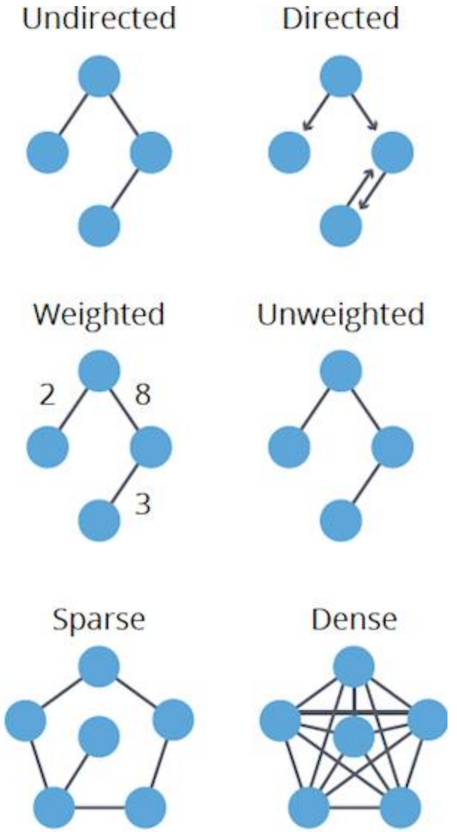
GRAPHS

- Graphs represent relational data
  - Entities → Nodes: $u \in \mathcal{V}$
    - Node features: $\boldsymbol{x}_u \in \mathbb{R}^{d_{\mathcal{V}}}$
  - Relations → Edges: $(u, v) \in \mathcal{E}$
    - Edge features: $\boldsymbol{e}_{uv} \in \mathbb{R}^{d_{\mathcal{E}}}$



Directed edges specify an incoming and outgoing node

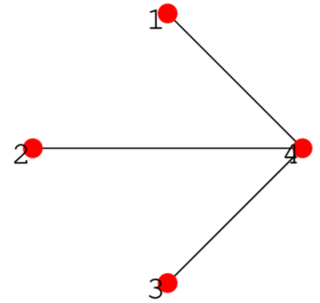
Edge features might be weights or otherwise more complicated attributes

Sparsity is problem-dependent… roughly, sparse if $|\mathcal{E}| \ll |\mathcal{V}|^2$

Graph Algorithms in Neo4j: Graph Algorithm Concepts

EDGE REPRESENTATIONS

- Graphs represent relational data
  - Entities → Nodes: $u \in \mathcal{V}$
    - Node features: $\boldsymbol{x}_u \in \mathbb{R}^{d_{\mathcal{V}}}$
  - Relations → Edges: $(u, v) \in \mathcal{E}$
    - Edge features: $\boldsymbol{e}_{uv} \in \mathbb{R}^{d_{\mathcal{E}}}$



Edge Representations

Adjacency Matrices

$A_{adjacency} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Incidence Matrices

$A_{incidence} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{E}|}$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

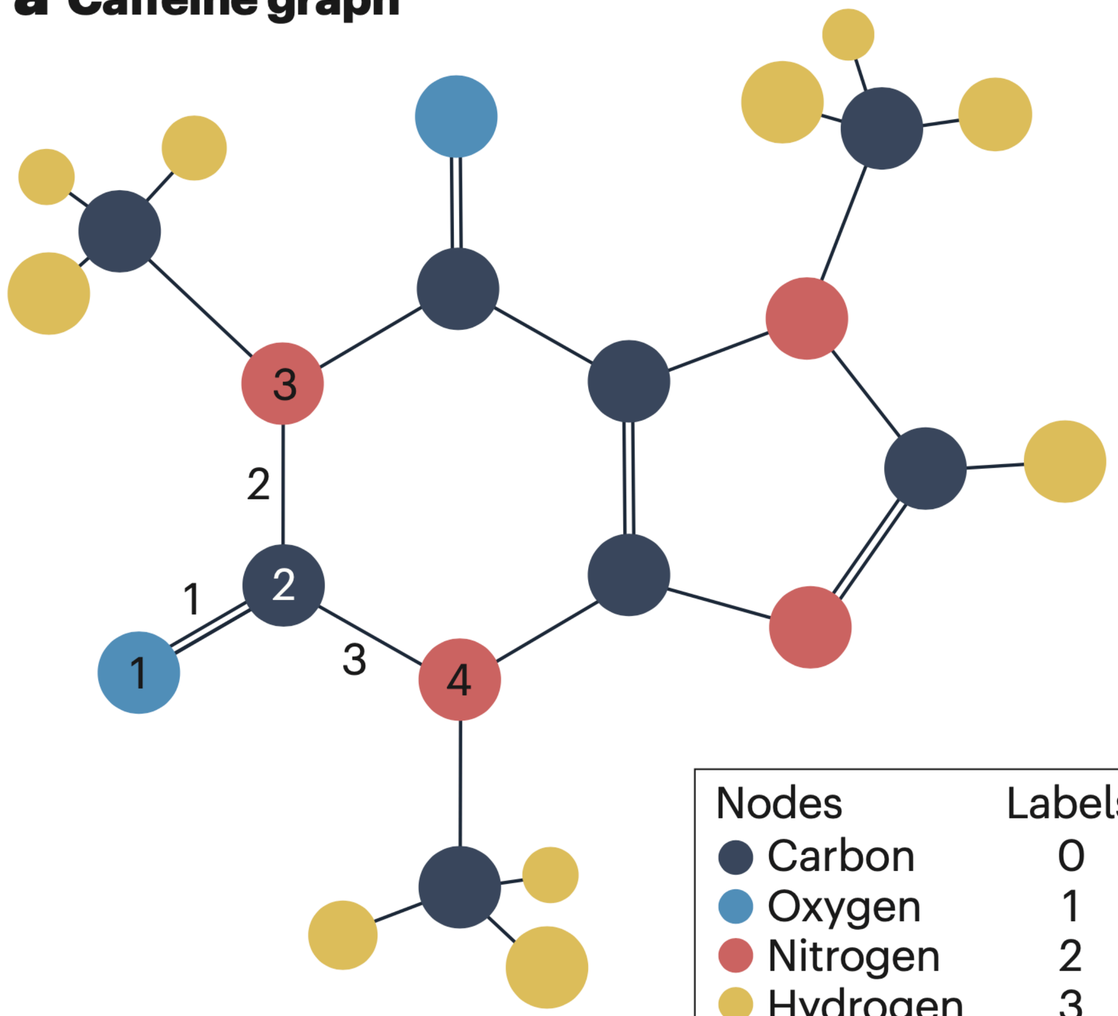
PyTorch geometric

Sparse Index Lists (COO)

$I_{COO} \in \mathbb{R}^{2 \times |\mathcal{E}|}$

$$\begin{bmatrix} [1 & 2 & 3] \\ [4 & 4 & 4] \end{bmatrix}$$

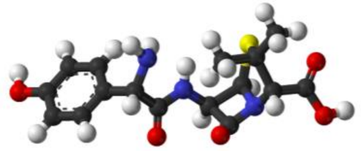
Graph Algorithms in Neo4j: Graph Algorithm Concepts



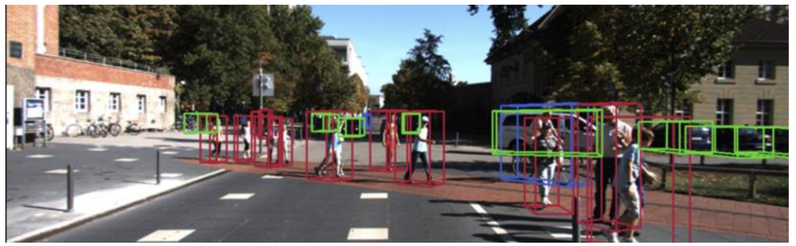
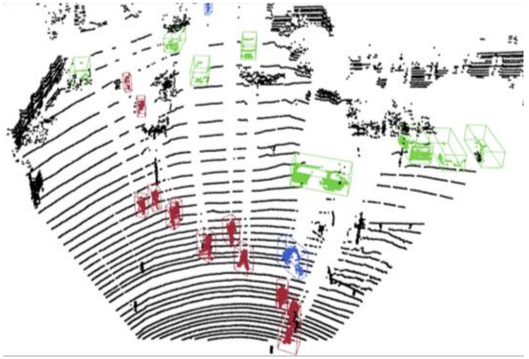
https://www.nature.com/articles/s42254-023-00569-0

GRAPH LEARNING TASKS

DRUG DISCOVERY

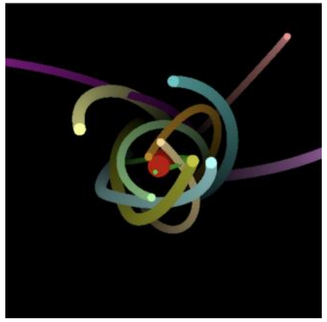
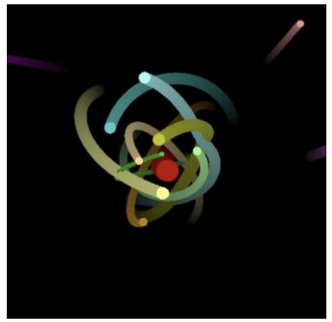
$GNN($  $) \rightarrow$ molecule property

INSTANCE SEGMENTATION

$GNN($  $) \rightarrow$ 

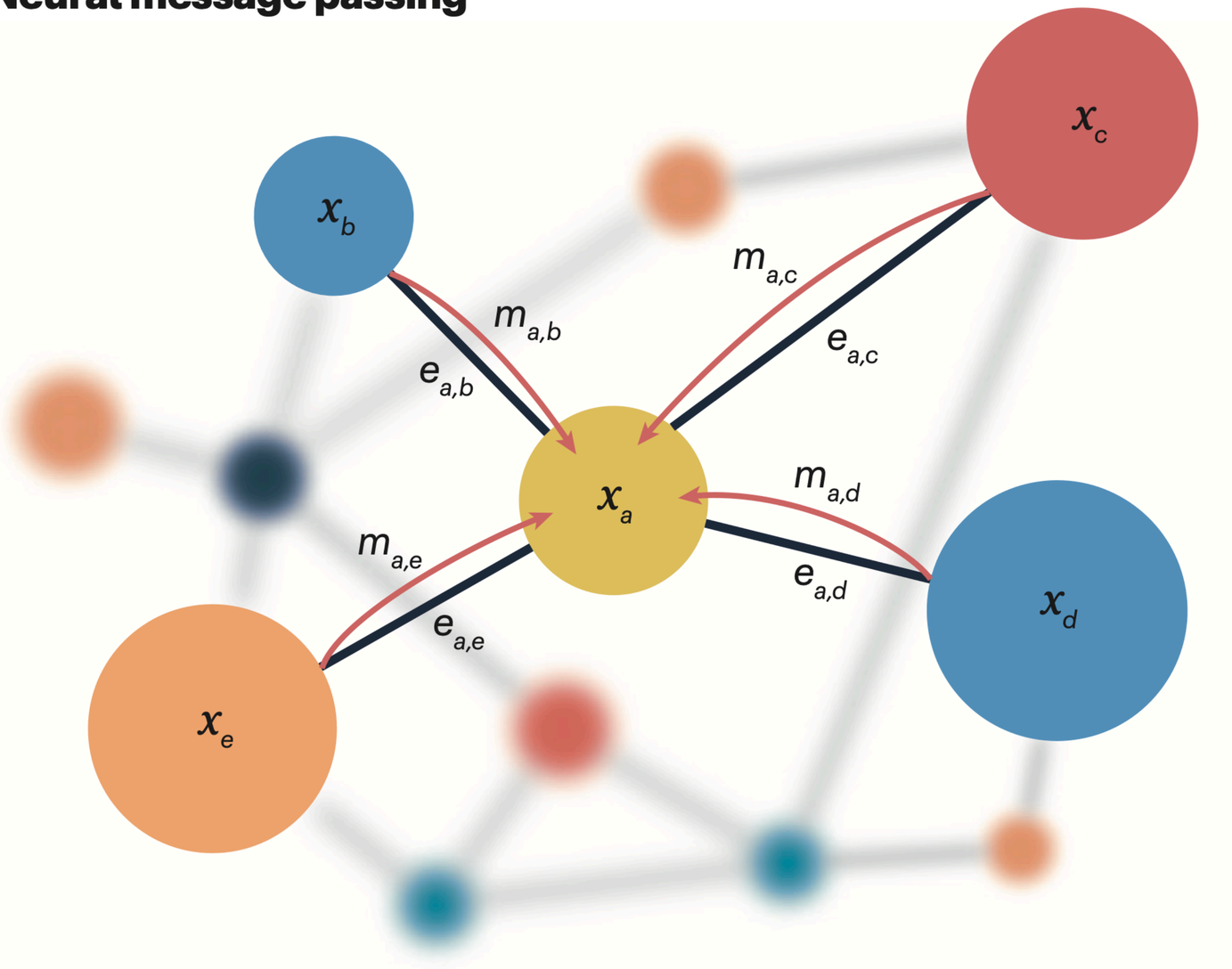
2003.01251.pdf (arxiv.org)

PHYSICS SIMULATION

$GNN($  $) \rightarrow$ 

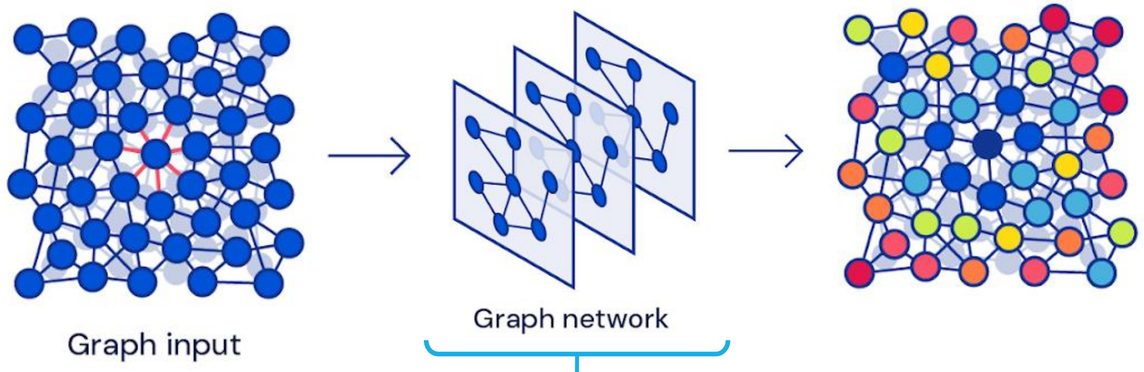
[1612.00222] Interaction Networks for Learning about Objects, Relations and Physics (arxiv.org)

b **Neural message passing**

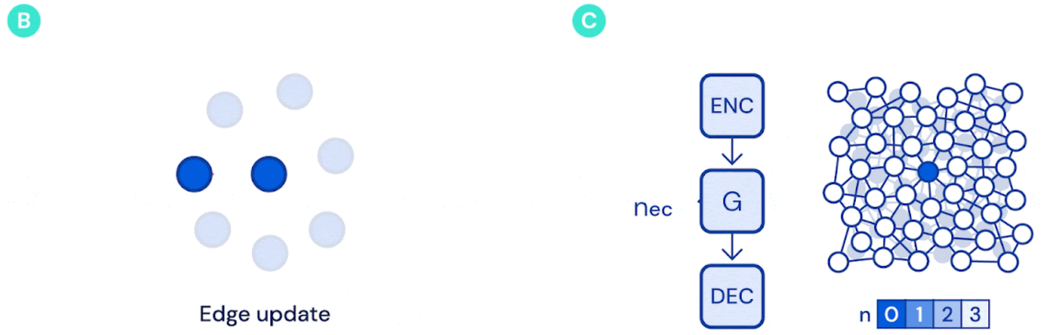


https://www.nature.com/articles/s42254-023-00569-0

GNNs: High-Level View



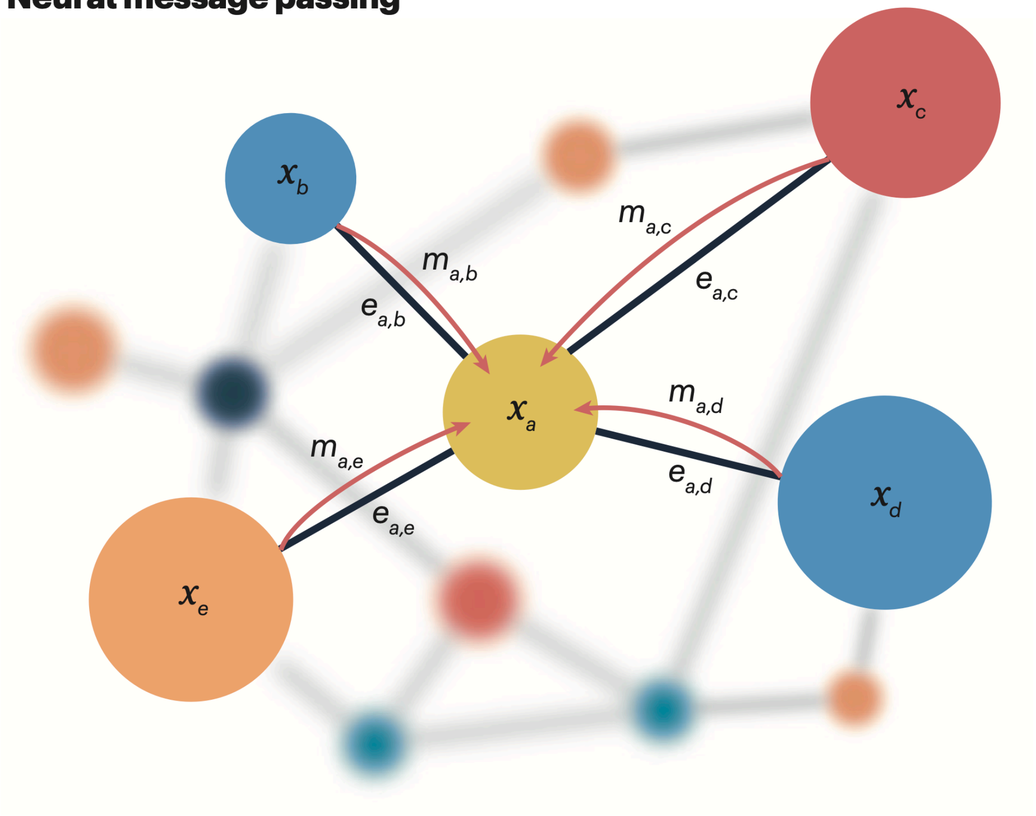
GNN Layers, Neural Message Passing



"Messages" computed from each node's neighborhood are used to update graph features… *k* iterations → info from *k*-hop neighborhoood

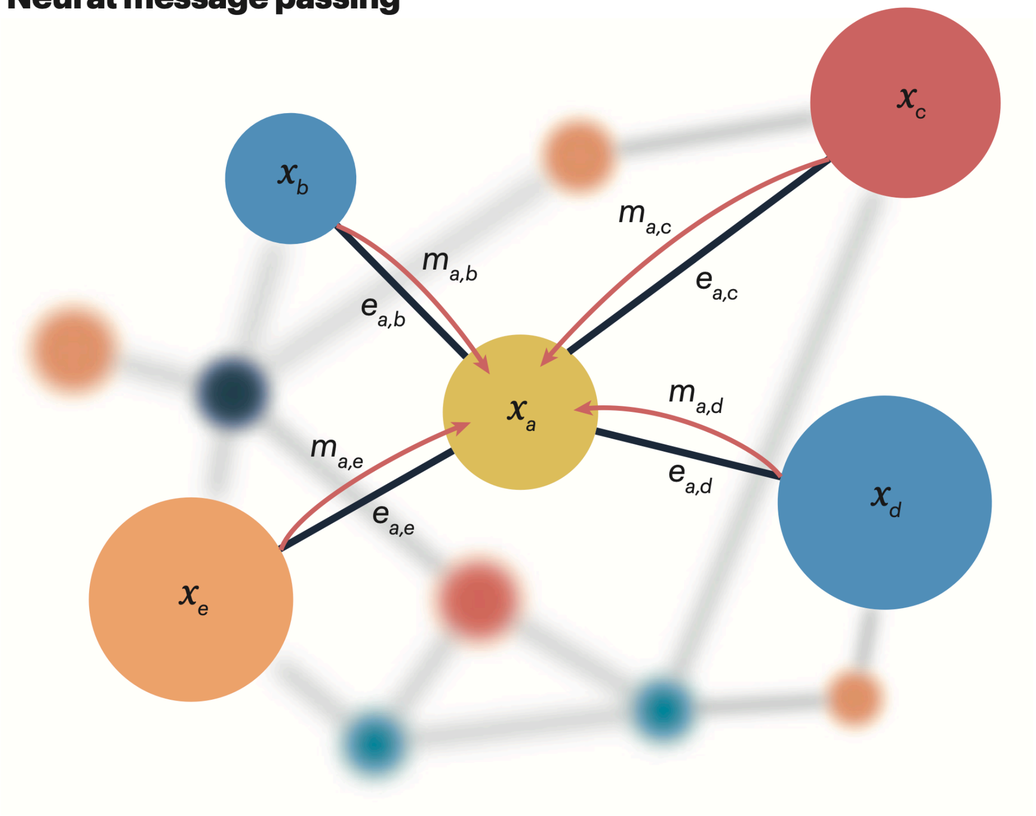
Figure Source: https://deepmind.com/blog/article/Towards-understanding-glasses-with-graph-neural-networks

b **Neural message passing**



$$\text{Message}_{ab}^{(\ell+1)} = \text{NN}_{\text{message}}^{(\ell+1)}(x_a^{(\ell)}, x_b^{(\ell)}, e_{a,b}^{(\ell)})$$

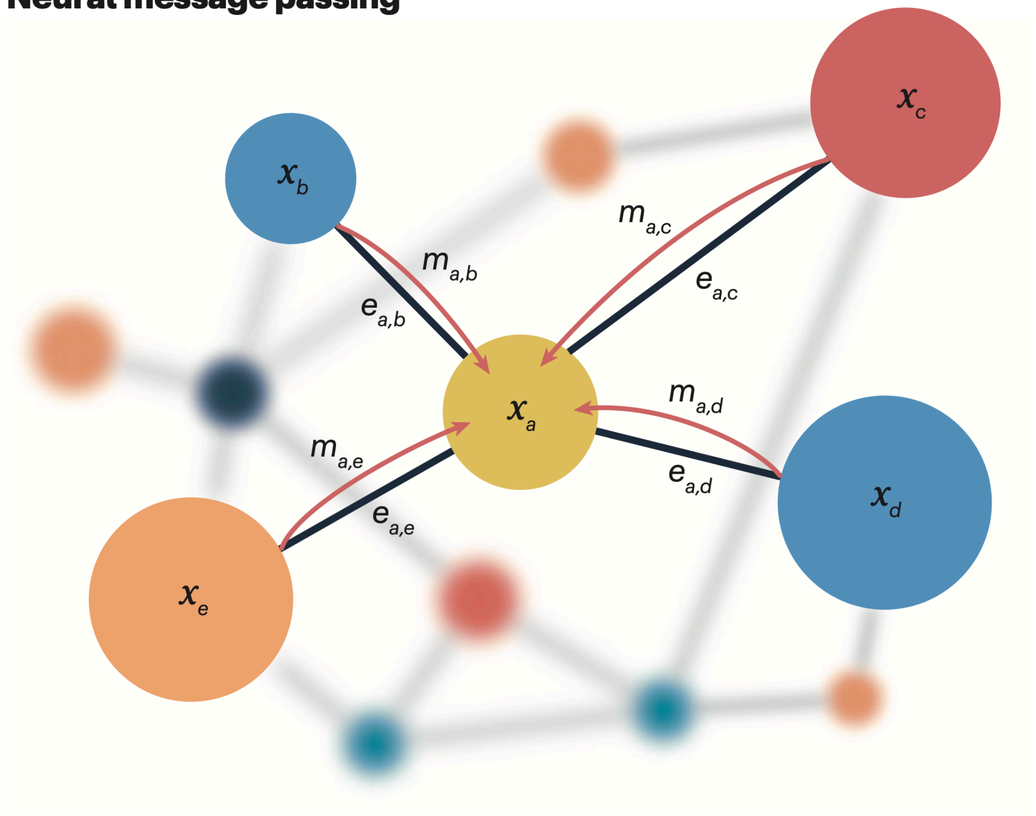
b **Neural message passing**



The goal is to update node *a*.

$$\text{Message}_{ab}^{(\ell+1)} = \text{NN}_{\text{message}}^{(\ell+1)}(x_a^{(\ell)}, x_b^{(\ell)}, e_{a,b}^{(\ell)})$$

b **Neural message passing**

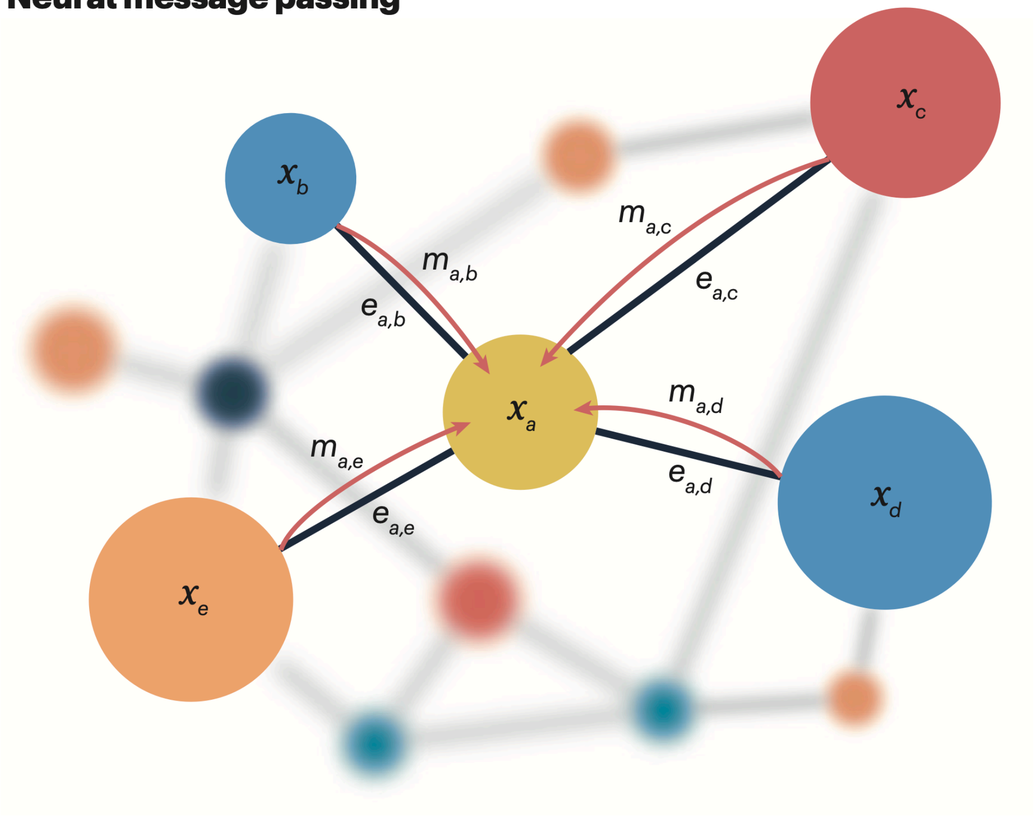


The goal is to update node *a*.

$$\text{Message}_{ab}^{(\ell+1)} = \text{NN}_{\text{message}}^{(\ell+1)}(x_a^{(\ell)}, x_b^{(\ell)}, e_{a,b}^{(\ell)})$$

…based on the values stored by its neighbors

b **Neural message passing**



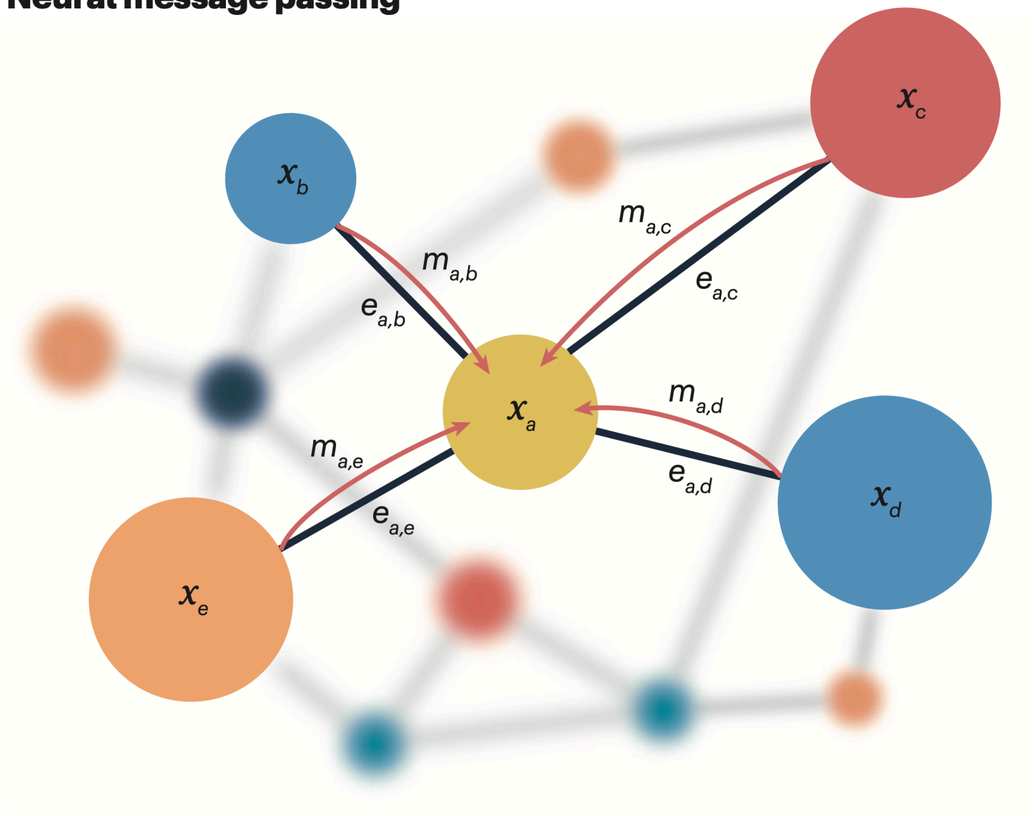
The goal is to update node *a*.

$$\text{Message}_{ab}^{(\ell+1)} = \text{NN}_{\text{message}}^{(\ell+1)}(x_a^{(\ell)}, x_b^{(\ell)}, e_{a,b}^{(\ell)})$$

…based on the values stored by its neighbors

We do this by computing messages from each neighbor *b* with a neural network

b **Neural message passing**

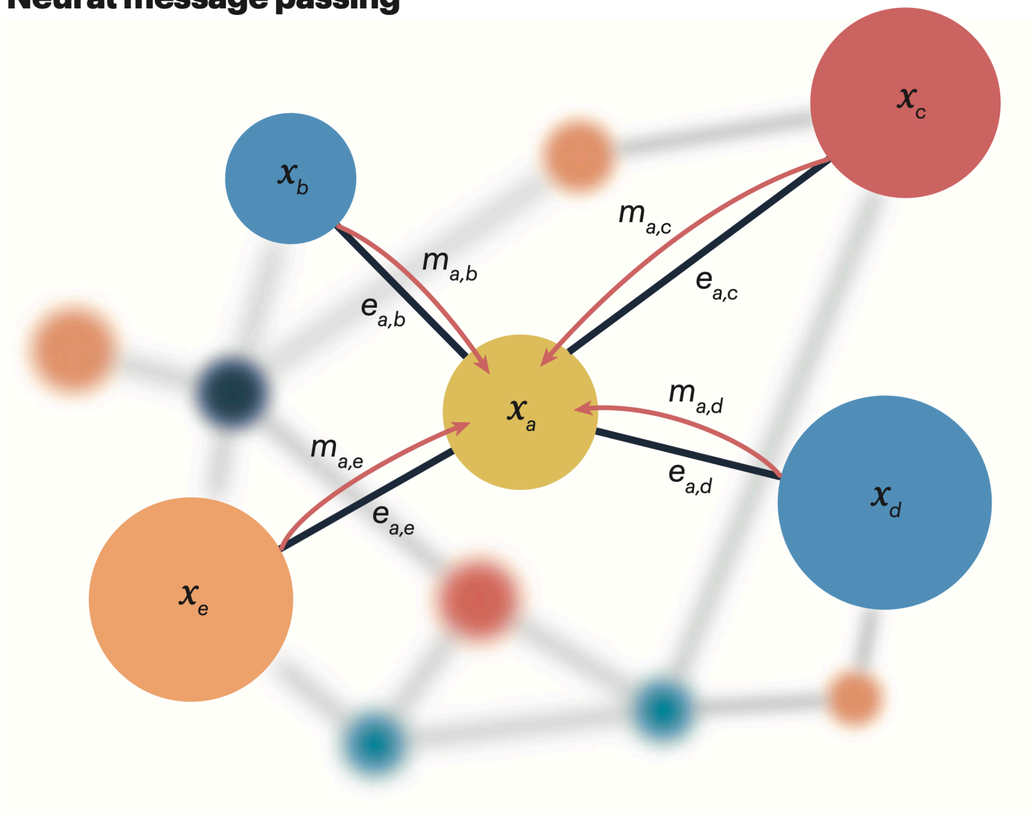


$$\text{Message}_{ab}^{(\ell+1)} = \text{NN}_{\text{message}}^{(\ell+1)}(x_a^{(\ell)}, x_b^{(\ell)}, e_{a,b}^{(\ell)})$$

$$\text{Aggr}_a^{(\ell+1)} = \sum_{b \in \mathcal{N}(a)} \text{Message}_{a,b}^{(\ell+1)}$$

Messages are *aggregated* in a permutation invariant way (here a sum) across the *neighborhood* of node *a* —> *N(a)*

b **Neural message passing**



$$\text{Message}_{ab}^{(\ell+1)} = \text{NN}_{\text{message}}^{(\ell+1)}(x_a^{(\ell)}, x_b^{(\ell)}, e_{a,b}^{(\ell)})$$

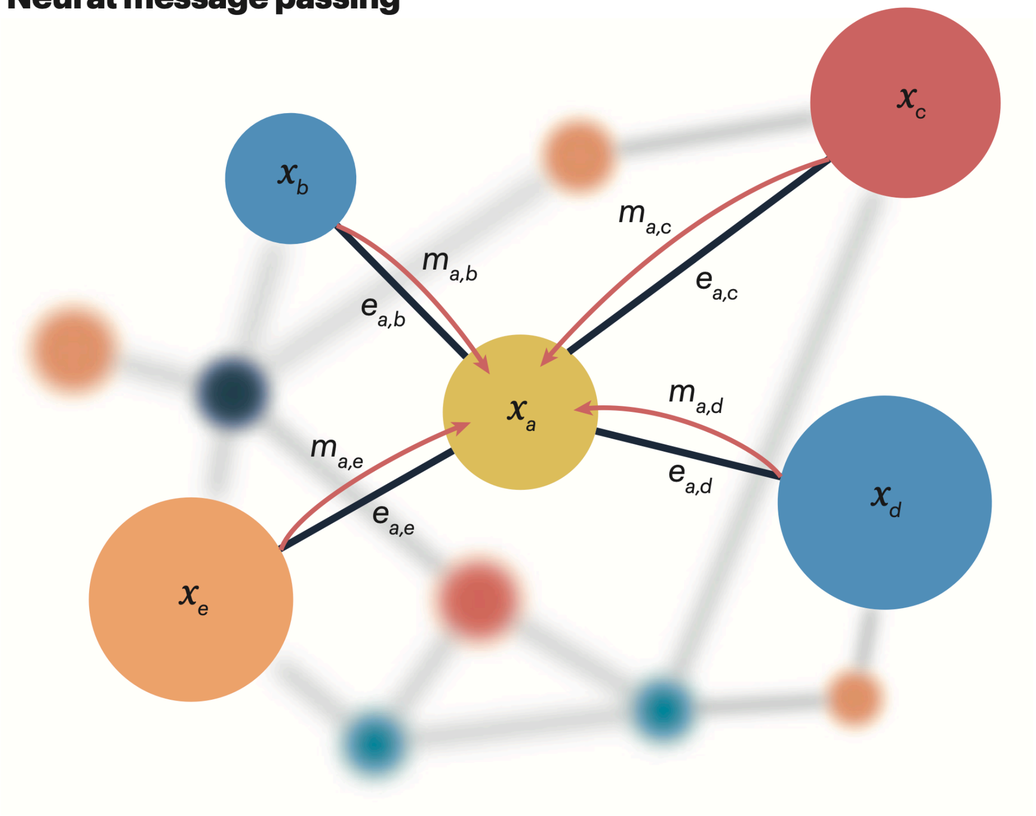
$$\text{Aggr}_a^{(\ell+1)} = \sum_{b \in \mathcal{N}(a)} \text{Message}_{a,b}^{(\ell+1)}$$

$$x_a^{(\ell+1)} = \text{NN}_{\text{update}}^{(\ell+1)}(x_a^{(\ell)}, \text{Aggr}_a^{(\ell+1)})$$

Finally, the aggregated information is used to update the state of node *a* via another NN…

this process repeats for every node in the graph!

b **Neural message passing**



$$\text{Message}_{ab}^{(\ell+1)} = \text{NN}_{\text{message}}^{(\ell+1)}(x_a^{(\ell)}, x_b^{(\ell)}, e_{a,b}^{(\ell)})$$

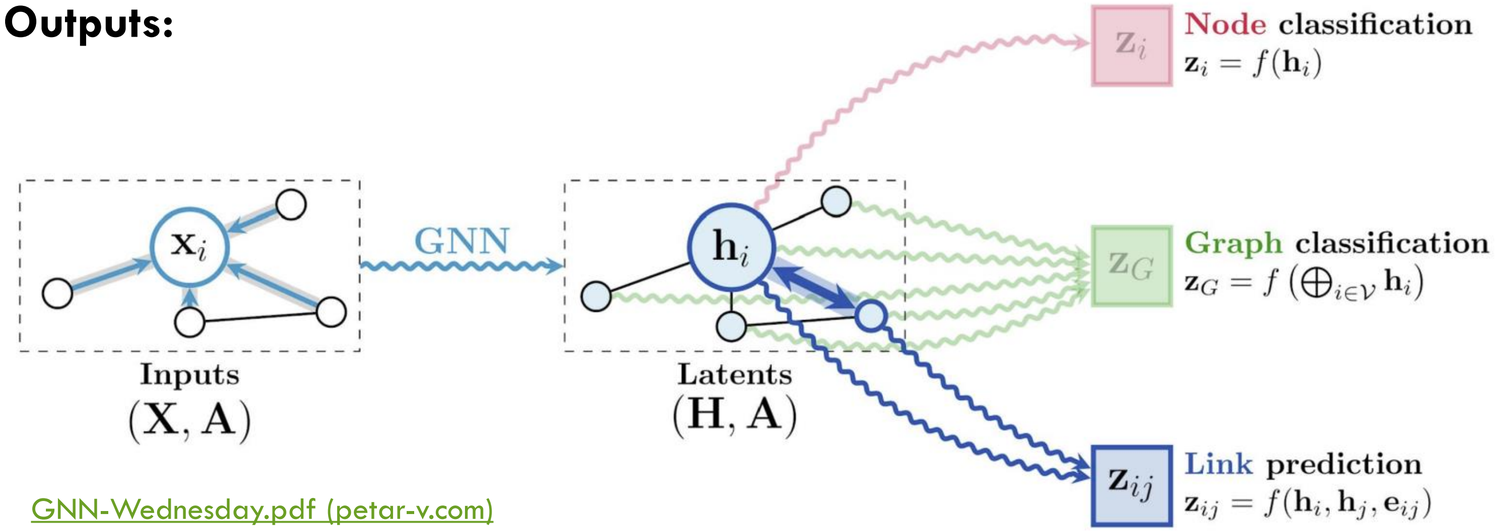
$$\text{Aggr}_a^{(\ell+1)} = \sum_{b \in \mathcal{N}(a)} \text{Message}_{a,b}^{(\ell+1)}$$

$$x_a^{(\ell+1)} = \text{NN}_{\text{update}}^{(\ell+1)}(x_a^{(\ell)}, \text{Aggr}_a^{(\ell+1)})$$

Generic MPNN Layers:

$$\boldsymbol{h}_u^{(k)} = \phi^{(k)}\left[\boldsymbol{h}_u^{(k-1)}, \bigoplus_{v \in N(u)} \psi^{(k)}\left(\boldsymbol{h}_u^{(k-1)}, \boldsymbol{h}_v^{(k-1)}, \boldsymbol{e}_{uv}^{(k-1)}\right)\right]$$

Outputs:



GNN-Wednesday.pdf (petar-v.com)

Tutorial Agenda

0. Software Setup
1. Graph-Structured Data
    1. Karate Club (node classification)
    2. ENZYMES (graph classification)
2. Graph Neural Networks
    1. GCN Math
    2. Training on Cora
3. Long Exercise: graph-level training with ENZYMES (Optional)
4. Message Passing Under the Hood (Optional)

Setup Instructions:

1. Navigate to https://github.com/GageDeZoort/prc_gnn_tutorial
2. Click on gnn_tutorial_3_27_23.ipynb
3. Change “github” to “githubtocolab” in your searchbar, i.e.:
https://colab.research.google.com/github/GageDeZoort/prc_gnn_tutorial
4. If you want to use a GPU (none of the cells require one), you can request one on Colab through Edit->Notebook Settings