ML Journal Club

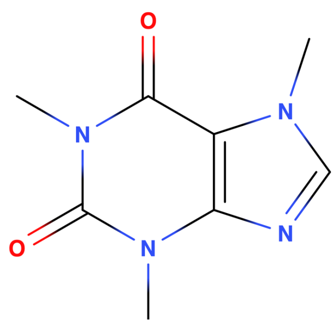
Graph Neural Networks 5/30

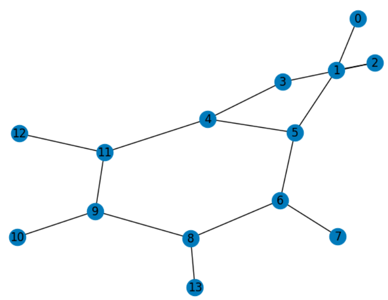
Content

- Topological Graph Neural Network
  - Define a graph
    - Topological graph
    - Geometrical graph
  - Graph convolutional network (GCN)
  - General architectures
- Geometrical Graph Neural Networks
- DEMO

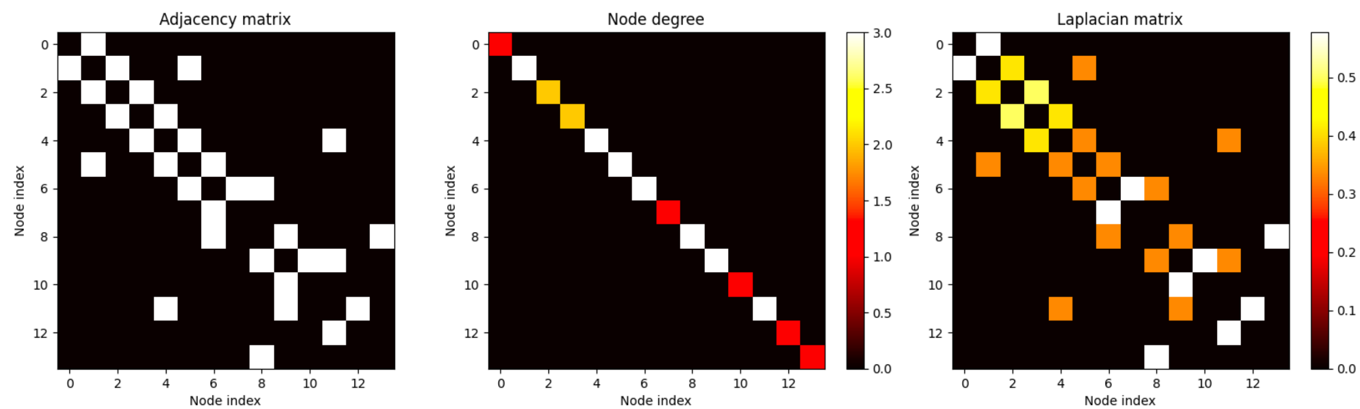
Basics

How to define a graph





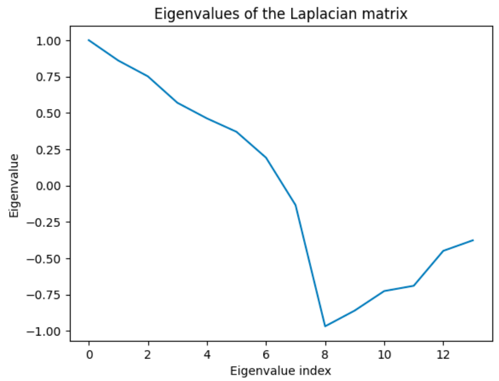
1,3,7-trimethylpurine-2,6-dione

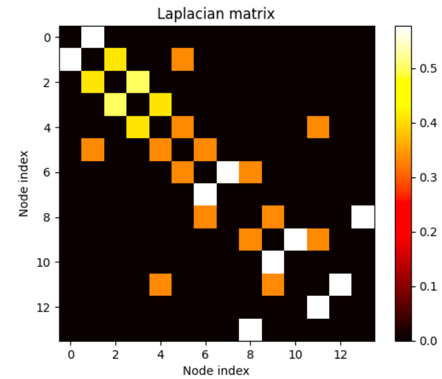


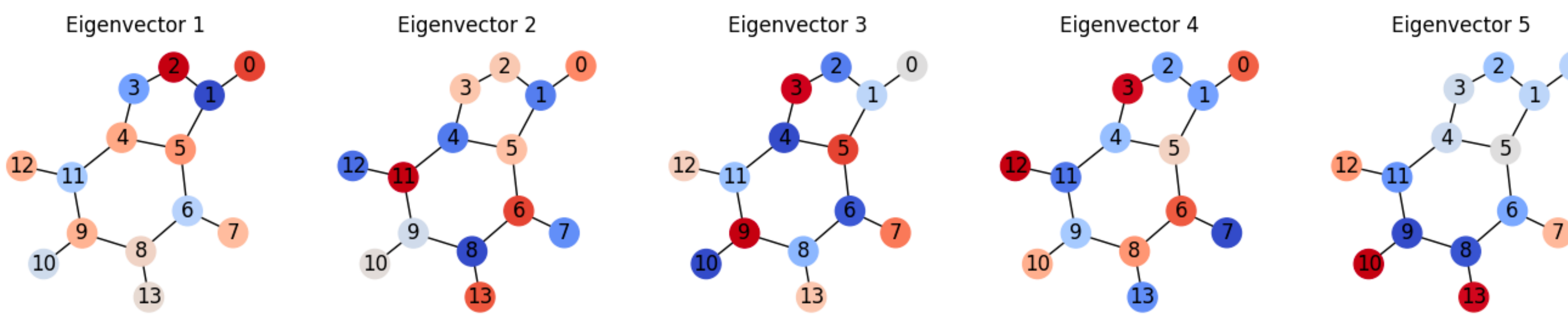
$$L = I - A$$

$$L_{sym} = I - D^{1/2} A D^{-1/2}$$

Basics
Eigen-decomposition of graph Laplacian







Basics

Eigen-decomposition of graph Laplacian

```
class LapTransform:
    def __call__(self,data):

        num_nodes = data.num_nodes
        edge_index, edge_weight = get_laplacian(
        data.edge_index,
        data.edge_weight,
        normalization='sym',
        num_nodes=num_nodes,
    )
        L = to_scipy_sparse_matrix(edge_index, edge_weight, num_nodes)
        eig_vals, eig_vecs = np.linalg.eigh(L.todense())
        eig_vecs = np.real(eig_vecs[:, eig_vals.argsort()])
        pe = torch.from_numpy(eig_vecs[:, 1:num_lap_vecs + 1])

        if pe.shape[1] < num_lap_vecs:
          pe = torch.nn.functional.pad(pe, (0, num_lap_vecs - pe.shape[1]), value=float(0))

        data = add_node_attr(data, pe, attr_name='pe')
        return data
```

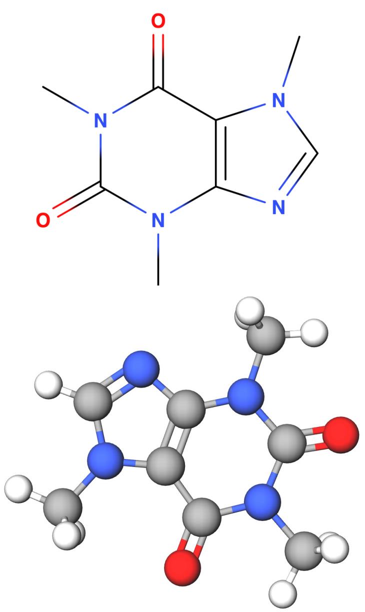
Basics
Eigen-decomposition of graph Laplacian

```
def forward(self,data):
    x = data.x
    edge_index = data.edge_index
    pe = data.pe


    x = torch.cat([x,pe],dim=-1)

    x = self.conv1(x,edge_index)
    x = self.activation(x)
    x = self.conv2(x,edge_index)
    x = self.dropout(x)
    x = scatter(x,data.batch,dim=0,reduce='sum')

    return x
```

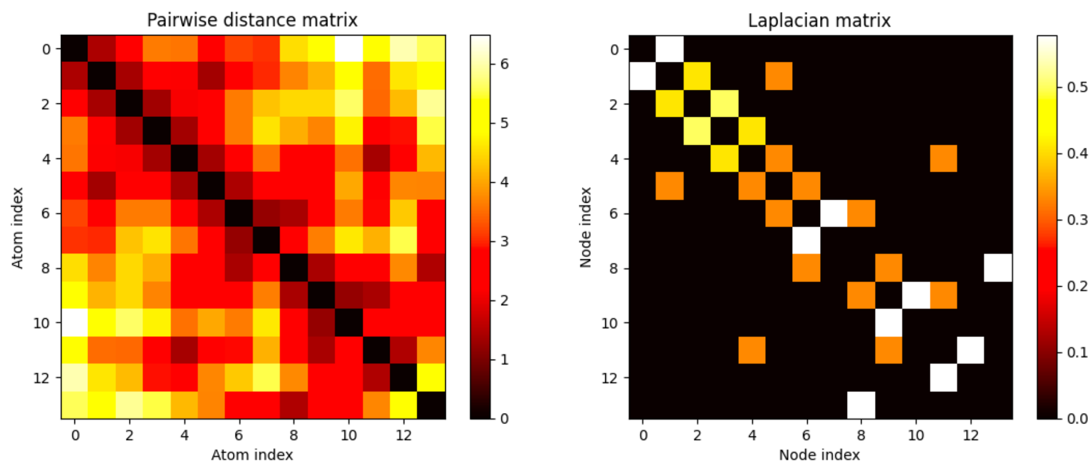
```
def forward(self,data):

    x = self.mp_block(data)
    graph_readout = scatter(x,data.batch,reduce='sum',dim=0)

    src, mask = to_dense_batch(x,data.batch)
    pe, _ = to_dense_batch(data.pe,data.batch)

    src = torch.cat([src,pe],dim=-1)

    if self.use_cls:
        cls_token = self.cls_token.expand(src.shape[0], -1, -1)
        src = torch.cat([cls_token, src], dim=1)
        mask = torch.cat([torch.ones(src.shape[0], 1).bool().to(s
        output = self.encoder(src, src_key_padding_mask=~mask)
        output = output[:, 0, :]
```

Use as the positional encoding for GNN and GT

Basics
How to use conformational information?

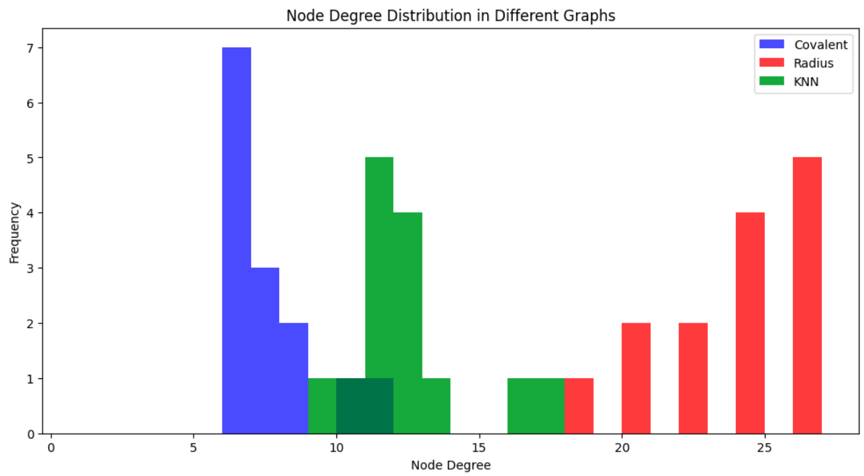


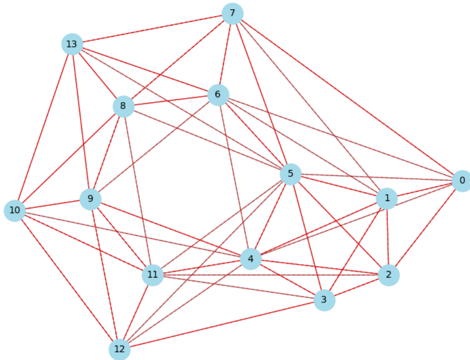
1,3,7-trimethylpurine-2,6-dione



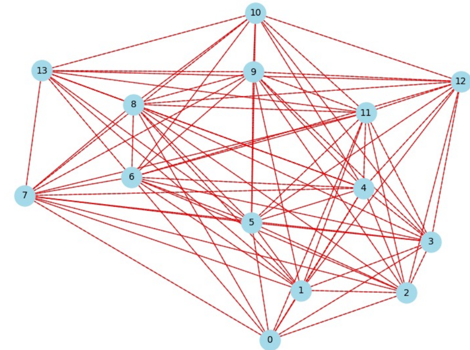
Basics

How to define connectivity in dynamics?

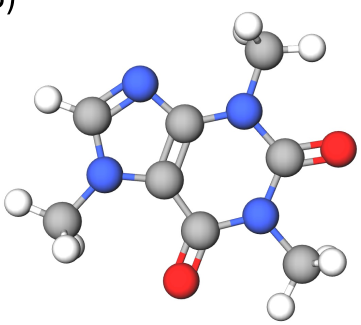




K-nearest neighbors (K=6)



Radius cutoff (R=5 Ang)

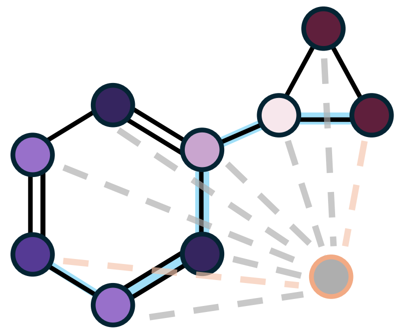


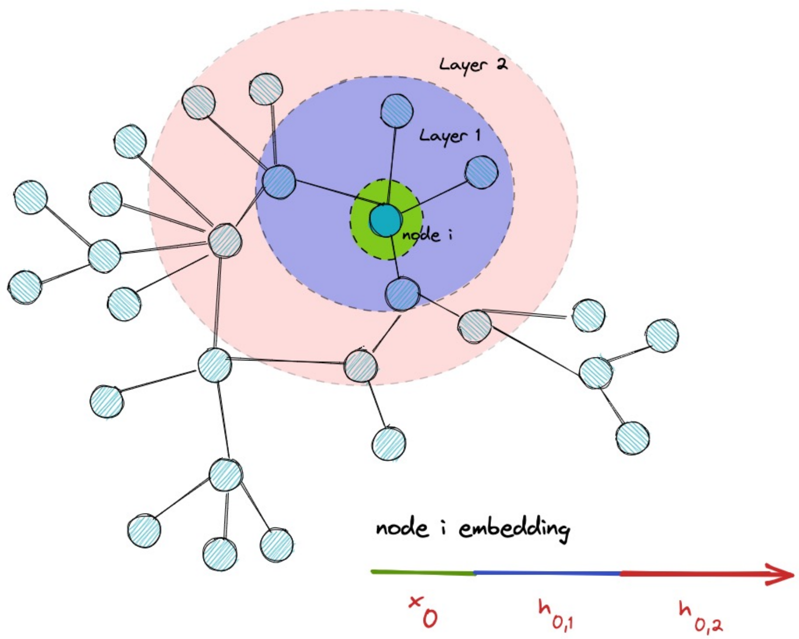
Covalent radii

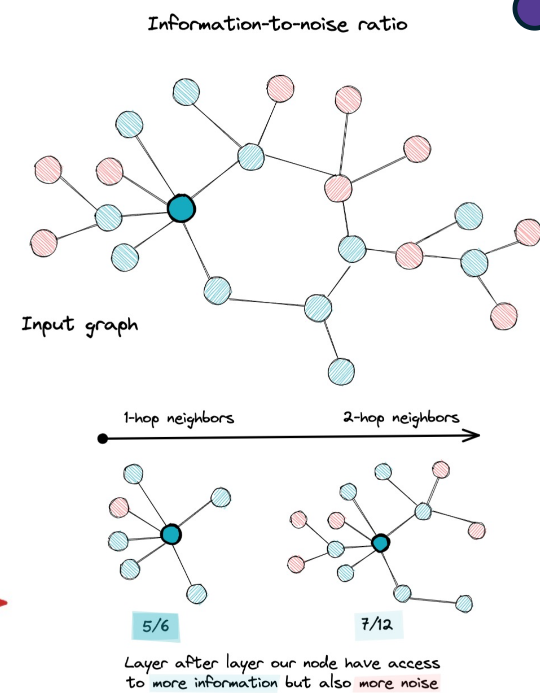
THE UNIVERSITY OF CHICAGO

Message-Passing Neural Networks

Mechanism and virtual node







Graph Convolutional Network (GCN)

Abbreviation to the graph spectral convolution

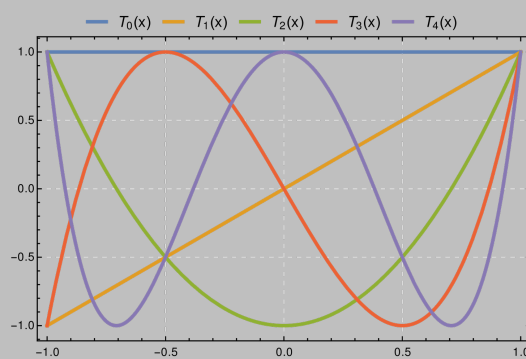
$$g_\theta x = U g_\theta U^T x \; (Graph\ Spectral\ Conv.)$$
$$g_\theta = \mathrm{diag}(\theta) \text{ in Fourier domain}$$
$$L = I - D^{\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^T$$
$$g_\theta(\Lambda) \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{\Lambda})$$
Insert back,
$$g_\theta x \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{L}) x \, , \tilde{L} = \frac{2}{\lambda_{max}} L - I$$
Truncate at order 1, *assume* $\lambda_{max} = 2$
$$g_\theta x \approx \theta_0 x + \theta_1 (L - I) x$$

$$Z = f(X, A) = \mathrm{softmax}\left(\hat{A}\ \mathrm{ReLU}\left(\hat{A} X W^{(0)}\right) W^{(1)}\right)$$

adj. mat.

Chebyshev polynomials

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_{n+1}(x) = 2x T_n(x) - T_{n-1}(x)$$



Graph Convolutional Network (GCN)

Abbreviation to the graph spectral convolution

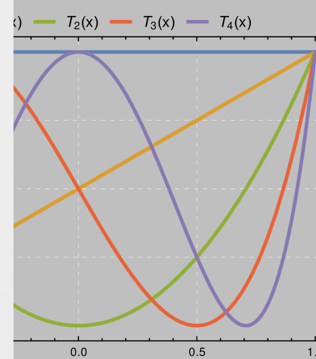
```
    x = self.lin(x)

    # propagate_type: (x: Tensor, edge_weight: OptTensor)
    out = self.propagate(edge_index, x=x, edge_weight=edge_weight)

    if self.bias is not None:
        out = out + self.bias

    return out


def message(self, x_j: Tensor, edge_weight: OptTensor) -> Tensor:
    return x_j if edge_weight is None else edge_weight.view(-1, 1) * x_j

def message_and_aggregate(self, adj_t: Adj, x: Tensor) -> Tensor:
    return spmm(adj_t, x, reduce=self.aggr)
```

ev polynomials

$_0(x) = 1$

$_1(x) = x$

$2xT_n(x) - T_{n-1}(x)$

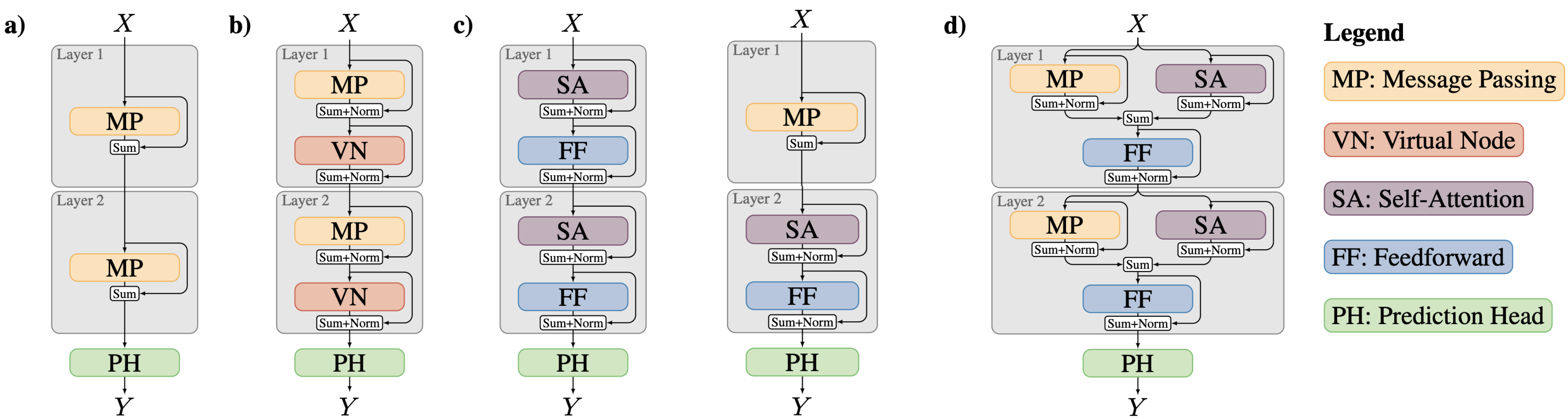


$$Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A}XW^{(0)}\Big)\,W^{(1)}\Big)$$

adj. mat.

Architectural design space of GNNs

MP and Graph Transformers



GCN/GAT/GIN…

GCN+VN
/GAT+VN
/GIN+VN
…

GT
SAN
GRIT

GraphTrans
SubFormer

GraphGPS
Exphormer
Etc.

Fundamental symmetries

Trans/Roto/Perm. Invariance/equivariance

- Translation: $T(d) = \{w \in \mathbb{R}^d\}$
  - Can be achieved by using relative displacement
- Rotation: $\mathrm{SO}(d) = \{Q \in \mathbb{R}^{d\times d}: Q^T Q = QQ^T = I_d, \det(Q) = 1\}$
  - Can be achieved by using scaler features (bond length, angle, etc.), vector features (relative displacement), and irreducible representations (spherical harmonics).
- Permutation: $S_n = \{\sigma: [n] \to [n]\ bijective\}$
  - MPNN/Transformer
- Invariance: $(f(gx)) = f(x)$
  - Energy
- Equivariance: $(f(gx)) \ = gf(x)$
  - Force, velocity, etc.

E(n)-GNN
Satorras et al. 2021

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

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

E(n)-GNN
Forward pass

EGNN
$\mathbf{m}_{ij} = \phi_e(\mathbf{h}_i^l, \mathbf{h}_j^l, \lVert \mathbf{r}_{ij}^l \rVert^2, a_{ij})$ $\hat{\mathbf{m}}_{ij} = \mathbf{r}_{ij}^l \phi_x(\mathbf{m}_{ij})$
$\mathbf{m}_i = \sum_{j \neq i} \mathbf{m}_{ij}$ $\hat{\mathbf{m}}_i = C \sum_{j \neq i} \hat{\mathbf{m}}_{ij}$
$\mathbf{h}_i^{l+1} = \phi_h\left(\mathbf{h}_i^l, \mathbf{m}_i\right)$ $\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \hat{\mathbf{m}}_i$

```python
def coord2radial(self, edge_index, coord):
    row, col = edge_index
    coord_diff = coord[row] - coord[col]
    radial = torch.sum(coord_diff**2, 1).unsqueeze(1)

    if self.normalize:
        norm = torch.sqrt(radial).detach() + self.epsilon
        coord_diff = coord_diff / norm

    return radial, coord_diff

def forward(self, h, edge_index, coord, edge_attr=None, node_attr=None):
    row, col = edge_index
    radial, coord_diff = self.coord2radial(edge_index, coord)

    edge_feat = self.edge_model(h[row], h[col], radial, edge_attr)
    coord = self.coord_model(coord, edge_index, coord_diff, edge_feat)
    h, agg = self.node_model(h, edge_index, edge_feat, node_attr)

    return h, coord, edge_attr
```

E(n)-GNN
Node model

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

```
self.node_mlp = nn.Sequential(
    nn.Linear(hidden_nf + input_nf, hidden_nf),
    act_fn,
    nn.Linear(hidden_nf, output_nf))

def node_model(self, x, edge_index, edge_attr, node_attr):
    row, col = edge_index
    agg = unsorted_segment_sum(edge_attr, row, num_segments=x.size(0))
    if node_attr is not None:
        agg = torch.cat([x, agg, node_attr], dim=1)
    else:
        agg = torch.cat([x, agg], dim=1)
    out = self.node_mlp(agg)
    if self.residual:
        out = x + out
    return out, agg
```

E(n)-GNN
Edge model (scaler)

$$\mathbf{m}_{ij} = \phi_e(\mathbf{h}_i^l, \mathbf{h}_j^l, \|\mathbf{r}_{ij}^l\|^2, a_{ij})$$

```
self.edge_mlp = nn.Sequential(
    nn.Linear(input_edge + edge_coords_nf + edges_in_d, hidden_nf),
    act_fn,
    nn.Linear(hidden_nf, hidden_nf),
    act_fn)

def edge_model(self, source, target, radial, edge_attr):
    if edge_attr is None:  # Unused.
        out = torch.cat([source, target, radial], dim=1)
    else:
        out = torch.cat([source, target, radial, edge_attr], dim=1)
    out = self.edge_mlp(out)
    if self.attention:
        att_val = self.att_mlp(out)
        out = out * att_val
    return out
```

E(n)-GNN

Edge model (Vector)

$$\hat{\mathbf{m}}_{ij} = \mathbf{r}^l_{ij} \phi_x(\mathbf{m}_{ij})$$

```
coord_mlp = []
coord_mlp.append(nn.Linear(hidden_nf, hidden_nf))
coord_mlp.append(act_fn)
coord_mlp.append(layer)
if self.tanh:
    coord_mlp.append(nn.Tanh())
self.coord_mlp = nn.Sequential(*coord_mlp)

if self.attention:
    self.att_mlp = nn.Sequential(
        nn.Linear(hidden_nf, 1),
        nn.Sigmoid())
```

```
def coord_model(self, coord, edge_index, coord_diff, edge_feat):
    row, col = edge_index
    trans = coord_diff * self.coord_mlp(edge_feat)
    if self.coords_agg == 'sum':
        agg = unsorted_segment_sum(trans, row, num_segments=coord.size(0))
    elif self.coords_agg == 'mean':
        agg = unsorted_segment_mean(trans, row, num_segments=coord.size(0))
    else:
        raise Exception('Wrong coords_agg parameter' % self.coords_agg)
    coord = coord + agg
    return coord
```

That's being saying, scaler * vector is still a vector, as long as you update the vector 'separately' on top of scaler features, your model is 'equivariant'

Group equivariant NN.

$$f(g \triangleright_X x) = g \triangleright_Y f(x) \qquad \forall\, g \in G,\ x \in X,$$

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[d, "g \triangleright_X"'] & Y \arrow[d, "g \triangleright_Y"] \\ X \arrow[r, "f"'] & Y \end{tikzcd}$$

$$\mathcal{F}_0 \xrightarrow{L_1} \mathcal{F}_1 \xrightarrow{L_2} \mathcal{F}_2 \xrightarrow{L_3} \dots \xrightarrow{L_{N-1}} \mathcal{F}_{N-1} \xrightarrow{L_N} \mathcal{F}_N$$

$$\begin{tikzcd} \mathcal{F}_0 \arrow[r, "L_1"] \arrow[d, "g \triangleright_0"'] & \mathcal{F}_1 \arrow[r, "L_2"] \arrow[d, "g \triangleright_1"'] & \mathcal{F}_2 \arrow[r, "L_3"] \arrow[d, "g \triangleright_2"'] & \dots \arrow[r, "L_{N-1}"] & \mathcal{F}_{N-1} \arrow[r, "L_N"] \arrow[d, "g \triangleright_{N-1}"'] & \mathcal{F}_N \arrow[d, "g \triangleright_N"'] \\ \mathcal{F}_0 \arrow[r, "L_1"'] & \mathcal{F}_1 \arrow[r, "L_2"'] & \mathcal{F}_2 \arrow[r, "L_3"'] & \dots \arrow[r, "L_{N-1}"'] & \mathcal{F}_{N-1} \arrow[r, "L_N"'] & \mathcal{F}_N \end{tikzcd}$$

Group function transform

Feature: $f(\vec{r}) = \vec{x}$

$$f(x, y, z) = (r, g, b)$$

A group element g can act on a function:

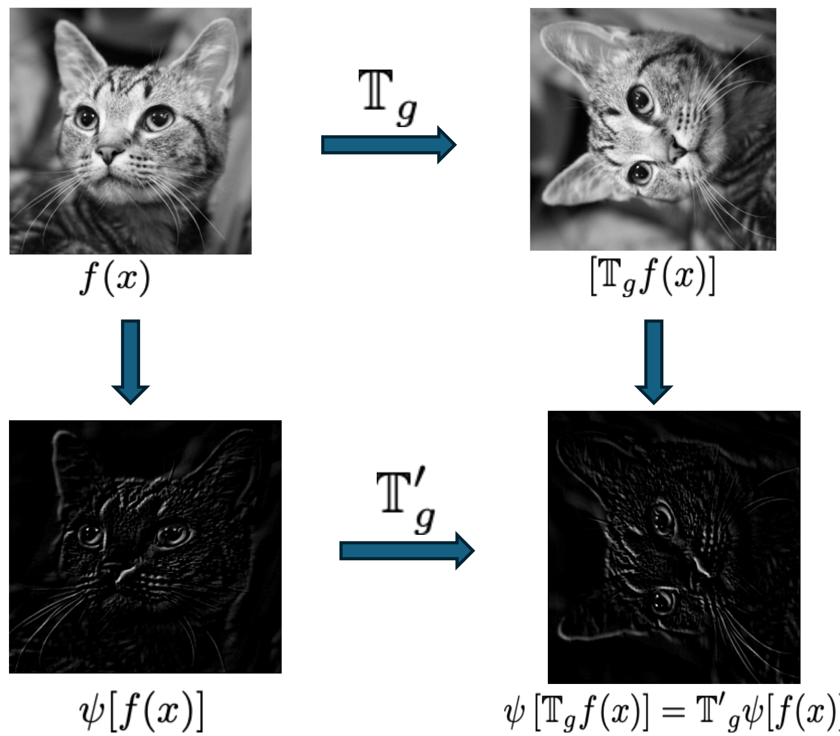
$$\mathbb{T}_g : f(\mathcal{X}) \rightarrow f'(\mathcal{X})$$

$$f'(x) = f\left(g^{-1}x\right)$$

A neural network is a map with an input of a function and an output of another function:

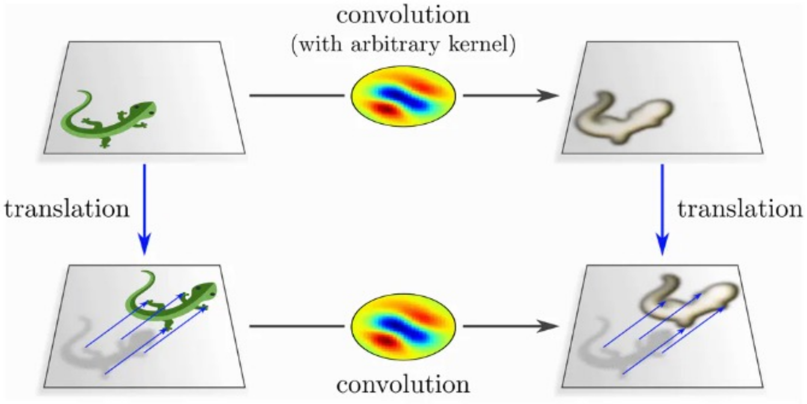
$$\psi : f\left(\mathcal{X}_1\right) \rightarrow f'\left(\mathcal{X}_2\right)$$

$$\psi\left[\mathbb{T}_g f(x)\right] = \mathbb{T}'_g \psi[f(x)] \quad \forall f(x)$$



Linear transformation in group equivariant NN.

Spatial weight sharing implies the translation equivariance of convolutional networks



$$(f * g)(x) = \int f(x-y)g(y)dy$$

A neural network layer (linear map) ψ is G-equivariant **if and only if** its form is a convolution operator:

$$\psi(f) = (f * \omega)(u) = \sum_{g \in G} f \uparrow^G (ug^{-1})\, \omega \uparrow^G (g)$$

It is complicated! It can be simplified using irreps:

$$f(g) = f_0 \cdot \rho_0(g) + \vec{f_1} \cdot \rho_1(g) + \ldots + \vec{f_k} \cdot \rho_k(g)$$

$$\psi(f) = f_0 w_0 \oplus \vec{f_1} w_1 \oplus \ldots \oplus \vec{f_k} w_k$$

For SO(3), spherical harmonics transforms in the same manner as irreps

Nonlinearity in group equivariant NN.

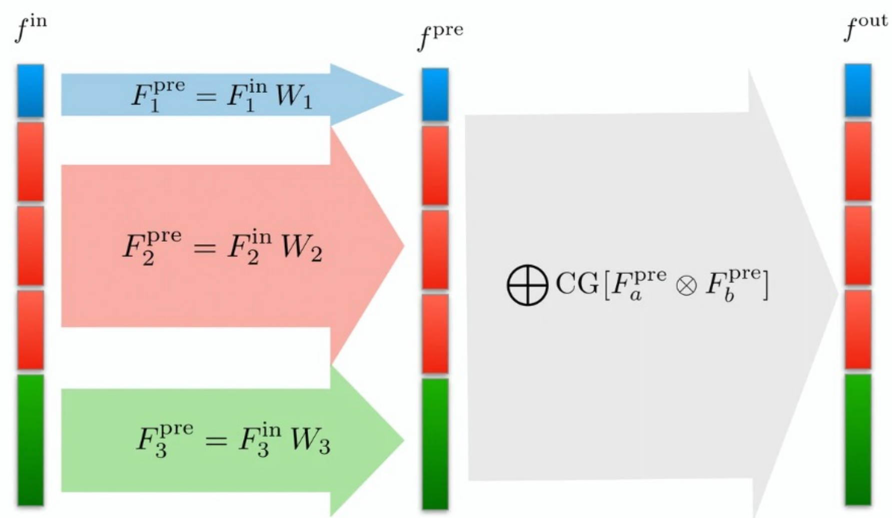
Clebsch-Gordan tensor product

$$\vec{f}_i' = \sum_j \sum_k \mathrm{CG}_{j,k,i} \cdot \vec{f}_j \vec{f}_k$$

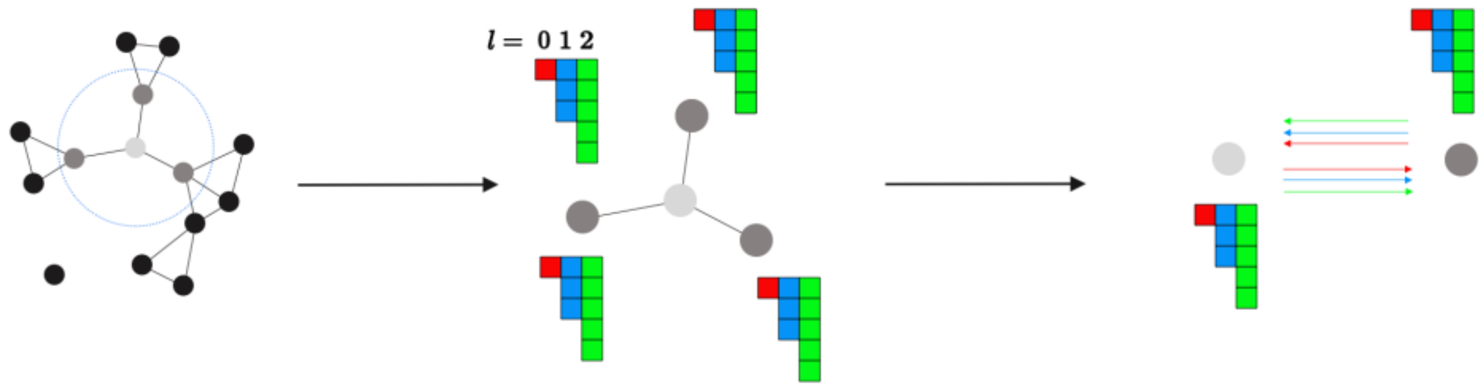
Gated nonlinearity:

$$\sigma_{\text{gated}} \left(\vec{f}_i\right) = \sigma\left(\left|\vec{f}_i\right|\right) \vec{f}_i$$

A "layer" of equivariant NN (Linear & nonlinear transformations)



Equivariant NN for molecular systems



A set of atoms is interpreted as an atomic graph.

Every atom carries a set of scalar, vector, and higher-order tensor features.

Atoms exchange information via filters.

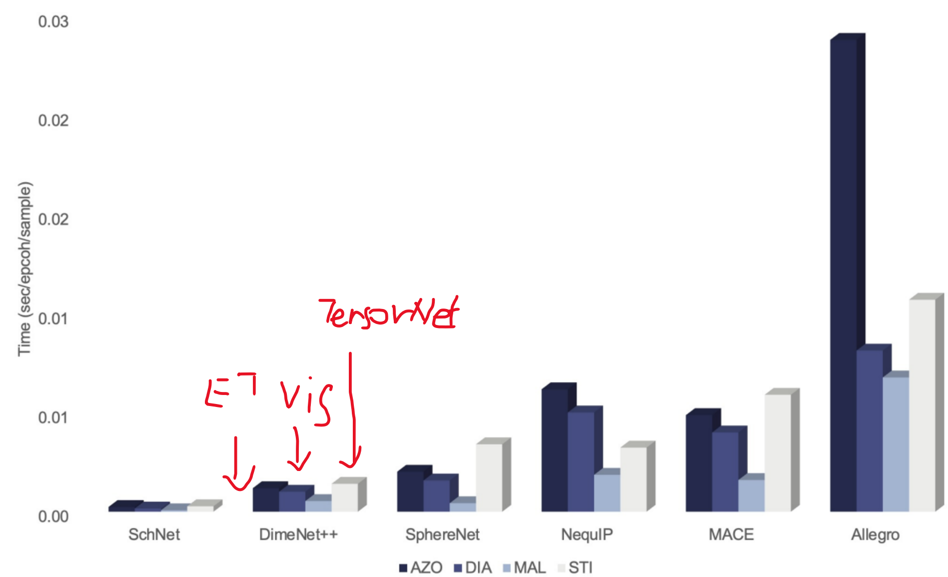
Roto-invariant/equivariant GNNs
Architectures

- Roto-invariant GNNs:
  - SchNet, DimeNet(++), PhysNet, GemNet, etc.
  - Increasing expressive power by using higher order scaler features
- Roto-equivariant GNNs
  - Scaler features + vectors features (my personal taste)
    - EGNN, GVP-GNN, PaiNN, Equivariant Transformer, ViSNet, TensorNet
    - Increasing expressive power by using higher-order feature and interactively update of scaler and vector features
  - Spherical harmonics + tensor product
    - TensorField Network, Cormorant, NequIP, Allegro, MACE, EquiFormer
    - Increasing expressive power by higher rank tensors or more complicated tensor operations

Roto-invariant/equivariant GNNs

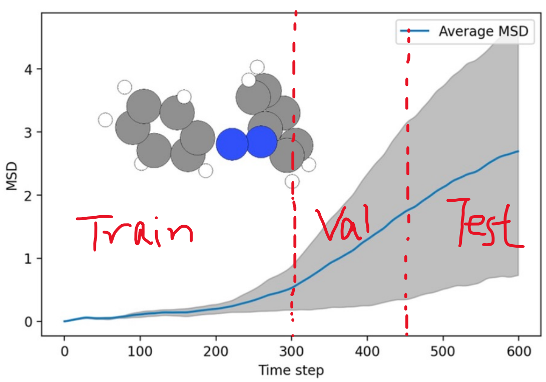
Personal opinions (don't take too much)

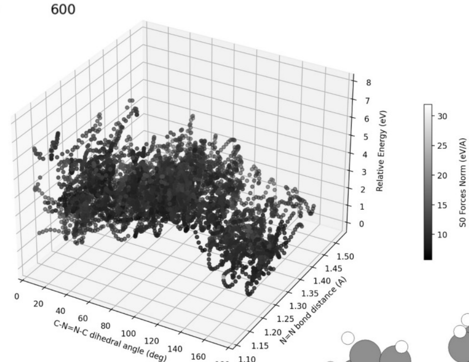
- Roto-invariant GNNs:
  - SchNet, DimeNet(++), PhysNet, GemNet, etc.
  - DON'T USE, OUT-OF-DATE
- Roto-equivariant GNNs
  - Scaler features + vectors features (my personal taste)
    - EGNN, GVP-GNN, PaiNN, Equivariant Transformer, ViSNet, TensorNet
    - Increasing expressive power by using higher-order feature and interactively update of scaler and vector features
  - Spherical harmonics + tensor product
    - TensorField Network, Cormorant, NequIP, Allegro, MACE, EquiFormer
    - **DON'T USE, 10x slower and memory-costly than what you expect (not joking)**

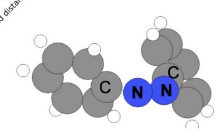


Roto-invariant/equivariant GNNs

Personal opinions (don’t take too much)







Not final results for PT models, just for a preview

	xxMD-Azobenzene (MAE, meV, meV/Ang)			
	Validation Set		Test Set	
	E	F	E	F
SchNet	539	248	722	283
DimeNet++	184	150	300	173
SphereNet	168	140	260	168
NequIP	393	119	1754	129
Allegro	106	98	174	110
MACE	257	71	292	85
ET (PT)	123	94	253	124
ViSNet (PT)	82	92	179	120
TensorNet-128	**45**	66	88	78
TensorNet-128 (PT)	50	**61**	**86**	**75**

Trajectories promoted by surface hopping, SA4-CASSCF(6e,6o)/6-31g, ground state energy and forces recomputed with uM06/6-31g