec{x}_{ij} \cdot \mathbf{v}_j, \vec{x}_{ij} \cdot \mathbf{v}_i, \mathbf{v}_i \cdot \mathbf{v}_j) \odot \vec{x}_{ij}, \end{aligned}$$



Equivariant GNNs

The high-level ideas in equivariant GNNs is that we keep track of the “types” of the objects and apply equivariant operations. Here, we introduce some concepts, Cartesian tensors, Spherical tensors, tensor product, Wigner-D matrices, and CG coefficients.

A tensor is a multi-dimensional array with directional information.

We have two key operators for Cartesian tensors: tensor product and tensor contraction. They allow us to move up and down the rank of Cartesian tensors to produce tensors of higher ranks or contract them down to lower ranks. Intuitively, we are creating new tensors using the indices.



Equivariant GNNs

We get two vectors in 3D space

```
▶ irreps_x = o3.Irreps('1o')  
  irreps_y = o3.Irreps('1e')
```

```
x = irreps_x.randn(-1)  
y = irreps_y.randn(-1)  
print(x)  
print(y)
```

```
⇒ tensor([-0.1222, -0.7573,  1.0416])  
   tensor([ 0.3303,  0.5315, -1.0235])
```


Equivariant GNNs

We get their outer, inner, and cross products.

```
[12] outter = torch.einsum('i,j', x, y)
      inner = torch.einsum('i,i', x, y)
      cross = torch.linalg.cross(x, y)
      print("=====  
Outter, Inner, Cross =====")
      print(outter)
      print(inner)
      print(cross)
```

```
⇒ ===== Outter, Inner, Cross =====
tensor([[ -0.0404, -0.0650,  0.1251],
        [ -0.2501, -0.4025,  0.7750],
        [  0.3440,  0.5536, -1.0661]])
tensor(-1.5089)
tensor([0.2214, 0.2189, 0.1851])
```



Equivariant GNNs

We get a random rotation matrix in $O(3)$ and rotate x, y and get the outer, inner, and cross

```
[13] R = o3.rand_matrix()
print(R)
r_x = torch.einsum('ij,j', R, x)
r_y = torch.einsum('ij,j', R, y)
r_outter = torch.einsum('i,j', r_x, r_y)
r_inner = torch.einsum('i,i', r_x, r_y)
r_cross = torch.linalg.cross(r_x, r_y)
print("=====  
Outter, Inner, Cross after Rotation =====")
print(r_outter)
print(r_inner)
print(r_cross)
```



```
print(r_x)
print(r_y)
```



```
tensor([0.7971, 0.0112, 1.0188])
tensor([-0.5431, 0.1576, -1.0580])
```



```
tensor([[ 0.1437, -0.9875, 0.0642],
        [ 0.9593, 0.1549, 0.2360],
        [-0.2430, 0.0276, 0.9696]])
=====  
Outter, Inner, Cross after Rotation =====
tensor([[ -0.4329, 0.1256, -0.8433],
        [-0.0061, 0.0018, -0.0119],
        [-0.5532, 0.1606, -1.0778]])
tensor(-1.5089)
tensor([-0.1725, 0.2900, 0.1317])
```

Equivariant GNNs

All of them transform in a certain way, or, all of these operations are equivariant. Outer product will transform with the "outer product" of the rotation matrices (i.e. Kronecker product if we flatten the resulting outer product of the vectors). Inner product are invariant. Cross product will transform with the original rotation matrix.

```

▶ R_outter = torch.einsum('ij,kl', R, R)
  rotate_outter = torch.einsum('ijkl,jl', R_outter, outter)
  rotate_inner = inner
  rotate_cross = torch.einsum('ij,j', R, cross)
  print("==== Rotate Outter, Inner, Cross =====")
  print(rotate_outter)
  print(rotate_inner)
  print(rotate_cross)

```

```

===== Outter, Inner, Cross after Rotation =====
tensor([[ -0.4329,  0.1256, -0.8433],
        [ -0.0061,  0.0018, -0.0119],
        [ -0.5532,  0.1606, -1.0778]])
tensor(-1.5089)
tensor([ -0.1725,  0.2900,  0.1317])

```

```

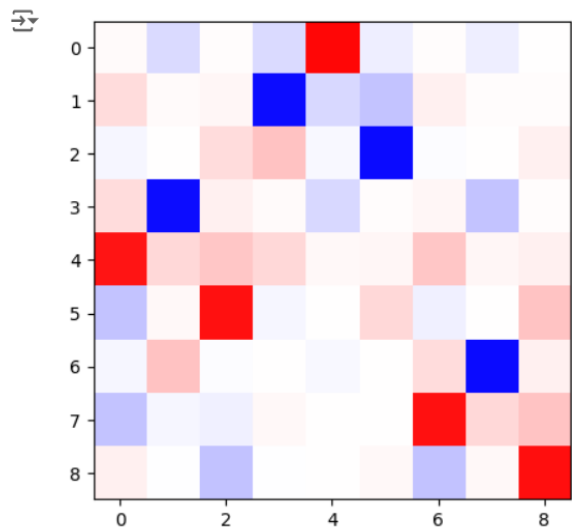
⇨ ===== Rotate Outter, Inner, Cross =====
tensor([[ -0.4329,  0.1256, -0.8433],
        [ -0.0061,  0.0018, -0.0119],
        [ -0.5532,  0.1606, -1.0778]])
tensor(-1.5089)
tensor([ -0.1725,  0.2900,  0.1317])

```

Equivariant GNNs

Visualize this rotation tensor for the outer product (a $3 \times 3 \times 3 \times 3$ tensor, we visualize as a 9 by 9 matrix)

```
plt.imshow(torch.kron(R, R), cmap='bwr', vmin=-1, vmax=1);
```





Equivariant GNNs

Now, for the outter product and the rotated one, we do a change of basis, and represent them in spherical basis.

```
[17] tp = o3.FullTensorProduct(irreps_x, irreps_y)
      # print(tp)
      irreps_xy = tp(x, y)
      irreps_rxy = tp(r_x, r_y)
      print("==== Spherical Outter =====")
      print(irreps_xy )
      print("==== Spherical Rotated Outter =====")
      print(irreps_rxy)
```



```
==== Spherical Outter =====
tensor([[-0.8712,  0.1566,  0.1548,  0.1309,  0.3317, -0.2228,  0.1231,  0.9395,
         -0.7253]])
==== Spherical Rotated Outter =====
tensor([[-0.8712, -0.1220,  0.2051,  0.0932, -0.9875,  0.0845,  0.6182,  0.1052,
         -0.4561]])
```



Equivariant GNNs

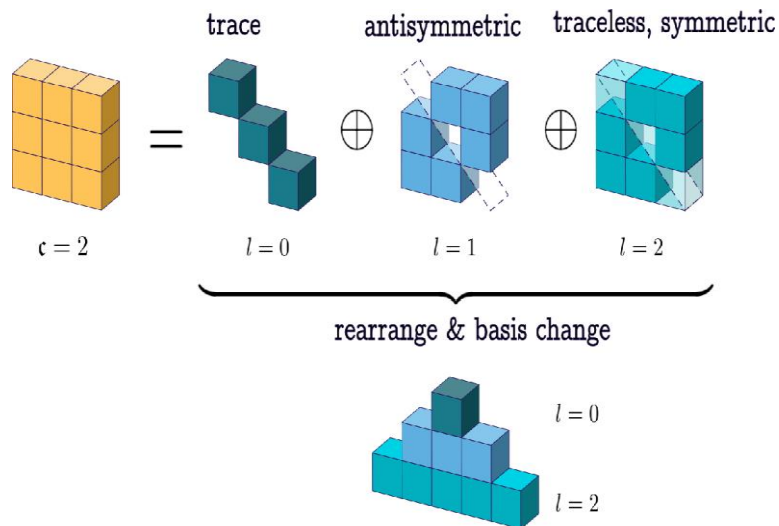
Now, for the outer product and the rotated one, we do a change of basis, and represent them in spherical basis.

```
[17] tp = o3.FullTensorProduct(irreps_x, irreps_y)
      # print(tp)
      irreps_xy = tp(x, y)
      irreps_rxy = tp(r_x, r_y)
      print("==== Spherical Outter =====")
      print(irreps_xy )
      print("==== Spherical Rotated Outter =====")
      print(irreps_rxy)
```



```
==== Spherical Outter =====
tensor([[-0.8712,  0.1566,  0.1548,  0.1309,  0.3317, -0.2228,  0.1231,  0.9395,
         -0.7253]])
==== Spherical Rotated Outter =====
tensor([[-0.8712, -0.1220,  0.2051,  0.0932, -0.9875,  0.0845,  0.6182,  0.1052,
         -0.4561]])
```

Equivariant GNNs



Equivariant GNNs

This spherical outter product is also equivariant, and the transformation matrix is the Wigner-D matrix

```
[18] D = tp.irreps_out.D_from_matrix(R)
      print("=====  
Rotate Spherical Outter by Wigner-D  
=====")
      irreps_rotate_xy = torch.einsum('ij,j', D, irreps_xy)
      print(irreps_rotate_xy)
```

```
⇒ ===== Rotate Spherical Outter by Wigner-D =====
   tensor([-0.8712, -0.1220,  0.2051,  0.0932, -0.9875,  0.0845,  0.6182,  0.1052,  
          -0.4561])
```


Equivariant GNNs

Visualize this rotation tensor for the spherical outter product:

```
plt.imshow(D, cmap='bwr', vmin=-1, vmax=1);
```

