Geometric and Topological Representation Learning

Semih Cantürk

Hamed Shirzad

Qi Yan

Université de Montréal

UBC

Mila

VECTOR INSTITUTE

MLSP 2025

35th IEEE International Workshop on Machine Learning for Signal Processing

"Signal Processing in the Age of Large Language Models"

AUGUST 31 - SEPTEMBER 3 ISTANBUL / TÜRKİYE

ISTANBUL LUTFI KIRDAR INTERNATIONAL CONVENTION AND EXHIBITION CENTRE – ICEC

mlsp2025graphs.github.io

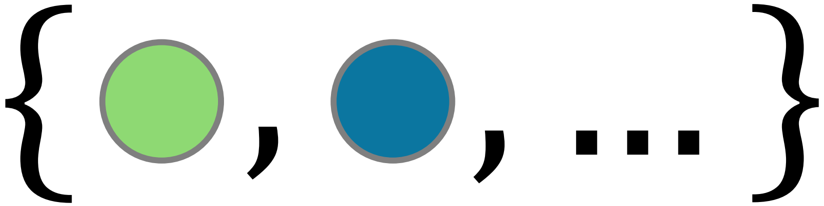
Part I
Graph Representations
A Primer

Part I: Outline

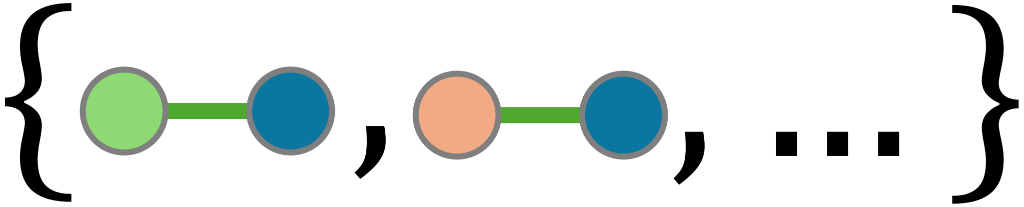
1. Primer on Graph Machine Learning
    1. Graph Representations: A Primer
    2. Early Methods
    3. Graph Neural Networks (GNN)
    4. Graph Convolutions
    5. Graph Isomorphism and Expressivity

Graphs

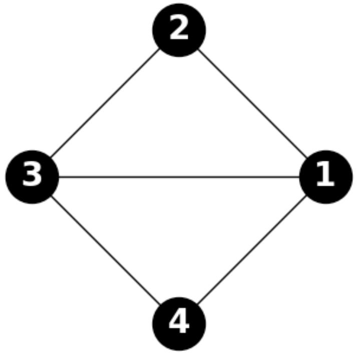
- **Set of Nodes =** V
  - Optionally with **features** X



- **Set of Edges =** E



- **Adjacency Matrix =** A



	1	2	3	4
1	0	1	1	1
2	1	0	1	0
3	1	1	0	1
4	1	0	1	0

Graphs

- Graphs ***G*** are defined by:
  - Vertices $V = \{v_1, \ldots, v_n\}$
  - Edges (undirected) $E = \{e_k = \{i, j\} : i, j \in V\} \subseteq V \times V$
  - Adjacency matrix $A : a_{i,j} = 1 \Leftrightarrow (i, j) \in E$
- Neighborhood: $\mathcal{N}(i) = \{j : (i, j) \in E\}$
- Degree: $d_i = |\mathcal{N}(i)|$
- Attributes:
  - Node features: $\mathbf{x} : V \to \mathbb{R}^d \quad \mathbf{X} : (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\top$
  - Edge features: $\mathbf{e} : E \to \mathbb{R}^{d'}$

$G = (V, E, A)$

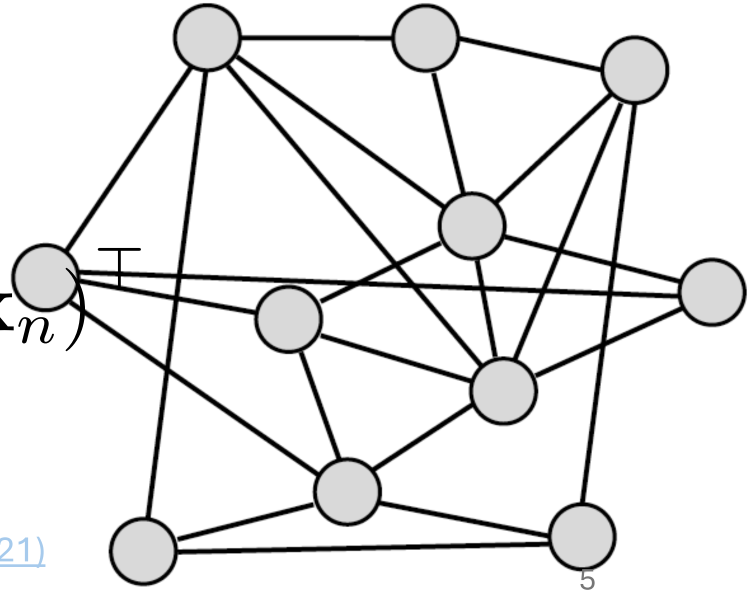
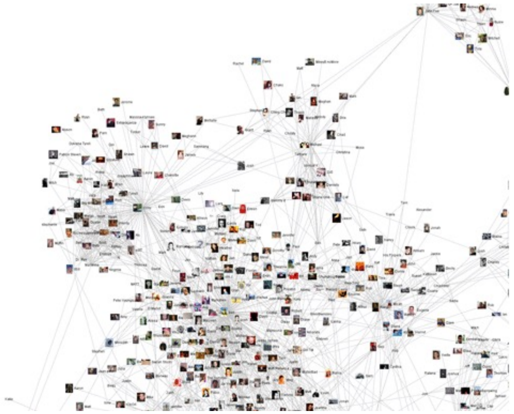
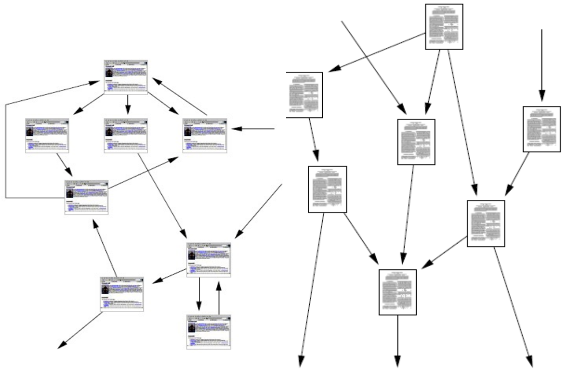


Illustration credit: Bronstein et al. (2021)

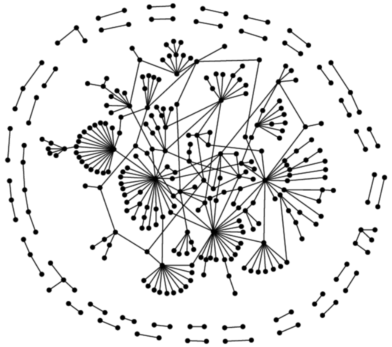
Abundance of graph-structured data



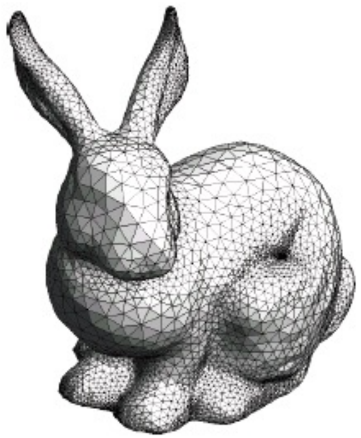
Social networks



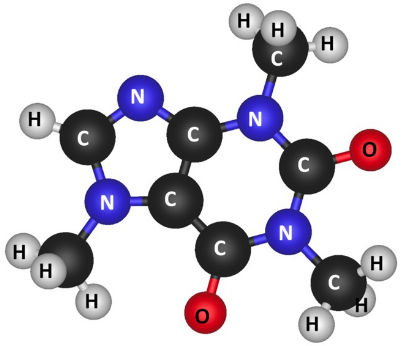
Knowledge graphs



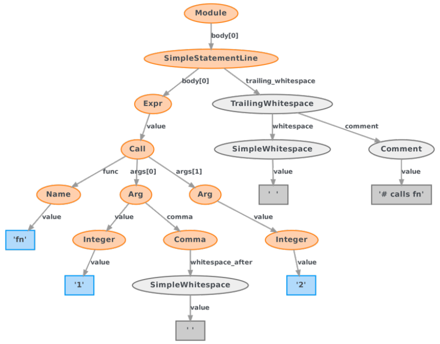
Biomedical networks



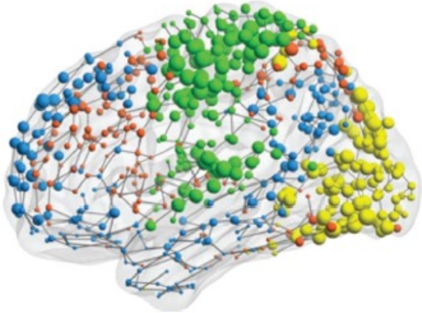
3D shapes



Molecular structure



The syntax of language and code



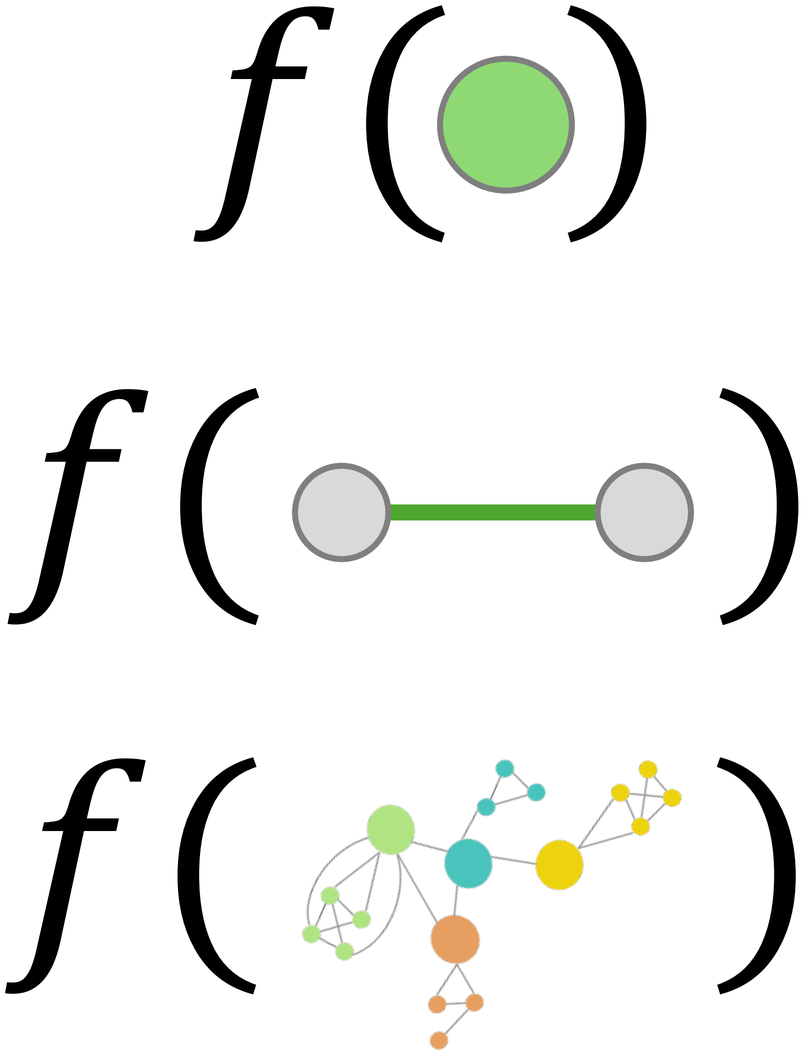
Networks of neurons

Types of tasks

- **Node-level tasks**
  - Node classification
  - Node clustering
  - Node regression

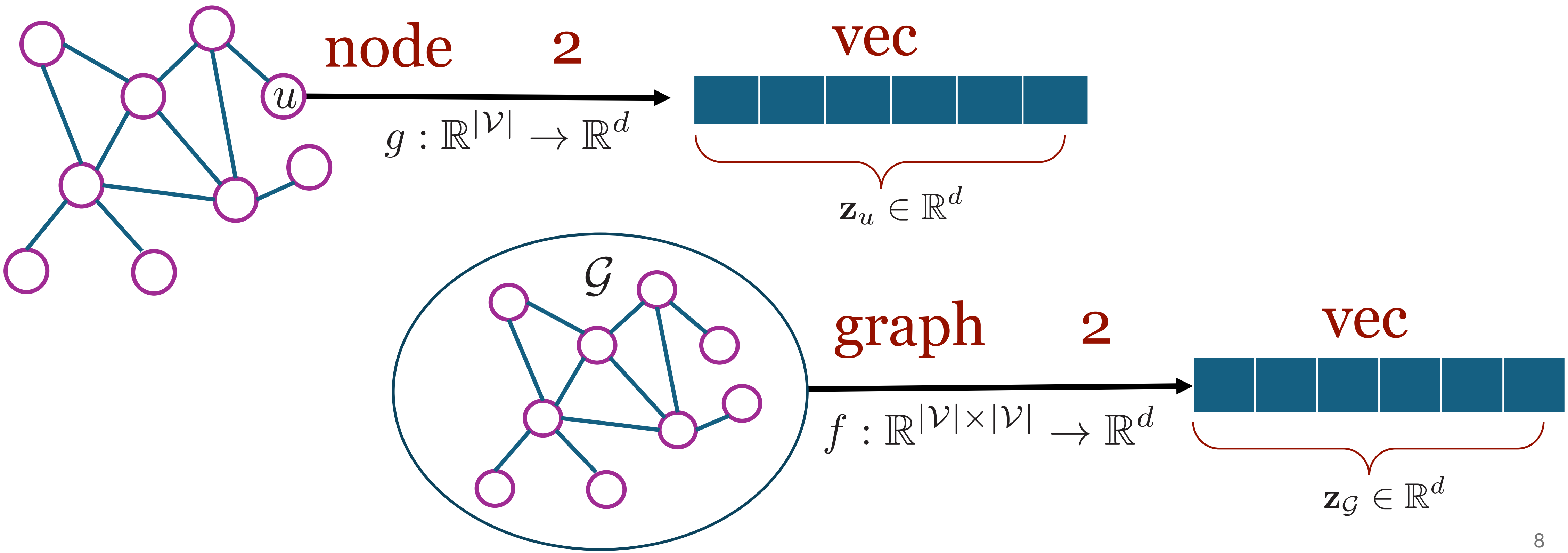
- **Edge-level tasks**
  - Link prediction
  - Edge classification
  - Knowledge graph completion

- **Graph-level tasks**
  - Graph classification
  - Graph regression



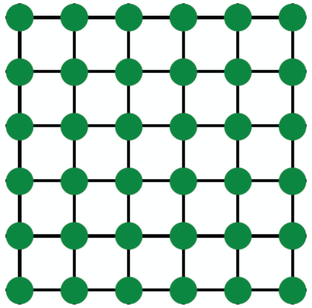
What is graph representation learning?

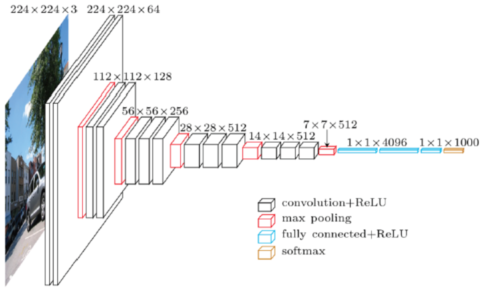
Goal: To learn useful node and graph representations without hand-crafted features



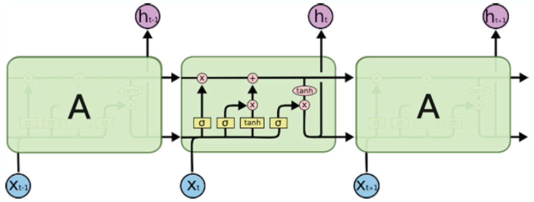
Why is it hard?

- Most deep learning methods are designed for regular sequences:
  - CNNs for regular grids: audio (1D), images (2D/3D), ...



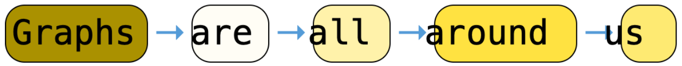
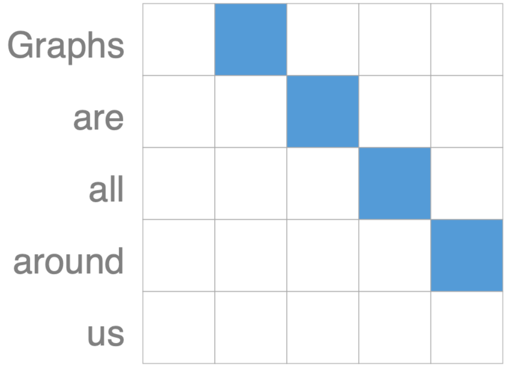
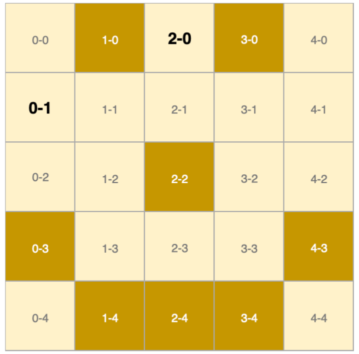
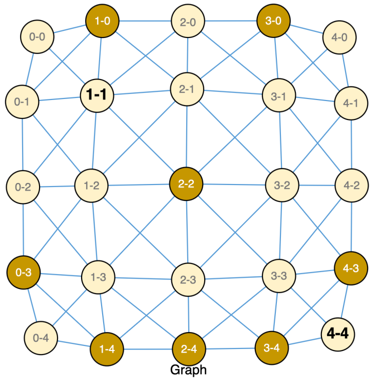
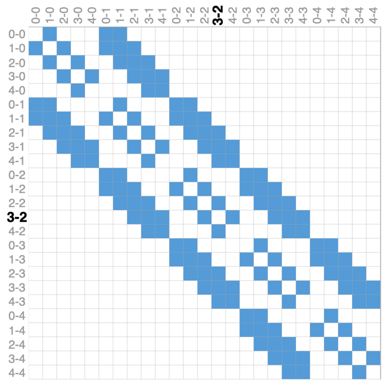
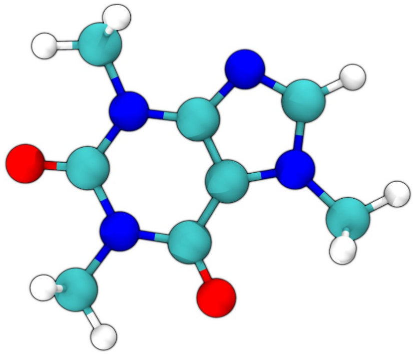
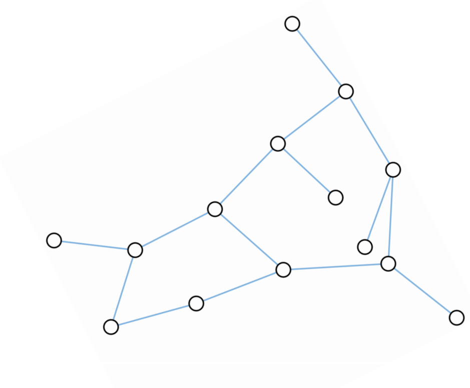
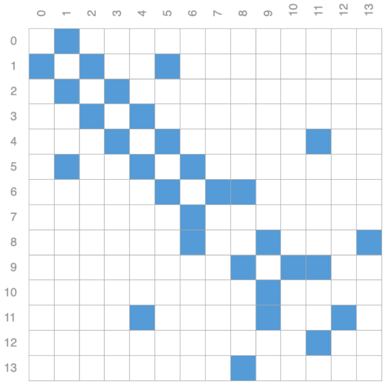


  - RNNs for sequences: text, audio, ...



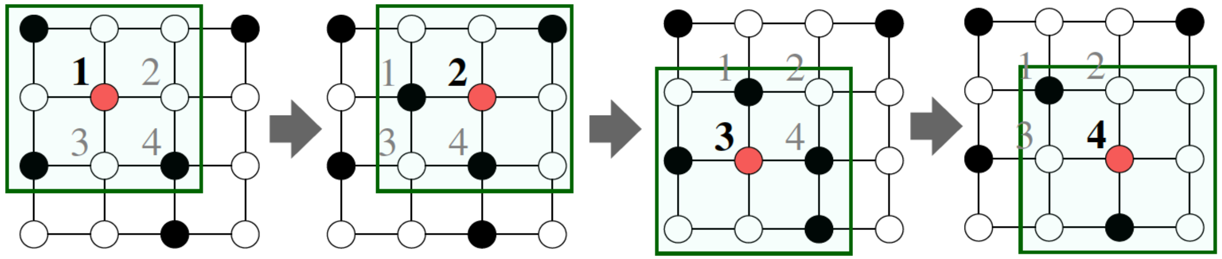
Graphs lack such regular geometric structure!

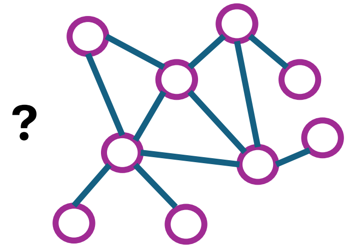
Figures from [Distill]

	Raw Data	Graph Repr.	Adjacency Matrix
Text	Graphs are all around us		
Image			
Molecule			

Key challenges of graph data

1. Complex geometric structure
   (often no straightforward notions of spatial locality)





2. Inherent symmetries lead to non-unique representations

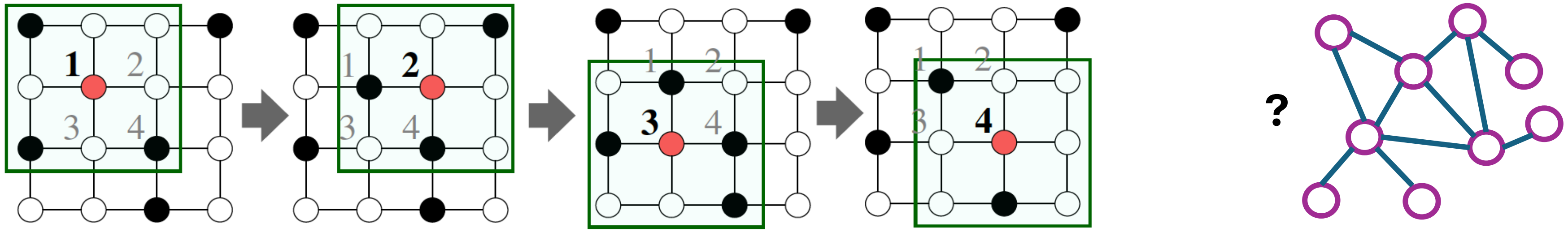
Graph-level functions must be **invariant** to node permutations

$$\boxed{f : \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|} \rightarrow \mathbb{R}^{d} \qquad f(\mathbf{PAP}^{\top}) = f(\mathbf{A}), \forall \mathbf{P} \in \mathbb{P}}$$

$$g : \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|} \rightarrow \mathbb{R}^{|\mathcal{V}|\times d} \qquad g(\mathbf{PAP}^{\top}) = \mathbf{P}g(\mathbf{A}), \forall \mathbf{P} \in \mathbb{P}$$

Key challenges of graph data

1. Complex geometric structure
   (often no straightforward notions of spatial locality)



2. Inherent symmetries lead to non-unique representations

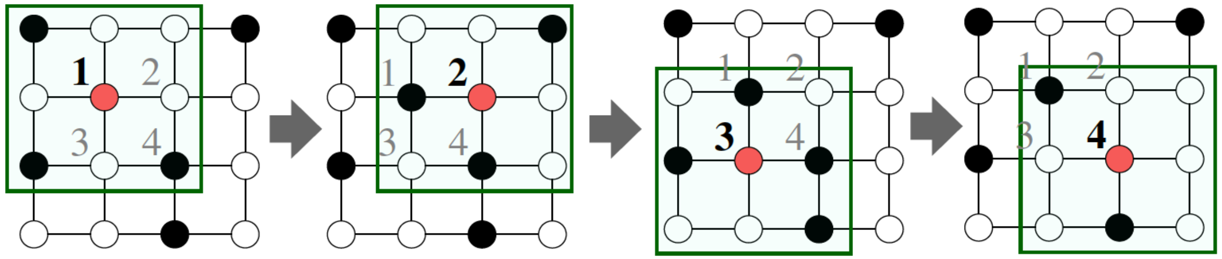
function on permuted adjacency matrix
=
function on original adjacency matrix

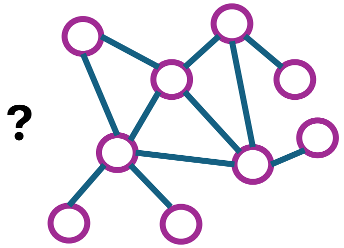
$$f : \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|} \to \mathbb{R}^d \qquad \boxed{f(\mathbf{PAP}^\top)} = \boxed{f(\mathbf{A})}, \forall \mathbf{P} \in \mathbb{P}$$

$$g : \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|} \to \mathbb{R}^{|\mathcal{V}|\times d} \qquad g(\mathbf{PAP}^\top) = \mathbf{P}g(\mathbf{A}), \forall \mathbf{P} \in \mathbb{P}$$

Key challenges of graph data

1. Complex geometric structure
(often no straightforward notions of spatial locality)





2. Inherent symmetries lead to non-unique representations

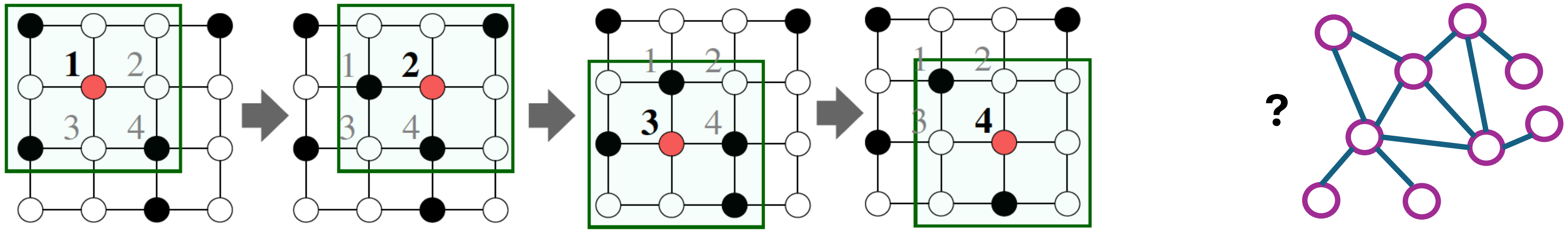
Node-level functions must be **equivariant** to node permutations

$$f : \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|} \to \mathbb{R}^{d} \qquad f(\mathbf{PAP}^{\top}) = f(\mathbf{A}), \forall \mathbf{P} \in \mathbb{P}$$

$$\boxed{g : \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|} \to \mathbb{R}^{|\mathcal{V}|\times d} \qquad g(\mathbf{PAP}^{\top}) = \mathbf{P}g(\mathbf{A}), \forall \mathbf{P} \in \mathbb{P}}$$

Key challenges of graph data

1. Complex geometric structure
(often no straightforward notions of spatial locality)



2. Inherent symmetries lead to non-unique representations

permutation of input
=
equivalent
permutation applied
to output

$$f : \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|} \to \mathbb{R}^{d} \qquad f(\mathbf{PAP}^\top) = f(\mathbf{A}), \forall \mathbf{P} \in \mathbb{P}$$

$$g : \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|} \to \mathbb{R}^{|\mathcal{V}|\times d} \qquad \boxed{g(\mathbf{PAP}^\top)} = \boxed{\mathbf{P}g(\mathbf{A})}, \forall \mathbf{P} \in \mathbb{P}$$

Data invariance & equivariance

Invariance

Equivariance

Computer Vision

Graph Learning

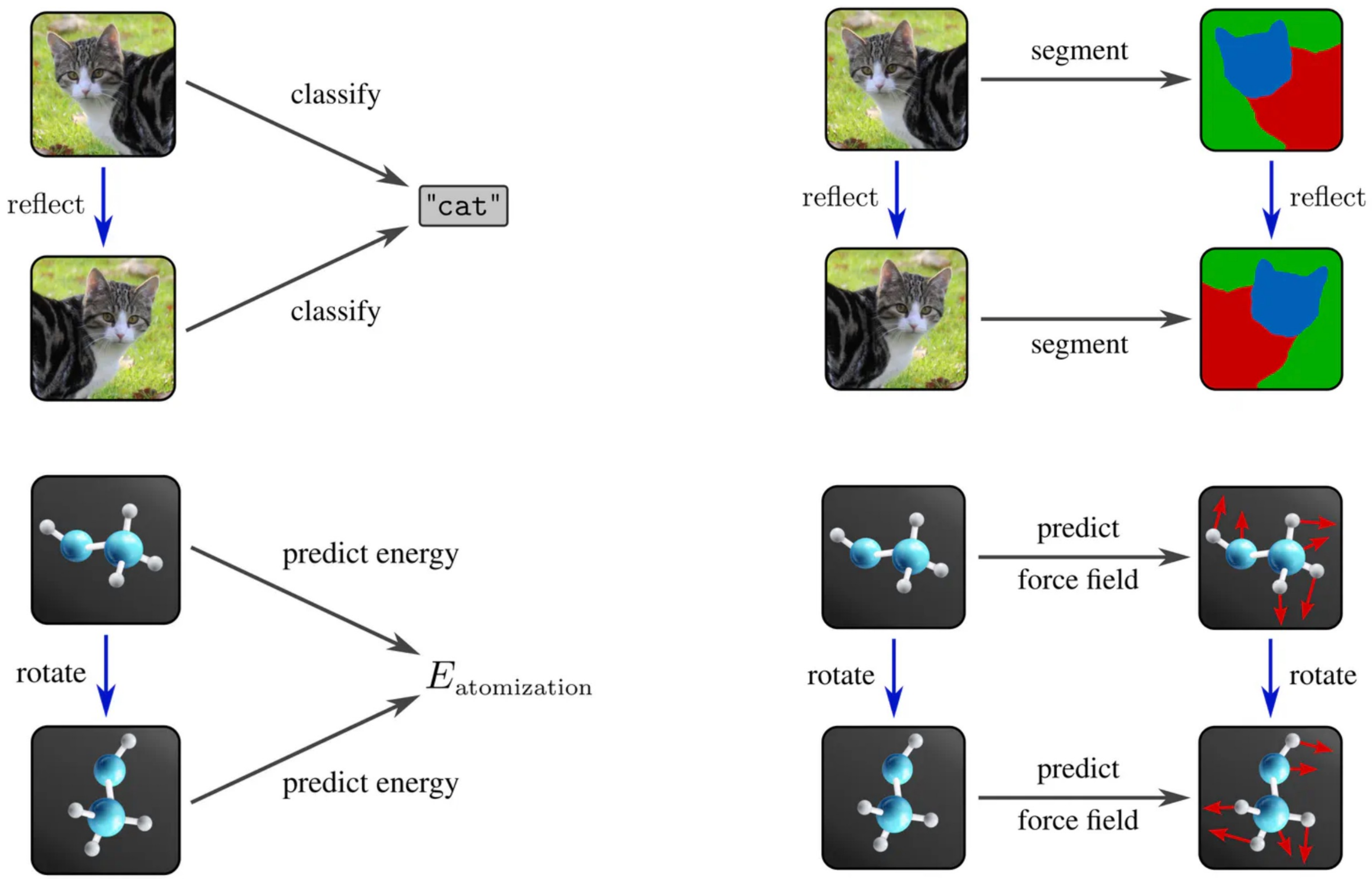


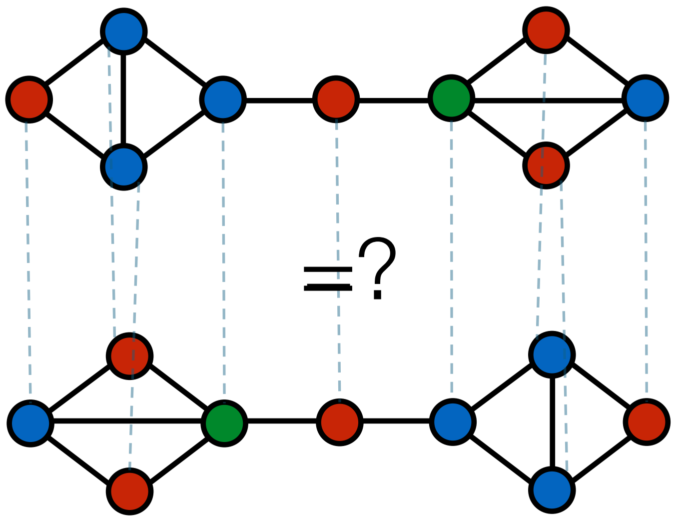
Illustration credit : [Maurice Weiler]

Isomorphism is often a bottleneck

- Even determining whether two graphs are the same (isomorphism testing) is computationally difficult!
- “NP-indeterminate” problem

$$\boxed{\min_{\mathbf{P}\in\mathbb{P}}} \left\| \boxed{\mathbf{A}_1} - \boxed{\mathbf{P}\mathbf{A}_2\mathbf{P}^\top} \right\| \boxed{\stackrel{?}{=} 0}$$

Does there exist a permutation matrix
such that
the adjacency matrix of graph one
equals
a permutation of the adjacency matrix of graph two?



Early Methods

Node Embeddings and Graph Kernels

Classic approach: Node Embeddings

- Goal is to encode nodes so that
  similarity in the embedding space (e.g., dot product)
  approximates similarity in the original graph

$$\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$

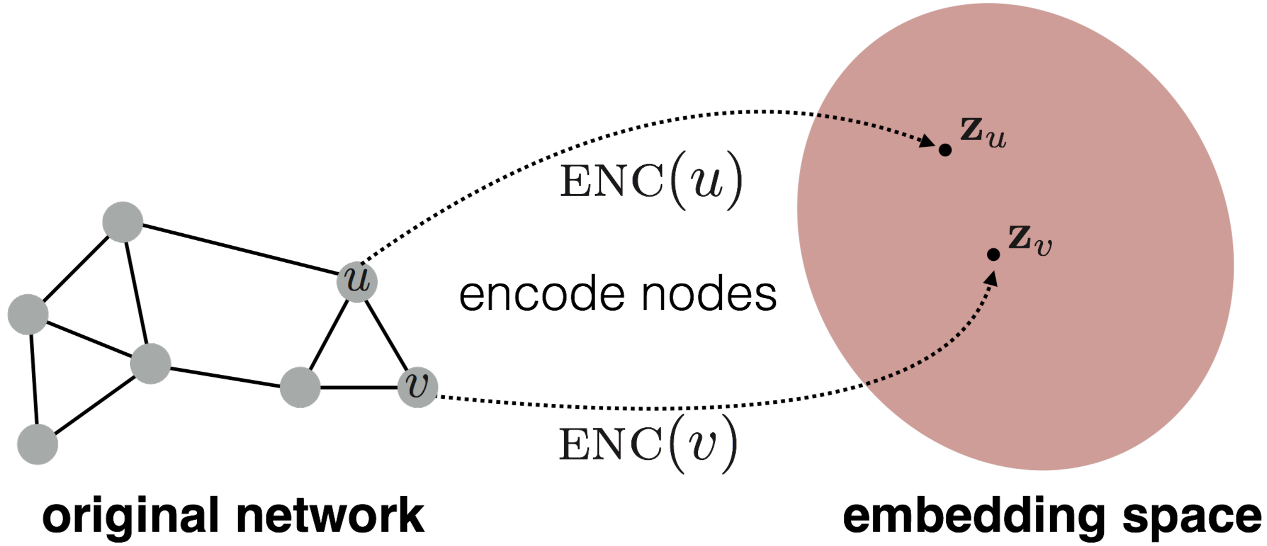


Illustration credit : [Stanford CS224W]

Node embeddings

- Traditional node embedding approaches use a *shallow encoder*: encoder is just an embedding-lookup

$$\mathrm{ENC}(v) = \mathbf{Z}\mathbf{v}$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ matrix, each column is node embedding **[what we learn!]**

$\mathbf{v} \in \mathbb{I}^{|\mathcal{V}|}$ indicator vector, all zeroes except one indicating node *v*

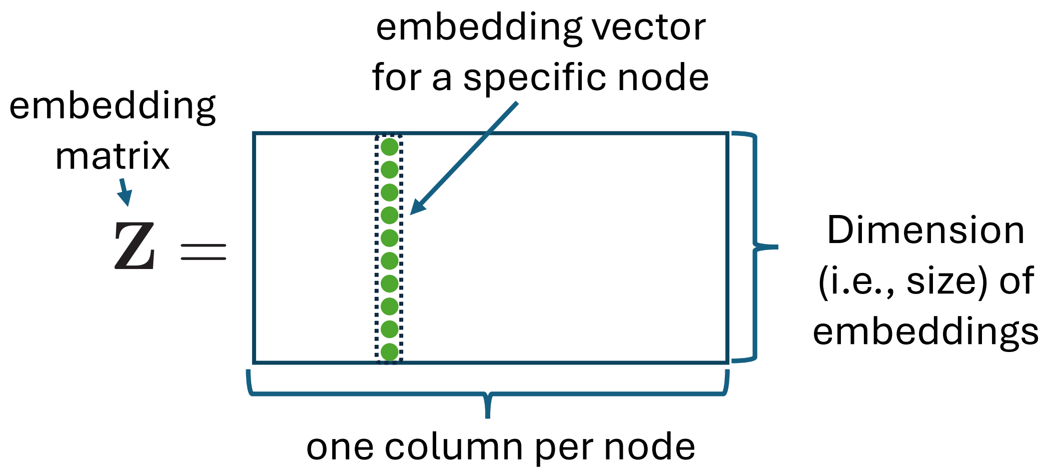


Illustration credit : Stanford CS224W

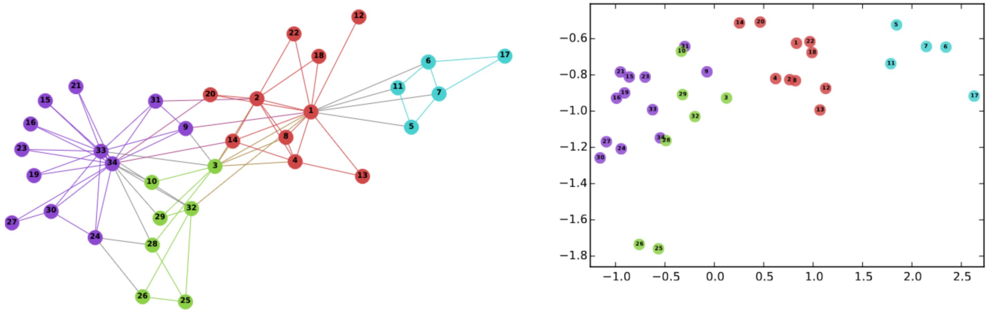
Node embeddings: Examples

- Adjacency-based similarity (HOPE; Ou et al. 2016):

$$\mathbf{z}_u^\top \mathbf{z}_v \approx \boxed{\mathbf{A}^k[u, v]}$$

k$^{\text{th}}$ power of the adjacency matrix
=
walks of length-*k* between ***u*** and ***v***

- Random-walk based similarity (DeepWalk; Perozzi et al. 2014):



(a) Input: Karate Graph (b) Output: Representation

$$\mathbf{z}_u^\top \mathbf{z}_v \approx$$

probability that *u* and *v* co-occur on a random walk over the network

Source: Perozzi et al. (2014)

Limitations of traditional node embeddings

1. O(|V|) parameters are needed:
   There is no parameter sharing and every node has its own unique embedding vector
2. Inherently "transductive":
   Cannot generate embeddings for nodes that were not seen during training
3. Does not incorporate node features:
   Many graphs have features that we can and should leverage

Towards Graph Neural Networks

Traditional node embeddings

Encoder is just a shallow lookup from an embedding matrix

$$\mathrm{ENC}(v) = \mathbf{Z}\mathbf{v}$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ Embedding matrix, each column is node embedding

$\mathbf{v} \in \mathbb{I}^{|\mathcal{V}|}$ indicator vector for node *v*

Graph neural networks

Encoder is complex function of graph structure and node features

$$\mathrm{ENC}(v) = f(\mathbf{A}, \mathbf{X})$$

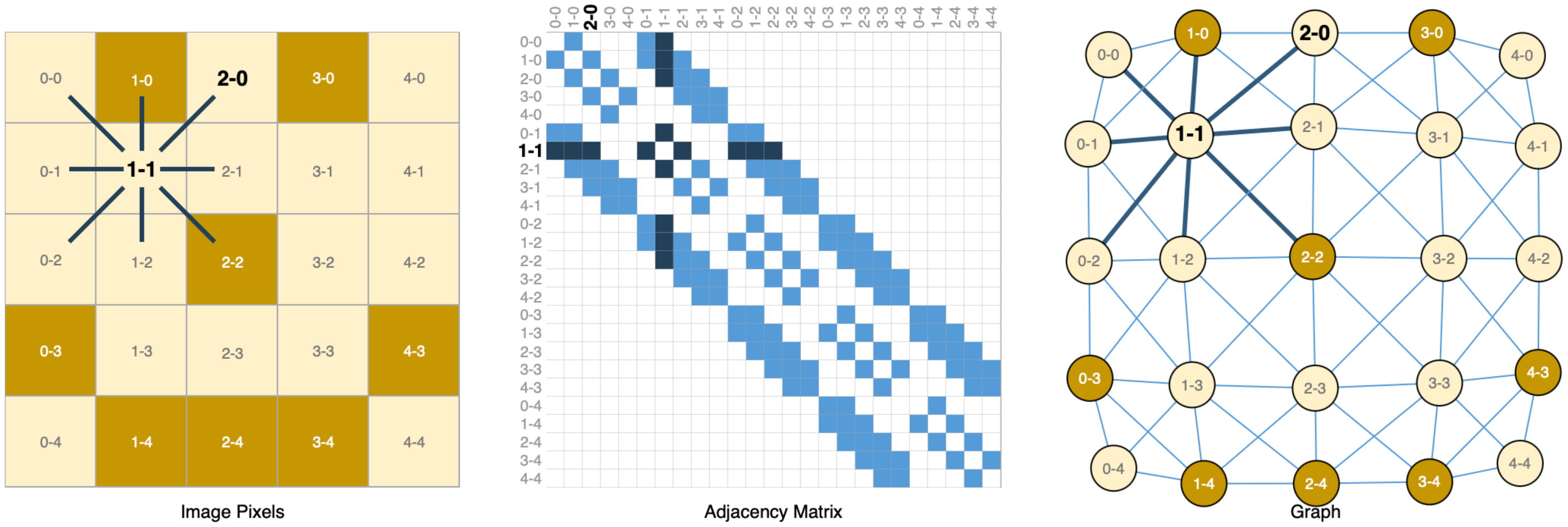
$\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ adjacency matrix for the graph

$\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$ matrix of node features

Graph Neural Networks

Intuition: Convolutional Networks

- Images as graphs: Filters capture neighborhood on a grid graph

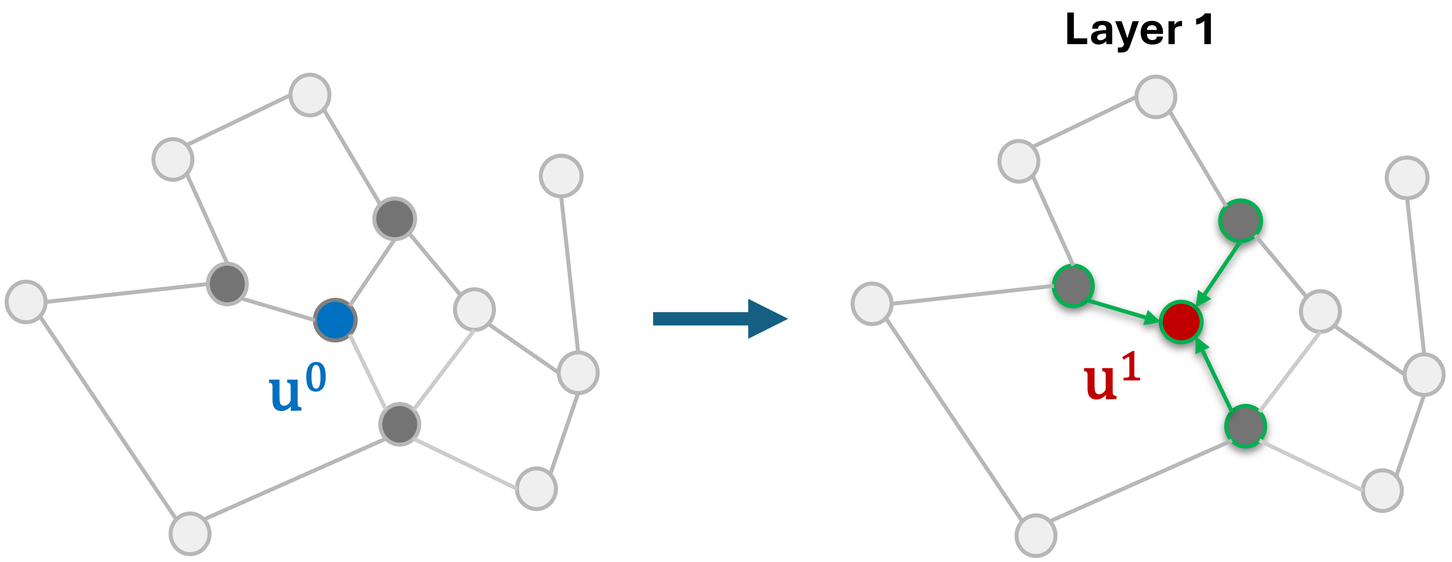


- GNNs generalize this intuition to *general* graphs
- Challenge: neighborhoods look different!

Image from [Distill]

Graph Neural Networks

- Most GNNs are based on the idea of iterative neighborhood aggregation, a.k.a. **Message Passing**



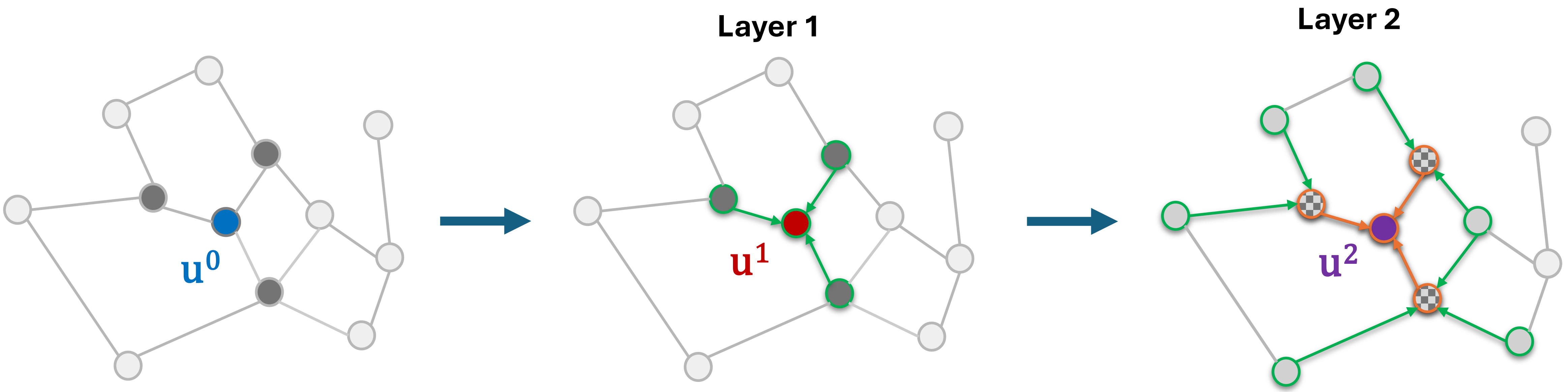
$$u^1 := f1\,(\,u^0, \mathrm{neighbors}(u)\,)$$

the update is done
in parallel for all nodes

Illustration credit: Ladislav Rampášek

Graph Neural Networks

- Most GNNs are based on the idea of iterative neighborhood aggregation, a.k.a. **Message Passing**



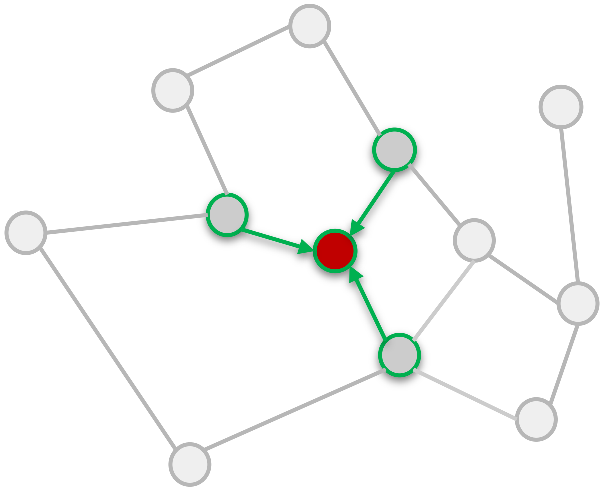
$u^1 := f1\,(\,u^0, \mathrm{neighbors}(u)\,)$ $u^2 := f2\,(\,u^1, \mathrm{neighbors}(u)\,)$

the update is done
in parallel for all nodes

Illustration credit: Ladislav Rampášek

The family of MPNNs

- Message Passing Neural Networks (MPNN) are a general class of GNNs



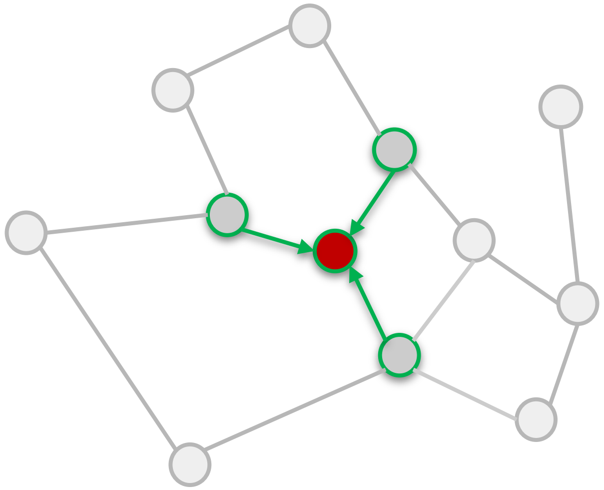
$$\begin{aligned}
\mathbf{h}_u^{(k+1)} &= \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \boxed{\text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\})}\right) \\
&= \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \boxed{\mathbf{m}_{\mathcal{N}(u)}^{(k)}}\right),
\end{aligned}$$

Information aggregated from neighboring nodes

(sometimes called the "messages" from the neighbors)

The family of MPNNs

- Message Passing Neural Networks (MPNN) are a general class of GNNs



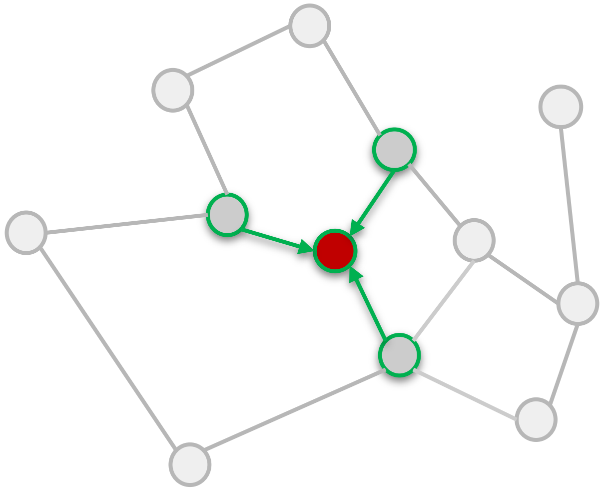
$$\mathbf{h}_u^{(k+1)} = \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\})\right)$$
$$= \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)}\right),$$

Information aggregated from neighboring nodes

Combined "new" embedding of this node with the previous layer

(sometimes called the "messages" from the neighbors)

The family of MPNNs



- Message Passing Neural Networks (MPNN) are a general class of GNNs

Updated node embedding

$$\boxed{\mathbf{h}_u^{(k+1)}} = \boxed{\text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \boxed{\text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\})}\right)}$$

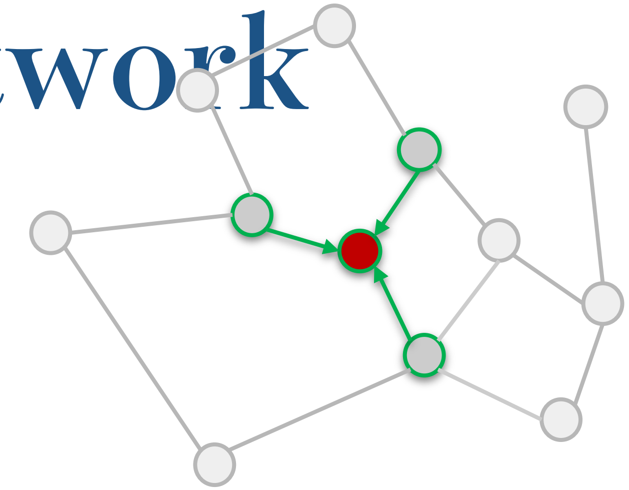
$$= \boxed{\text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \boxed{\mathbf{m}_{\mathcal{N}(u)}^{(k)}}\right)},$$

Information aggregated from neighboring nodes

Combined "new" embedding of this node with the previous layer

(sometimes called the "messages" from the neighbors)

The basic Graph Neural Network



- A "basic" GNN (Scarselli et al., 2008)

$$\boxed{\mathbf{h}_u^{(0)} = \mathbf{x}_u}$$

Initial "layer 0" embeddings are equal to node features

$$\boxed{\mathbf{h}_u^{(k)}} = \boxed{\sigma}\left(\mathbf{W}_{\text{self}}^{(k)} \boxed{\mathbf{h}_u^{(k-1)}} + \mathbf{W}_{\text{neigh}}^{(k)} \boxed{\sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)}} + \mathbf{b}^{(k)}\right)$$

kth layer embedding of *v*

non-linearity (e.g., ReLU)

previous layer embedding of *u*

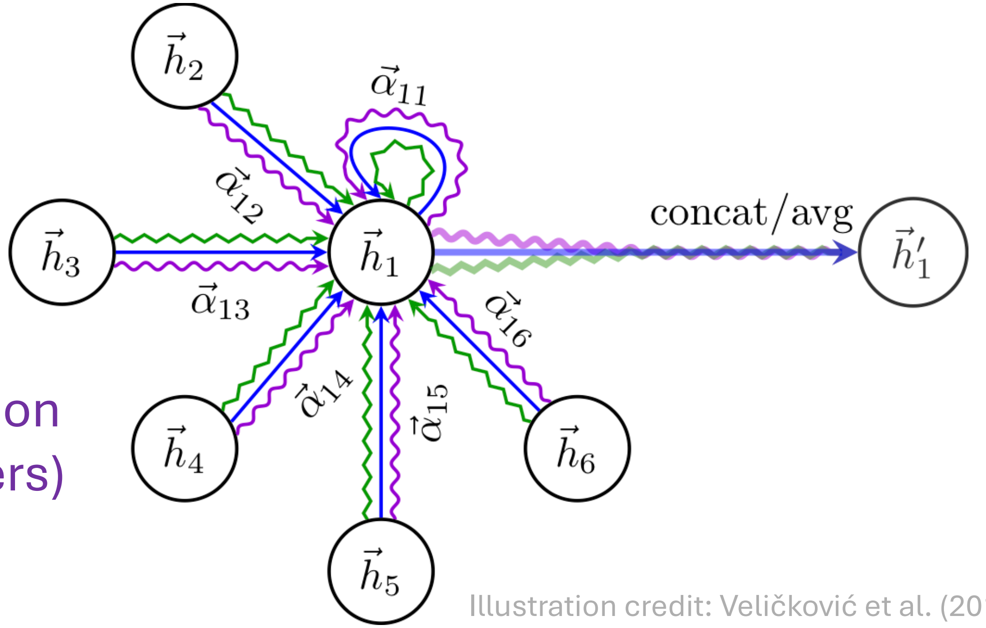
sum of neighbor's previous layer embeddings

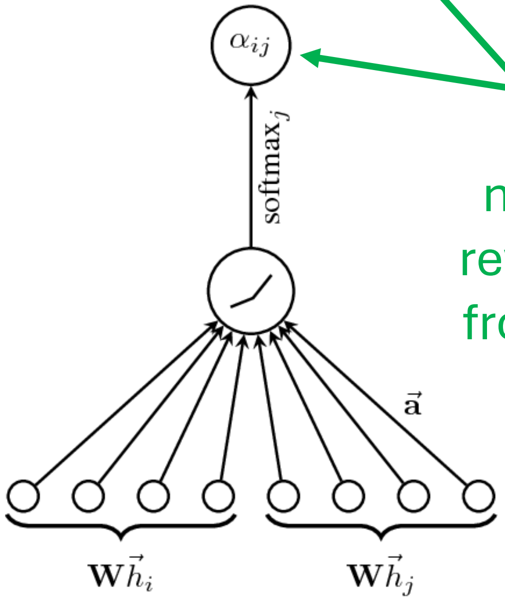
Graph Attention Networks (GATs)

- Aggregating with self-attention (local Transformer on graphs), Veličković et al. (2018)

$$\text{AGGREGATE}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\}) = \sum_{v \in \mathcal{N}(u)} \boxed{\alpha(\mathbf{h}_u^{(k)}, \mathbf{h}_v^{(k)})}\,\mathbf{h}_v^{(k)}$$

Multi-head attention (like in Transformers)





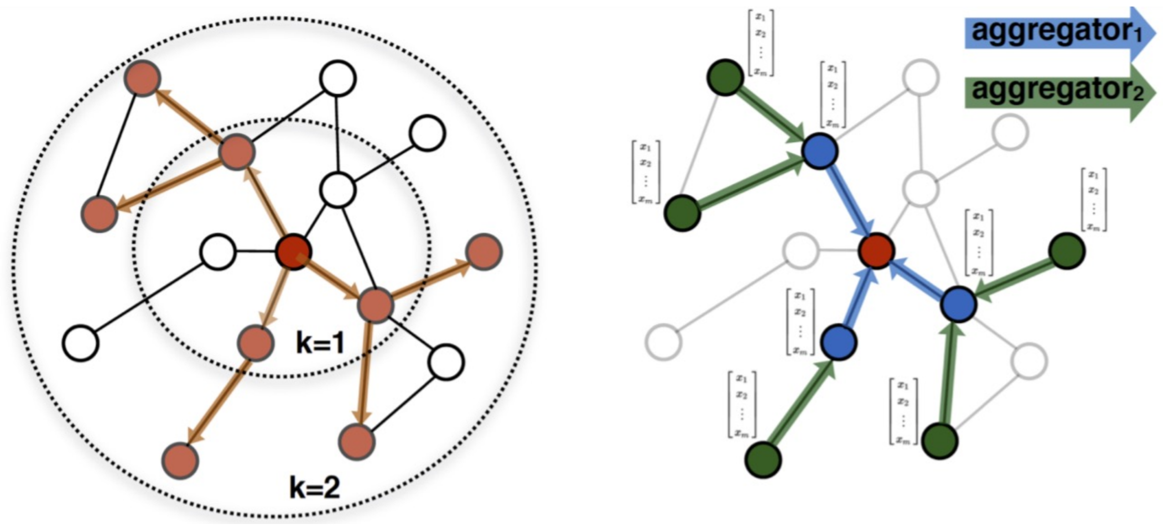
Pair-wise node attention to reweight messages from the neighbors

Illustration credit: Veličković et al. (2018)

Generalized neighborhood aggregation

- Aggregating more complex transformations of the neighbor features, as in Hamilton et al. (2017)'s **GraphSAGE**:

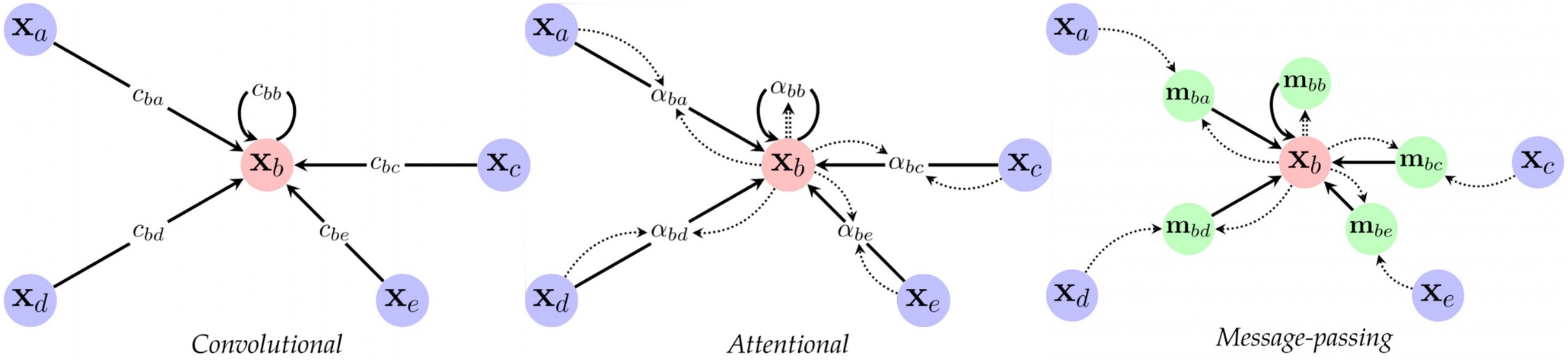
$$\text{AGGREGATE}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\}) = \frac{1}{|\mathcal{N}(u)|} \sum_{v \in \mathcal{N}(u)} \text{MLP}(\mathbf{h}_v^{(k)})$$



Explore the neighborhood (e.g., random walks) and aggregate with e.g. MLP or RNN instead of sum or average

Illustration credit: Hamilton et al. (2017)

List of most popular MPNNs



$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} c_{ij}\psi(\mathbf{x}_j)\right) \qquad \mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} a(\mathbf{x}_i, \mathbf{x}_j)\psi(\mathbf{x}_j)\right) \qquad \mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} \psi(\mathbf{x}_i, \mathbf{x}_j)\right)$$

- **GCN**: Graph Convolutional Network (Kipf and Welling, 2017)
- **GAT**: Graph Attention Network (Veličković et al., 2018)
- **GatedGCN**: Residual Gated Graph ConvNets (Bresson & Laurent, 2018)
- **GIN**: Graph Isomorphism Network (Xu et al., 2018)

Illustration credit: Hamilton et al. (2020)

A few useful tricks

- Include **edge** and/or **graph-level features** during message-passing (e.g., Battaglia et al., 2018)
- Residual connections between layers, GraphNorm, ...
- Jumping knowledge (JK) connections (Xu et al., 2018):

$$\boxed{\mathbf{z}_u = \mathbf{h}^{(K)}}$$

Instead of using the final GNN layer output as the node embeddings

$$\boxed{\mathbf{z}_u = f([\mathbf{x}_u, \mathbf{h}_u^{(1)}, \ldots, \mathbf{h}_u^{(K)}])}$$

Define the node embeddings as a function of the representations at all intermediate layers!

Advantages of MPNNs

1. Number of parameters is independent of the graph size:
   The parameters in the GNN layers depend only on the dimension of the feature inputs
2. Inherently “inductive”:
   After training GNNs can be used to infer embeddings on unseen graphs
3. Naturally incorporate node features:
   GNNs learn by aggregating node features over local neighborhoods

Readout layer

- After L message-passing layers, we get embeddings $h_u{}^L$ at each u
- How to convert to a final prediction?
- Use a readout layer, aka prediction/output heads

Node-level task:

$$\mathbf{g}_v = f_{\text{readout}}(\mathbf{h}_v^{(L)})$$

Graph-level task:

$$\mathbf{h}_G = f_{\text{readout}}(\{\!\{\mathbf{h}_v^{(L)} \mid v \in V\}\!\})$$

Edge-level task:

$$\mathbf{g}_{u,v} = f_{\text{readout}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$$

Graph classification: Node pooling

- Pool node embeddings $\longrightarrow$ get embedding for the whole graph
- The pooling function must be invariant w.r.t. node ordering
- Sum pooling: simply sum the node embeddings:

Embedding for the entire graph

$$\boxed{\mathbf{z}_{\mathcal{G}}} = \boxed{\sum_{v \in \mathcal{V}} \mathbf{h}_v^{(K)}}$$

Sum over all the node embeddings in the graph

Graph classification: Node pooling

- Hierarchical pooling (e.g., Ying et al. (2018)'s DiffPool)
  1. Coarsen graph by clustering or removing nodes based on their embeddings
  2. Run another GNN on coarsened graph
  3. Repeat until graph is just a single node (or use a final sum pooling layer)

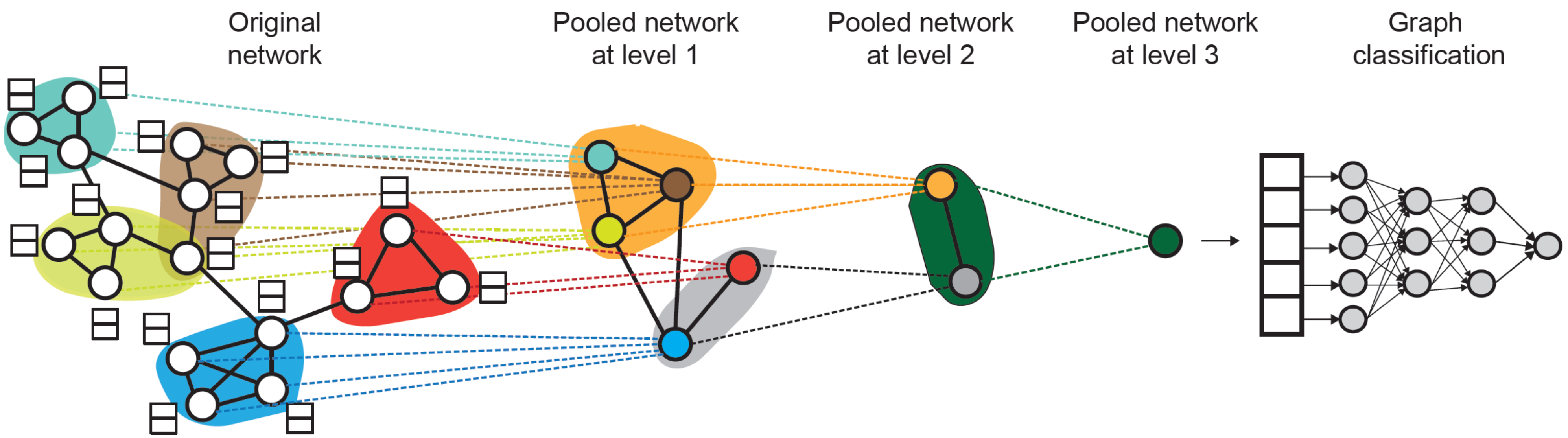
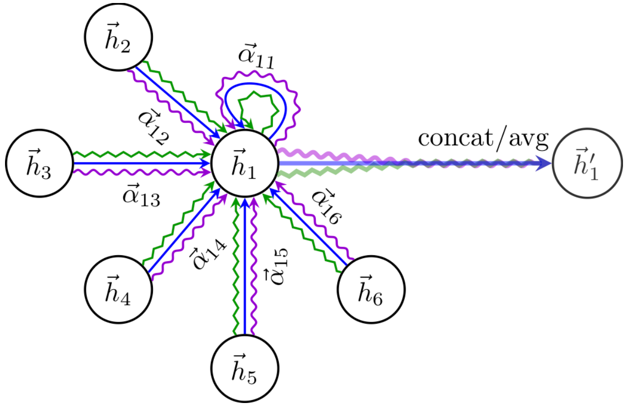


Illustration credit: Ying et al. (2018)

Summary of MPNNs

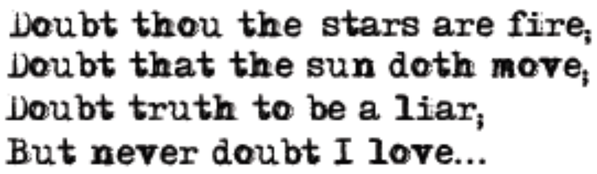


- MPNN = GNN based on "Neural Message Passing" paradigm
- A layer of MPNN is principally composed from:
  - Message block: computes a message along an edge between nodes *u* and *v*
  - Aggregation block: pools all incoming messages from the neighborhood of *u*
  - Update block: combines node representation with the neighborhood info
- With appropriate "readout" block, MPNN can be trained end-to-end for any of the 3 graph tasks (node, link, or graph –level prediction)

Graph Convolutions

A Spectral Perspective

Deep Learning success on Euclidean domains



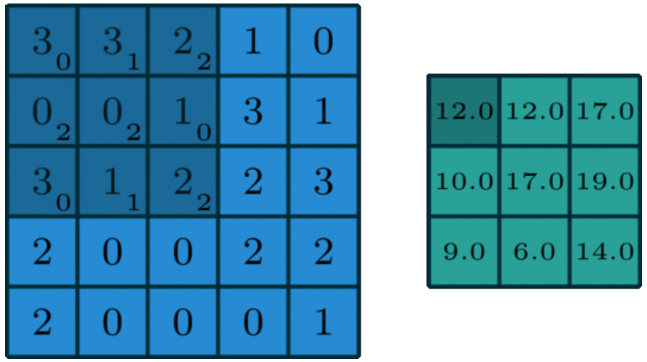
Text

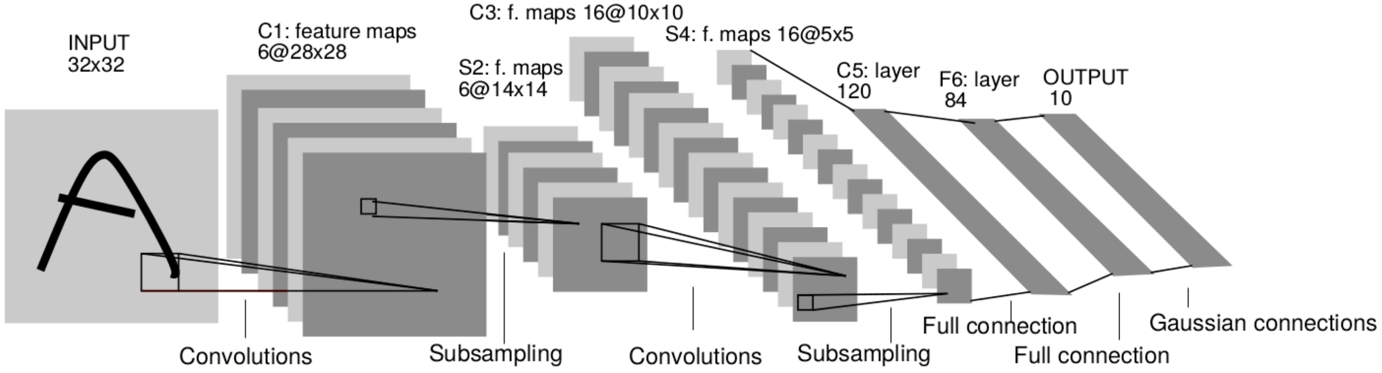
Audio signals

Images

- What geometric structure in images, speech, video, text, is exploited by CNNs?

Key properties of CNNs





Convolutional **(Translation invariance)**

Scale Separation **(Compositionality)**

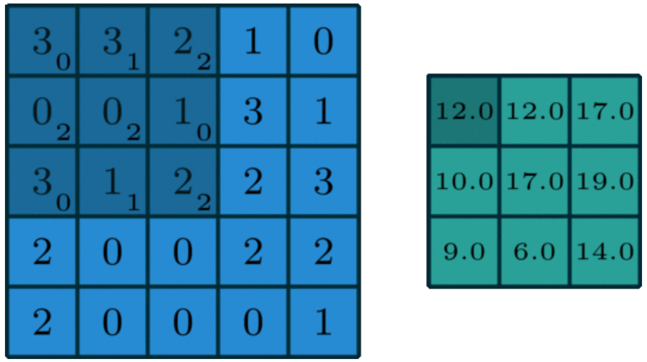
Filters localized in space **(Deformation Stability)**

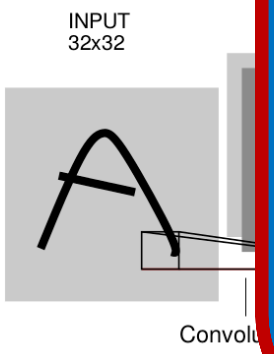
$O(1)$ parameters per layer (independent of input image size n)

$O(n)$ complexity per layer (filtering done in the spatial domain)

Illustration credit : Dumoulin & Visin (2016)

Key properties of C





Can we extend convolutions to the graph domain?

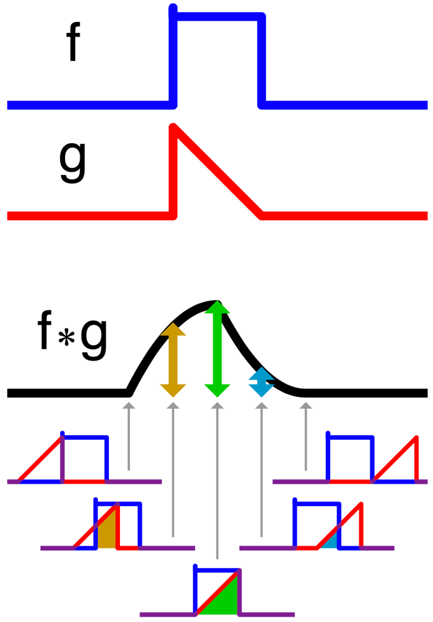
How do the GNNs we just saw fit into this?

- ☺️Convolutional **(Translation invariance)**
- ☺️Scale Separation **(Compositionality)**
- ☺️Filters localized in space **(Deformation Stability)**
- ☺️$O(1)$ parameters per layer (independent of input image size n)
- ☺️$O(n)$ complexity per layer (filtering done in the spatial domain)

Illustration credit : [Dumoulin & Visin (2016)]

Convolutions

- Mathematical operation on two functions that expresses how the *shape* of one is modified by the other
- **Amount of overlap** of one function as it is shifted over another
- This makes them great feature extractors via **template matching**



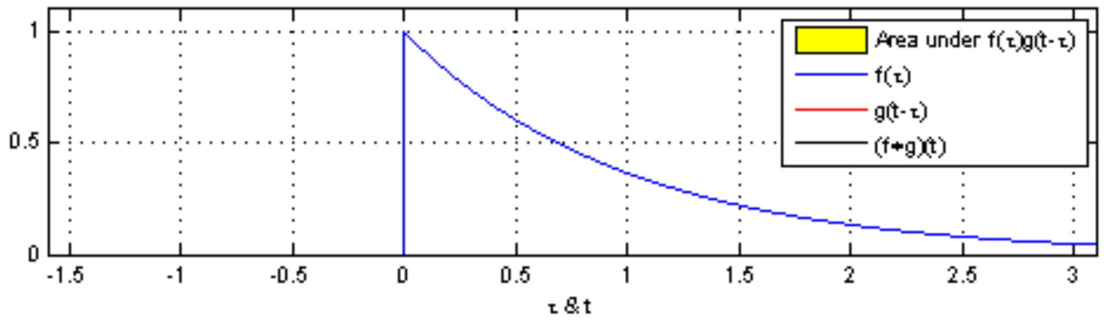
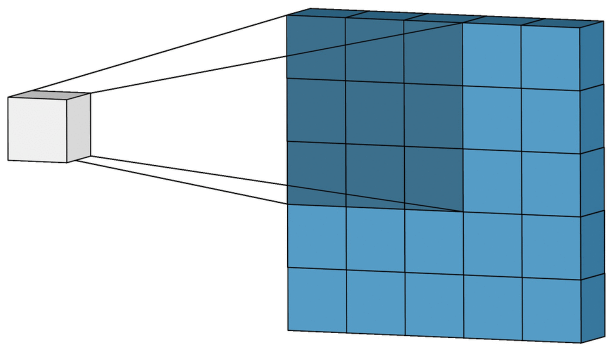


Illustration credit : Wikimedia Commons

Convolutions

- Convolution as **template matching:**



$$h_i^{\ell+1} = w^\ell * h_i^\ell$$

$$= \sum_{j \in \Omega} \langle w_j^\ell, h_{i-j}^\ell \rangle$$

$$= \sum_{j \in \mathcal{N}_i} \langle w_j^\ell, h_{ij}^\ell \rangle$$

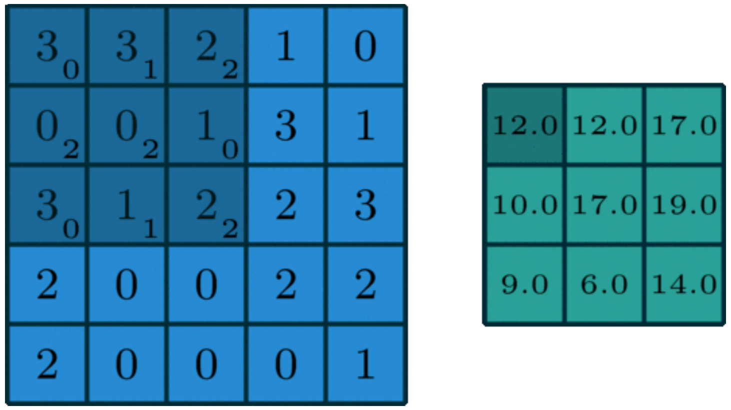


Illustration credit : [Dumoulin & Visin (2016)]()

Convolutions & the Fourier Transform

- The **Convolution Theorem** states that the Fourier transform of the convolution of two functions is the pointwise product of their Fourier transforms:

$$\mathcal{F}(w * h) = \mathcal{F}(w) \odot \mathcal{F}(h) \implies w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

- In general, the Fourier transform has $O(n^2)$ complexity, but if the domain is a grid, then the complexity can be reduced to $\boldsymbol{O(n \log n)}$ **with FFT**

Convolutions & the Fourier Transform

- FT decomposes a signal into frequency components
- A mapping from "time domain" to "frequency" domain…
  - … but wait – what do these even mean on a static graph?
- Our signals are defined over *nodes* instead

Illustration credit: [Wikimedia Commons]

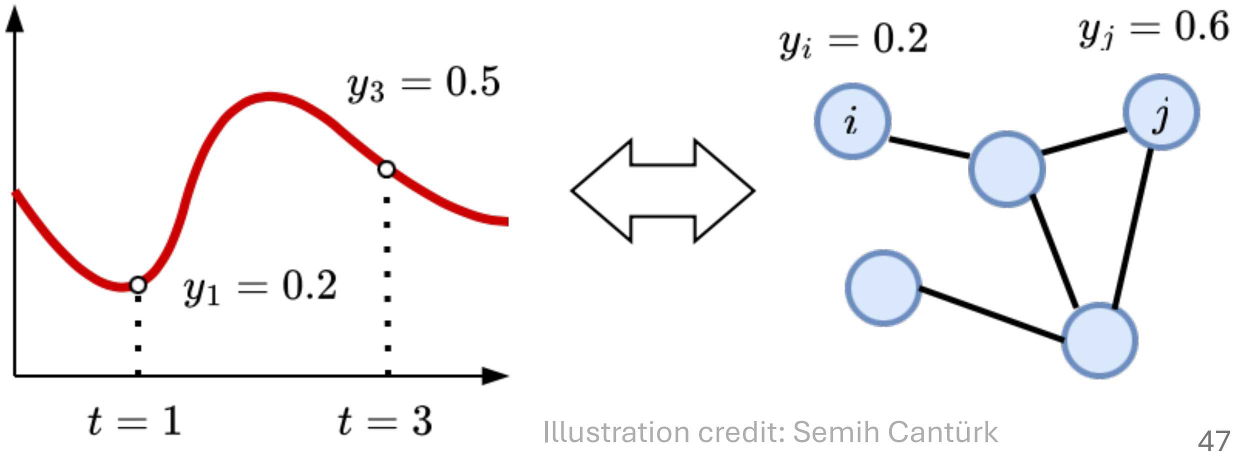
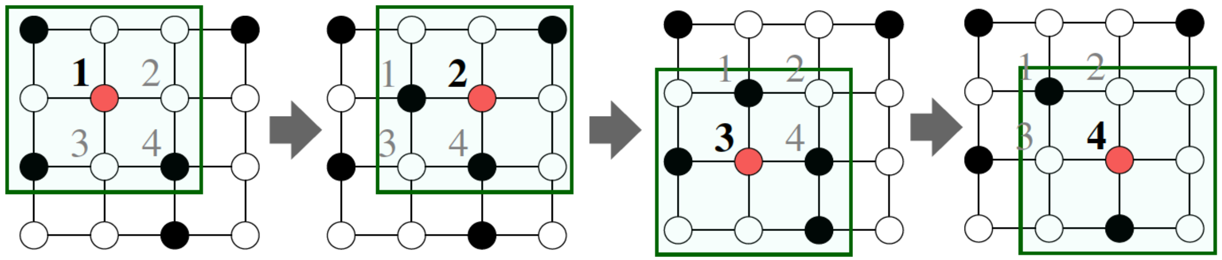


Illustration credit: Semih Cantürk

GNNs and graph convolutions

- **Intuition:** The aggregation over node neighbors in a GNN is akin to a "center-surround" convolutional filter



- **Theory (at a high-level):** GNNs can be derived as a generalization of convolutions to the graph domain, based on *graph Fourier analysis*

Illustration credit : Stanford CS224W

Hamilton, *2020*. “Graph Representation Learning”

Graph Fourier analysis

- **Fact 1:** We can define the **Laplacian operator** for a graph $G = (V, E, A)$

Identity

Adjacency matrix

Symmetric normalized Laplacian

Diagonal degree matrix

$$\mathbf{L}_{\text{sym}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$$

$$= \mathbf{I} - \mathbf{A}_{\text{sym}}$$

Intuition: This Laplacian operator measures how much a signal differs between a node and its immediate neighborhood

Graph Fourier analysis

- **Fact 1:** We can define the **Laplacian operator** for a graph $G = (V, E, A)$

Identity

Adjacency matrix

Symmetric normalized Laplacian

$$\mathbf{L}_{\text{sym}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$$

$$= \mathbf{I} - \mathbf{A}_{\text{sym}}$$

Diagonal degree matrix

Symmetric normalized adjacency matrix

Intuition: This Laplacian operator measures how much a signal differs between a node and its immediate neighborhood

Graph Fourier analysis

- **Fact 1:** We can define the **Laplacian operator** for a graph $G = (V, E, A)$

Symmetric normalized Laplacian

$$\boxed{\mathbf{L}_{\text{sym}}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} = \mathbf{I} - \mathbf{A}_{\text{sym}}$$

$$\boxed{\mathbf{L}_{\text{sym}}\mathbf{x}[u]} = \boxed{\sum_{v \in \mathcal{N}(u)} \frac{\mathbf{x}[u] - \mathbf{x}[v]}{\sqrt{d_u d_v}}}$$

Multiplying a node signal $x \in \mathbb{R}^d$ by the Laplacian

Intuition: This Laplacian operator measures how much a signal differs between a node and its immediate neighborhood

Graph Fourier analysis

- **Fact 2:** The (generalized) eigenfunctions of the Laplace operator on Euclidean space are the Fourier modes (i.e., sinusoidal plane waves)

$$\mathbf{L}_{\text{sym}} = \mathbf{U}\Lambda\mathbf{U}^\top$$

- Fourier analysis in Euclidean space involves projecting an input signal onto the eigenbasis of the Laplace operator

Graph Fourier analysis

- **Key idea:** Define the graph Fourier transform (GFT) by representing a signal in the eigenbasis of the graph Laplacian:

Graph Fourier transform (GFT) of a signal $\boldsymbol{x} \in \mathbb{R}^{|V|}$ on the nodes of graph

Multiplication by the inverse Laplacian eigenvectors.

$$\boxed{\hat{\mathbf{x}} = \mathrm{GFT}(\mathbf{x})} = \boxed{\mathbf{U}^\top \mathbf{x}} \qquad \text{where } \mathbf{L}_{\text{sym}} = \mathbf{U}\Lambda\mathbf{U}^\top$$

$$\boxed{\mathbf{x} = \mathrm{GFT}^{-1}(\hat{\mathbf{x}}) = \mathbf{U}\hat{\mathbf{x}}}$$

Inverse graph Fourier transform

Key interpretation:

Laplacian eigenvectors = Fourier basis

Laplacian eigenvalues = Frequencies

Graph Fourier analysis

- **Key idea:** Define the graph Fourier transform (GFT) by representing a signal in the eigenbasis of the graph Laplacian:

Graph Fourier transform (GFT)

Multiplication by the inverse Laplacian eigenvectors

$$\boxed{\hat{\mathbf{x}} = \mathrm{GFT}(\mathbf{x})} = \boxed{\mathbf{U}^\top \mathbf{x}} \qquad \text{where } \mathbf{L}_{\text{sym}} = \mathbf{U}\Lambda\mathbf{U}^\top$$

$$\boxed{\mathbf{x} = \mathrm{GFT}^{-1}(\hat{\mathbf{x}}) = \mathbf{U}\hat{\mathbf{x}}}$$

Inverse graph Fourier transform

- A **graph convolution** can then be defined by **element-wise multiplication in the graph Fourier domain:**

Convolution of a signal $\boldsymbol{x} \in \mathbb{R}^{|V|}$ by a filter $\mathbf{f} \in \mathbb{R}^{|V|}$

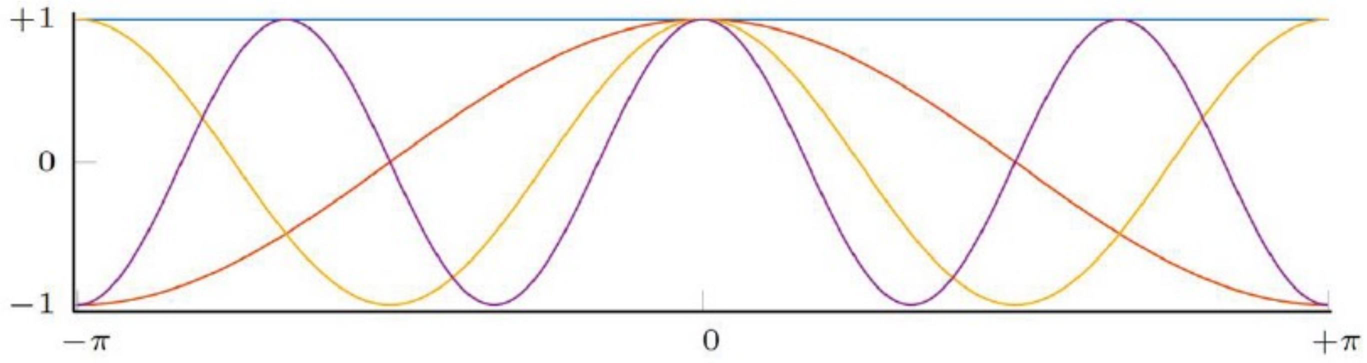
$$\boxed{\mathbf{x} \star_{\mathcal{G}} \mathbf{f}} = \mathbf{U}\left(\boxed{\mathbf{U}^\top \mathbf{x} \circ \mathbf{U}^\top \mathbf{f}}\right)$$

Element-wise multiplication between GFT of the signal and a filter

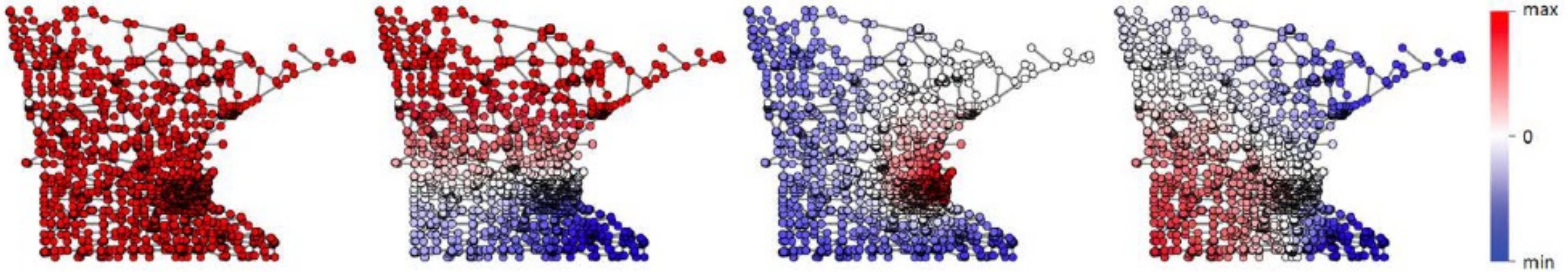
$$= \mathbf{U}\left(\hat{\mathbf{x}} \times \boxed{\mathrm{diag}(\hat{\mathbf{f}})}\right)$$

Representation of a filter as a diagonal matrix of frequencies

Graph Fourier analysis: Examples



First eigenvectors of ring graph Laplacian = classical Fourier basis



ϕ_0 ϕ_1 ϕ_2 ϕ_3

First eigenvectors of the Minnesota road network Laplacian

Illustration credit: [Bronstein et al. (2017)]

Connecting Spectral Convolutions and GNNs

- In practice, we can define graph convolutions as **multiplications by polynomials of the (normalized) adjacency matrix**, since

$$\boxed{p(\mathbf{A}_{\text{sym}})} = \boxed{\mathbf{U}p'(\Lambda)\mathbf{U}^{-1}} \qquad \text{where } \boxed{\mathbf{L}_{\text{sym}} = \mathbf{U}\Lambda\mathbf{U}^{-1}}$$

Any polynomial of the adjacency matrix can be simultaneously diagonalized (i.e., share the same eigenvectors) with the Laplacian

Connecting Spectral Convolutions and GNNs

- In practice, we can define graph convolutions as **multiplications by polynomials of the (normalized) adjacency matrix**, since

$$p(\mathbf{A}_{\text{sym}}) = \mathbf{U}p'(\Lambda)\mathbf{U}^{-1} \qquad \text{where } \mathbf{L}_{\text{sym}} = \mathbf{U}\Lambda\mathbf{U}^{-1}$$

$$\Rightarrow \boxed{p(\mathbf{A}_{\text{sym}})\mathbf{x}} = \boxed{\mathbf{U}}\boxed{p'(\Lambda)}\boxed{\mathbf{U}^{-1}\mathbf{x}}$$

Multiplying a signal by a polynomial of the adjacency matrix is equivalent to multiplying the graph Fourier transform of that signal
by a diagonal matrix (i.e. performing an elementwise multiplication)
and then performing an inverse graph Fourier transform

Connecting Spectral Convolutions and GNNs

- In practice, we can define graph convolutions as **multiplications by polynomials of the (normalized) adjacency matrix**, since

$$p(\mathbf{A}_{\text{sym}}) = \mathbf{U}p'(\Lambda)\mathbf{U}^{-1} \qquad \text{where } \mathbf{L}_{\text{sym}} = \mathbf{U}\Lambda\mathbf{U}^{-1}$$

$$\Rightarrow p(\mathbf{A}_{\text{sym}})\mathbf{x} = \mathbf{U}p'(\Lambda)\mathbf{U}^{-1}\mathbf{x}$$

Multiplying by a **polynomial of the adjacency matrix** $p(\mathbf{A}_{\text{sym}})\mathbf{x}$

gives a valid graph convolution $\mathbf{x} \star_{\mathcal{G}} \mathbf{f} = \mathbf{U}\left(\hat{\mathbf{x}} \times \text{diag}(\hat{\mathbf{f}})\right)$

Comparing the perspectives: Basic MPNN

Message-passing

$$\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}_{\text{self}}^{(k)}\mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)}\sum_{v\in\mathcal{N}(u)}\mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)}\right)$$

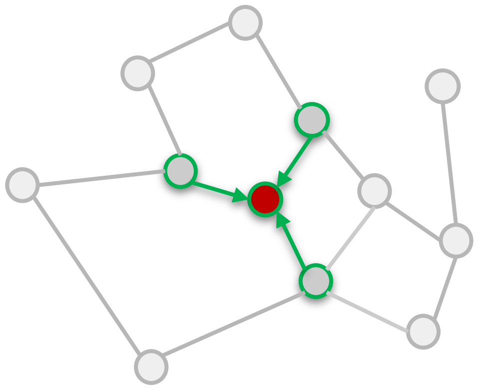
k[th] layer embedding of *v*

non-linearity (e.g., ReLU)

previous layer embedding of *u*

sum of neighbor's previous layer embeddings

$$\mathbf{h}_u^{(k+1)} = \sigma(\mathbf{W}\mathbf{h}_u^{(k)} + \mathbf{W}\mathbf{m}_{\mathcal{N}(u)})$$



Spectral Conv

$$\mathbf{H}^{(k+1)} = \sigma((\mathbf{A}+\mathbf{I})\mathbf{H}^{(k)}\mathbf{W}^{(k)})$$

$$= \sigma(\tilde{\mathbf{A}}\mathbf{H}^{(k)}\mathbf{W}^{(k)})$$

Graph signal to convolve: node embeddings

Degree 1 polynomial filter $p(\mathbf{A}_{\text{sym}})$: 1-hop

Summary of Convolutional View

- GNNs can be derived as a generalization of convolutions to the graph domain, based on graph Fourier analysis
- Multiplying by a polynomial of the adjacency matrix gives a valid graph convolution
- We can do convolutions also in the spectral domain, instead of the graph domain, however without “tricks” it is too expensive
- Both graph domain and spectral domain filtering is an active area of research

Graph Isomorphism and Expressivity

What functions can GNNs represent?

Weisfeiler-Lehman algorithm

- The WL algorithm is used to extract a discrete feature vector for a graph, which can be used to test if two graphs are isomorphic

WL Algorithm

- Initialize each node by the same color.
- For $t=1..T$:
  - For each node:
    - New label ←
      hash (multiset of labels in local neighborhood)
- Represent graph has multiset of labels obtained after T iterations

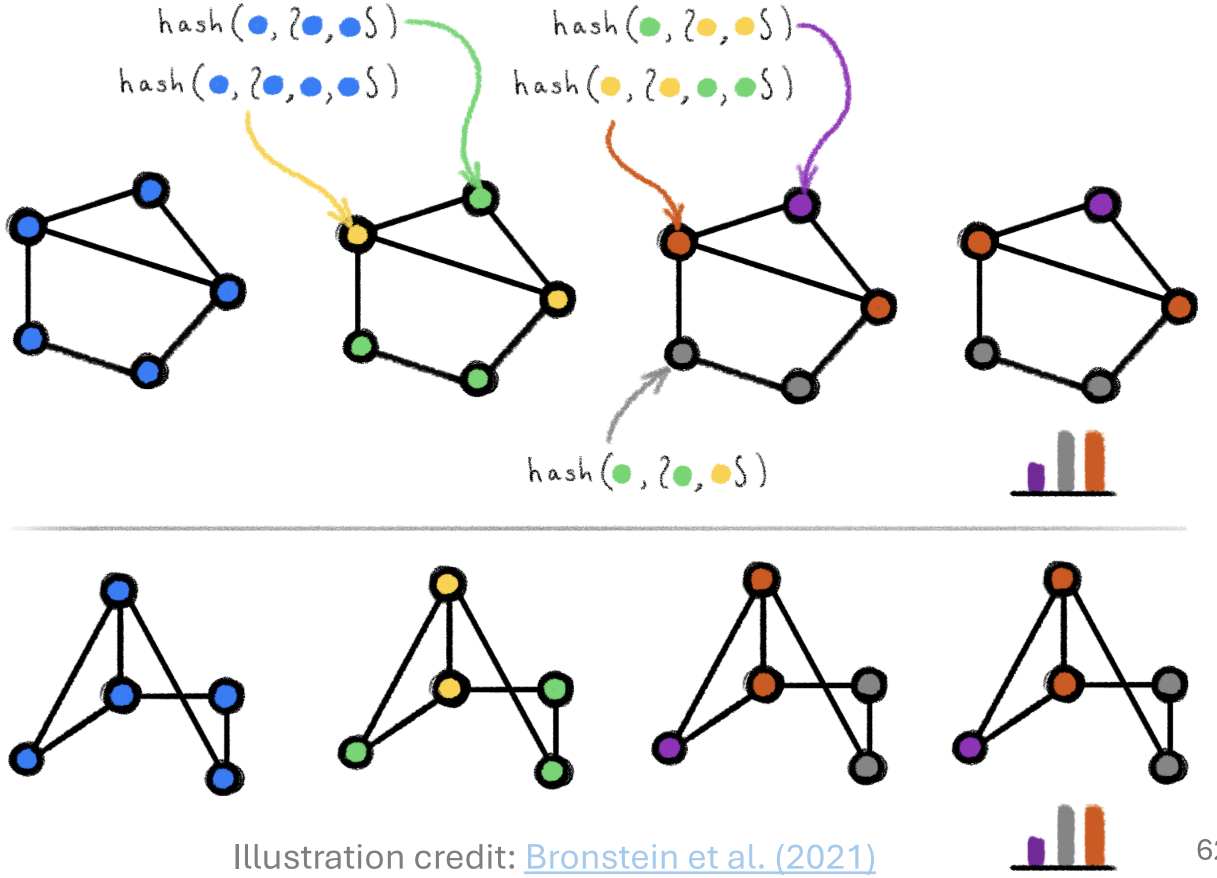


Illustration credit: [Bronstein et al. (2021)]

Weisfeiler-Lehman and GNNs

- GNNs can be viewed as a continuous and differentiable version of the WL test!

WL Coloring Algorithm

- Initialize each node with a discrete label
- For each node:
  - New label ← ***hash***(multiset of labels in local neighborhood)

GNN Message Passing

- Initialize each node with a continuous embedding
- For each node:
  - New embedding ← ***NN***(set of embeddings in local neighborhood)

1-WL Isomorphism Test & Limitations

- Graph isomorphism is hard!
- 1-WL test fails to distinguish some pairs of non-isomorphic graphs (e.g, regular graphs)
- A vanilla message-passing GNN also cannot distinguish such graphs!

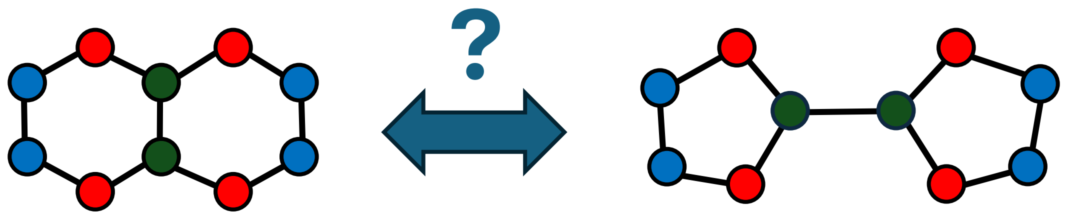


Illustration credit: [Arnaiz-Rodríguez & Velkinger (2024)]

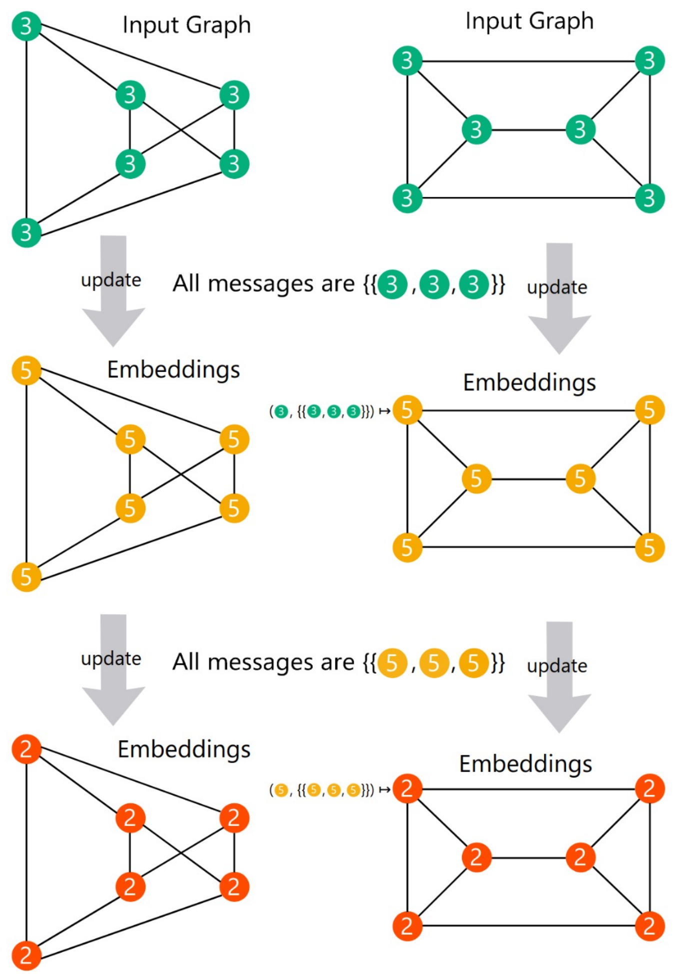


Illustration credit: [Sato (2020)]

1-WL Limitations

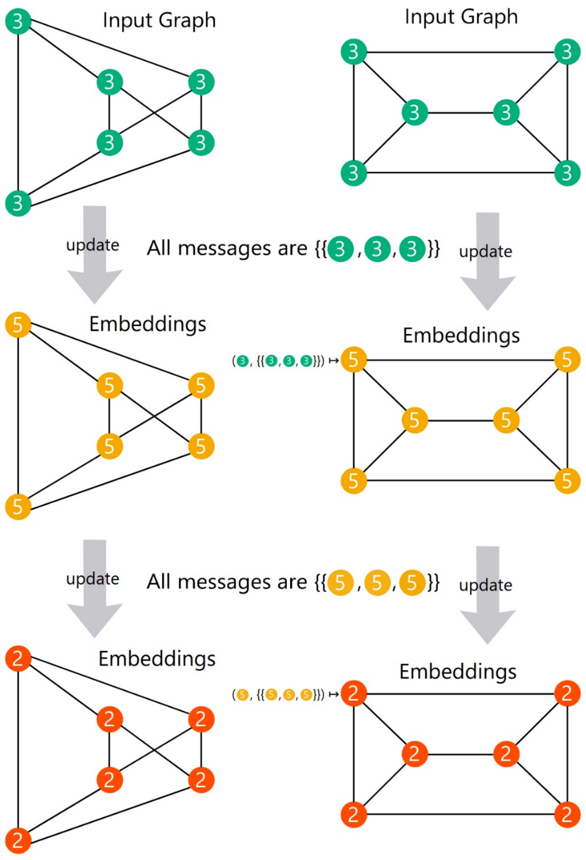


Figure 4: Message passing GNNs cannot distinguish any pair of regular graphs with the same degree and size even if they are not isomorphic.

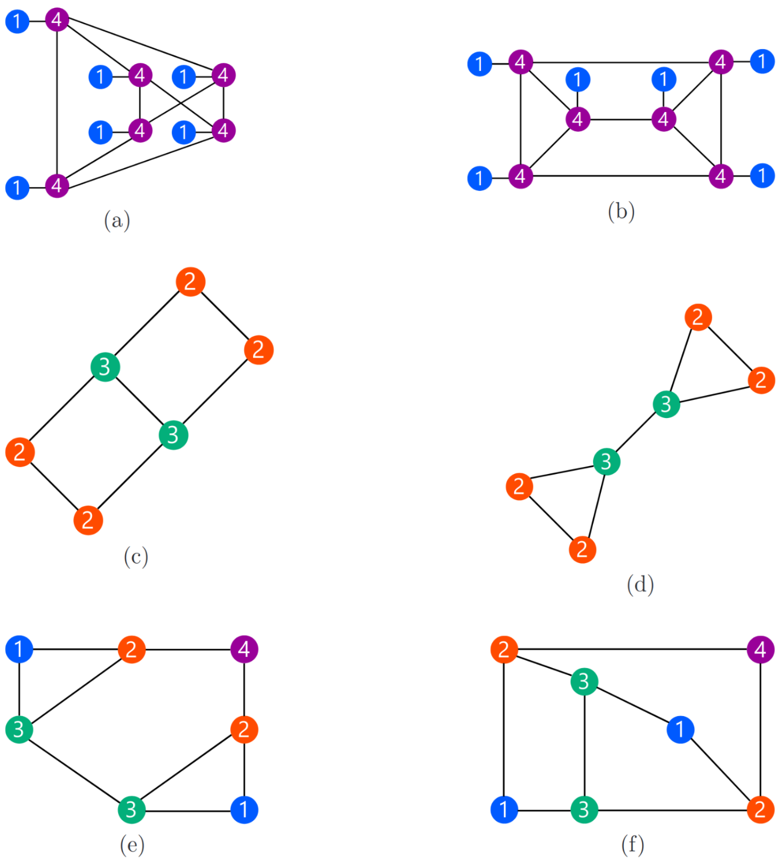


Figure 5: Although these graphs are not isomorphic or regular, GNNs cannot distinguish (a) from (b), (c) from (d), and (e) from (f)

Illustration credit: Sato (2020)

How to Enhance Expressivity?

- Most standard GNNs limited by 1-WL graph isomorphism test
- Ways to improve GNN expressivity:
  - Add features: Easiest way -- even random features breaks symmetries & distinguish/count local structures
    - Positional/structural encodings! More on that later...
  - Modulate message-passing: E.g. anisotropic message aggregation like GAT/attention
  - Modify underlying graph: k-GNNs, graph transformers, expander graphs, graph rewiring strategies

More Resources

- **Distill:** Intro to GNNs, Convolutions on graphs.
  - Excellent introductory material with interactive visualizations
- **PyG Docs:** Colab notebooks & video tutorials
  - More content available in the main documentation directory
- **ICML 2024** Graph Learning Tutorial
  - Similar introductory content, but (a) less applications & no coding content, but (b) goes deeper into specific challenges in graph learning
- **William L. Hamilton**'s GRL book, GRL keynote talk
- **Geometric Deep Learning Course:** Book, Keynote, Lectures

Live coding: Building GNNs with PyG

[Link]

Part II
Challenges in Graph Learning
(and How to Solve Them)

Part II: Outline

Caveats of Message Passing Neural Networks

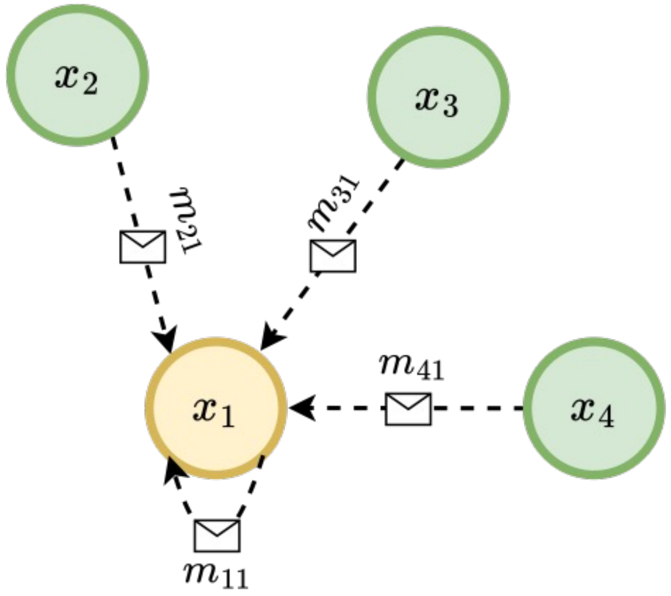
- Under-reaching
- Over-smoothing
- Over-squashing

Solutions

- Rewiring
- Advanced GNNs
- Graph Transformers

Message Passing Neural Networks (MPNN)

- Features at each node x_i, maybe also each edge $e_{i \to j}$
- Message from node j to node i: $m_{j \to i} = f_e(x_j, x_i, e_{j \to i})$
- New features $x_i^+ = f_v(x_i, \sum_j m_{j \to i})$ for some NN f_v



Some Parameters

1. Graph Diameter (d)

2. Problem Radius (r ≦ d)

    - Local dependencies
    - Long range dependencies

3. Reach capacity of GNN (k)

    - Number of layers in MPNNs

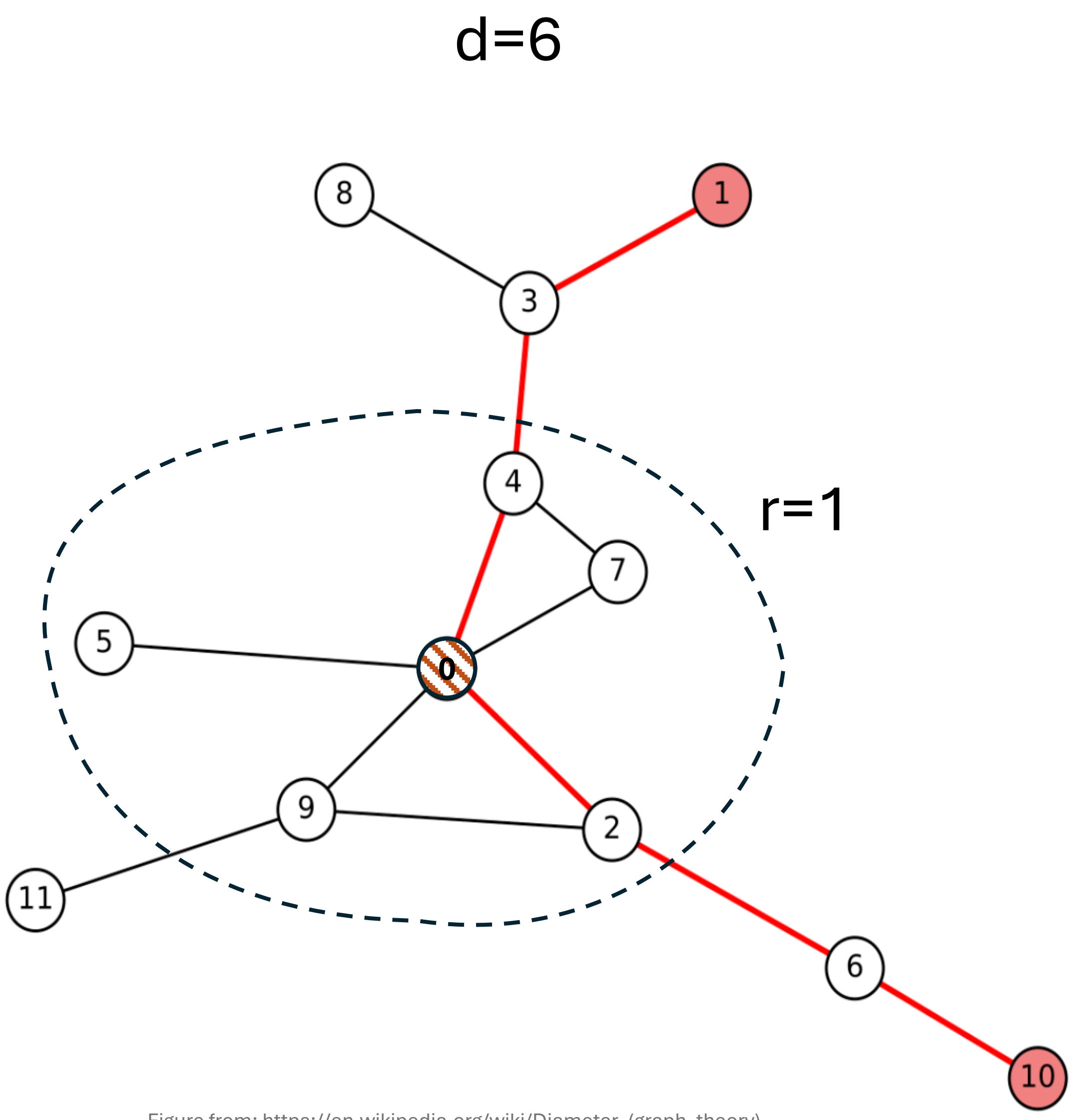
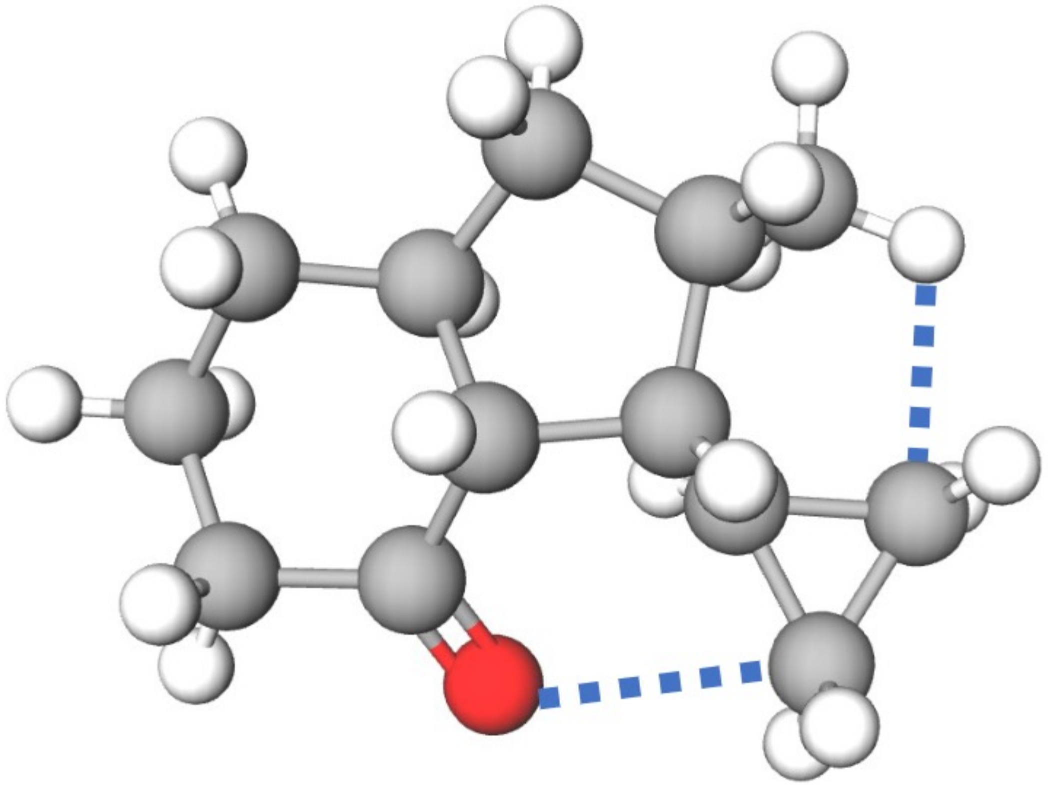


Figure from: https://en.wikipedia.org/wiki/Diameter_(graph_theory)

Under-reaching



Long-range 3D atomic contact not captured by the structure
[Dwivedi et al., 2022]

Under-reaching

- **Problem**
  - Nodes can only access information within k hops (limited receptive field).
  - Information cannot propagate beyond k GNN layers.
  - If k < r (where r = problem radius), the model under-reaches.
- **Implications**
  - Long-range dependencies are missed.
  - r often grows with graph size → required k scales with graph.
- **Solution**
  - Stack more layers (k > r) to enable information exchange across distant nodes.

⚠ Spoiler alert: there are some consequences

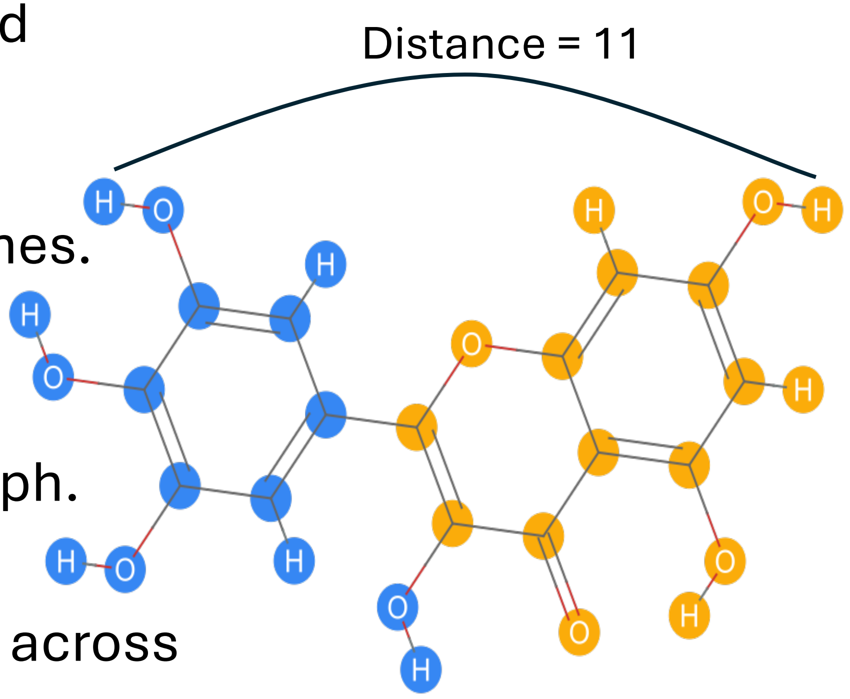
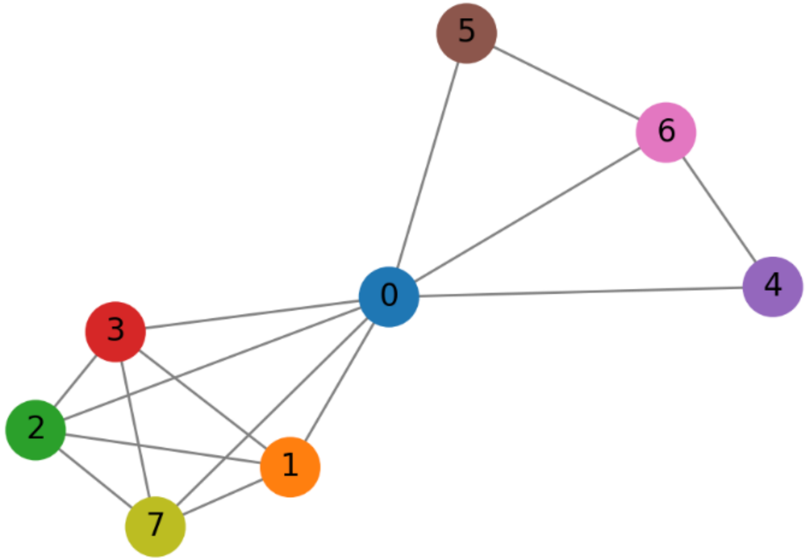


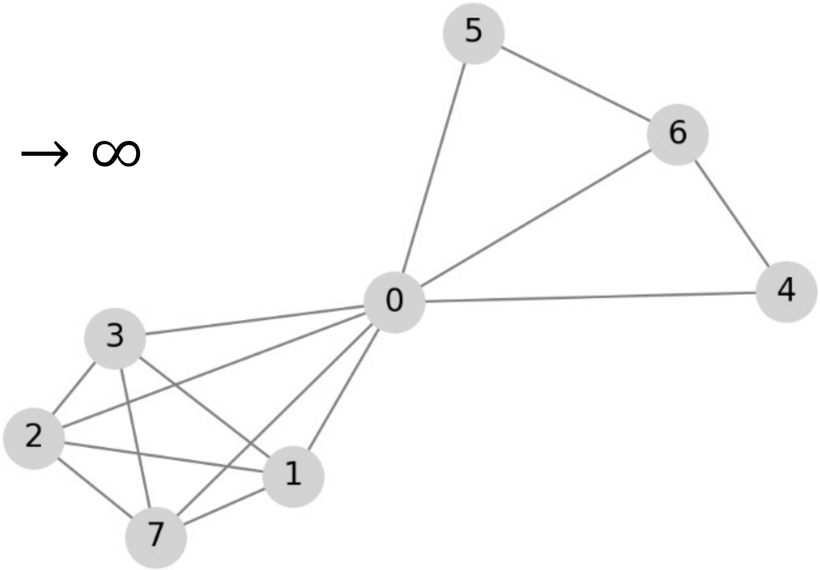
Figure from: https://www.wolfram.com/language/12/molecular-structure-and-computation/molecule-graphs.html.en

Over-smoothing

With many stacked layers, node embeddings become **indistinguishable**.

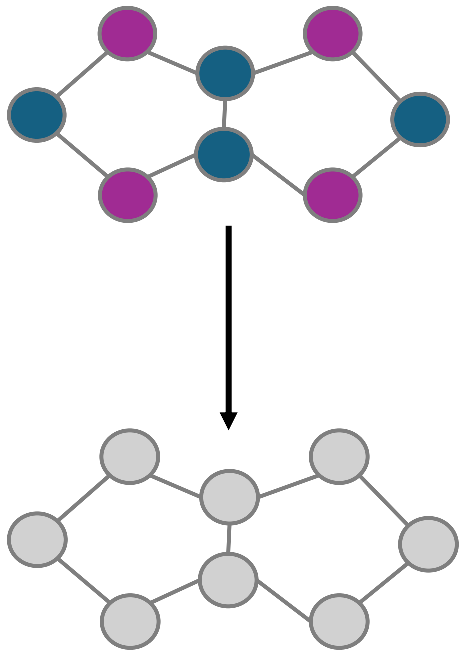


$$\sum_{(u,v)\in E} | h_u^k - h_v^k | \to 0 \quad as\ k \to \infty$$



Over-smoothing in GNNs

- **Problem**
  - Stacking many GNN layers → node embeddings become **indistinguishable**
- **Cause**
  - Too much mixing of information
  - Depends on:
    - Graph connectivity (*G*)
    - GNN architecture
- **Key properties**
  - Independent of problem radius (*not about k < r*)
  - Leads to convergence of embeddings across all nodes
- **Takeaway**
  - More layers ≠ more expressive → risk of over-smoothing

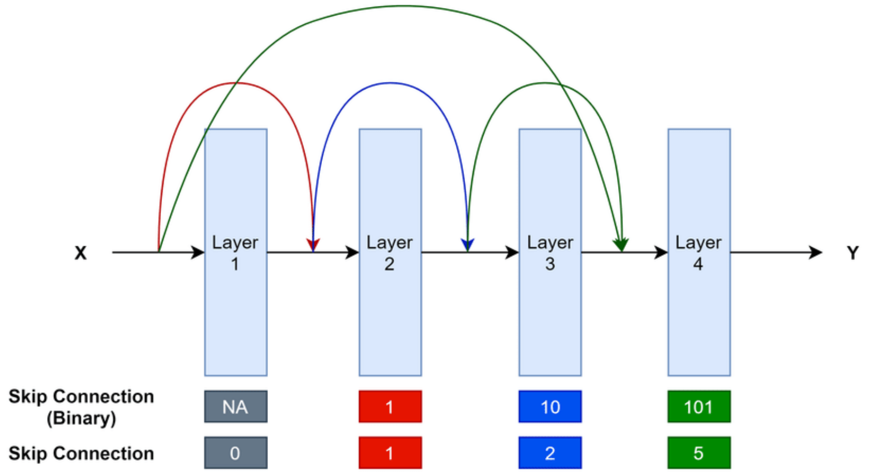


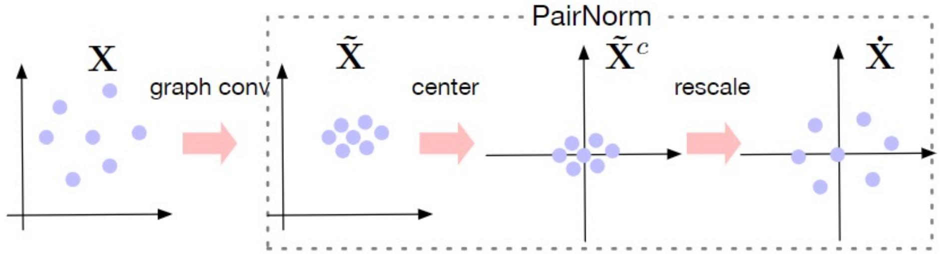
$$\sum_{(u,v)\in E} | h_u^k - h_v^k | \to 0 \quad as\ k \to \infty$$

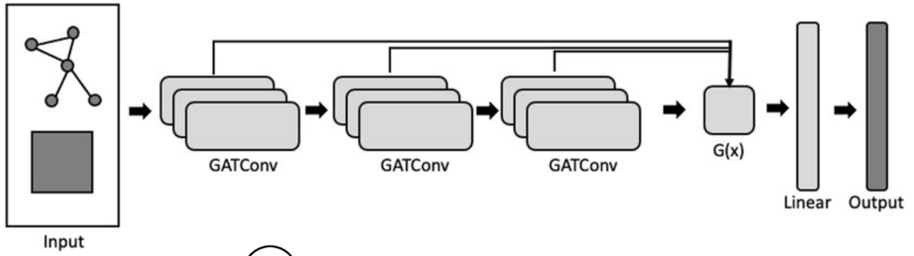
Figure and equation from: ICML 2024 tutorial on Graph Learning

Some over-smoothing solutions

- **Architectural techniques**
  - **Residual / skip connections** → preserve raw node features across layers
  - **Normalization layers** (BatchNorm, GraphNorm, PairNorm) & **Regularization** → stabilize training
  - **Jumping knowledge networks** → adaptively combine representations from different layers
- **Alternative designs**
  - **Attention mechanisms** → selective information propagation
  - **Hybrid Graph Filters** → Low pass filters cause oversmoothing, combine them with band-pass filters (Graph Scattering)







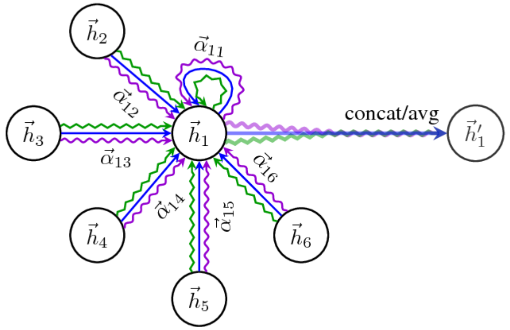
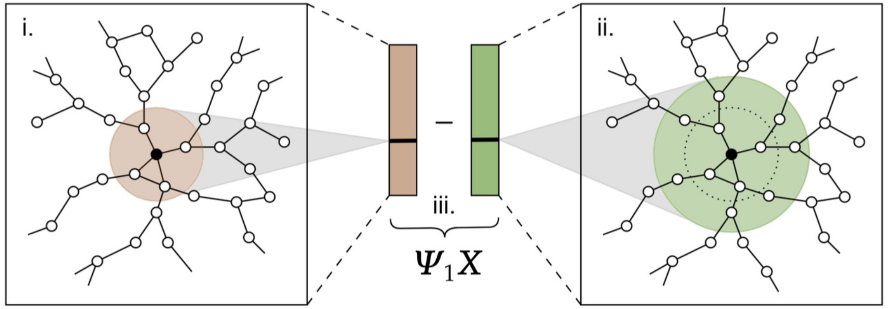
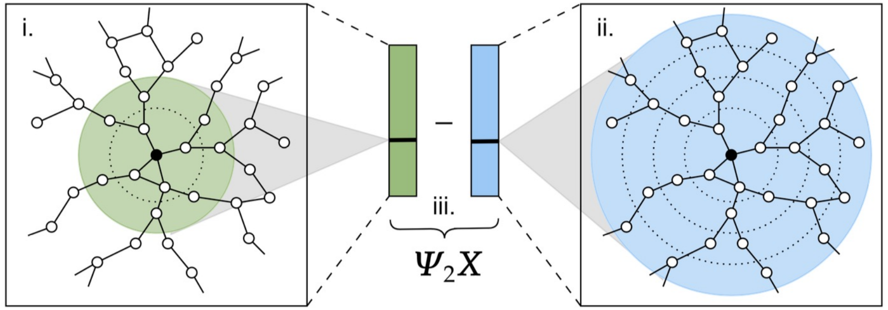


Figure sources: 1. Li, Yaoman, and Irwin King. "Autograph: Automated graph neural network." International conference on neural information processing.
2. Zhao, Lingxiao, and Leman Akoglu. "Pairnorm: Tackling oversmoothing in gnns. (ICLR 202)
3. Eschenburg, et al. "Learning cortical parcellations using graph neural networks." Frontiers in neuroscience 15
4. Veličković, Petar, et al. "Graph attention networks."

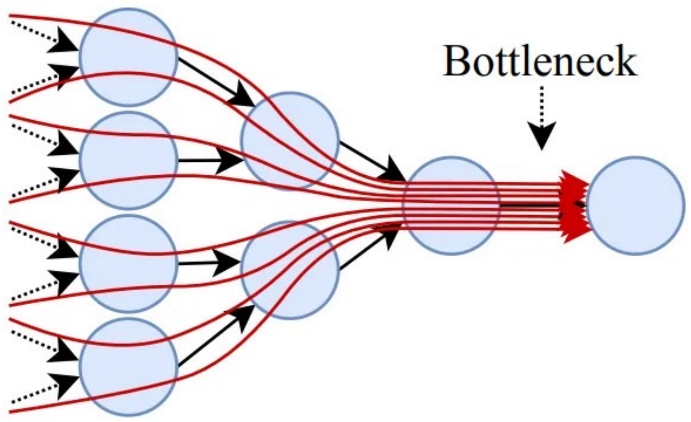
Geometric scattering

- Can we handle over-smoothing and under-reaching by better filter design?
- **GCN filters**: Low-pass filtering *only*
- GS based on **graph diffusion wavelets :**
  - Filters based on powers 2^i of the the lazy random walk matrix $\boldsymbol{P} := \frac{1}{2}(\boldsymbol{I}_n + \boldsymbol{W}\boldsymbol{D}^{-1})$
  - Subtract these low-pass filters at different scales
  - Capture information at different frequency bands $\boldsymbol{\Psi}_k := \boldsymbol{P}^{2^{k-1}} - \boldsymbol{P}^{2^k}$
- Larger receptive fields resolve under-reaching
- **Band-pass filtering** avoids over-smoothing
- **Hybrid scattering** combines both low- and band-pass filters

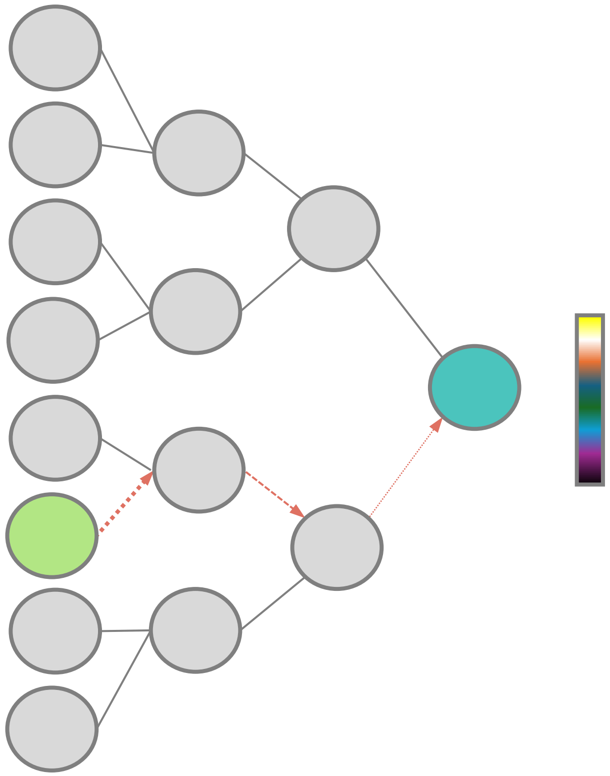




Figures from: Fredrick Wenkel PhD thesis



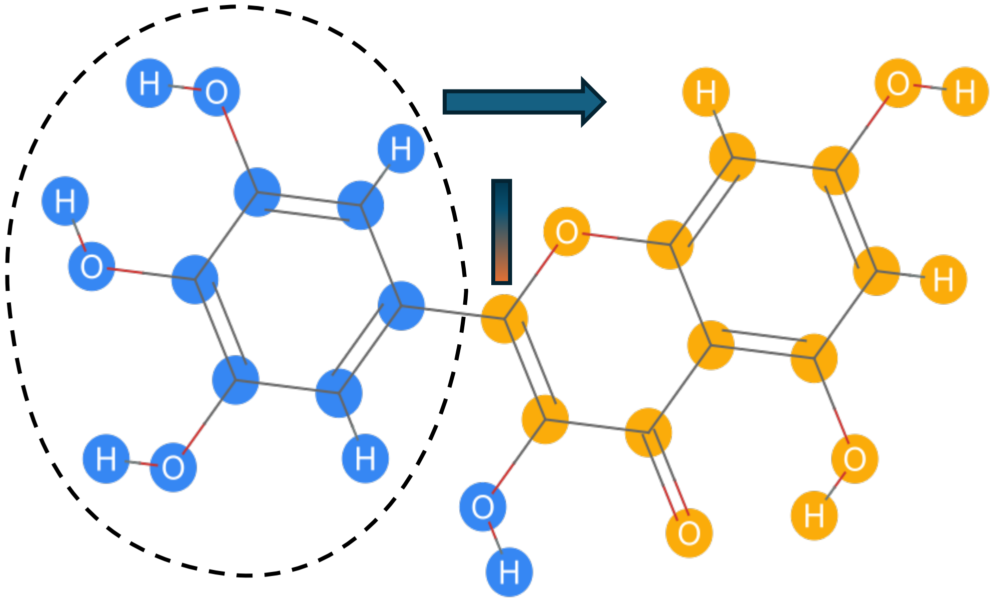
Over-squashing



Information bottlenecks appear in the message passing process

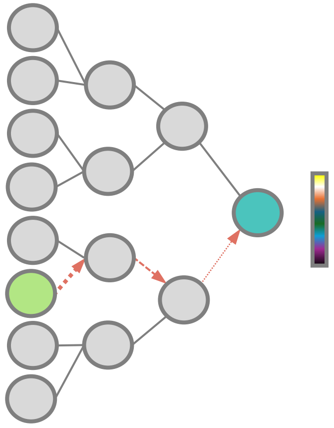
Number of nodes in the receptive field increases exponentially with the depth

Figures from: 1. Alon et al. "On the bottleneck of graph neural networks and its practical implications."
2. : ICML 2024 tutorial on Graph Learning



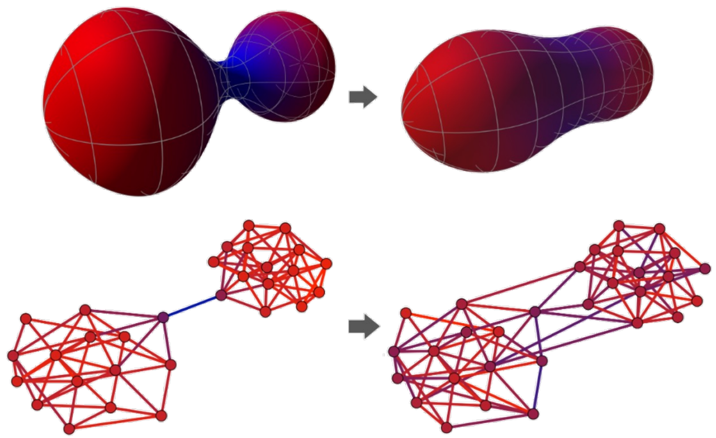
Over-squashing

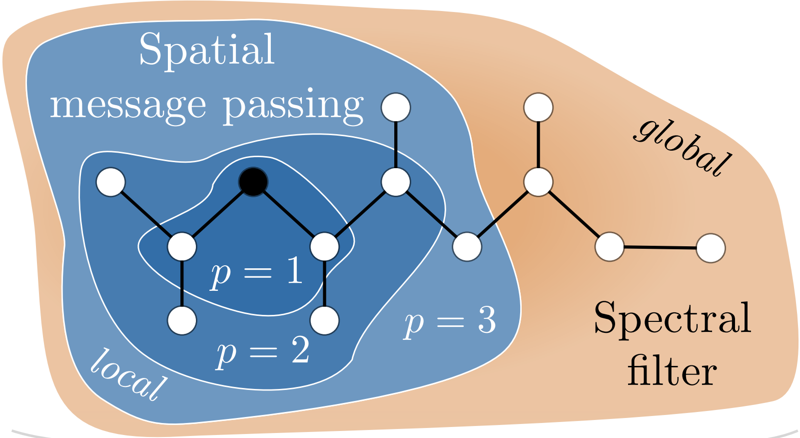
- **Problem**
  - Number of nodes in the receptive field grows **exponentially** with depth.
  - Information from many distant nodes must be **compressed** into fixed-size node embeddings.
  - Too much information needs to pass through a single node or a small number of nodes, causing **information bottlenecks**.
- **Effects**
  - Information from distant nodes is lost or distorted.
  - Long-range dependencies fail.
- **Key takeaway**
  - Even with enough layers, bottlenecks limit effective information flow.



Solutions to over-squashing

- **Graph rewiring**
  - Add edges/nodes to bypass bottlenecks (e.g., curvature-based, spectral, or diffusion-based rewiring).
  - Improves connectivity and reduces path lengths.
- **Model capacity**
  - Increase hidden dimension size
    → more room to encode distant information.
  - Use attention to prioritize important signals.
- **Advanced Architectures**
  - Higher order GNNs.
  - Combining with spectral methods.
  - Graph Transformers ...

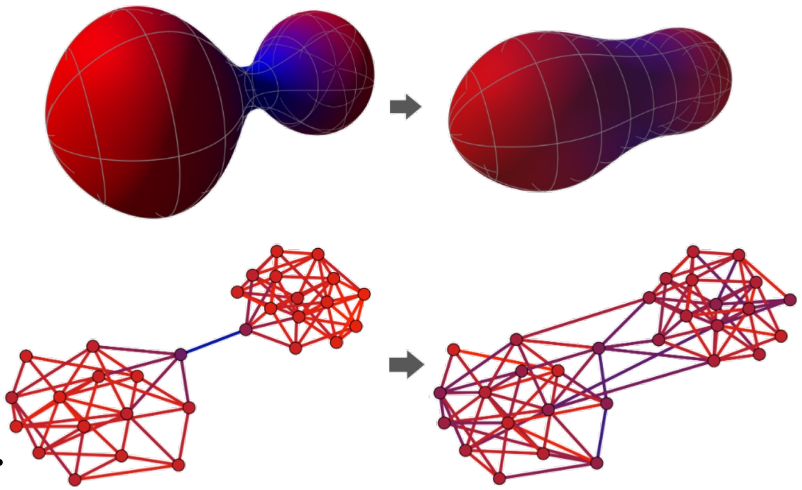


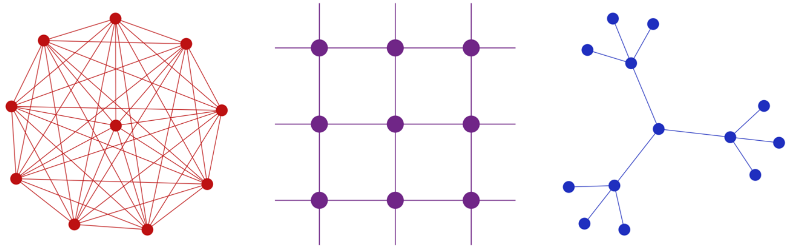


Figures from: 1. Topping, Jake, et al. "Understanding over-squashing and bottlenecks on graphs via curvature."
2. : Geisler, Simon Markus, et al. "Spatio-spectral graph neural networks."

Graph rewiring

- **Idea**
  - Modify graph edges to improve message passing.
  - Helps connect distant nodes, reduce bottlenecks, and highlight relevant interactions.
- **Types of Rewiring**
  - **Spatial**: add edges locally (within *k*-hop).
  - **Spectral**: add edges based on global connectivity measures.
- **Static vs Dynamic**
  - **Static**: rewiring fixed before training.
  - **Dynamic**: edges updated during training, adapt to task.
- **Benefit**
  - Mitigates over-squashing by improving long-range communication.



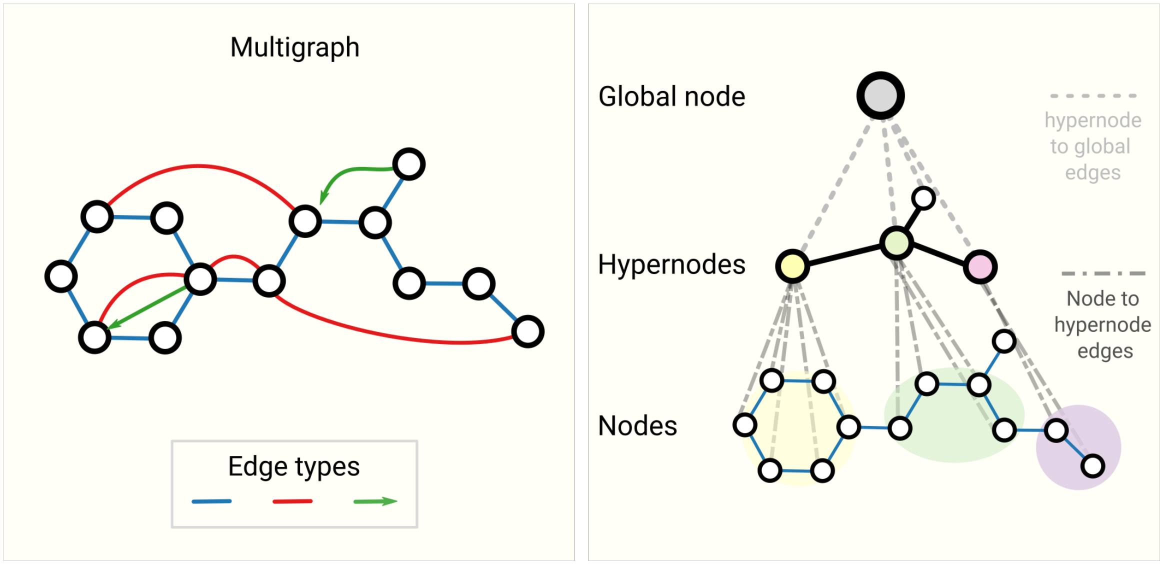


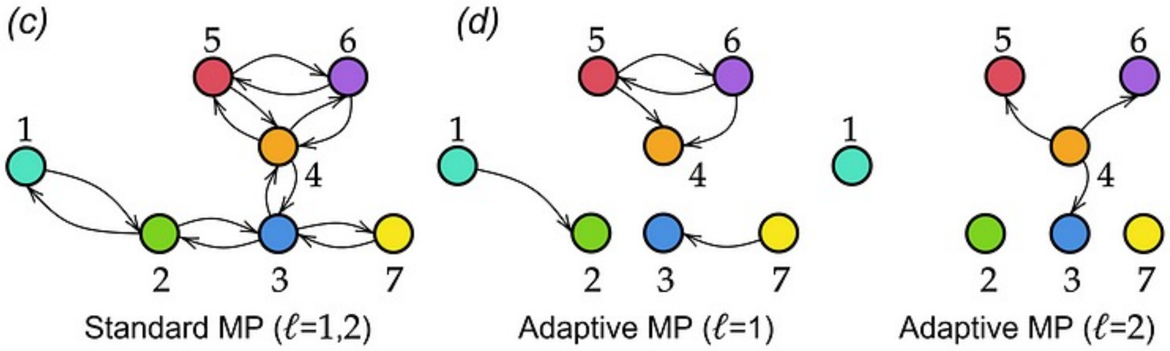
(a) Clique (> 0) (b) Grid (= 0) (c) Tree (< 0)

Figures from: 1 & 2. Topping, Jake, et al. "Understanding over-squashing and bottlenecks on graphs via curvature."

Advanced GNNs

- Higher order GNNs
- Hierarchical GNNs
- Adaptive Message Passing
- …





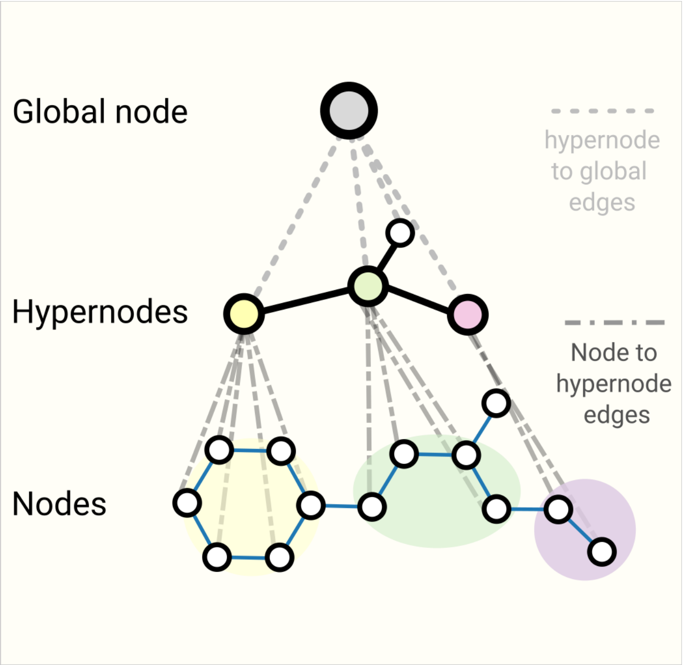
Figures from: 1. https://distill.pub/2021/gnn-intro/
2. :Errica, Federico, et al. "Adaptive message passing: A general framework to mitigate oversmoothing, oversquashing, and underreaching."

Higher Order GNNs

- **Motivation**
  - Standard GNNs operate on **pairs of nodes (edges)**.
  - Higher-order GNNs extend message passing to **k-tuples of nodes**.
  - Captures **richer relational structures** (motifs, subgraphs, hyperedges).
- **Examples**
  - **2-WL GNNs**: work on pairs of nodes, more expressive than standard MPNNs.
  - **Subgraph GNNs**: operate on induced subgraphs around nodes or edges.
- **Key advantage**
  - Higher expressive power → can distinguish graphs that standard GNNs cannot (e.g., certain regular graphs).
  - Gives a **richer communication channel** that bypasses narrow single-edge bottlenecks.

Hierarchical GNNs

- **Motivation**
  - Capture information at **different levels of granularity**
  - From small neighborhoods → larger scale structures
- **Methods**
  - **Graph clustering**: group nodes into clusters
  - Combine **original nodes** and **cluster-level nodes**
  - Multi-level hierarchy possible (recursive clustering)
- **Advantages**
  - Long-distance nodes become **closer in higher levels**
  - Improves efficiency and long-range information flow



Adaptive GNNs

- **Motivation**
  - Not all nodes need to send/receive messages in every round.
  - Each node can act as **sender**, **receiver**, both, or nothing.
- **Methods**
  - Train an auxiliary GNN to decide **when to pass messages**.
  - Introduce **half-hop nodes** for flexible communication.
- **Advantages**
  - Receptive fields grow **adaptively and controllably**.
  - Enables deeper GNNs while limiting unnecessary message passing.
  - Focuses long-path communication only where needed.

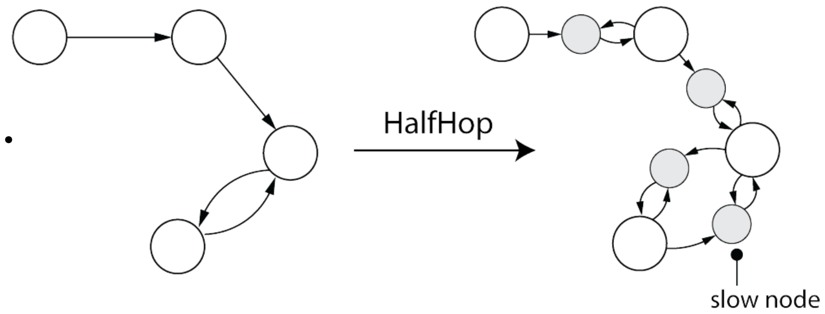
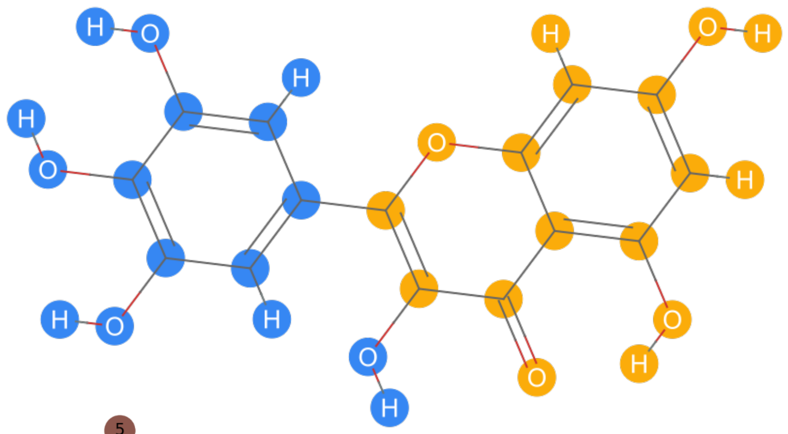
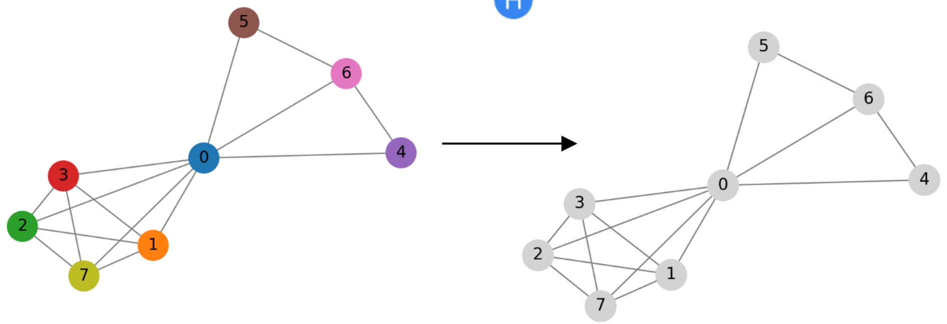


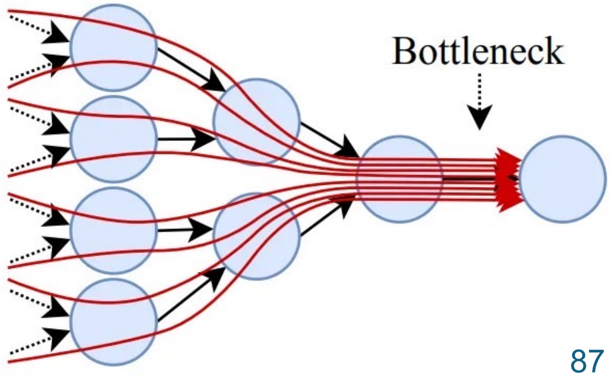
Figure from: Azabou, Mehdi, et al. "Half-hop: A graph upsampling approach for slowing down message passing."

Summary of challenges

- **Propagation limits**
  - Information flows only along graph edges → hard to capture **long-range dependencies**
  - **Solution**: deeper GNNs (more layers)
- **Over-smoothing**
  - Too many layers → node features become **indistinguishable**
  - **Solutions**: residual/skip connections, normalization, jumping knowledge, alternative designs
- **Over-squashing**
  - Graph **bottlenecks** restrict information flow
  - Large receptive fields → information **over-compressed** into embeddings
  - **Solutions**: graph rewiring, larger hidden dimensions, advanced architectures (e.g., higher-order, attention-based)





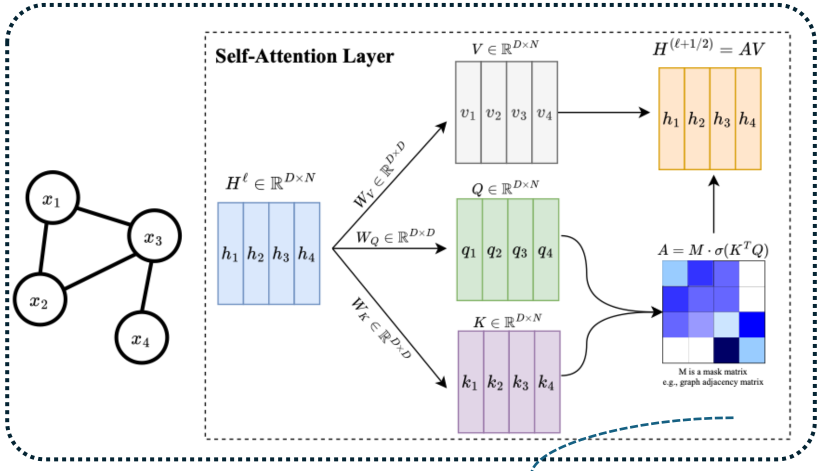


Graph Transformers

An alternative to MPNNs?

Graph Transformers

- Decoupling **graph structure** from the **computation graph**



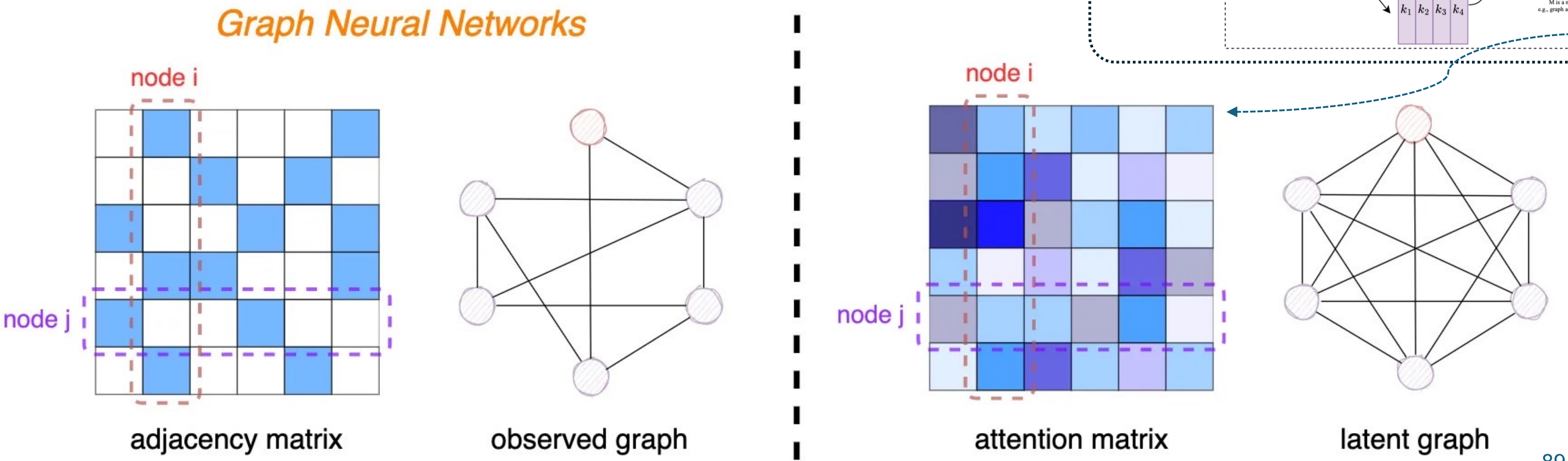
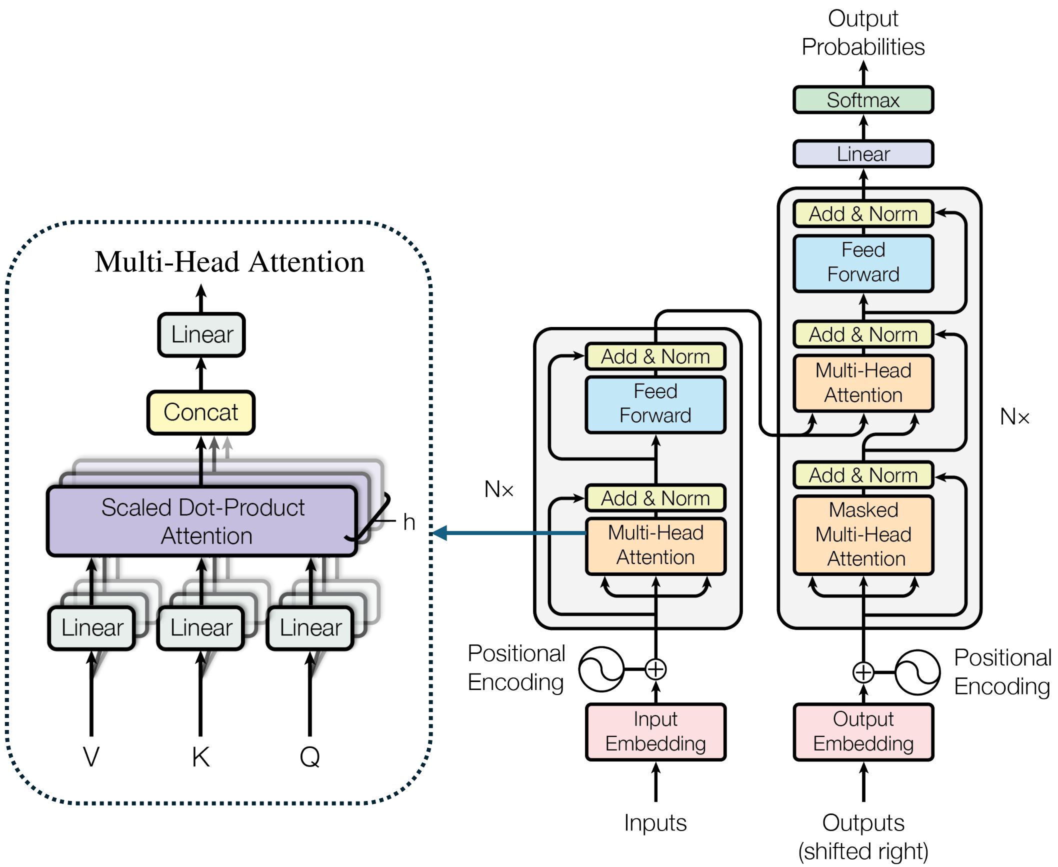


Figure from: How to Build Graph Transformers with O(N) Complexity, Qitian Wu, Medium blog post, with some modification

Transformers

- LLM era: Next token prediction based on a past context window
- Key element: “Attention is All You Need!”
- All pair attention mechanism!

- Sequence of words: aka line graph
- Dense attention



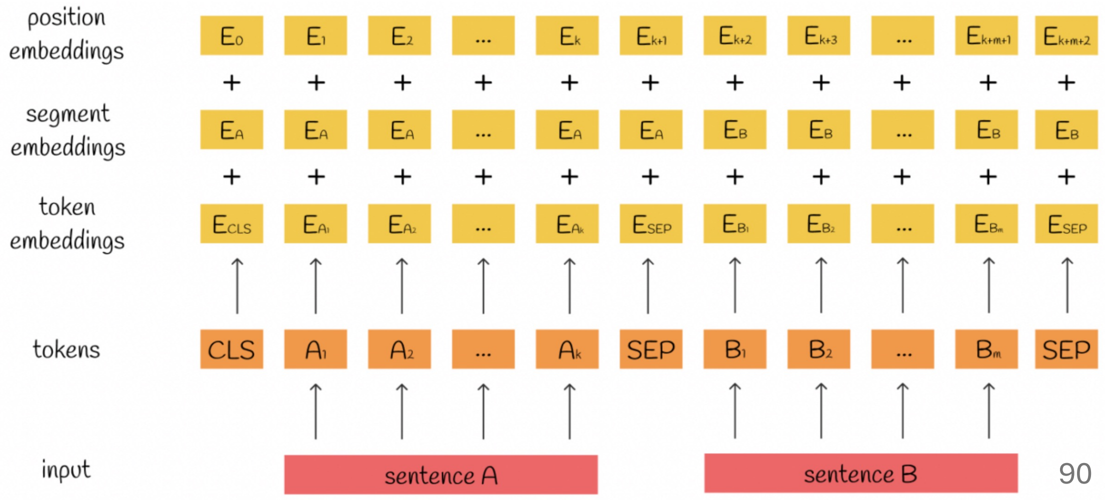


Figure from: 1. Vaswani et al. “Attention is All you Need”
2. Large Language Models: BERT — Bidirectional Encoder Representations from Transformer, Medium blog post by Vyacheslav Efimov

Graph Transformers

- Similar Attention mechanism
- Tokens:
  - Nodes
  - Edges
  - Subgraphs
  - ...

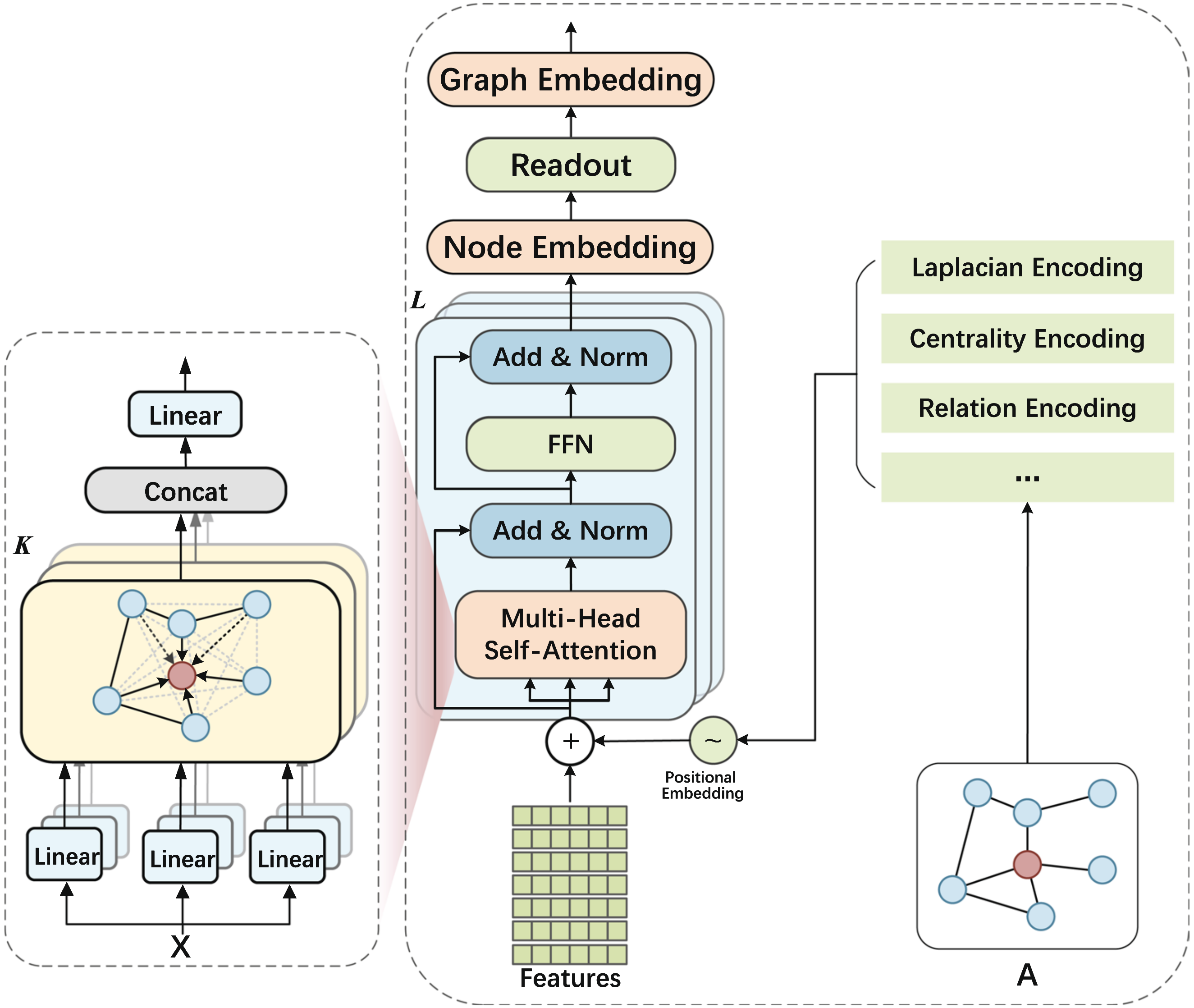


Figure from: Sun, Chengcheng, et al. "Attention-based graph neural networks: a survey." *Artificial intelligence* 2023

Message Passing vs. Graph Transformers

Message Passing

Updates across edges of input graph

✅ Captures inductive bias from input graph topology

✅ Efficient computation: $\mathcal{O}(nd^2 + md)$

❌ Difficulty with long-range dependencies

❌ Oversmoothing, oversquashing

Graph Transformers

Use global attention

✅ All-pair attention, no information bottleneck

✅ Long-range modelling

❌ Loss of inductive bias from graph

❌ Inefficient computation: $\mathcal{O}(nd^2 + n^2d)$

- Usually in graphs $m \ll n^2$

n: number of nodes, m: number of edges, d: hidden dimension of the network

Message Passing vs. Graph Transformers

Message Passing

Updates across edges of input graph

✅ Captures inductive bias from input graph topology

✅ Efficient computation: $\mathcal{O}(nd^2 + md)$

❌ Difficulty with long-range dependencies

❌ Oversmoothing, oversquashing

Graph Transformers

Use global attention

✅ All-pair attention, no information bottleneck

✅ Long-range modelling

❌ Loss of inductive bias from graph

Positional and structural encodings

❌ Inefficient computation: $\mathcal{O}(nd^2 + n^2 d)$

- Usually in graphs $m \ll n^2$

n: number of nodes, m: number of edges, d: hidden dimension of the network

Positional/ Structural Encodings

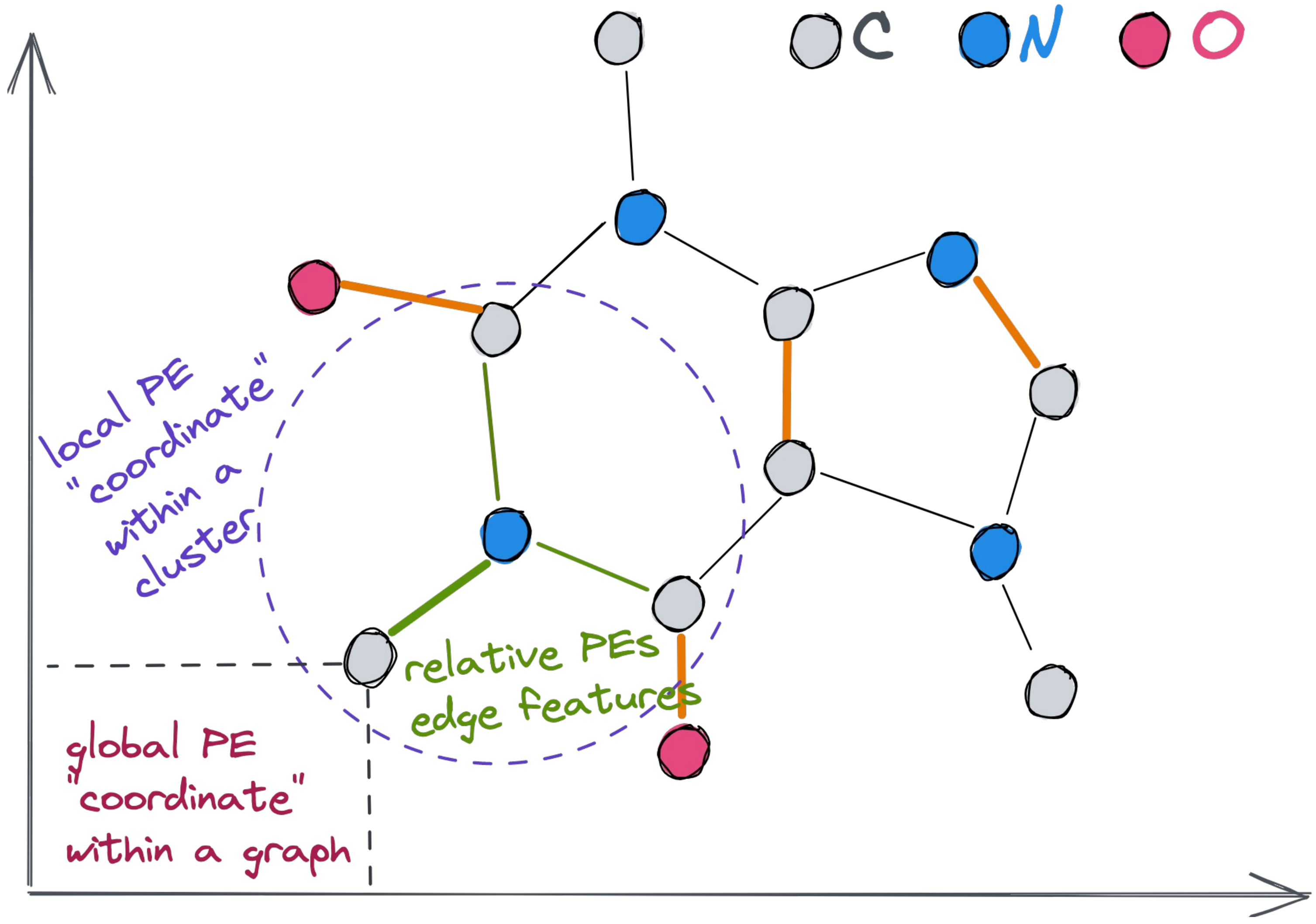
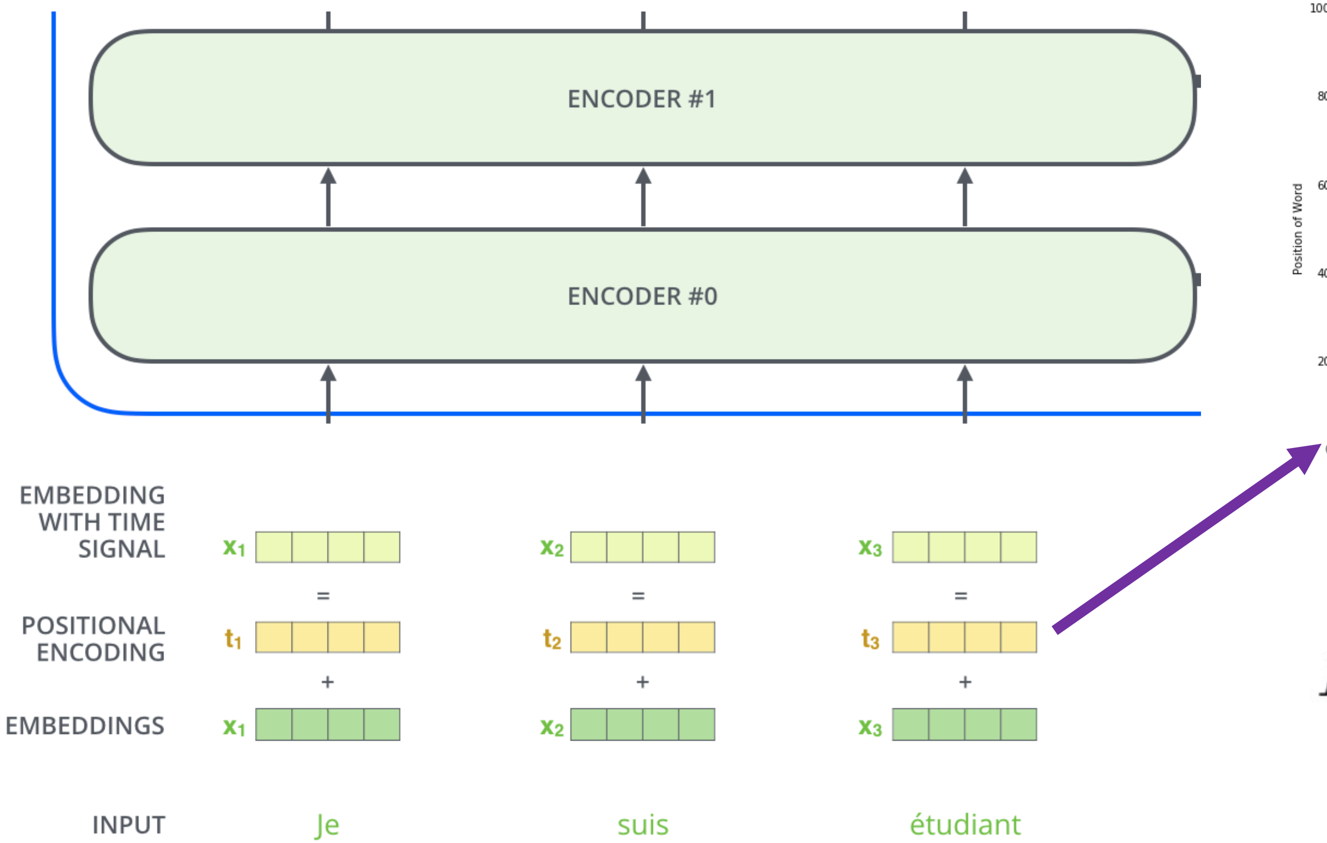
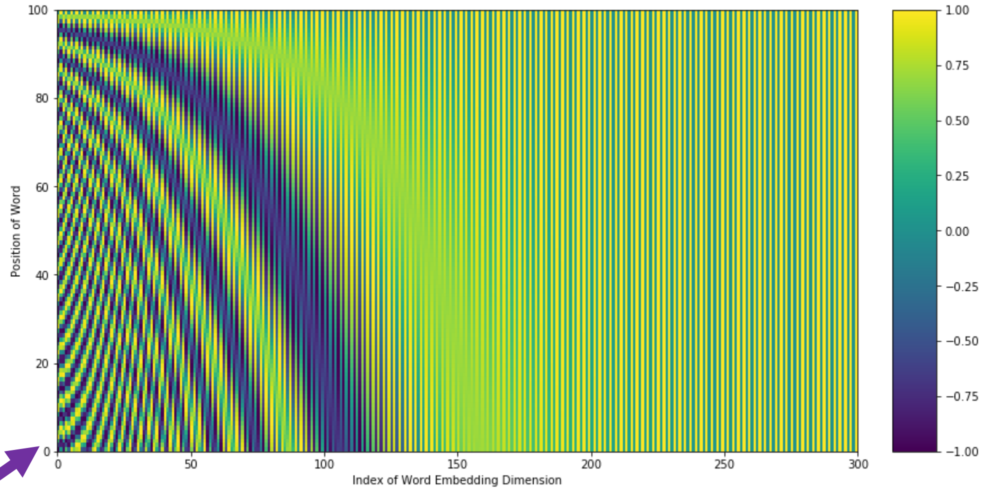


Figure from: GraphGPS: Navigating Graph Transformers, blog post by Michael Galkin

Positional Encodings (PE) for Sequences





$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

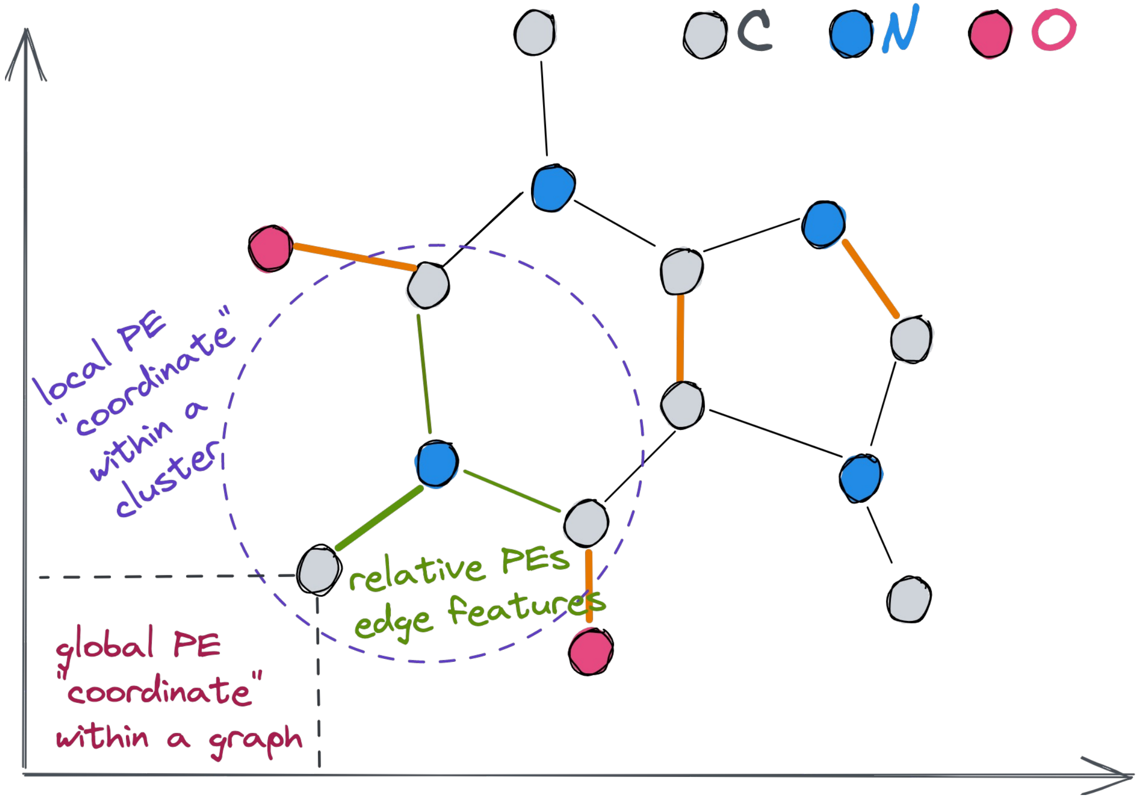
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

PEs borrowed from NLP, where they denote the position of each word, making the embeddings of closer-by words more similar

Figure from: Jay Alammar. The Illustrated Transformer

Graph Positional and Structural Encodings

- Can provably improve expressive power of GNNs
- Organized into 3 categories based on their locality:
  - **Local**
  - **Global**
  - **Relative**



Laplacian Eigenvectors and Eigenvalues

$$L = D - A = U^T \Lambda U$$

Laplacian eigenvectors = Fourier basis
Laplacian eigenvalues = Frequencies

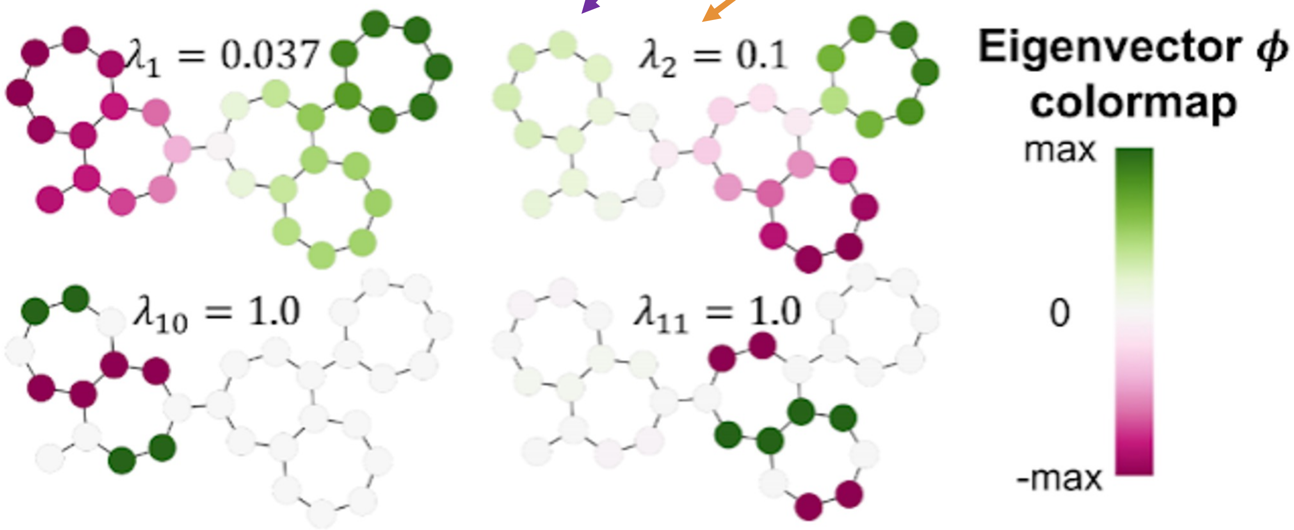


illustration: Kreuzer et al. Rethinking Graph Transformers with Spectral Attention, 2021
Belkin and Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation, 2003

Many more: RWSE, HKSE, ElectrostaticSE

- **RWSE:** Random walk landing probabilities after   steps: Diagonal of RW matrix
- **HKSE:** Represents a Gaussian in Euclidean space, solves the diffusion equation
- **ElstaticSE:** Kernel based on the electrostatic interaction between nodes

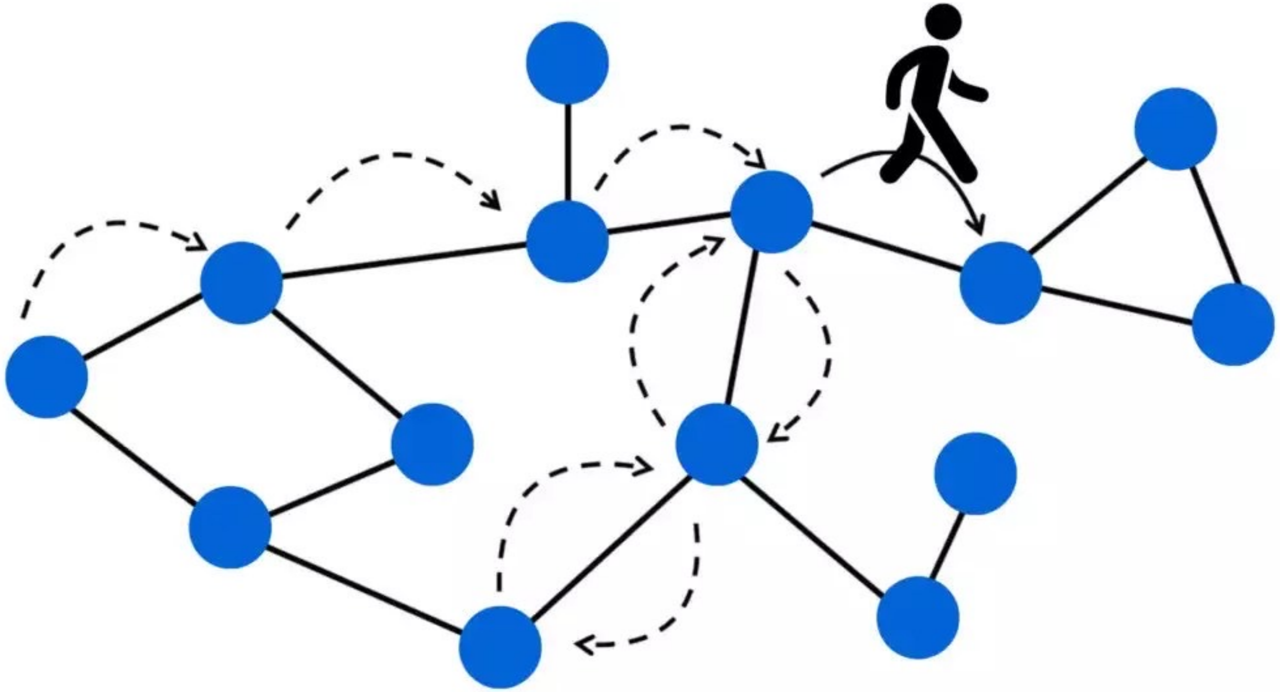
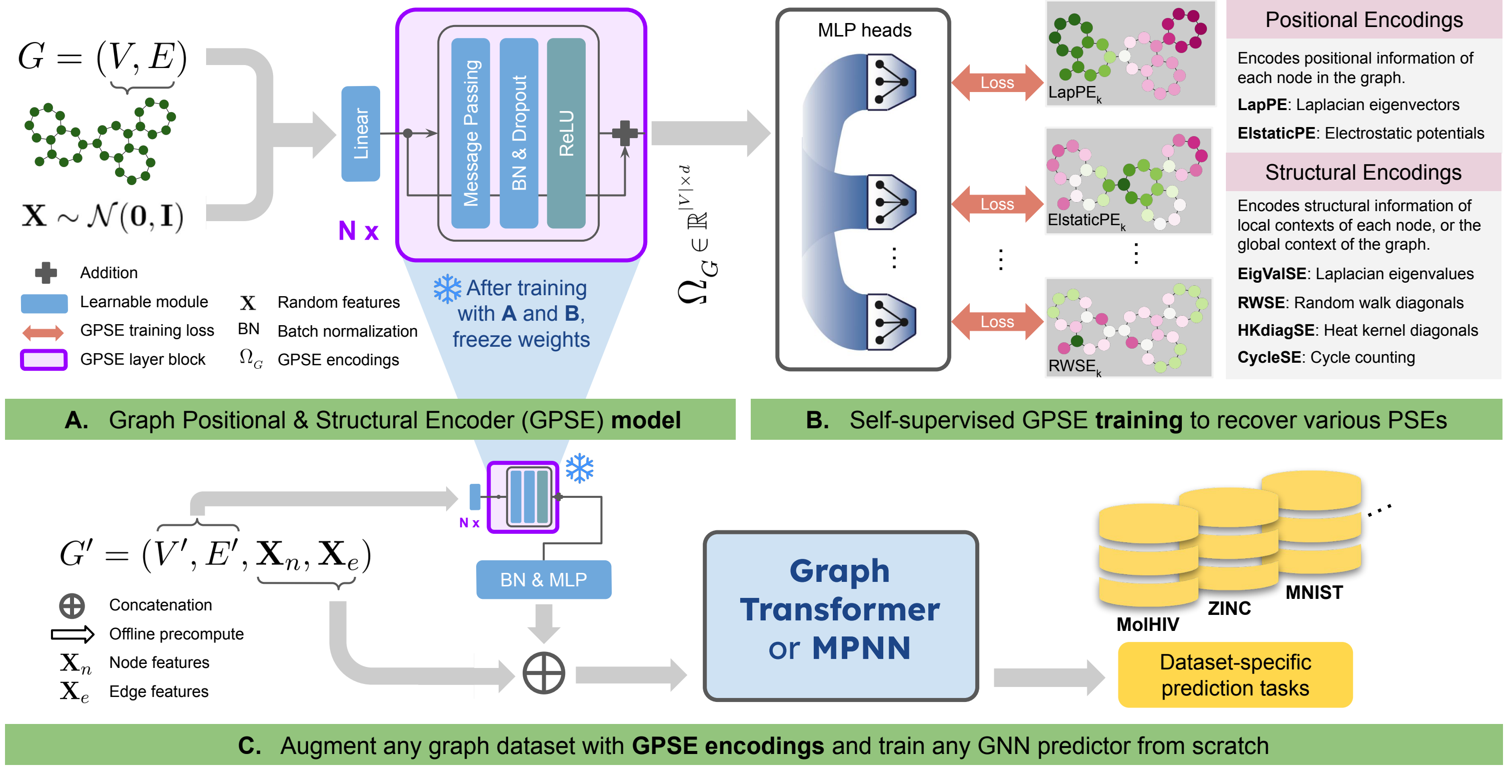


Figure from: What is Node2Vec?, blog post by Neri Van Otten

GPSE

- Graph Positional and Structural Encoder (GPSE, Cantürk et al., 2024): A self-supervised learning of various positional encodings.
- An MPNN leans to generate the positional encodings from Gaussian initializations

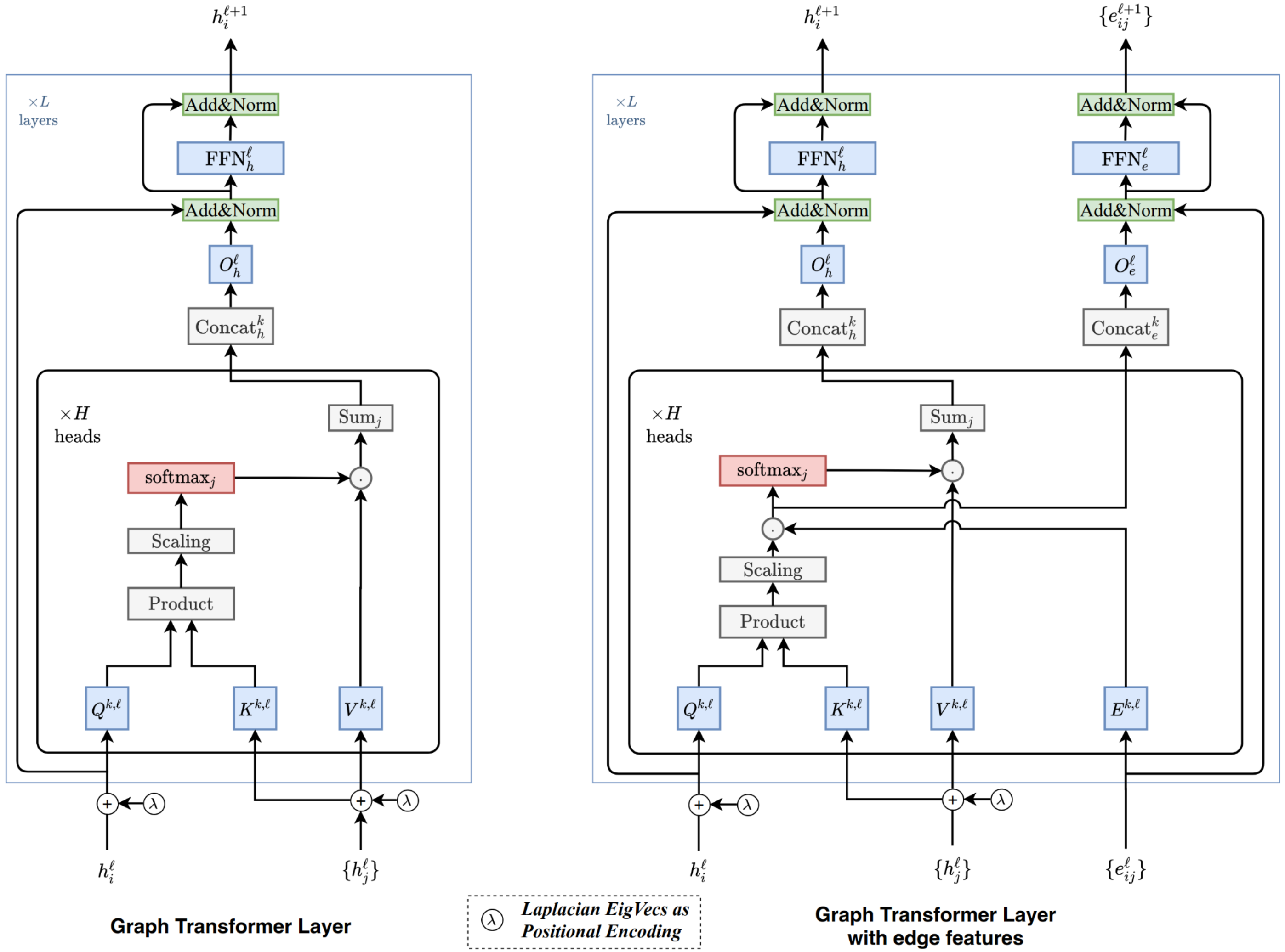


Graph Transformers: Examples

GT / Graphormer / SAN / GraphGPS

Graph Transformers

- The first design by Dwivedi and Bresson, 2021



Graph Transformer Layer

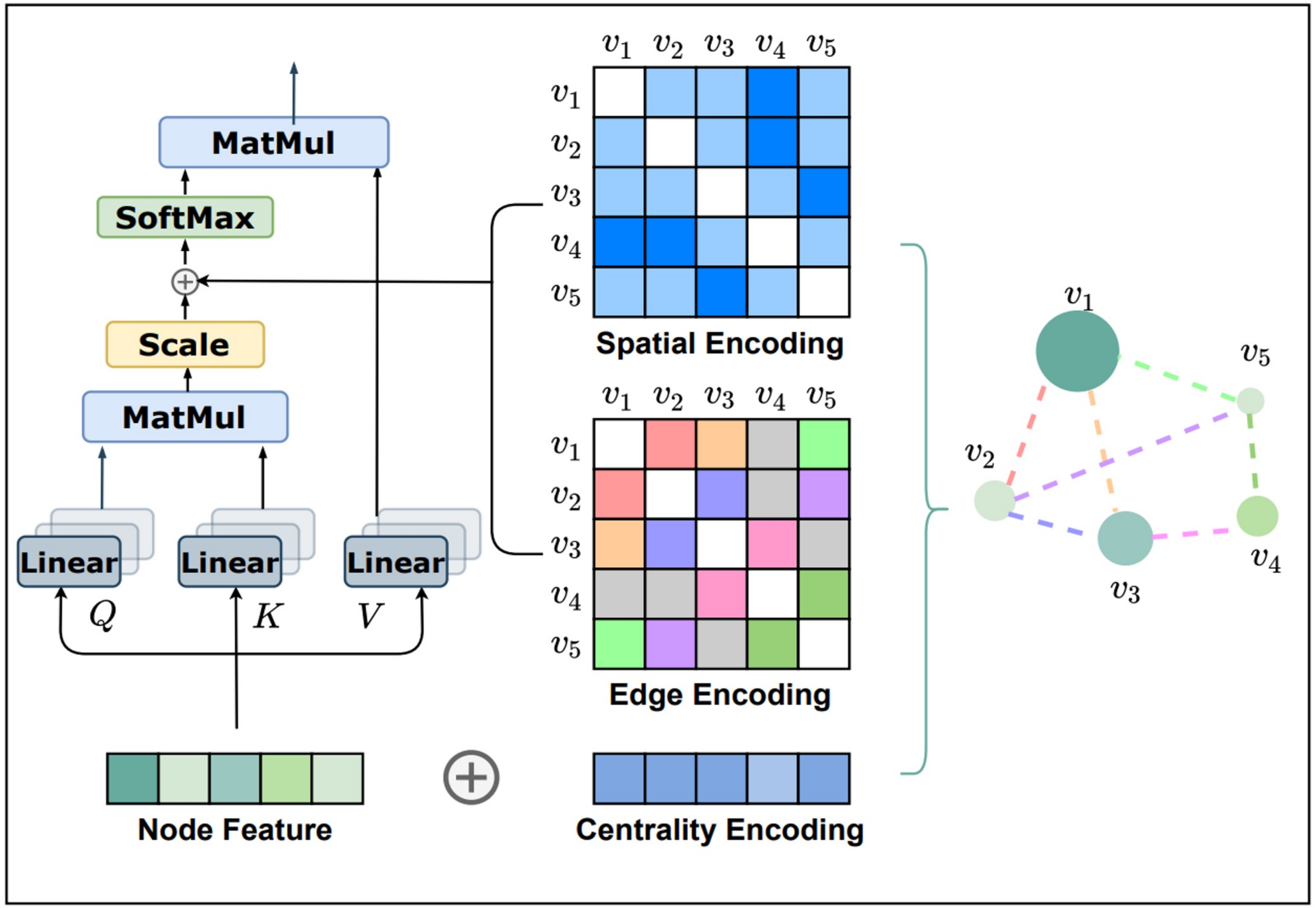
Graph Transformer Layer with edge features

Ying et al. Do Transformers Really Perform Bad for Graph Representation? NeurIPS 2021

Graphormer

At the time SoTA:

- OGB-LSC graph property
- Open Catalyst NeurIPS’21 Challenge



Positional Encodings used:

- Node centrality

$$h_i^{(0)} = x_i + z^-_{\deg^-(v_i)} + z^+_{\deg^+(v_i)}$$

- Spatial encoding:

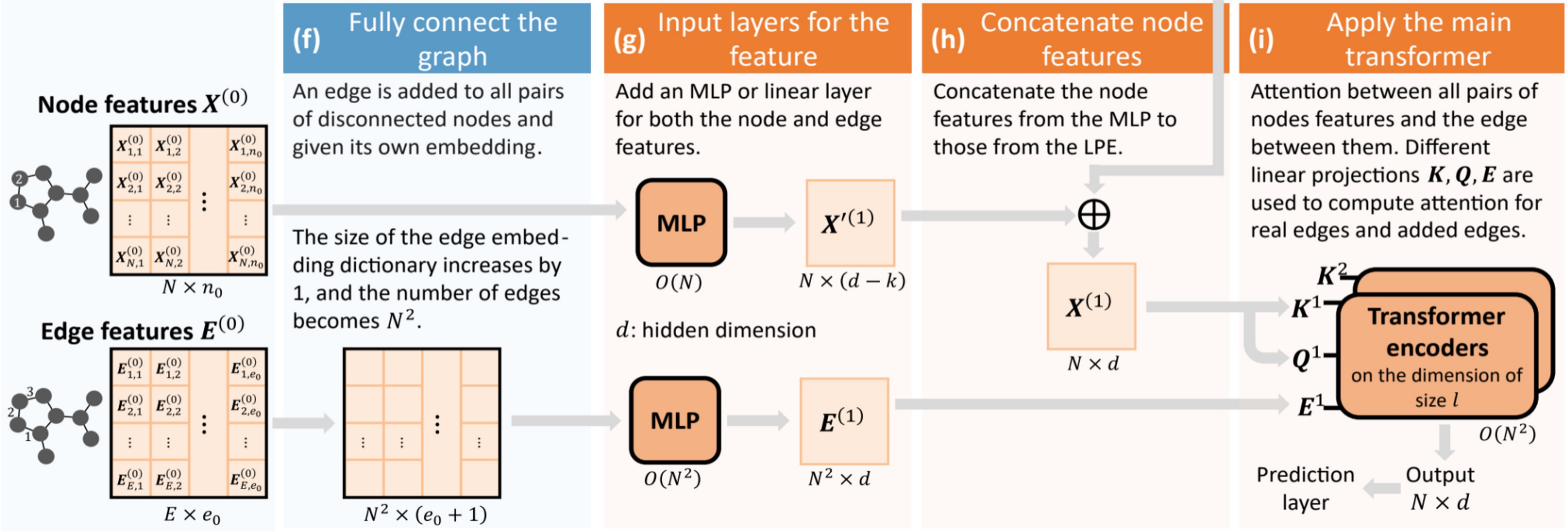
    shortest paths distances

- Edge encoding

    enc. edges on a shortest path

$$A_{ij} = \frac{(h_i W_Q)(h_j W_K)^T}{\sqrt{d}} + b_{\phi(v_i,v_j)} + c_{ij}, \text{ where } c_{ij} = \frac{1}{N}\sum_{n=1}^{N} x_{e_n}(w_n^E)^T$$

Spectral Attention Network



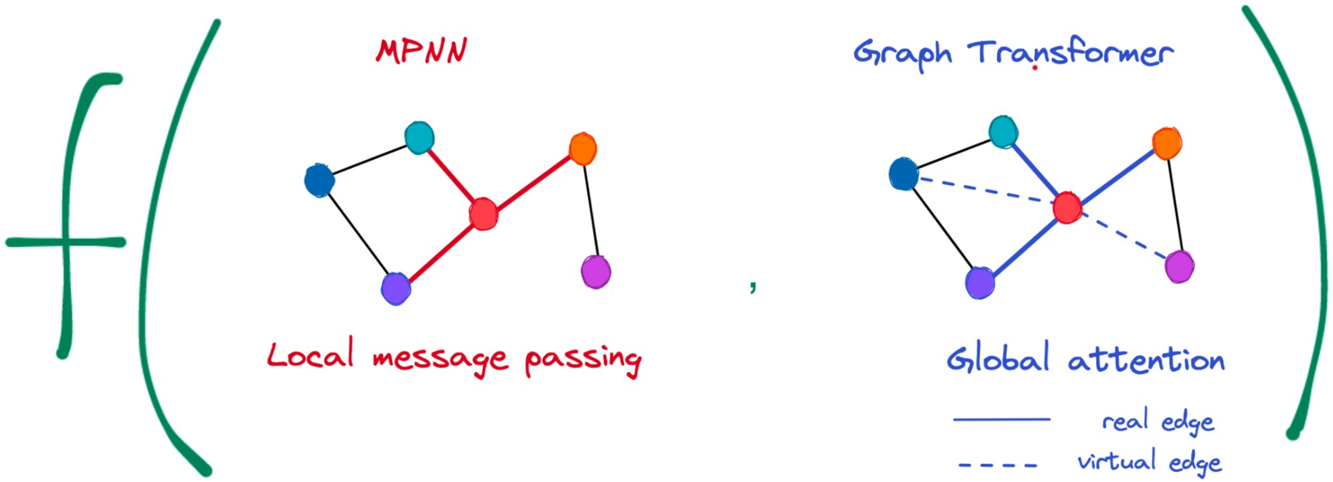
Kreuzer, Beaini et al. Rethinking Graph Transformers with Spectral Attention, 2021

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., & Beaini, D. Recipe for a General, Powerful, Scalable Graph Transformer. NeurIPS 2022

GraphGPS

1. Positional and Structural Features

2. GPS layer: Combine MPNN and Transformer (Global Attention)



Message Passing vs. Graph Transformers

Message Passing

Updates across edges of input graph

✅ Captures inductive bias from input graph topology

✅ Efficient computation: $\mathcal{O}(nd^2 + md)$

❌ Difficulty with long-range dependencies

❌ Oversmoothing, oversquashing

Graph Transformers

Use global attention

✅ All-pair attention, no information bottleneck

✅ Long-range modelling

~~❌ Loss of inductive bias from graph~~

❌ Inefficient computation: $\mathcal{O}(nd^2 + n^2 d)$

- Usually in graphs $m \ll n^2$

n: number of nodes, m: number of edges, d: hidden dimension of the network

Message Passing vs. Graph Transformers

Message Passing

Updates across edges of input graph

✅ Captures inductive bias from input graph topology

✅ Efficient computation: $\mathcal{O}(nd^2 + md)$

❌ Difficulty with long-range dependencies

❌ Oversmoothing, oversquashing

Graph Transformers

Use global attention

✅ All-pair attention, no information bottleneck

✅ Long-range modelling

~~❌ Loss of inductive bias from graph~~

❌ Inefficient computation: $\mathcal{O}(nd^2 + n^2d)$

- Usually in graphs $m \ll n^2$

Efficient Transformers

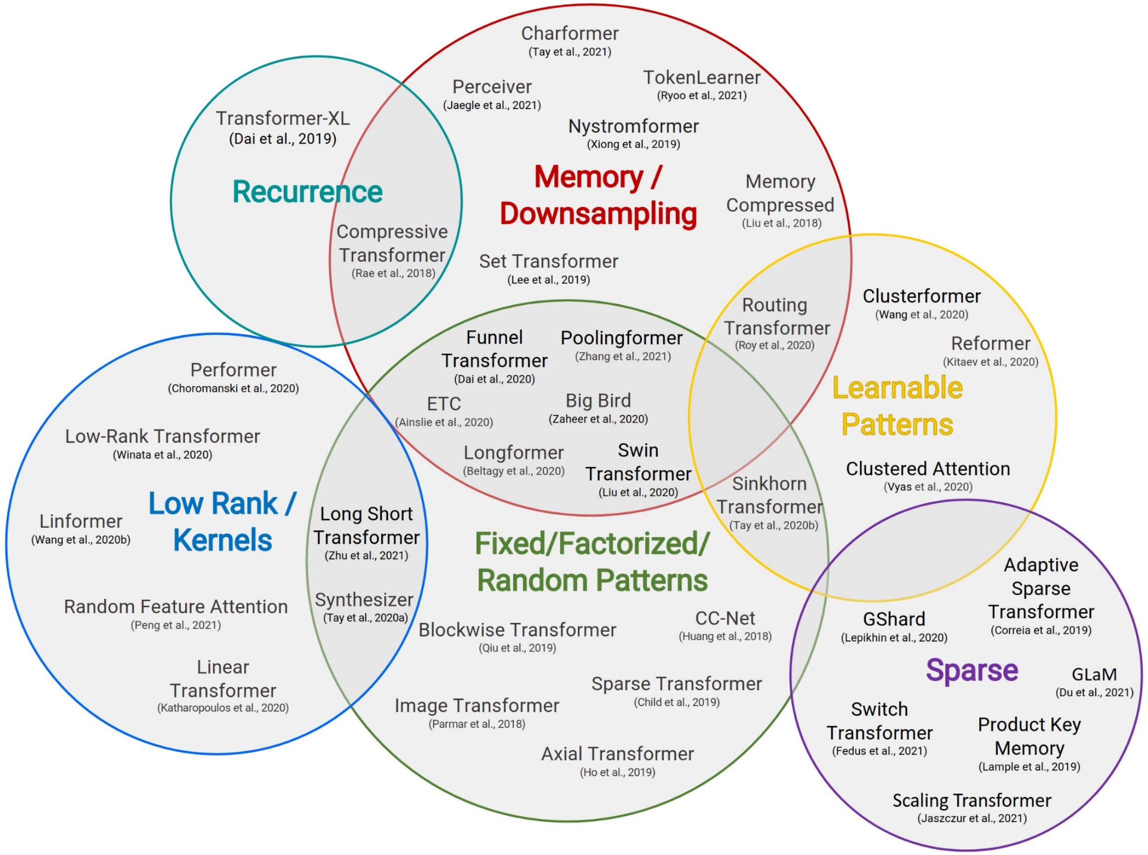
n: number of nodes, m: number of edges, d: hidden dimension of the network

Efficient Graph Transformers

Sparse Transformers / Linear Transformers / Sampling

Scaling

- Typical transformer has $O(N^2)$ dependence – prohibitive for long sequences or large graphs
- Dense attention ⟶ **efficient attention**
    - Sparse attention
    - Low-rank attention
    - Sampling based
    - Different design
- Many efficient attention mechanisms proposed for *sequence* transformers



Y. Tay, M. Dehghani, D. Bahri, and D. Metzler. Efficient Transformers: A Survey. ACM Computing Survey, volume 55. 2022

Efficient Transformers for *Graphs*

- **Sparse patterns**
  - **Exphormer** (Shirzad et al., 2023) → efficient sparse attention pattern
- **Low-rank / efficient attention**
  - **Nodeformer** (Wu et al., 2023) → low-rank attention using kernelized Gumbel-Softmax. Inspired by **Performer** (Choromanski et al., 2021)
  - **Difformer** → diffusion-based attention
  - **Polynormer** → polynomial + linear attention
- **Sampling-based**
  - **Gophormer** (Zhao et al., 2021) → neighbor sampling strategy
  - **Spexphormer** (Shirzad et al., 2024) → Importance Sampling + Exphormer
- **Alternative designs**
  - **NAGphormer** (Chen et al., 2022) → k-hop neighborhood summary
- **And many more…**

Exphormer

Sparse attention pattern → reduces computation

Linear complexity with respect to the graph size

Virtual nodes → shrink graph diameter, enable long-range communication

Expander graphs → prevent over-squashing by improving information flow

Video source: https://research.google/blog/exphormer-scaling-transformers-for-graph-structured-data/

Nodeformer

Why Attention is Quadratic

$$\text{Att}(Q, K, V) = softmax\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

$Q, K, V \in \mathbb{R}^{n\times d}$, where n= number of nodes/tokens.

Computing $QK^\top$gives an $n\times n$ matrix → $\mathcal{O}(n^2 d)$.

Linear Attention

$$\text{Att}(Q, K, V) \approx \phi(Q)(\phi(K)^\top V)$$

Replace softmax with kernel feature map $\phi(\cdot)$.

Avoids explicit $n\times n$ matrix.

Complexity: $\mathcal{O}(nd^2)$→ linear in n.

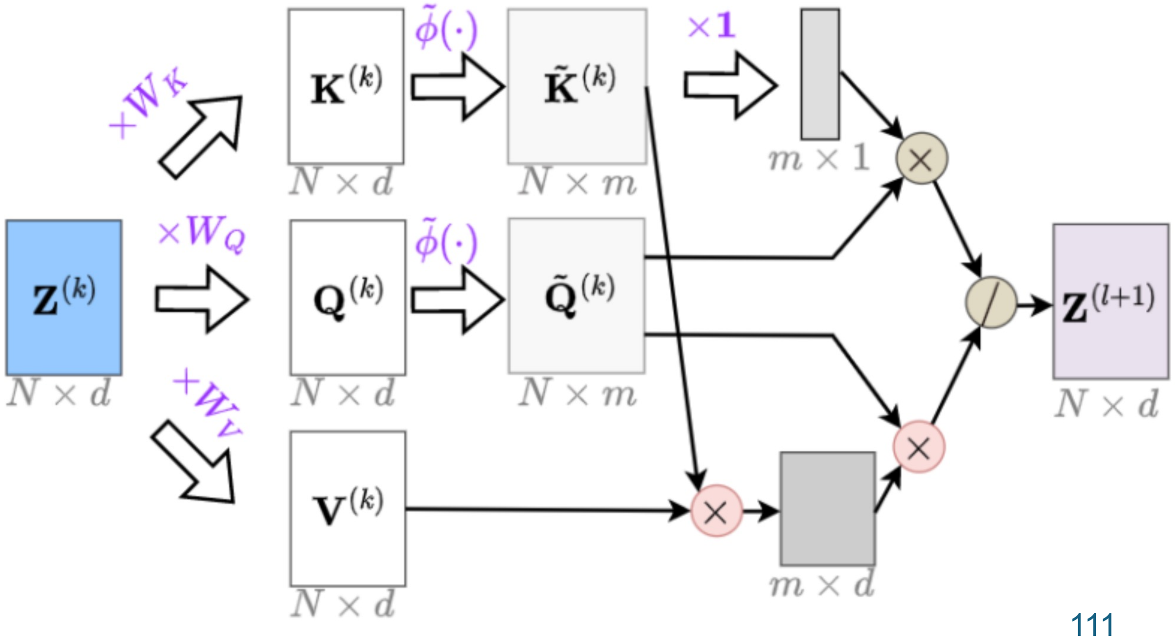
The global attention layer of NodeFormer

$$\mathbf{Q}^{(k)} = \mathbf{W}_Q\mathbf{Z}^{(k)},\ \ \mathbf{K}^{(k)} = \mathbf{W}_K\mathbf{Z}^{(k)},\ \ \mathbf{V}^{(k)} = \mathbf{W}_V\mathbf{Z}^{(k)}$$

$$\tilde{\mathbf{Q}}^{(k)} = \phi(\mathbf{Q}^{(k)}),\ \ \tilde{\mathbf{K}}^{(k)} = \phi(\mathbf{K}^{(k)})$$

$$\mathbf{D}^{(k)} = \text{diag}^{-1}\left(\tilde{\mathbf{Q}}^{(k)}\left((\tilde{\mathbf{K}}^{(k)})^\top\mathbf{1}\right)\right)$$

$$\mathbf{Z}^{(k+1)} = \mathbf{D}^{(k)}\left[\tilde{\mathbf{Q}}^{(k)}\left((\tilde{\mathbf{K}}^{(k)})^\top\mathbf{V}^{(k)}\right)\right]$$



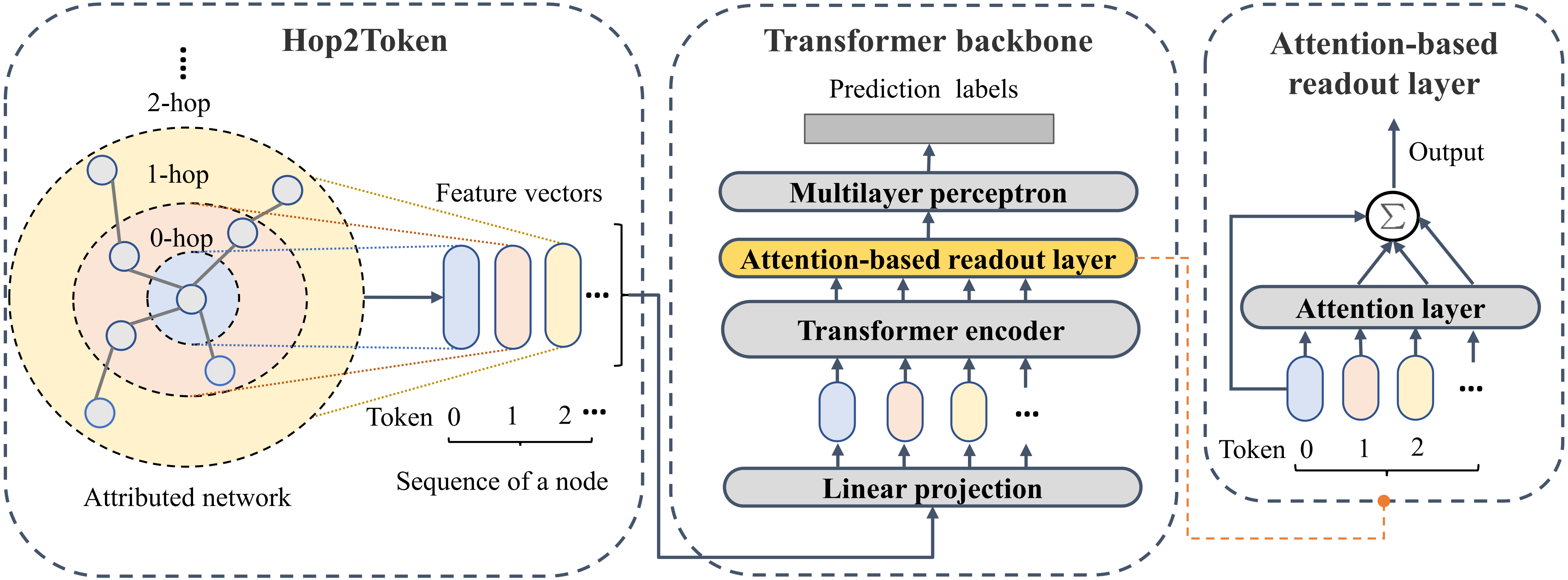
Figures from: How to Build Graph Transformers with O(N) Complexity, Qitian Wu, Medium blog post

NAGphormer

Hop-to-token: convert *k-hop neighborhoods* into tokens.

Summarization: aggregate all nodes within distance *k* into a single embedding.

Attention: model interactions between different hop-level embeddings.



Message Passing vs. Graph Transformers

Message Passing

Updates across edges of input graph

✅ Captures inductive bias from input graph topology

✅ Efficient computation: $\mathcal{O}(nd^2 + md)$

❌ Difficulty with long-range dependencies

❌ Oversmoothing, oversquashing

Graph Transformers

Use global attention

✅ All-pair attention, no information bottleneck

✅ Long-range modelling

~~❌ Loss of inductive bias from graph~~

❌ Inefficient computation: $\mathcal{O}(nd^2 + n^2 d)$

- Usually in graphs $m \ll n^2$

n: number of nodes, m: number of edges, d: hidden dimension of the network

Message Passing vs. Graph Transformers

Message Passing

Updates across edges of input graph

✅ Captures inductive bias from input graph topology

✅ Efficient computation: $\mathcal{O}(nd^2 + md)$

❌ Difficulty with long-range dependencies

❌ Oversmoothing, oversquashing

Graph Transformers

Use global attention

✅ All-pair attention, no information bottleneck

✅ Long-range modelling

~~❌ Loss of inductive bias from graph~~

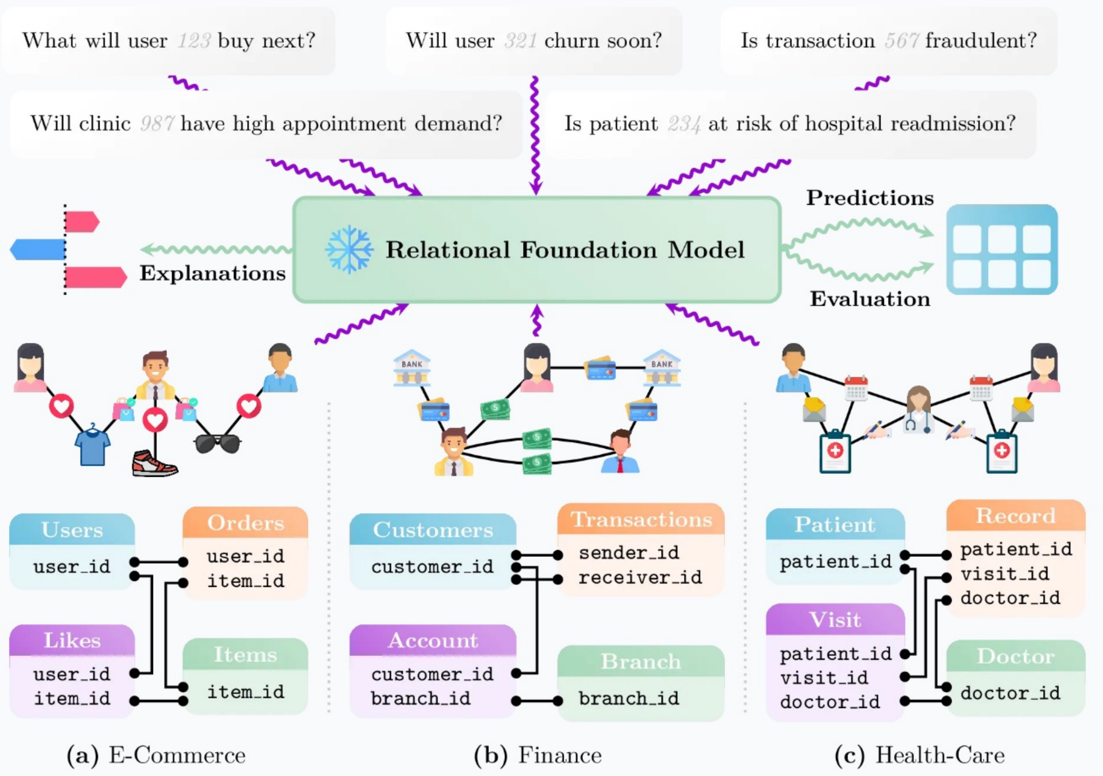
~~❌ Inefficient computation:~~ $\cancel{\mathcal{O}(nd^2 + n^2d)}$

- Usually in graphs $m \ll n^2$

n: number of nodes, m: number of edges, d: hidden dimension of the network

Graph Transformers in Industry

- **KumoRFM:** A Large Relational Foundation Model
- For large scale tabular graphs
- Generalizable for many different tasks



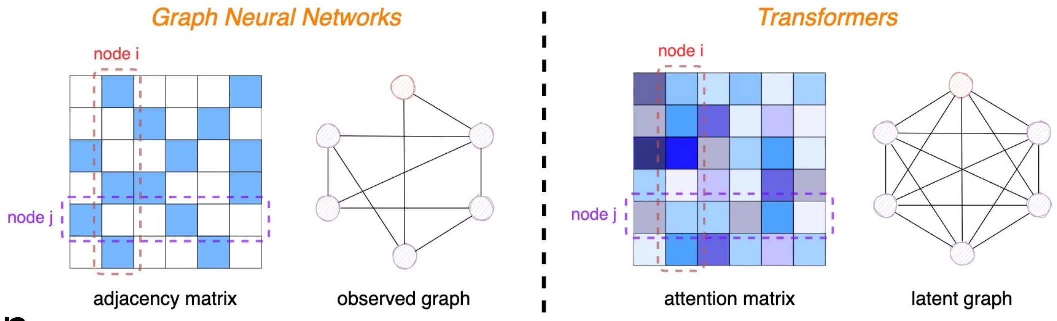
Key capabilities of Relational Foundation Models.

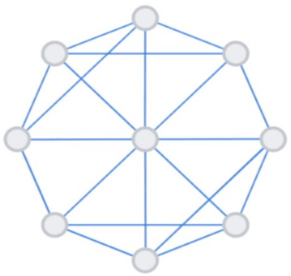
Summary

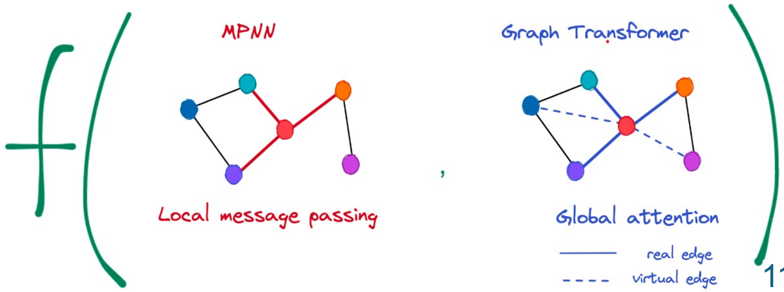
Revolution
- Bring the success of Transformers to **graph learning**
- Decoupling **graph structure** from the computation grapn

Challenges
- **Loss of inductive bias**
  - Need **positional** and **structural encodings**
- **Scalability issues**
  - Graphs are large → require **efficient Transformers**
  - Explore **alternative designs**
- **Hybrid Approaches**
  - Combine **GNN message passing** with **Transformer attention**

Industry Impact
- Deployed in large-scale systems (e.g., **KumoRFM**)







More Resources

- **Kumo AI:** Graph Transformers blog post
  - Excellent introductory material with interactive visualizations
- **Google Research:** Blog post on Exphormer model
- **ICML 2024** Graph Learning Tutorial
  - Similar introductory content, but (a) less applications & no coding content, but (b) goes deeper into specific challenges in graph learning
- **William L. Hamilton**'s GRL book, GRL keynote talk
- **Geometric Deep Learning Course:** Book, Keynote, Lectures

Live coding: Building Graph Transformers

[Link]

Part III
Beyond Message Passing
New Research Areas & Applications

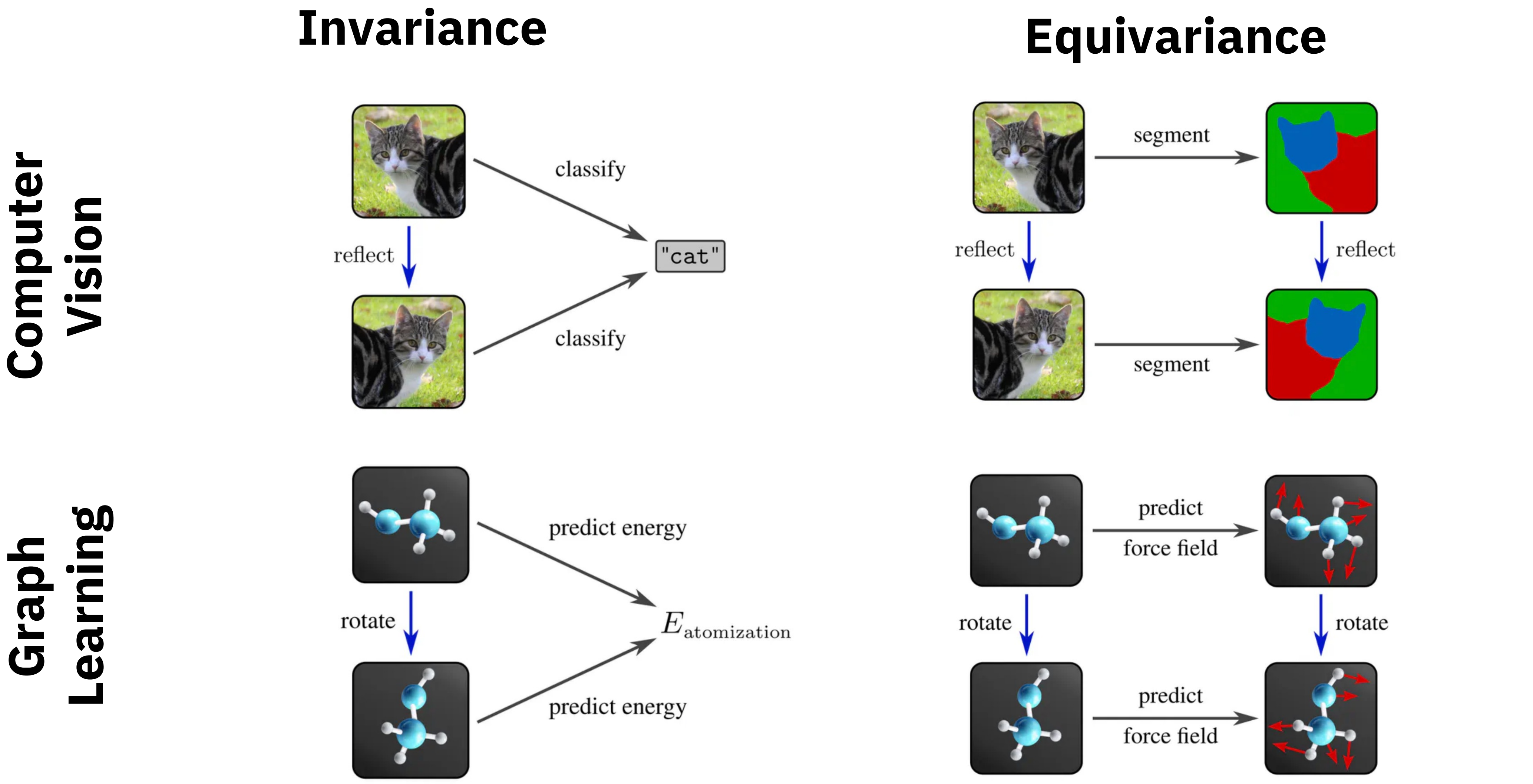
Part III: Outline

1. Translational and Rotational Symmetries
2. Geometric Generative Modeling
    1. Background
    2. Graph Generation
    3. Protein Design Problem

Roto-Translational Symmetries

Invariance & Equivariance over SO(3)/SE(3)

Data invariance & equivariance



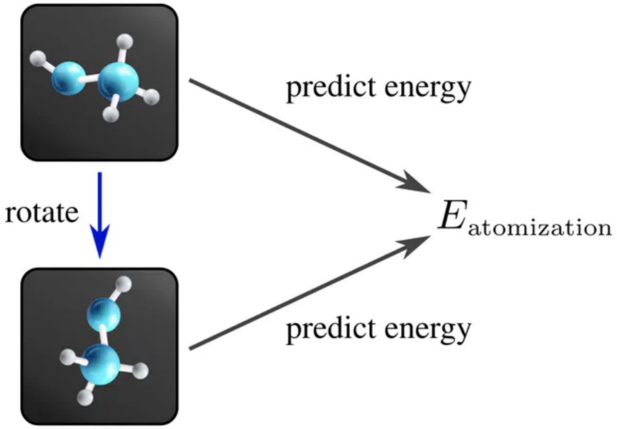
Data invariance & equivariance

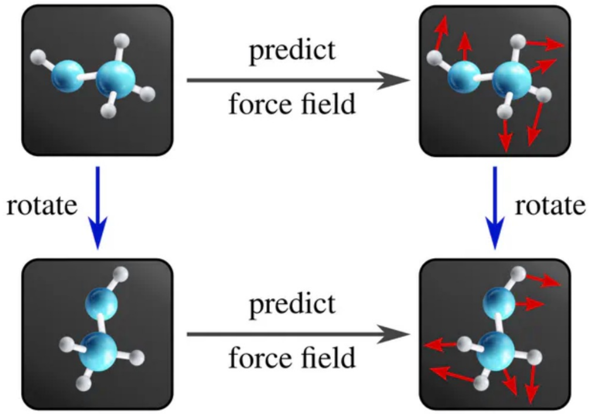
Invariance | **Equivariance**

Computer Vision

How to maintain invariance or equivariance with $x - y - z$ coordinates as features?

Graph Learning





GNN equivariance w.r.t. specific transformations

- Equivariant models in some group of transformations (in addition to permutations)
  - rotations:
    SO(3), SO(2), ...
  - rotations + translations:
    SE(3)
  - rotations + translations + reflections:
    E(n)

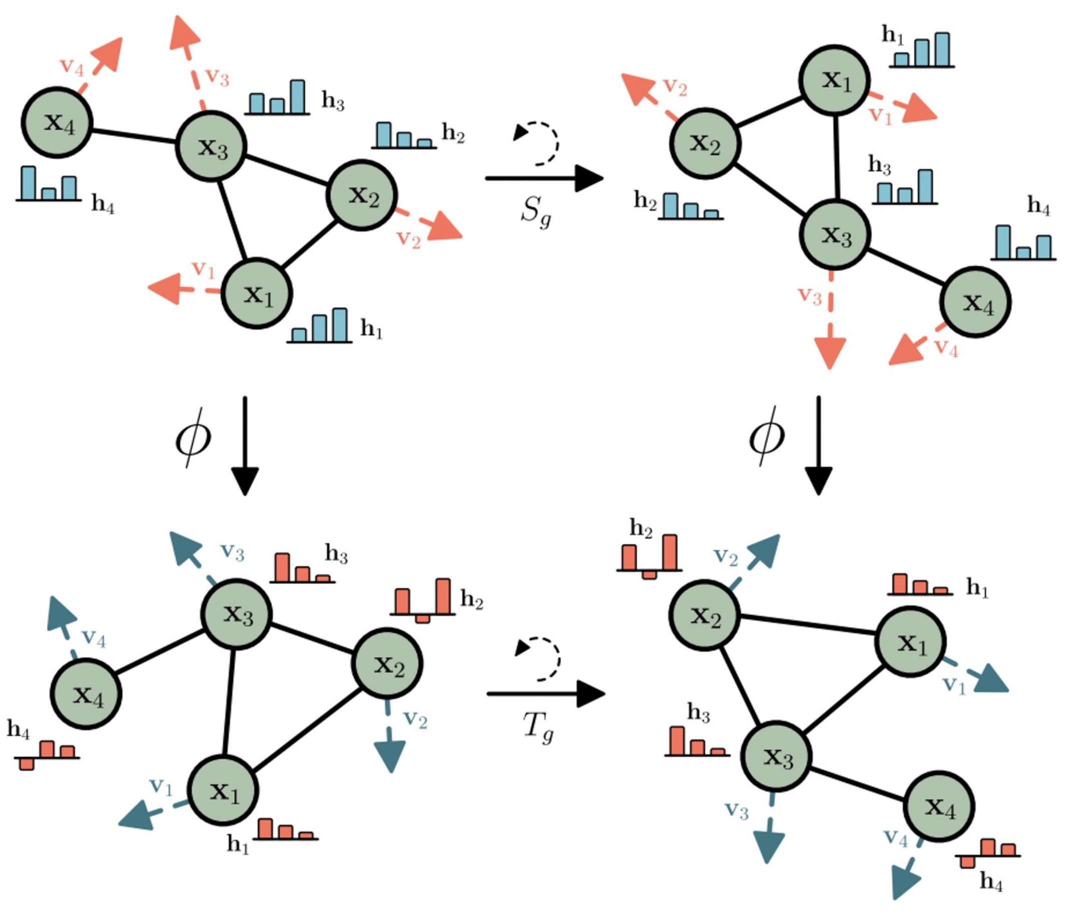


Figure 1. Example of rotation equivariance on a graph with a graph neural network ϕ

$$\phi(T_g(\mathbf{x})) = S_g(\phi(\mathbf{x}))$$

Figure from: Satorras et al. E(n) Equivariant Graph Neural Networks. ICML 2021

E(n) Equivariant GNN

Equivariant to:

- rotations, translations, reflections and permutations

Applied to tasks such as:

- Simulation of N-body system dynamics
- In Quantum Chemistry (QM9)

GNN

$$m_{i,j} = \phi_e(h_i^l, h_j^l, a_{ij})$$

$$m_i = \sum_{j \in \mathcal{N}(i)} m_{i,j}$$

$$h_i^{l+1} = \phi_h(h_i^l, m_i)$$

E(n)-GNN

Relative Squared Position

$$m_{i,j} = \phi_e(h_i^l, h_j^l, \| x_i^l - x_j^l \|^2, a_{ij})$$

$$x_i^{l+1} = x_i^l + C \sum_{j \neq i} (x_i^l - x_j^l)\phi_x(m_{i,j})$$

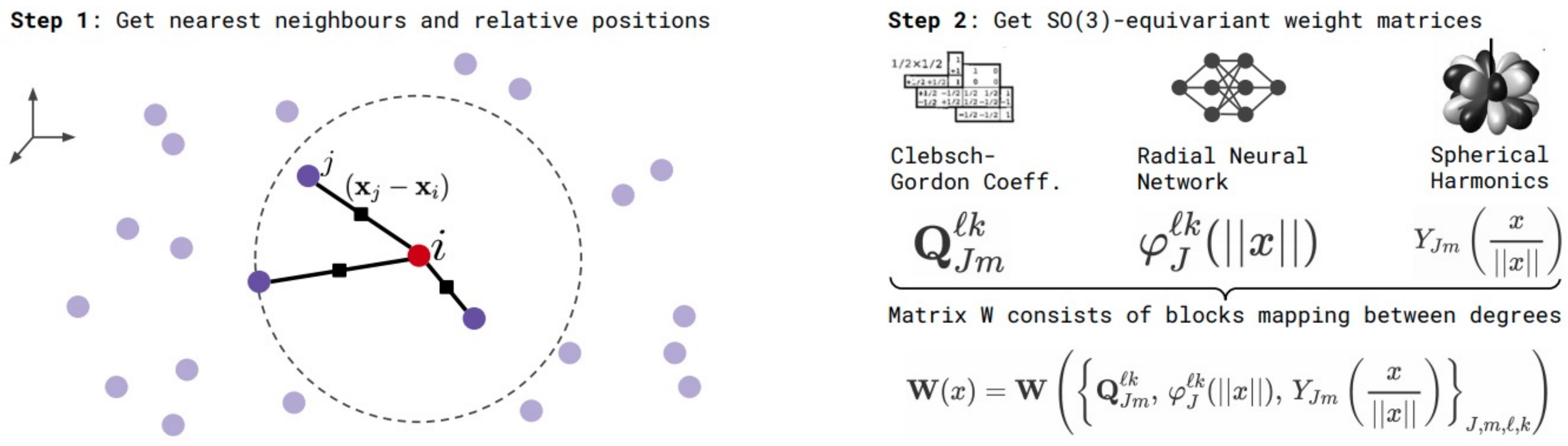
Position Update

$$m_i = \sum_{j \in \mathcal{N}(i)} m_{i,j}$$

$$h_i^{l+1} = \phi_h(h_i^l, m_i)$$

Figure from: Satorras et al. E(n) Equivariant Graph Neural Networks. ICML 2021

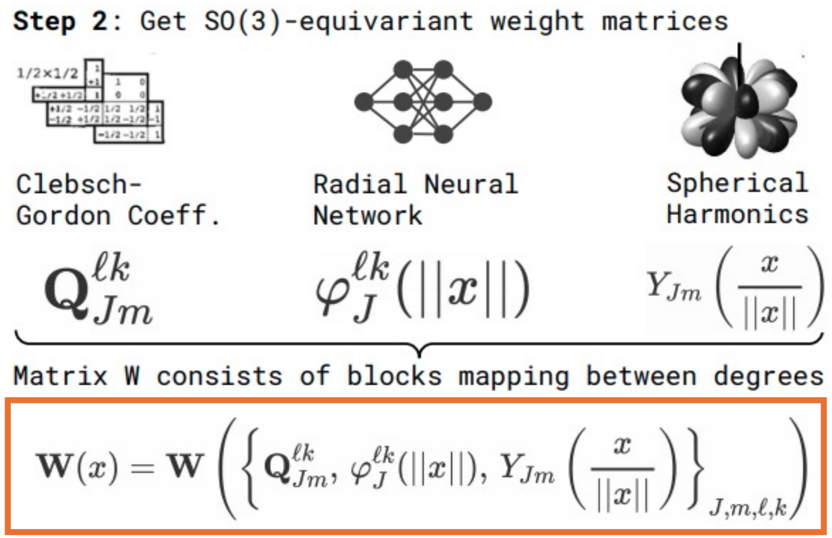
SE(3)-Transformer



- **Spherical harmonics:** Orthonormal basis for rotations in SO(3). Functions that project 3-dimensional vectors into spherical tensors (equivariant)
- **Radial NN**: Uses RBFs to learn distance-based features (invariant)
- **Clebsch-Gordon Coefficients:** Physics-derived change-of-basis matrices to ensure invariant and equivariant weights

Figure from: Fuchs et al. SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks. NeurIPS 2020

SE(3)-Transformer



$$\mathbf{W}^{\ell k}(\mathbf{x}) = \sum_{J=|\ell-k|}^{\ell+k} \varphi_J^{\ell k}(\|\mathbf{x}\|)\, \mathbf{W}_J^{\ell k}(\mathbf{x}),$$

where

$$\mathbf{W}_J^{\ell k}(\mathbf{x}) = \sum_{m=-J}^{J} Y_{Jm}\left(\frac{\mathbf{x}}{\|\mathbf{x}\|}\right) \mathbf{Q}_{Jm}^{\ell k}.$$

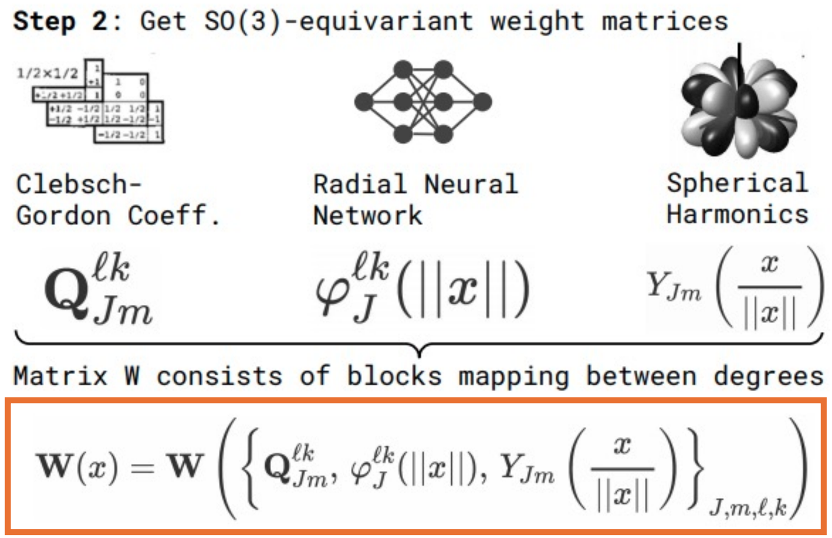
The TFN layer output is

$$\mathbf{f}_{\text{out},i}^{\ell} = \underbrace{w^{\ell\ell}\mathbf{f}_{\text{in},i}^{\ell}}_{\text{self-interaction}} + \sum_{k\geq 0}\sum_{j\neq i} \mathbf{W}^{\ell k}(\mathbf{x}_j - \mathbf{x}_i)\, \mathbf{f}_{\text{in},j}^{k}.$$

Symbol	Meaning
$\mathbf{W}^{\ell k}(\mathbf{x})$	Kernel mapping between input type k and output type ℓ
J	Index of equivariant basis kernel (runs from $\lvert\ell - k\rvert$ to $\ell + k$)
$\varphi_J^{\ell k}(\|\mathbf{x}\|)$	Learnable radial function depending only on distance $\|\mathbf{x}\|$
$\mathbf{W}_J^{\ell k}(\mathbf{x})$	Basis kernel determined by angular structure
$Y_{Jm}(\mathbf{x}/\|\mathbf{x}\|)$	Spherical harmonic (angular dependence)
$\mathbf{Q}_{Jm}^{\ell k}$	Clebsch–Gordan coefficient matrices, shape $(2\ell + 1) \times (2k + 1)$
$\mathbf{f}_{\text{in},j}^{k}$	Input feature of type k at node j
$\mathbf{f}_{\text{out},i}^{\ell}$	Output feature of type ℓ at node i
$w^{\ell\ell}$	Scalar self-interaction parameter (for $k = \ell$, $J = 0$)
$\mathbf{x}_j - \mathbf{x}_i$	Relative position vector between nodes i and j

Figure from: Fuchs et al. SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks. NeurIPS 2020

SE(3)-Transformer



$$\mathbf{W}^{\ell k}(\mathbf{x}) = \sum_{J=|\ell-k|}^{\ell+k} \varphi_J^{\ell k}(\|\mathbf{x}\|)\, \mathbf{W}_J^{\ell k}(\mathbf{x}),$$

where

$$\mathbf{W}_J^{\ell k}(\mathbf{x}) = \sum_{m=-J}^{J} Y_{Jm}\left(\frac{\mathbf{x}}{\|\mathbf{x}\|}\right) \mathbf{Q}_{Jm}^{\ell k}.$$

The TFN layer output is

$$\mathbf{f}_{\text{out},i}^{\ell} = \underbrace{w^{\ell\ell}\mathbf{f}_{\text{in},i}^{\ell}}_{\text{self-interaction}} + \sum_{k\geq 0}\sum_{j\neq i} \mathbf{W}^{\ell k}(\mathbf{x}_j - \mathbf{x}_i)\, \mathbf{f}_{\text{in},j}^{k}.$$

The kernel $\mathbf{W}^{\ell k}(\mathbf{x})$ mapping features of type k to type ℓ is expressed as a sum over basis kernels indexed by J. Each feature $\mathbf{f}_j$ a concatenation of vectors of different types. Each basis kernel separates **radial dependence**, captured by a learnable function $\varphi_J^{\ell k}(\|\mathbf{x}\|)$, and **angular dependence**, determined by spherical harmonics Y_{Jm} and Clebsch–Gordan coefficients $\mathbf{Q}_{Jm}^{\ell k}$. This structure ensures equivariance to rotations. The special case $J = 0, k = \ell$ corresponds to a learnable scalar times the identity, known as **self-interaction**. Altogether, the roto-translation equivariant layer combines self-interaction with message passing via equivariant kernels depending on relative positions $\mathbf{x}_j - \mathbf{x}_i$. Note that this design comes from a prior work Tensor Field Network.

In practice, we have type-0 vectors that are invariant under rotations (like node type) and type-1 vectors that rotate according to 3D rotation matrices (positions and velocities).

Figure from: Fuchs et al. SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks. NeurIPS 2020

SE(3)-Transformer

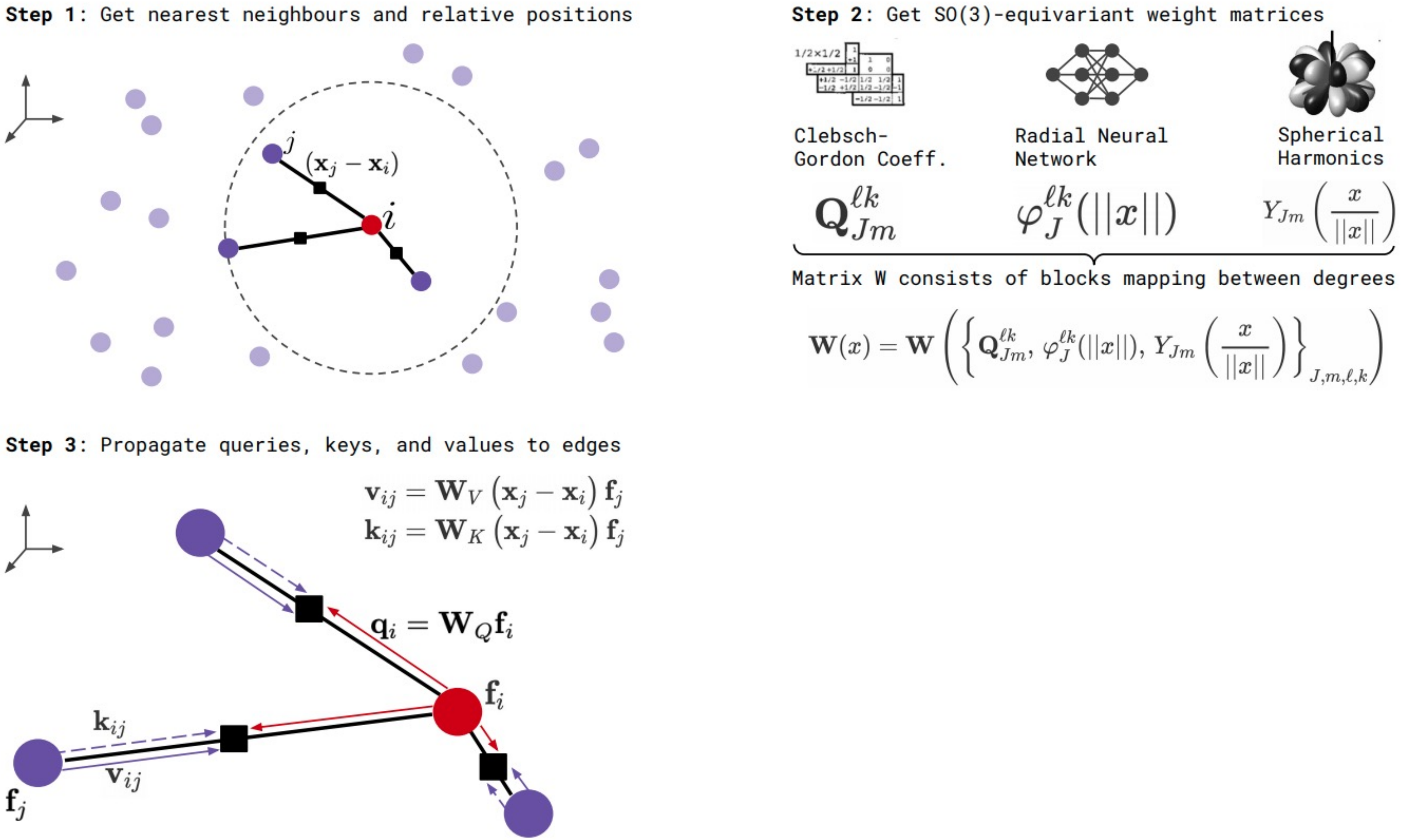


Figure from: Fuchs et al. SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks. NeurIPS 2020

SE(3)-Transformer

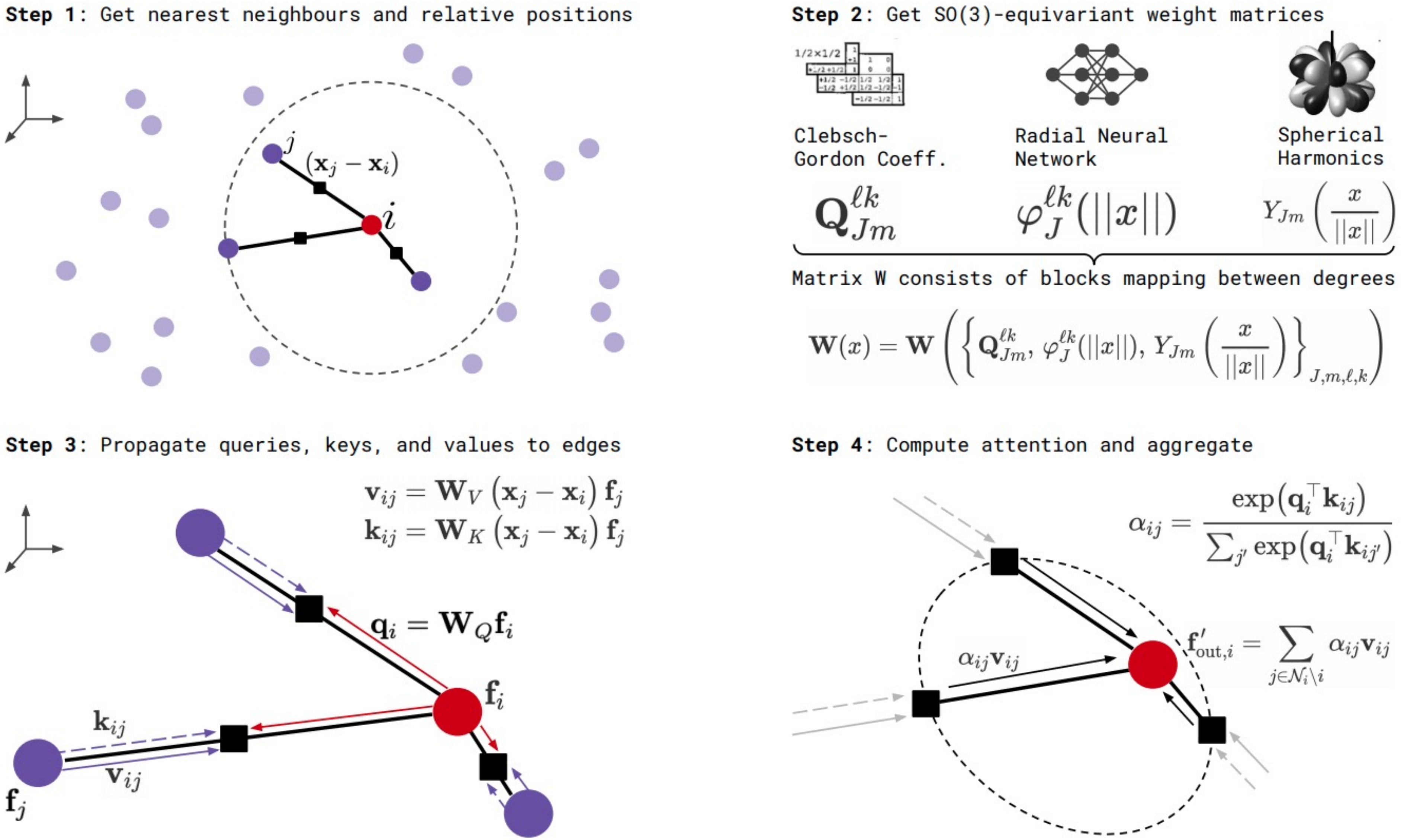
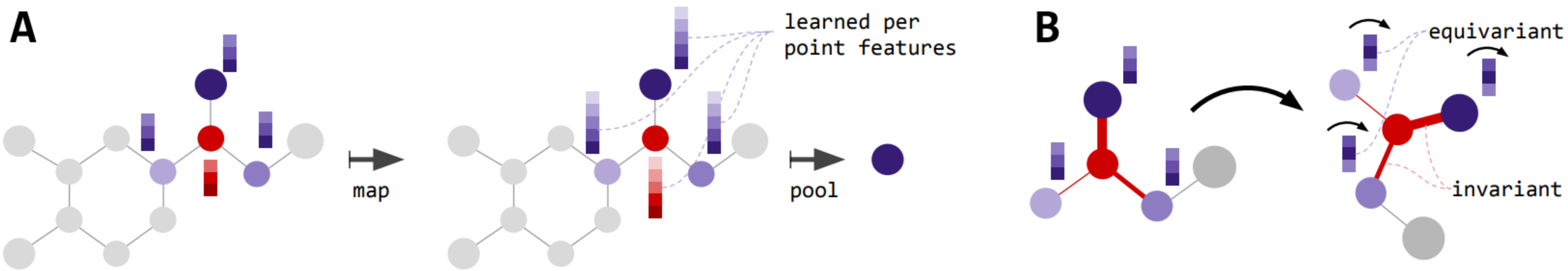


Figure from: Fuchs et al. SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks. NeurIPS 2020

SE(3)-Transformer



- Each layer maps from a point cloud to a point cloud (or graph to graph) while guaranteeing equivariance.
- Attention weights (indicated by line thickness) are invariant w.r.t. input rotation, MPNN *messages* are equivariant
- **Main idea:** Obtain invariant/equivariant final node/graph level representations, apply usual prediction heads

Figure from: Fuchs et al. SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks. NeurIPS 2020

Live coding: SE(3)-Transformer

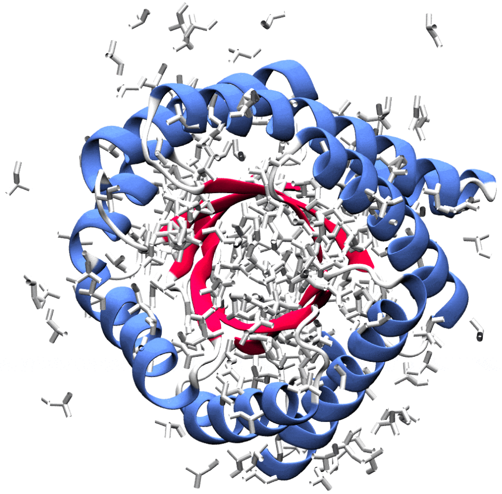
[Link]

Geometric Generative Modeling: Background

Fundamentals of Diffusion & Flows

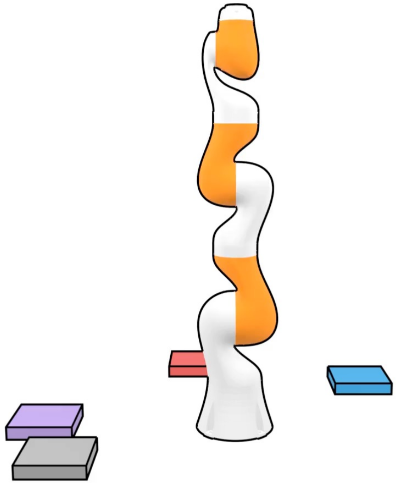
Generative Models Beyond Images and Text

Scientific Data



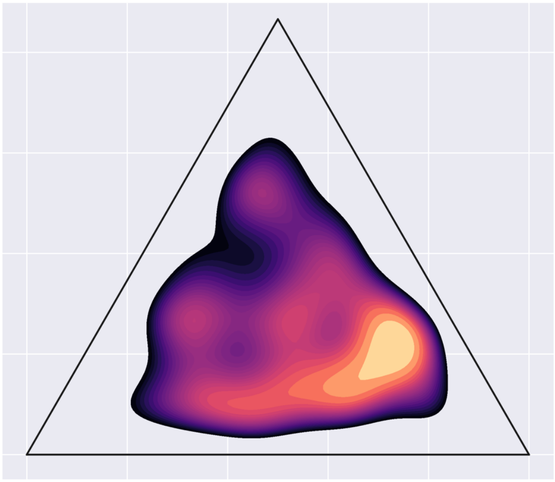
SE(3) invariant
Protein structure generation

Robotics



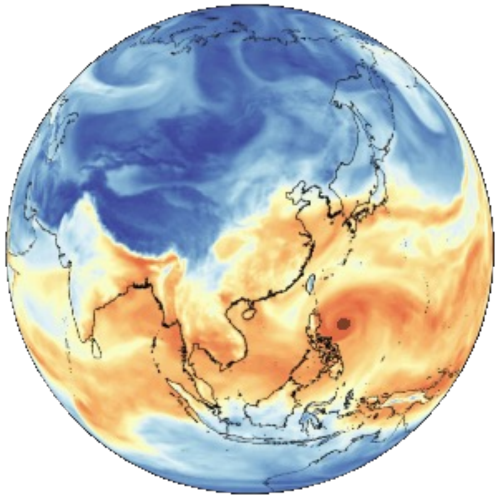
SO(2) invariant
Block stacking

Information Geometry



Fisher-Rao geometry
On the probability Simplex

Climate Modeling



Spherical Geometry $\mathbb{S}^2$

Image credit: Joey Bose

Problem Setting: Generative Modeling

- **Unknown:** data distribution $\mathbf{q}$

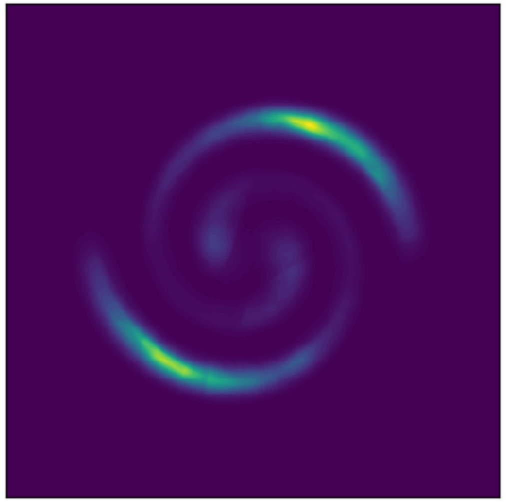
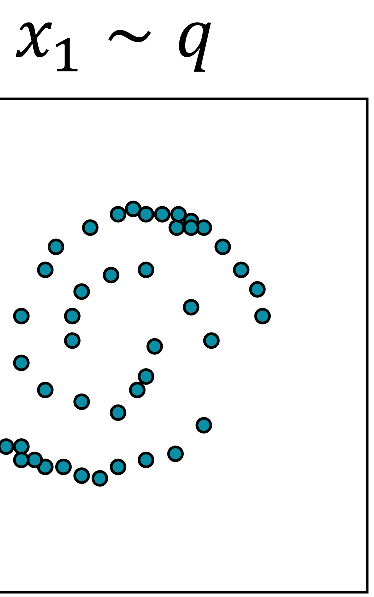


Image credit: Joey Bose

Problem Setting: Generative Modeling

- **Unknown:** data distribution q
- **Given:** samples $\mathrm{x}_1 \sim \mathrm{q}$



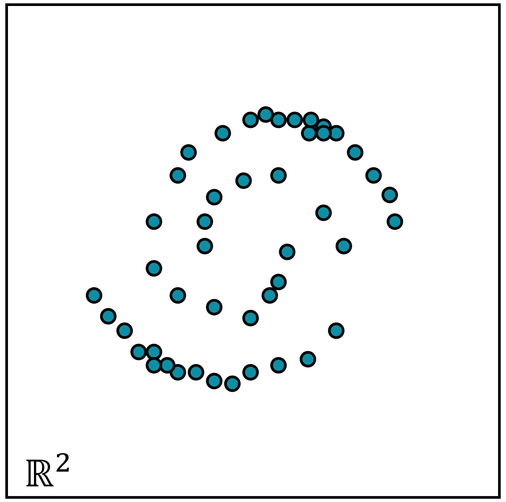
Goal: learn a sampler from the unknown q

Image credit: Joey Bose

Problem Setting: Generative Modeling

- **Unknown:** data distribution q
- **Given:** samples $x_1 \sim q$
- **Learn:** neural network with parameters θ

$x_1 \sim q$



Generative Model

(ψ_θ, p_θ)

Generator

Underlying Density

Goal: find parameters θ s.t. $p_\theta \approx q$

Image credit: Joey Bose

Problem Setting: Generative Modeling

- **Coupling:** Sample from a noise-data pair (X_0, X_1)
- **Interpolation:** Construct interpolation:

$$X_t = tX_1 + (1-t)X_0$$

- **Learn:** neural network with parameters

$$\dot{Z}_t = v_t(Z_t)$$

by minimizing

$$\min_v \int_0^1 \mathbb{E}_{(X_0,X_1)}\left[\left\|\dot{X}_t - v_t(X_t)\right\|^2\right]$$

where $\dot{X}_t = X_1 - X_0$ are the line directions.

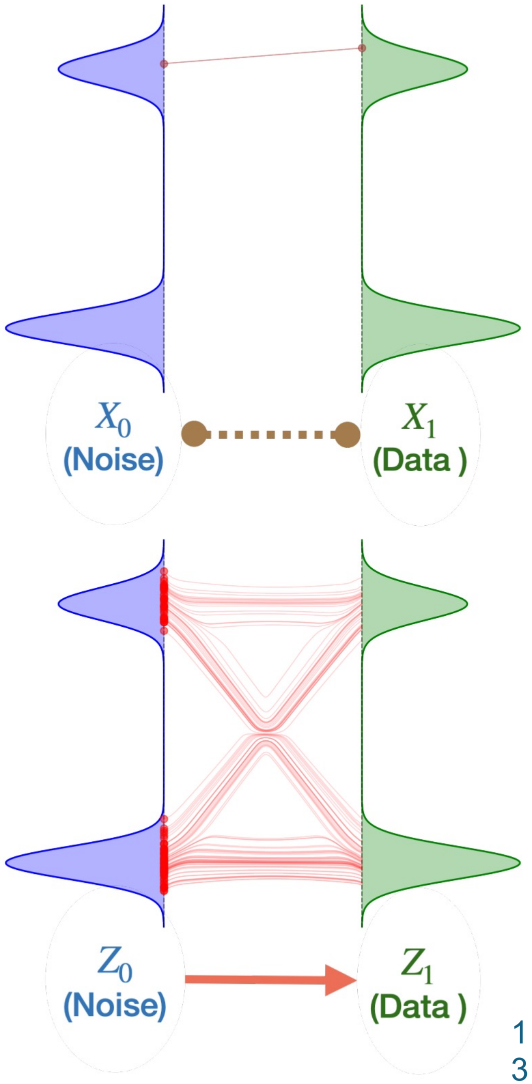


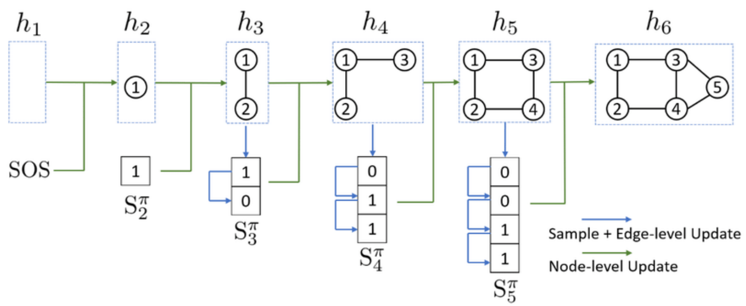
Image credit: https://rectifiedflow.github.io/

Geometric Generative Modeling: Graph Generation

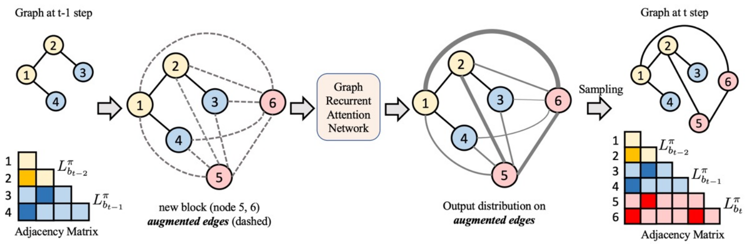
Generating graph data with desired properties.

Sequential Methods

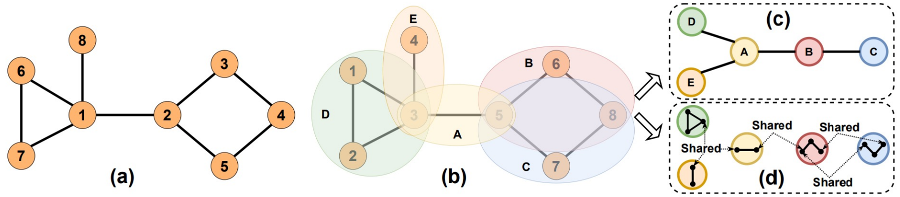
GraphRNN: Add nodes one by one, each connects to previous nodes sequentially using an RNN



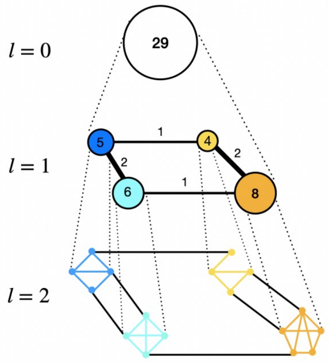
GRAN: Add nodes (or groups), then decide edges in parallel with a GNN



TD-Gen: Generate a tree decomposition first, map each tree node to a subgraph via GraphRNN-style process



HiGen: Hierarchical generation from coarse to fine, capturing global structure and local details



VAE and GAN based Approaches

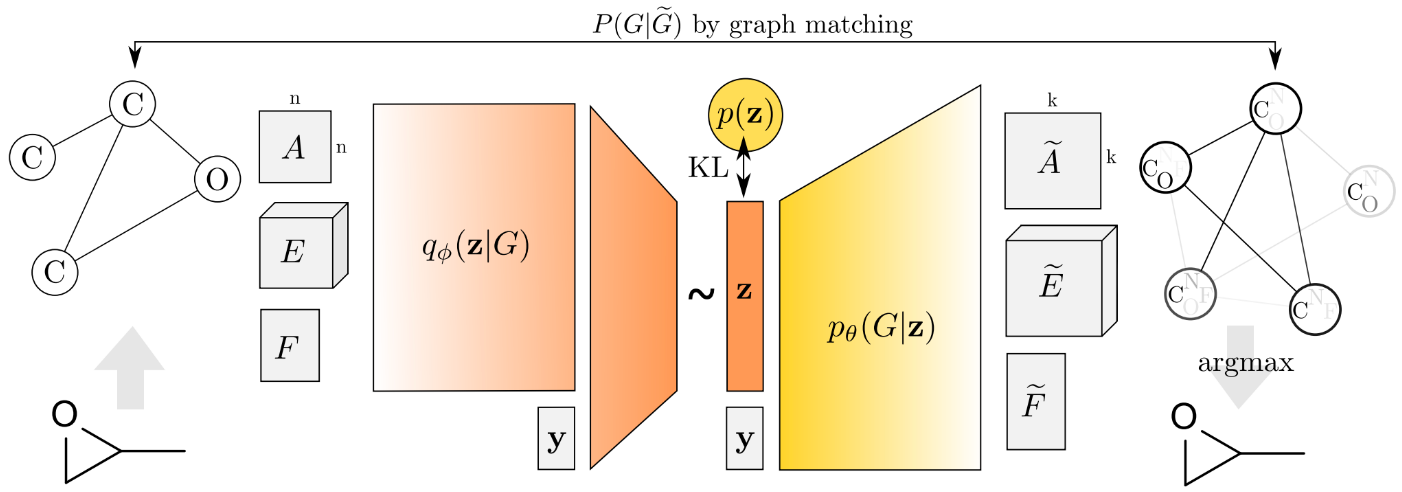
GraphVAE:

Encode adjacency matrix into a latent space with a GNN

Decode back to reconstruct the graph

Challenge: permutation invariance ⇒ multiple valid reconstructions

Solution: high-complexity graph matching to align outputs

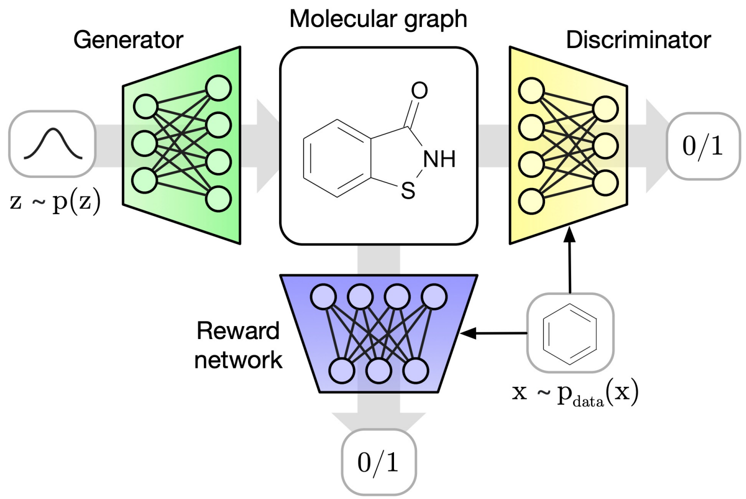


MolGAN:

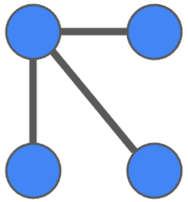
GAN framework for molecular graphs

Learns graph distribution without explicit likelihood

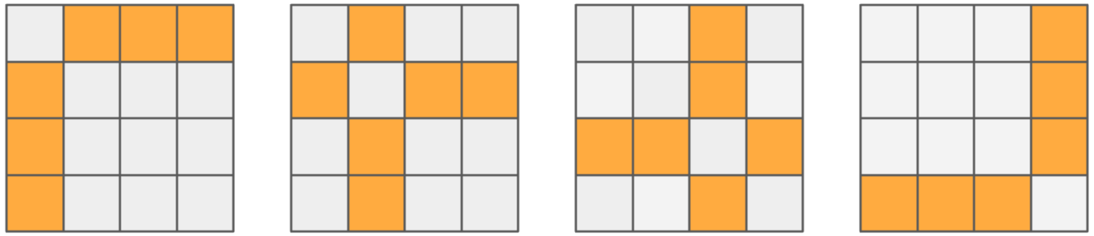
Limitation: training instability, mode collapse issues



Permutation Invariant Graph Generation



An example graph w/ 4 nodes.



$$\underbrace{\begin{bmatrix}0&1&1&1\\1&0&0&0\\1&0&0&0\\1&0&0&0\end{bmatrix}}_{\mathbf{A}^{\pi_1}} \quad \underbrace{\begin{bmatrix}0&1&0&0\\1&0&1&1\\0&1&0&0\\0&1&0&0\end{bmatrix}}_{\mathbf{A}^{\pi_2}} \quad \underbrace{\begin{bmatrix}0&0&1&0\\0&0&1&0\\1&1&0&1\\0&0&1&0\end{bmatrix}}_{\mathbf{A}^{\pi_3}} \quad \underbrace{\begin{bmatrix}0&0&0&1\\0&0&0&1\\0&0&0&1\\1&1&1&0\end{bmatrix}}_{\mathbf{A}^{\pi_4}}$$

These 4 distinct adjacency matrices stand for the same graph.

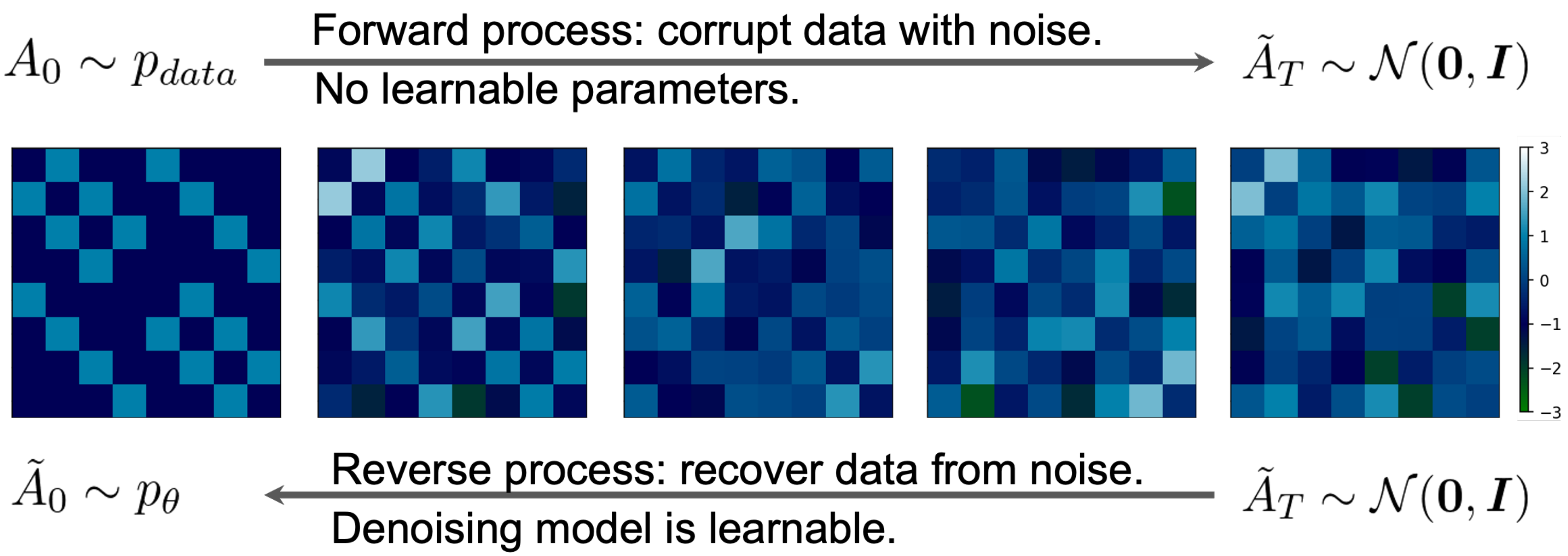
When represented in adjacency matrices, one graph has up to O(n!) distinct representations.

Permutation invariance: how to let generative models treat as “equally likely to happen”?

$$p_\theta(A^{\pi_1}) = p_\theta(A^{\pi_2}) = p_\theta(A^{\pi_3}) = \cdots$$
$$p_\theta(A) = p_\theta(PAP^\top) \quad \forall P \in \mathcal{S}_n$$

Image credit: https://qiyan98.github.io/blog/2025/gen-graph

Permutation Invariant Graph Generation



- Pipeline of denoising generative models (flow matching / diffusion) on graph data.

Image credit: https://qiyan98.github.io/blog/2025/gen-graph

Permutation Invariant Graph Generation

Theorem 1. *If* $\mathbf{s} : \mathbb{R}^{N\times N} \to \mathbb{R}^{N\times N}$ *is a permutation equivariant function, then the scalar function* $f_s = \int_{\gamma[\mathbf{0},A]} \langle \mathbf{s}(X), \mathrm{d}X \rangle_F + C$ *is permutation invariant, where* $\langle A, B \rangle_F = \mathrm{tr}(A^\top B)$ *is the Frobenius inner product,* $\gamma[\mathbf{0}, A]$ *is any curve from* $\mathbf{0} = \{0\}_{N\times N}$ *to* A*, and* $C \in \mathbb{R}$ *is a constant.*

- For generative models, we use the score function $\mathrm{s(X)} = \nabla \mathrm{log p_t(X)}$.

Niu et al., **Permutation invariant graph generation via score-based generative modeling**. (AISTATS 2020)

Diffusion and Score Models

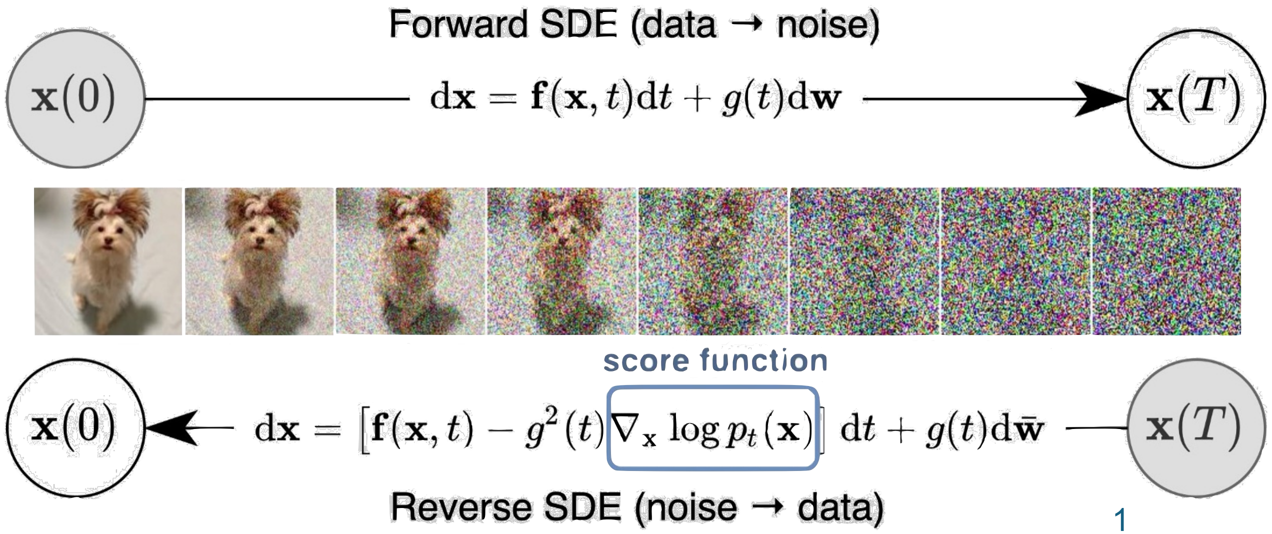
Forward SDE $$dx_t = f_t(x_t)dt + g_t dw_t$$ Data → Noise

Reverse SDE $$d\bar{x}_t = (f_t(x_t) - \boxed{g_t^2 \nabla \log p_t})dt + g_t d\bar{w}_t$$ Noise → Data

The Fokker-Planck Equation

$$\partial_t p_t = -\mathrm{div}(p_t f_t) + \frac{1}{2} g_t^2 \nabla^2 p_t$$

The Score!



Sohl-Dickstein et al., **Deep unsupervised learning using nonequilibrium thermodynamics**. (ICML 2015)
Ho et at., **Denoising Diffusion Probabilistic Models**. (NeurIPS 2020)
Song et al., **Score-Based Generative Modeling through Stochastic Differential Equations**. (ICLR 2021)

Permutation Invariant Graph Generation

Theorem 1. *If* $\mathbf{s}: \mathbb{R}^{N\times N} \to \mathbb{R}^{N\times N}$ *is a permutation equivariant function, then the scalar function* $f_s = \int_{\gamma[\mathbf{0},A]} \langle \mathbf{s}(X), \mathrm{d}X\rangle_F + C$ *is permutation invariant, where* $\langle A, B\rangle_F = \mathrm{tr}(A^\top B)$ *is the Frobenius inner product,* $\gamma[\mathbf{0}, A]$ *is any curve from* $\mathbf{0} = \{0\}_{N\times N}$ *to* A*, and* $C \in \mathbb{R}$ *is a constant.*

- For generative models, we use the score function $\mathrm{s}(\mathrm{X}) = \nabla \mathrm{log} \mathrm{p_t}(\mathrm{X})$.
- The (implicit) induced density function defined below is permutation invariant!

$$\log p_\theta(A) = \int_{\gamma[0,A]} \langle \mathbf{s}_\theta(X), \mathrm{d}X\rangle_F + \log p_\theta(\mathbf{0}).$$

Niu et al., **Permutation invariant graph generation via score-based generative modeling.** (AISTATS 2020)

Permutation Invariant Graph Generation

- Open question: does it hinder or boost performance for graph generative models?
  Recent counter-examples: AlphaFold 3, DiffAlign, SwinGNN, etc.

- AlphaFold 3: “The diffusion module operates directly on raw atom coordinates, and on a coarse abstract token representation, **without rotational frames or any equivariant processing.”**

More Resources

- **UvA GeDL:** Introduction to group equivariant deep learning
  - Excellent lecture notes & resources
- **Fabian Fuchs:** AlphaFold 2 & Equivariance
- **AlchemyBio:** Deconstructing SE(3)-Transformer
  - Very in-depth into the invariant/equivariant tools discussed
- **LoG 2024:** Geometric Generative Models Tutorial
  - A very comprehensive tutorial our GGM notes are based on – goes into derivations of diffusion, flows and molecular applications
  - Also has additional coding tutorials!
- Several extensive blog posts on diffusion/scores/flows:
  - What are Diffusion Models?
  - Building Diffusion Models' theory from ground up
  - Flow With What You Know
  - A visual dive into conditional flow matching

References

Abramson, J., Adler, J., Dunger, J., Evans, R., Green, T., Pritzel, A., Ronneberger, O., Willmore, L., Ballard, A. J., Bambrick, J., & others. (2024). Accurate structure prediction of biomolecular interactions with AlphaFold 3. *Nature*, *630*(8016), 493–500.

Alon, U., & Yahav, E. (2020). On the bottleneck of graph neural networks and its practical implications. *ArXiv Preprint ArXiv:2006.05205*.

Arnaiz-Rodríguez, A., & Velingker, A. (n.d.). *Graph Learning: Principles, Challenges, and Open Directions*. Tutorial presented at 41st International Conference on Machine Learning (ICML), Vienna, Austria, 22 July 2024. [Online]. Available: https://icml2024graphs.ameyavelingker.com/.

Azabou, M., Ganesh, V., Thakoor, S., Lin, C.-H., Sathidevi, L., Liu, R., Valko, M., Veličković, P., & Dyer, E. L. (2023). Half-hop: A graph upsampling approach for slowing down message passing. *International Conference on Machine Learning*, 1341–1360.

Battaglia, P., Hamrick, J. B. C., Bapst, V., Sánchez, Á., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, Ç., Song, F., Ballard, A., Gilmer, J., Dahl, G. E., Vaswani, A., Allen, K., Nash, C., Langston, V. J., … Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks. *ArXiv Preprint ArXiv:1806.01261*. https://arxiv.org/pdf/1806.01261.pdf

Bresson, X., & Laurent, T. (2018). *Residual Gated Graph ConvNets*. https://arxiv.org/abs/1711.07553

Bronstein, M. M., Bruna, J., Cohen, T., & Veličković, P. (2021). *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., & Vandergheynst, P. (2017). Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, *34*(4), 18–42.

Cantürk, S., Liu, R., Lapointe-Gagné, O., Létourneau, V., Wolf, G., Beaini, D., & Rampášek, L. (2023). Graph positional and structural encoder. *ArXiv Preprint ArXiv:2307.07107*.

Chen, J., Gao, K., Li, G., & He, K. (2022). NAGphormer: A tokenized graph transformer for node classification in large graphs. *ArXiv Preprint ArXiv:2206.04910*.

Daigavane, A., Ravindran, B., & Aggarwal, G. (2021). Understanding Convolutions on Graphs. *Distill*. https://doi.org/10.23915/distill.00032

De Cao, N., & Kipf, T. (2018). MolGAN: An implicit generative model for small molecular graphs. *ArXiv Preprint ArXiv:1805.11973*.

Dumoulin, V., & Visin, F. (2018). *A guide to convolution arithmetic for deep learning*. https://arxiv.org/abs/1603.07285

Dwivedi, V. P., & Bresson, X. (2020). A generalization of transformer networks to graphs. *ArXiv Preprint ArXiv:2012.09699*.

Dwivedi, V. P., Rampášek, L., Galkin, M., Parviz, A., Wolf, G., Luu, A. T., & Beaini, D. (2022). Long range graph benchmark. *Advances in Neural Information Processing Systems*, *35*, 22326–22340.

Errica, F., Christiansen, H., Zaverkin, V., Maruyama, T., Niepert, M., & Alesiani, F. (2023). Adaptive message passing: A general framework to mitigate oversmoothing, oversquashing, and underreaching. *ArXiv Preprint ArXiv:2312.16560*.

Eschenburg, K. M., Grabowski, T. J., & Haynor, D. R. (2021). Learning cortical parcellations using graph neural networks. *Frontiers in Neuroscience*, *15*, 797500.

Fey, M., Kocijan, V., Lopez, F., Lenssen, J. E., & Leskovec, J. (n.d.). *KumoRFM: A Foundation Model for In-Context Learning on Relational Data*.

Fuchs, F., Worrall, D., Fischer, V., & Welling, M. (2020). Se (3)-transformers: 3d roto-translation equivariant attention networks. *Advances in Neural Information Processing Systems*, *33*, 1970–1981.

Geisler, S. M., Kosmala, A., Herbst, D., & Günnemann, S. (2024). Spatio-spectral graph neural networks. *Advances in Neural Information Processing Systems*, *37*, 49022–49080.

References

Hamilton, W.L. (2020). Graph Representation Learning. In *Synthesis Lectures on Artifical Intelligence and Machine Learning*. Morgan & Claypool.

Hamilton, W.L., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 1025–1035.

Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, *33*, 6840–6851.

Karami, M. (2023). Higen: Hierarchical graph generative networks. *ArXiv Preprint ArXiv:2305.19337*.

Kipf, T. N., & Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. *International Conference on Learning Representations*. https://openreview.net/forum?id=SJU4ayYgl

Kreuzer, D., Beaini, D., Hamilton, W., Létourneau, V., & Tossou, P. (2021). Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, *34*, 21618–21629.

Laabid, N., Rissanen, S., Heinonen, M., Solin, A., & Garg, V. (2024). Equivariant Denoisers Cannot Copy Graphs: Align Your Graph Diffusion Models. *ArXiv Preprint ArXiv:2405.17656*.

Leskovec, J. (n.d.). *CS224W: Machine Learning with Graphs*. Stanford University, Stanford, CA, Fall 2024. [Online]. Available: https://web.stanford.edu/class/cs224w/.

Li, Y., & King, I. (2020). Autograph: Automated graph neural network. *International Conference on Neural Information Processing*, 189–201.

Liao, R., Li, Y., Song, Y., Wang, S., Hamilton, W., Duvenaud, D. K., Urtasun, R., & Zemel, R. (2019). Efficient graph generation with graph recurrent attention networks. *Advances in Neural Information Processing Systems*, *32*.

Lipman, Y., Chen, R. T., Ben-Hamu, H., Nickel, M., & Le, M. (2022). Flow matching for generative modeling. *ArXiv Preprint ArXiv:2210.02747*.

Min, Y., Wenkel, F., & Wolf, G. (2020). Scattering gcn: Overcoming oversmoothness in graph convolutional networks. *Advances in Neural Information Processing Systems*, *33*, 14498–14508.

Niu, C., Song, Y., Song, J., Zhao, S., Grover, A., & Ermon, S. (2020). Permutation invariant graph generation via score-based generative modeling. *International Conference on Artificial Intelligence and Statistics*, 4474–4484.

Ou, M., Cui, P., Pei, J., Zhang, Z., & Zhu, W. (2016). Asymmetric Transitivity Preserving Graph Embedding. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1105–1114. https://doi.org/10.1145/2939672.2939751

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). DeepWalk: online learning of social representations. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 701–710. https://doi.org/10.1145/2623330.2623732

Qiang Liu. (2024). *PyTorch RectifiedFlow*. https://github.com/lqiang67/rectified-flow

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., & Beaini, D. (2022). Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, *35*, 14501–14515.

Sanchez-Lengeling, B., Reif, E., Pearce, A., & Wiltschko, A. B. (2021). A Gentle Introduction to Graph Neural Networks. *Distill*. https://doi.org/10.23915/distill.00033

Sato, R. (2020). *A Survey on The Expressive Power of Graph Neural Networks*. https://arxiv.org/abs/2003.04078

References

Satorras, V. G., Hoogeboom, E., & Welling, M. (2021). E (n) equivariant graph neural networks. *International Conference on Machine Learning*, 9323–9332.

Scarselli, F., Gori, M., Ah Chung Tsoi, Hagenbuchner, M., & Monfardini, G. (2009). The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, *20*(1), 61–80. https://doi.org/10.1109/TNN.2008.2005605

Shirzad, H., Hajimirsadeghi, H., Abdi, A. H., & Mori, G. (2022). Td-gen: graph generation using tree decomposition. *International Conference on Artificial Intelligence and Statistics*, 5518–5537.

Shirzad, H., Velingker, A., Venkatachalam, B., Sutherland, D. J., & Sinop, A. K. (2023). Exphormer: Sparse transformers for graphs. *International Conference on Machine Learning*, 31613–31632.

Simonovsky, M., & Komodakis, N. (2018). Graphvae: Towards generation of small graphs using variational autoencoders. *International Conference on Artificial Neural Networks*, 412–422.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., & Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. *International Conference on Machine Learning*, 2256–2265.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., & Poole, B. (2020). Score-based generative modeling through stochastic differential equations. *ArXiv Preprint ArXiv:2011.13456*.

Sun, C., Li, C., Lin, X., Zheng, T., Meng, F., Rui, X., & Wang, Z. (2023). Attention-based graph neural networks: a survey. *Artificial Intelligence Review*, *56*(Suppl 2), 2263–2310.

Topping, J., Di Giovanni, F., Chamberlain, B. P., Dong, X., & Bronstein, M. M. (2021). Understanding over-squashing and bottlenecks on graphs via curvature. *ArXiv Preprint ArXiv:2111.14522*.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2018). Graph attention networks. *The 6th ICLR*.

Weiler, M. (n.d.). *Equivariant neural networks – what, why and how? An introduction to equivariant & coordinate independent CNNs – Part 1*. Blog post, Maurice Weiler's website, Nov. 7, 2023. [Online]. Available: https://maurice-weiler.gitlab.io/blog_post/cnn-book_1_equivariant_networks/.

Wu, Q., Zhao, W., Li, Z., Wipf, D. P., & Yan, J. (2022). Nodeformer: A scalable graph structure learning transformer for node classification. *Advances in Neural Information Processing Systems*, *35*, 27387–27401.

Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019). How powerful are graph neural networks? *Proc. of ICLR*.

Xu, Keyulu, Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K., & Jegelka, S. (2018). Representation learning on graphs with jumping knowledge networks. *International Conference on Machine Learning*, 5453–5462.

Yan, Q., Liang, Z., Song, Y., Liao, R., & Wang, L. (2023). Swingnn: Rethinking permutation invariance in diffusion models for graph generation. *ArXiv Preprint ArXiv:2307.01646*.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., & Liu, T.-Y. (2021). Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, *34*, 28877–28888.

Ying, R., You, J., Morris, C., Ren, X., Hamilton, W. L., & Leskovec, J. (2018). Hierarchical Graph Representation Learning with Differentiable Pooling. *Advances in Neural Information Processing Systems*, *31*. https://proceedings.neurips.cc/paper/2018/file/e77dbaf6759253c7c6d0efc5690369c7-Paper.pdf

You, J., Ying, R., Ren, X., Hamilton, W., & Leskovec, J. (2018). Graphrnn: Generating realistic graphs with deep auto-regressive models. *International Conference on Machine Learning*, 5708–5717.

Zhao, L., & Akoglu, L. (2019). Pairnorm: Tackling oversmoothing in gnns. *ArXiv Preprint ArXiv:1909.12223*.