GRAPH NEURAL NETWORK

Alireza Akhavanpour
http://Class.vision

AGENDA

An Introduction to Graphs and their Applications in Machine Learning

Graph Neural Networks and Implementation in TensorFlow/Keras

Implementing Graph Neural Networks in PyG

Training and Using Graph Neural Networks at Scale

Edge Features

Link Prediction and Implementing Recommender Systems

Spatio-Temporal Graph Neural Networks

Conclusion

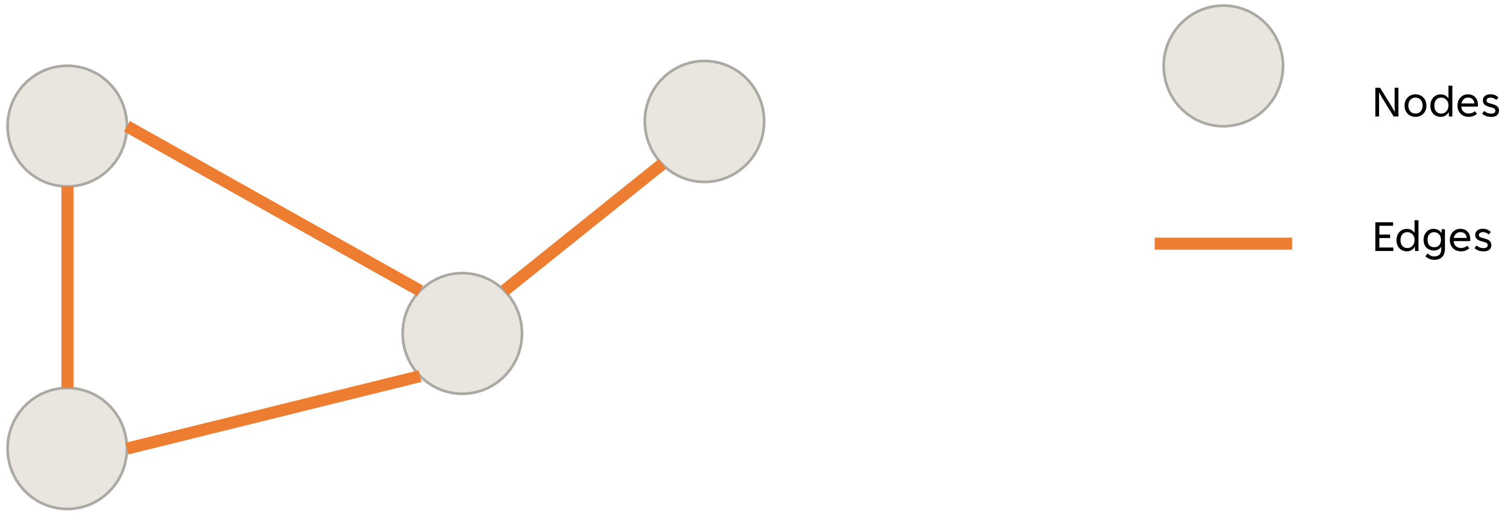
GRAPH TERMINOLOGY

What is Node, Edge, and ...

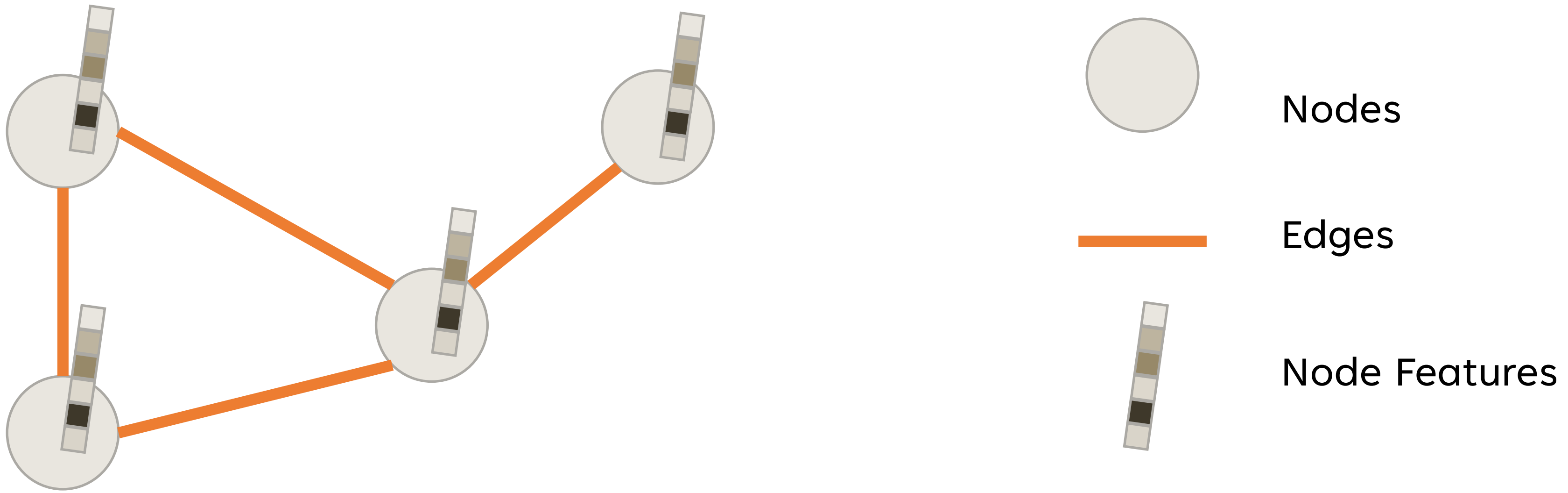
How we can store graphs?

...

GRAPH DEFINITION

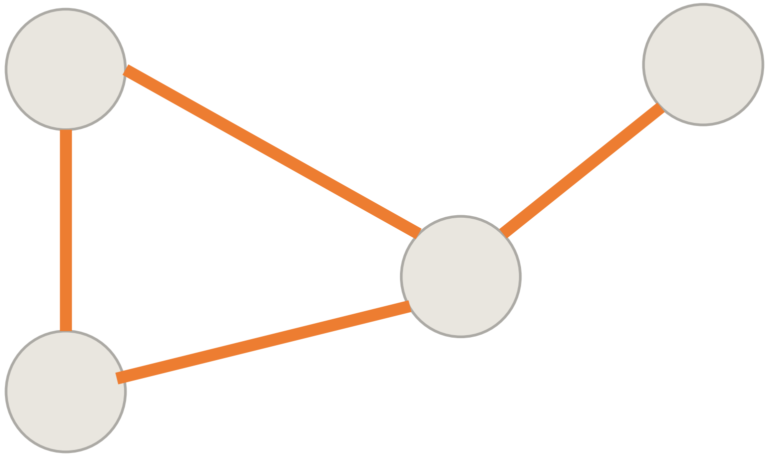


GRAPH DEFINITION



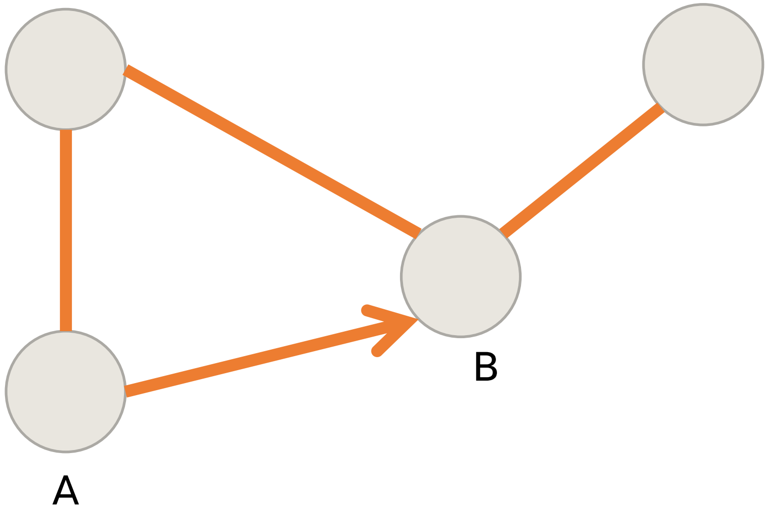
TYPES OF GRAPH

- Undirected graph
- Directed graph



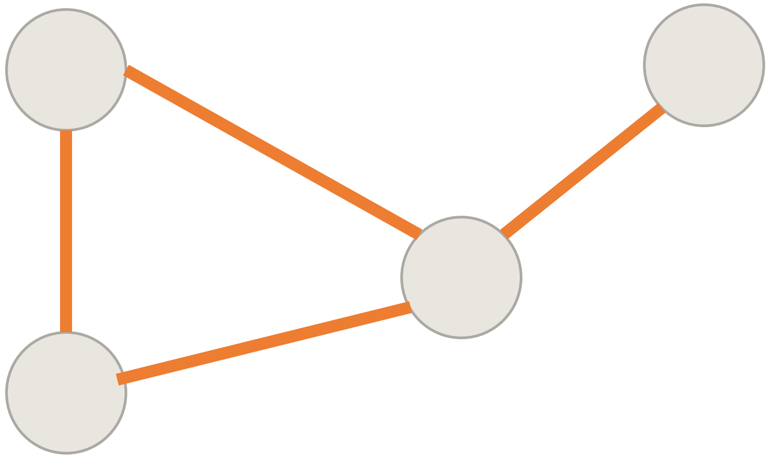
TYPES OF GRAPH

- Undirected graph ↔ or —
- Directed graph →

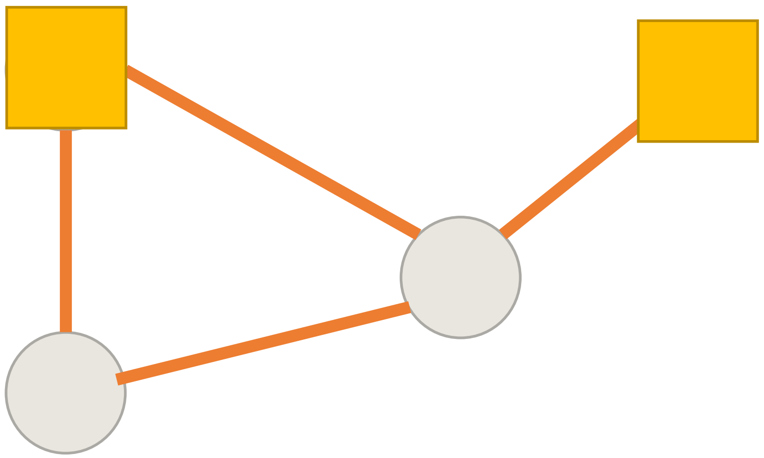


TYPES OF GRAPH

- Homogeneous graph
- Heterogeneous graph



Homogeneous

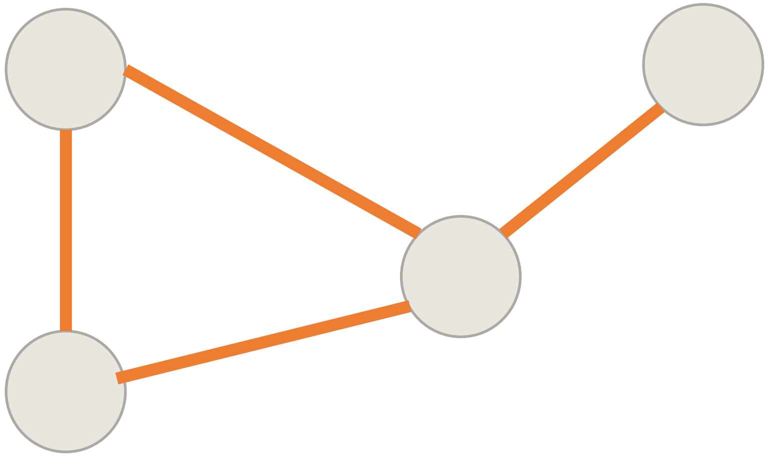


Heterogeneous

GRAPH EXAMPLE

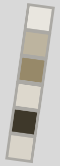
$$G = (V, E, u)$$

- u → FEATURE VECTORS
- E → EDGES (Adjacency, Weight) = (A,W)
- V → VERTECIES (NODES)



Nodes
Social media accounts

Edges
People connection

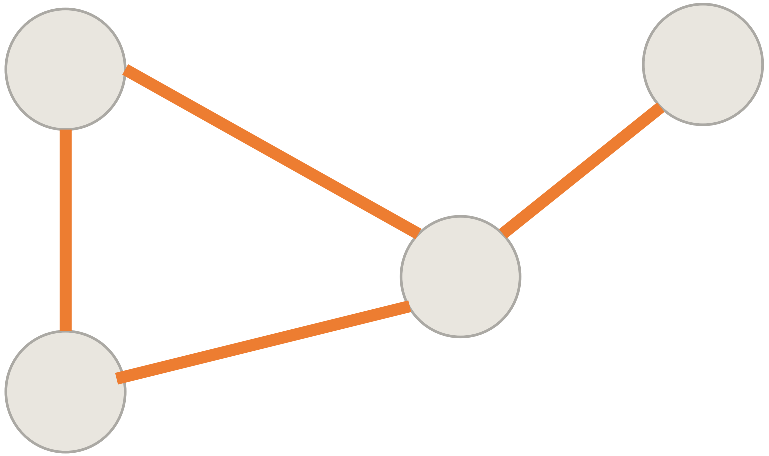


Node Features
Age, Gender, ...

GRAPH EXAMPLE

$$G = (V, E, u)$$

- u → FEATURE VECTORS
- E → EDGES (Adjacency, Weight) = (A,W)
- V → VERTECIES (NODES)



Nodes
Social media accounts

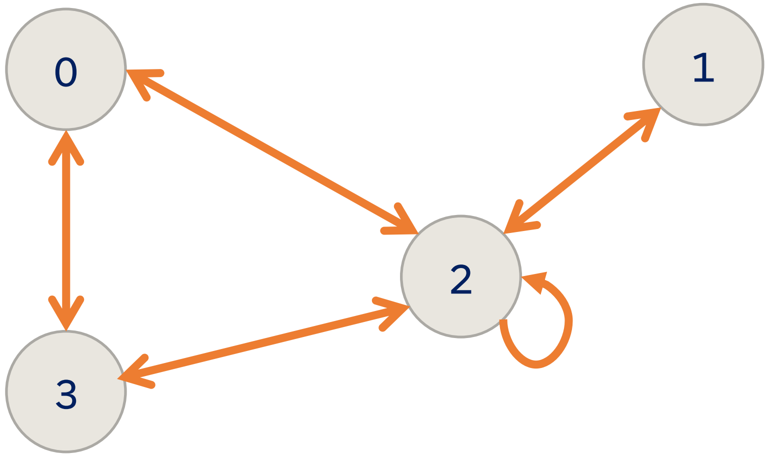
Edges
People connection

Node Features
Age, Gender, ...

Undirected graph ⇔ or —
Facebook

Directed graph →
Instagram

STORING GRAPH

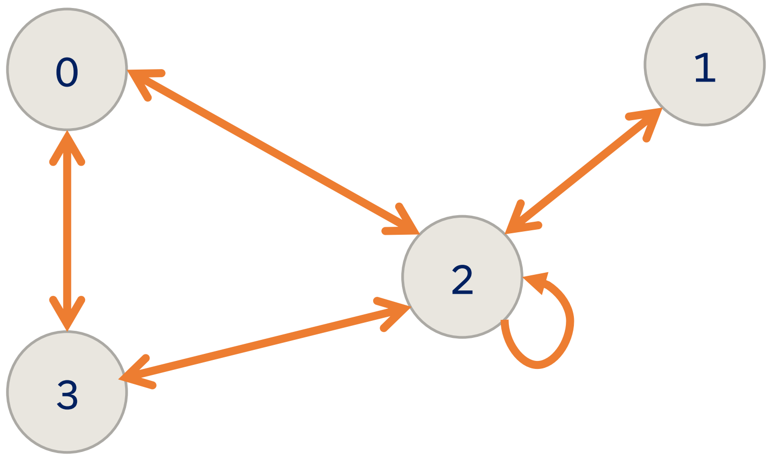


Homogeneous

Source Node, Target Node

Edge list: $\begin{bmatrix} (0,1) \\ (0,2) \\ (0,3) \\ (1,0) \\ (2,0) \\ (2,2) \\ (2,3) \\ (3,0) \\ (3,2) \end{bmatrix}$

STORING GRAPH



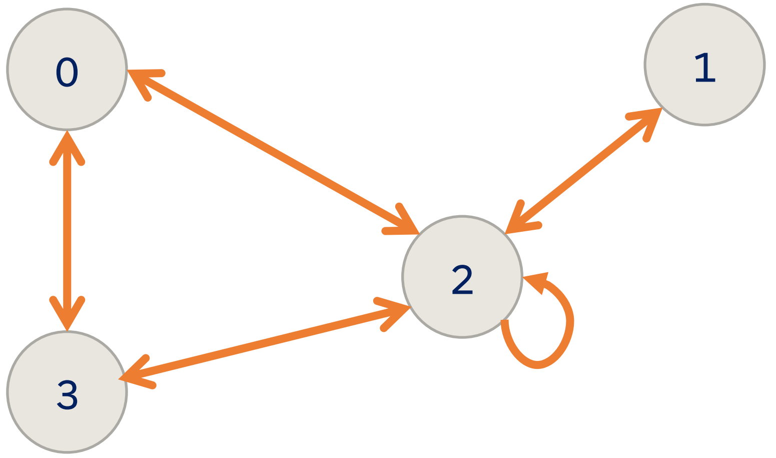
Homogeneous

Adjacency Matrix:

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

$$V \times V$$

STORING GRAPH



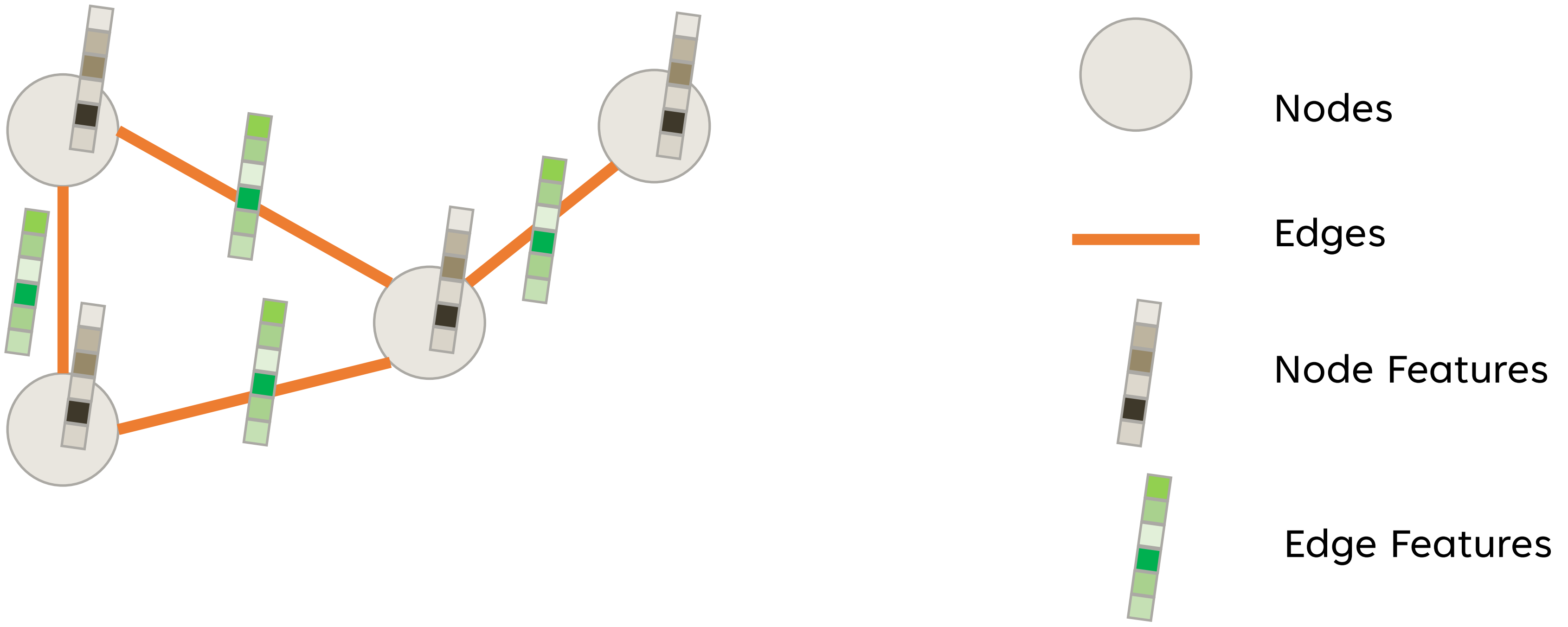
Homogeneous

Adjacency Matrix:

	0	1	2	3
0	0	2	1.5	4
1	5	0	0	0
2	1.5	0	1	1
3	12	0	0	1

We can use **weight** instead of Boolean!
To show how strong the connection is!

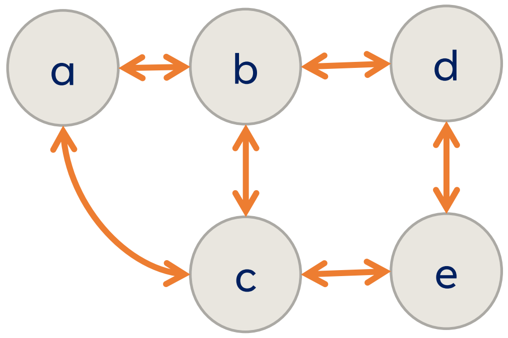
EDGE FEATURES



YOU CAN MODEL COMPLEX SYSTEMS, DEPENDING ON HOW YOU CHOOSE TO DEFINE THE GRAPH

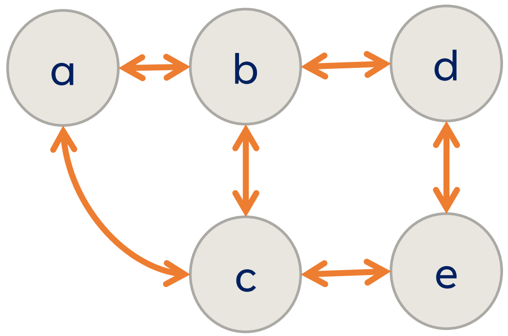
- **Edge type**:
  weighted vs binary
- **Edge directionality**:
  undirected vs directed
- **Features:**
  None, node-based, edge-based
- **Temporal Aspects**:
  Features, topology
- **Others**:
  Multi-graphs, hypergraphs, complex networks

GRAPH DEGREE



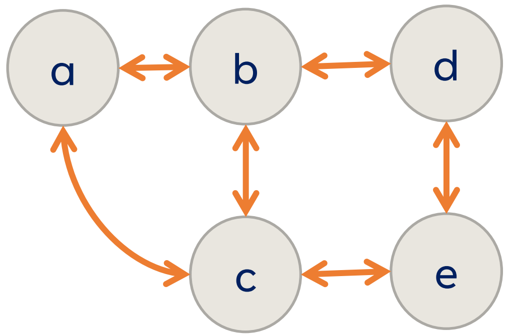
$$x = \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix}$$

GRAPH DEGREE



$$x = \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} \qquad A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

GRAPH DEGREE



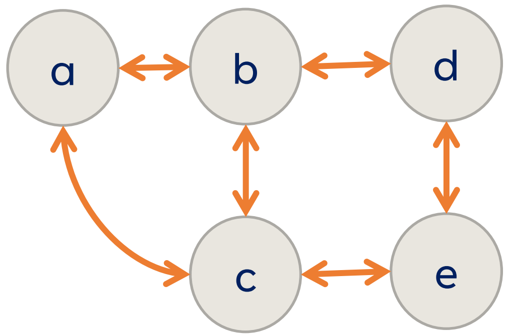
$$x = \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} \qquad A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Degree matrix (D) is a **diagonal matrix** defining number of connection per node

$$D = \begin{bmatrix} \mathbf{2} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{3} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{3} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{2} & 0 \\ 0 & 0 & 0 & 0 & \mathbf{2} \end{bmatrix}$$

Degree matrix shows influence of each node on the whole graph

LAPLACIAN OF GRAPH

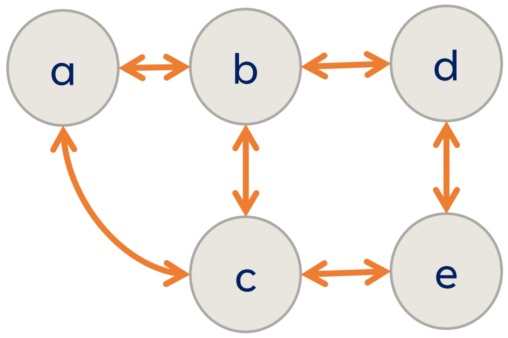


$$x = \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} \quad A = \begin{bmatrix} 0&1&1&0&0 \\ 1&0&1&1&0 \\ 1&1&0&0&1 \\ 0&1&0&0&1 \\ 0&0&1&1&0 \end{bmatrix} \quad D = \begin{bmatrix} 2&0&0&0&0 \\ 0&3&0&0&0 \\ 0&0&3&0&0 \\ 0&0&0&2&0 \\ 0&0&0&0&2 \end{bmatrix}$$

Laplacian matrix (L) is a L = D — A OR L = D — W in weighted matrix

$$L = \begin{bmatrix} 2 & -1 & -1 & 0 & 1 \\ -1 & 3 & -1 & -1 & 0 \\ -1 & -1 & 3 & 0 & -1 \\ 0 & -1 & 0 & 2 & -1 \\ 0 & 0 & -1 & -1 & 2 \end{bmatrix}$$

NORMALIZED GRAPH

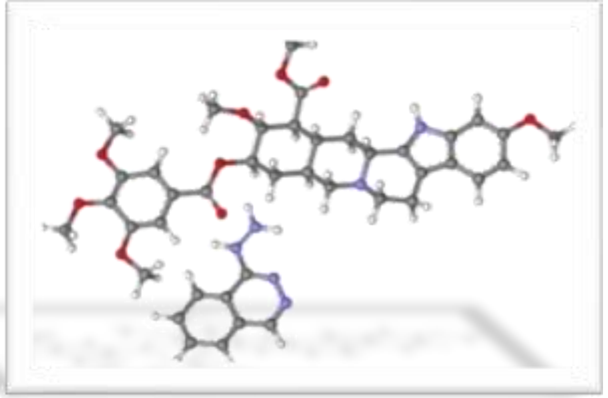


We can decide to show the relation between of the nodes, with any of the following matrices:

$$A, L, \bar{A}, \bar{L}$$

GRAPH USAGE AND APPLICATIONS

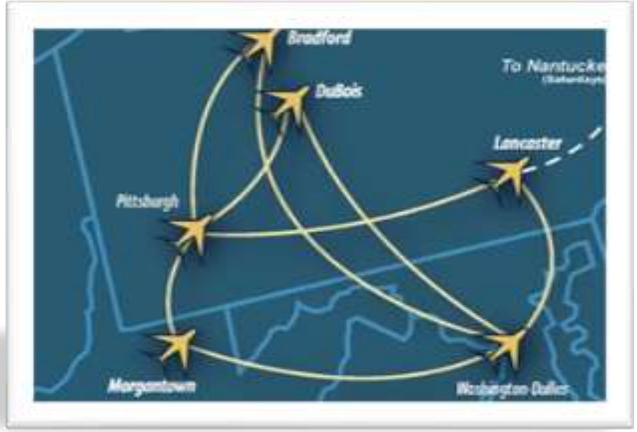
GRAPH DATA IS EVERYWHERE



Medicine/ pharmacy

Recommender system

Social Networks



Airports connection

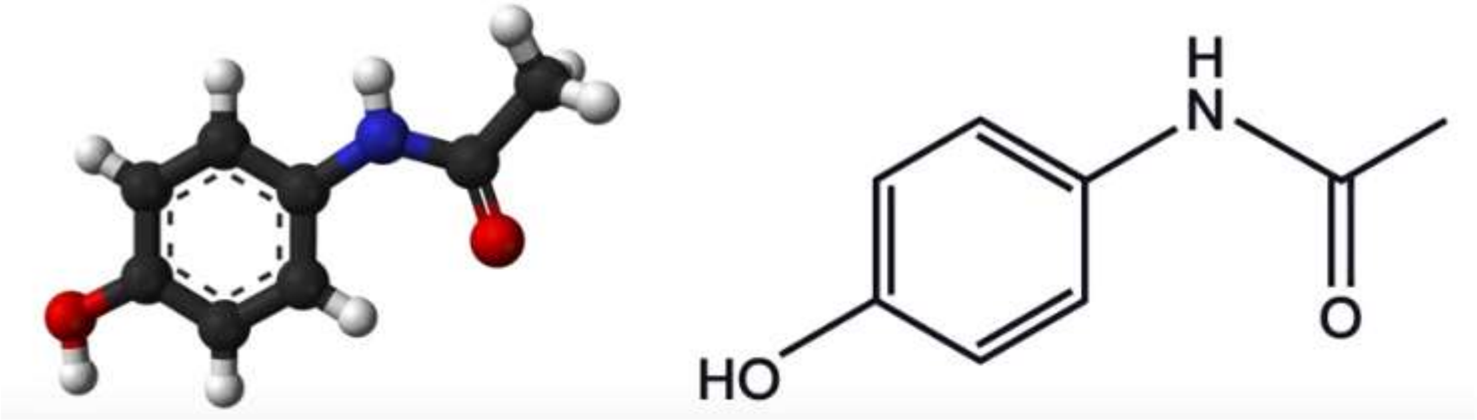
Brain cortex

Traffic map

MOLECULES ARE GRAPHS!

A very natural way to represent molecules is as a **graph**

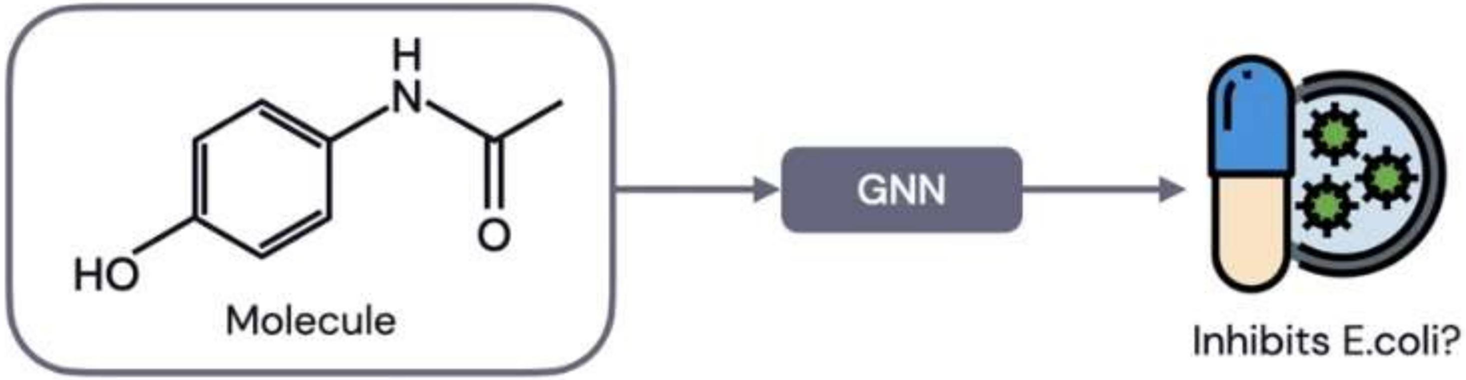
- **Atoms** as nodes, **bonds** as edges
- Features such as atom type, charge, bond type..



GNNS FOR MOLECULE CLASSIFICATION

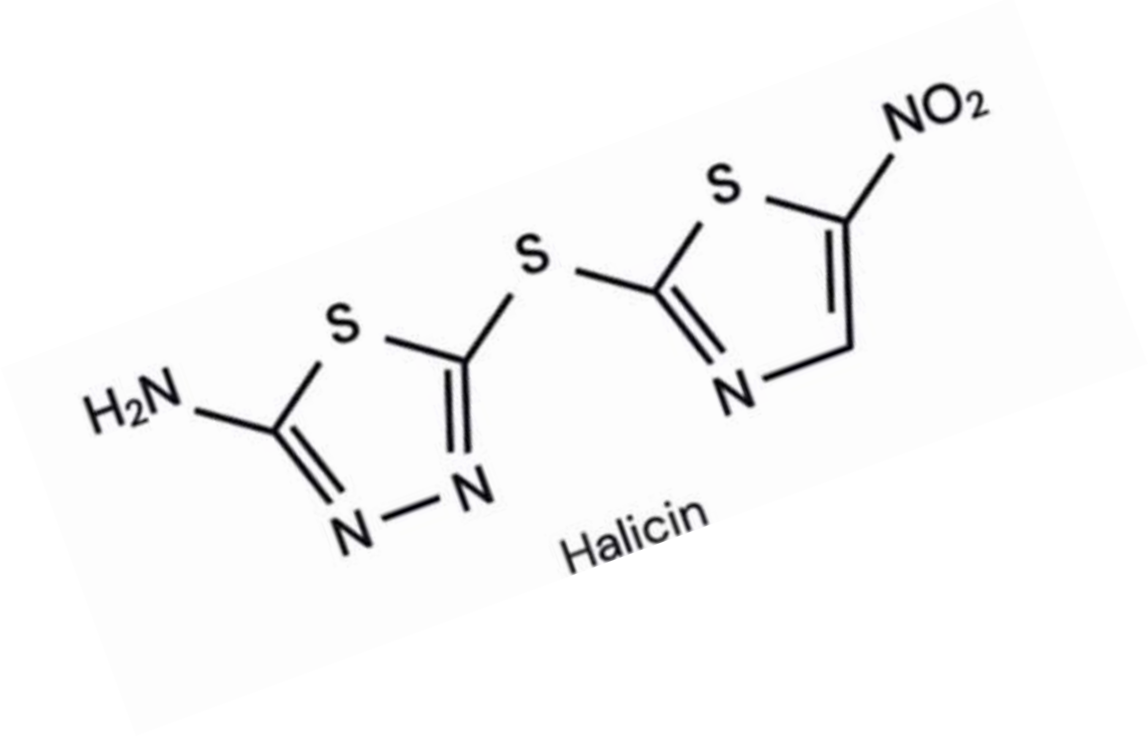
Interesting task to predict is, for example, whether the molecule is a potent drug

- Can do binary classification on whether the drug will inhibit certain bacteria. (E.coli)
- Train on a curated dataset for compounds where response is known.



FOLLOW-UP STUDY

- Once trained, the model can be applied to any molecule.
  - Execute on a large dataset of known candidate molecules.
  - Select the —top-100 candidates from your GNN model.
  - Have chemists thoroughly investigate those (after some additional filtering).
- Discover a previously overlooked compound that is a highly potent antibiotic!



SUCCESS STORY!

Cell

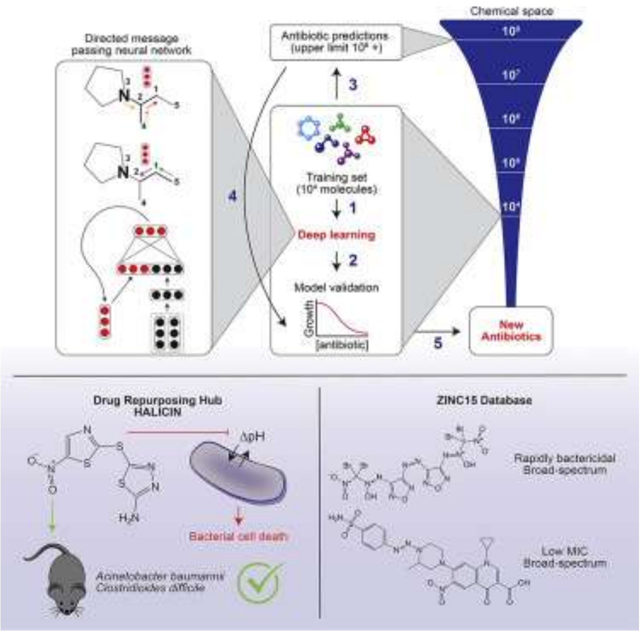
CellPress

Volume 180, Issue 4, 20 February 2020, Pages 688-702.e13

Article

A Deep Learning Approach to Antibiotic Discovery

Jonathan M. Stokes [1 2 3], Kevin Yang [3 4 10], Kyle Swanson [3 4 10], Wengong Jin [3 4], Andres Cubillos-Ruiz [1 2 5], Nina M. Donghia [1 5], Craig R. MacNair [6], Shawn French [6], Lindsey A. Carfrae [6], Zohar Bloom-Ackermann [2 7], Victoria M. Tran [2], Anush Chiappino-Pepe [5 7], Ahmed H. Badran [2], Ian W. Andrews [1 2 5], Emma J. Chory [1 2], George M. Church [5 7 8], Eric D. Brown [6], Tommi S. Jaakkola [3 4], Regina Barzilay [3 4 9], James J. Collins [1 2 5 8 9 11]



SUCCESS STORY!

nature

Subscribe

NEWS · 20 FEBRUARY 2020

Powerful antibiotics discovered using AI

Machine learning spots molecules that work even against 'untreatable' strains of bacteria.

SUCCESS STORY!

SUCCESS STORY!

BBC Sign in | News | Sport | Reel | Worklife | Travel | Future

NEWS

Home | Video | World | UK | Business | Tech | Science | Stories | Entertainment & Arts

BBC WORKLIFE Our new guide for getting ahead

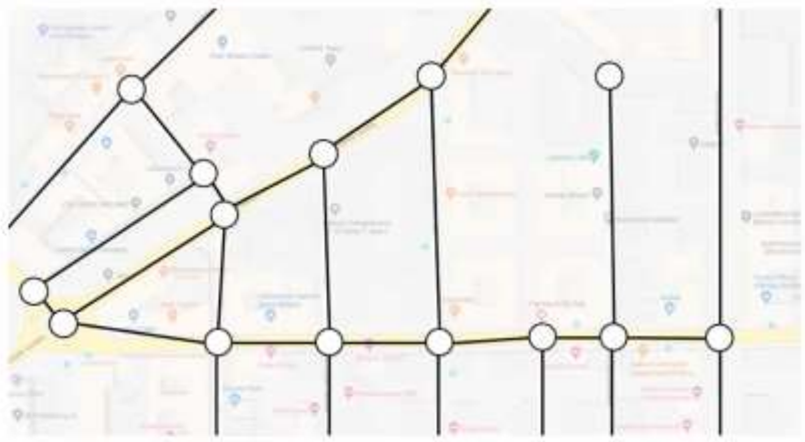
Scientists discover powerful antibiotic using AI

21 February 2020

Share

TRAFFIC MAPS ARE GRAPHS!

Transportation maps (e.g. the ones found on Google Maps) naturally modeled as graphs.



Nodes could be **intersections**, and **edges** could be **roads**.
(Relevant **node features**: road length, current speeds, historical speeds)

DEEPMIND'S ETA PROBLEM!

Partition candidate route into super-segments, sampled proportionally to (est.) traffic density.

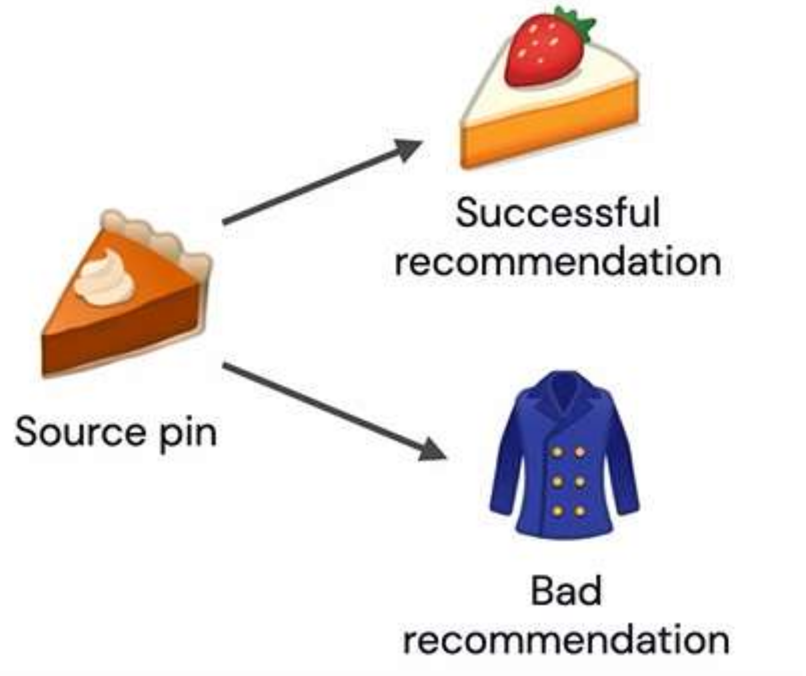
Run GNN on super-segment graph to estimate estimated time of arrival (ETA) (graph regression).

https://class.vision/blog/گوگل-مپ-ترافیک-شبکه-عصبی-گرافی/

RECOMMENDER SYSTEMS

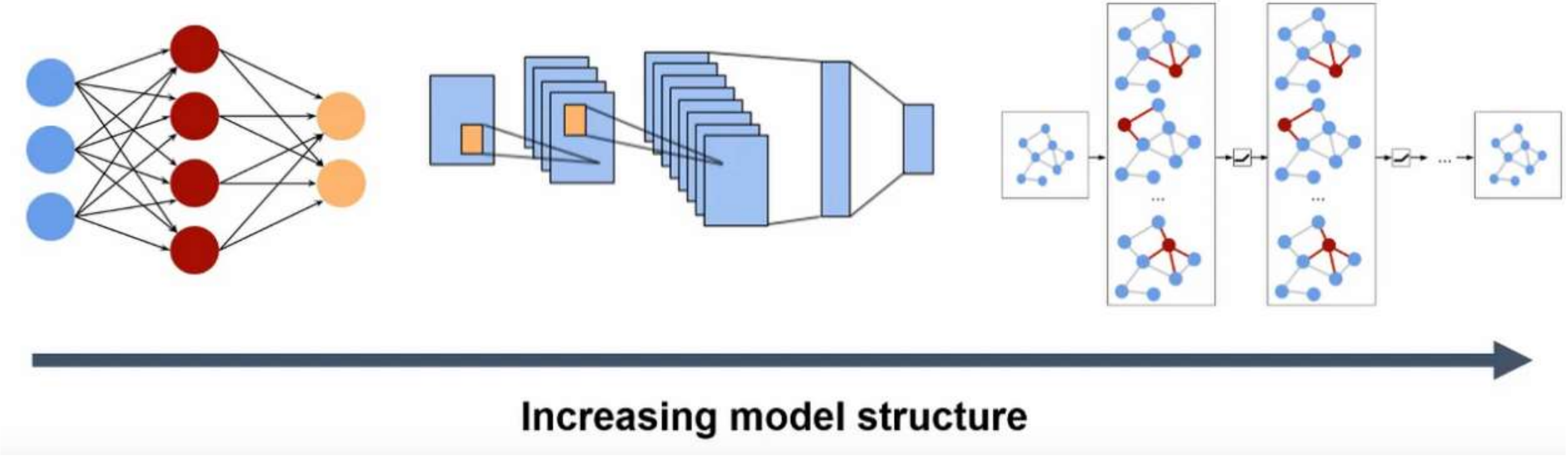
A common task on **social networks** is **recommendation.**

- Based on a user's preferences, recommend new content
- Can leverage existing links as adjacency input to a (link-prediction) GNN!
- Major issue: our methods (so far) assume the graph is processed all-at- once! (one solution is GraphSAGE)



GRAPH CHALLENGE AND PROBLEMS

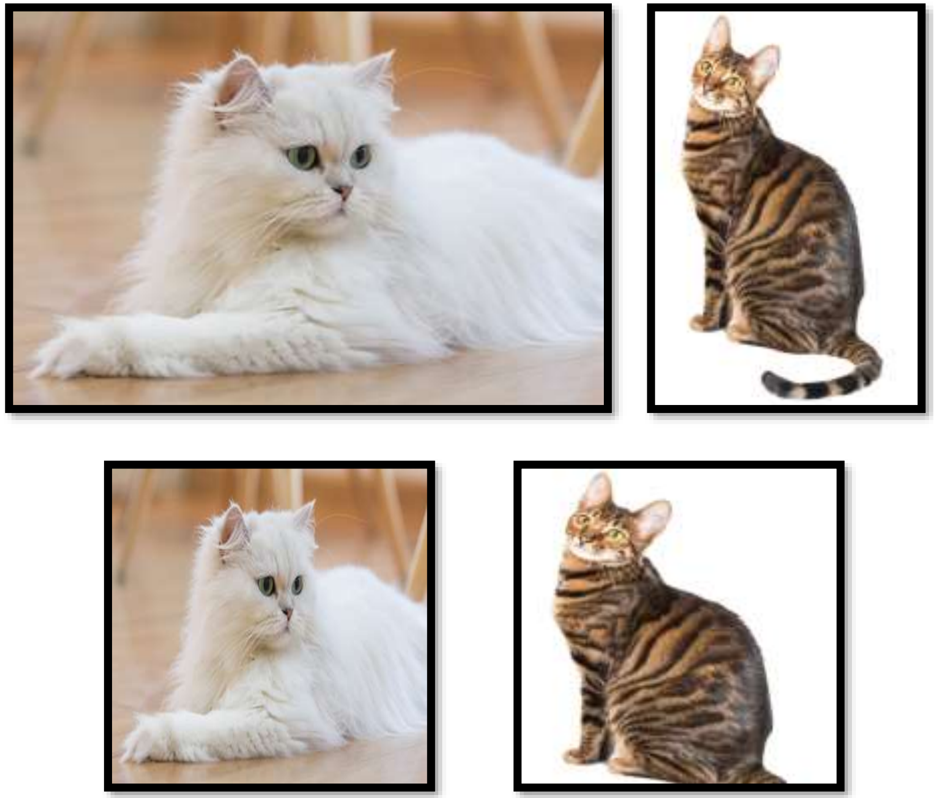
WHY USE GRAPHS?
WHY NOT JUST USE MLP OR ATTENTION AND LEARN "EVERYTHING" END-TO-END?

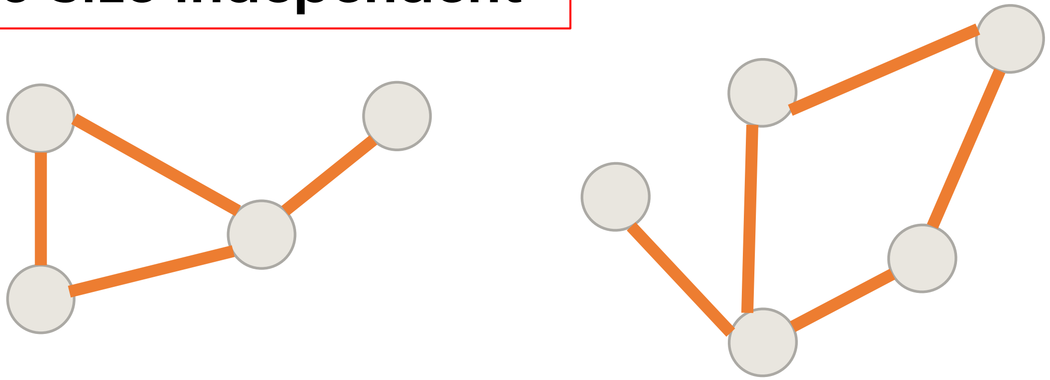


PROBLEM: GRAPH DATA IS DIFFERENT

Challenge 1: Data size and shape

It should be **Size independent**

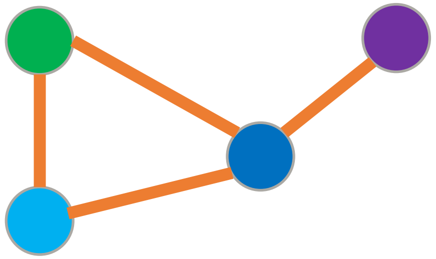




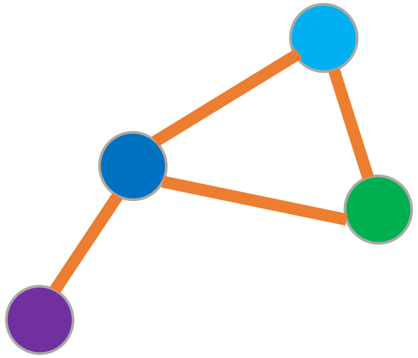
PROBLEM: GRAPH DATA IS DIFFERENT

Challenge 2: Isomorphism

It should be **Permutation invariance**

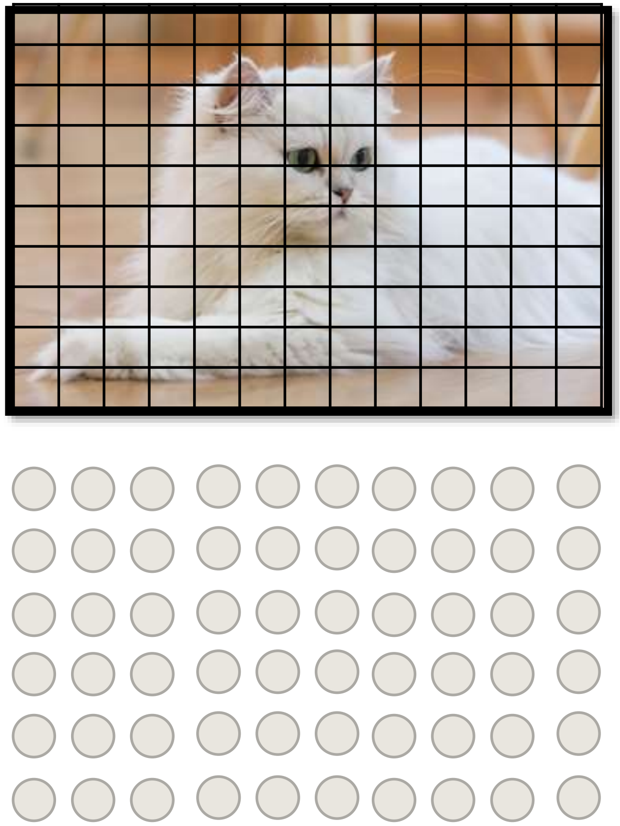


We cannot feed adjacency matrix to MLP

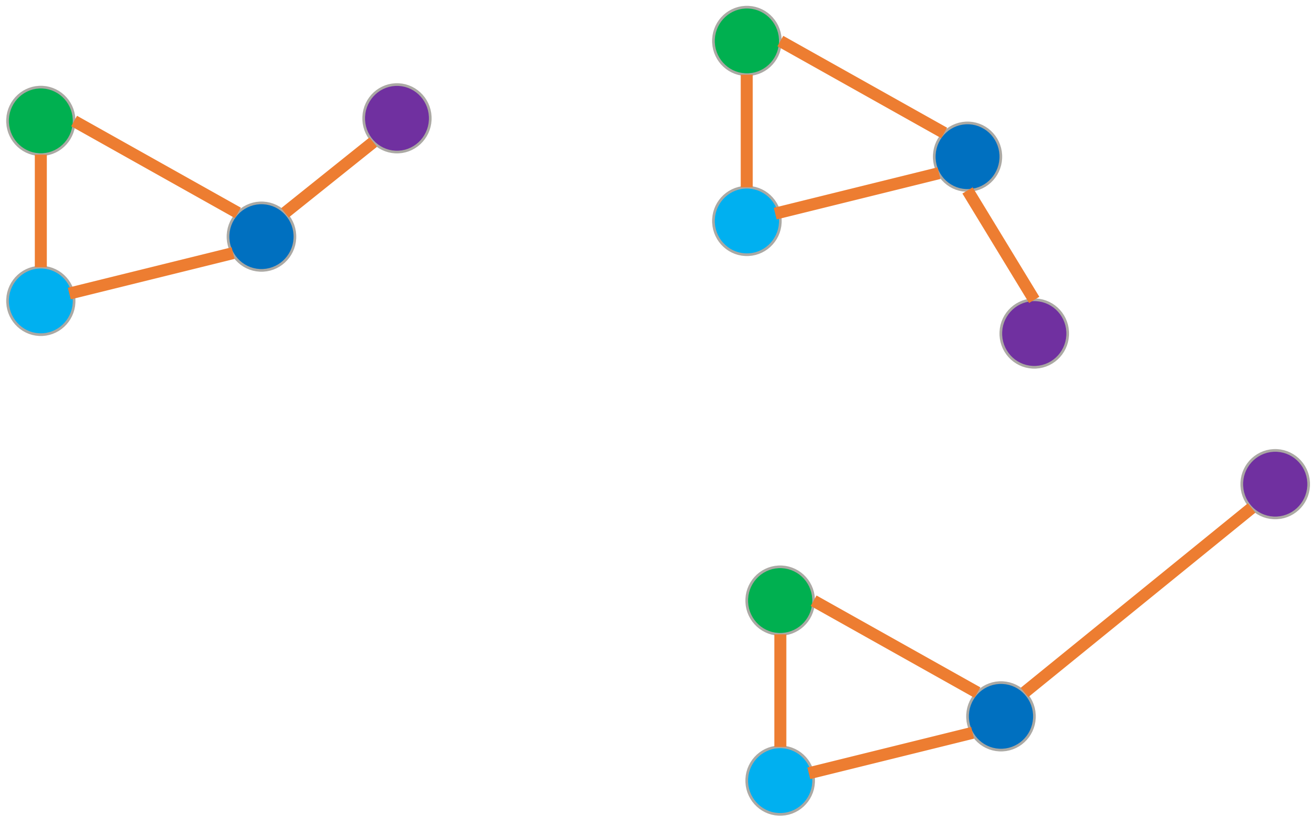


PROBLEM: GRAPH DATA IS DIFFERENT

Challenge 3: Grid structure



It should be in **Non-Euclidean space**



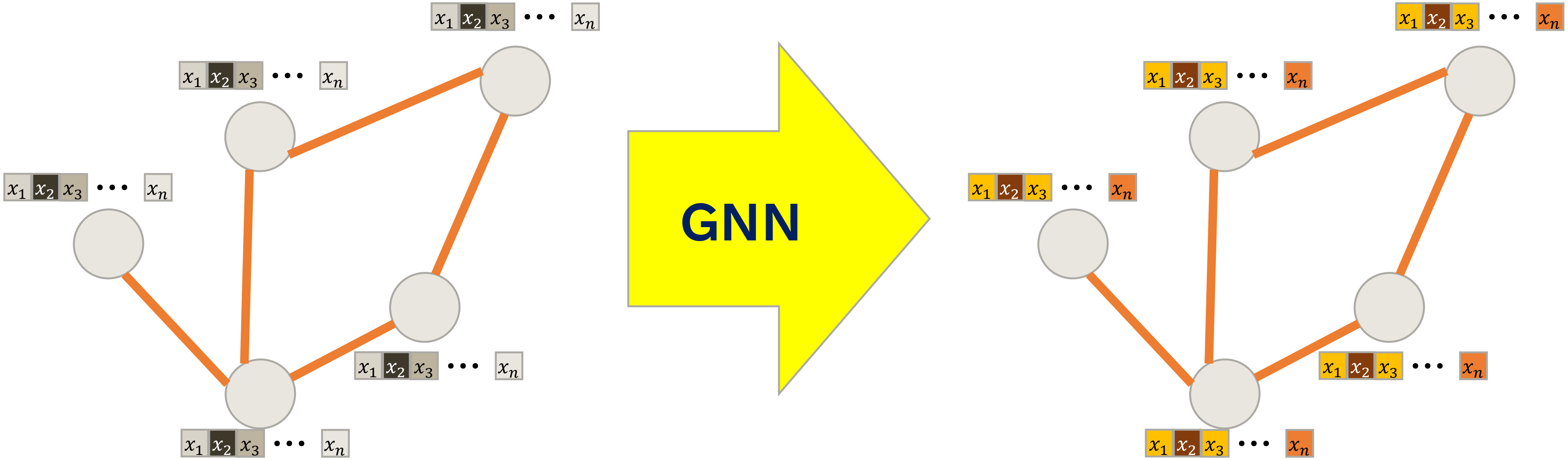
OTHER CHALLENGES WITH GRAPH CONVOLUTIONS

Desirable properties for a graph convolutional layer:

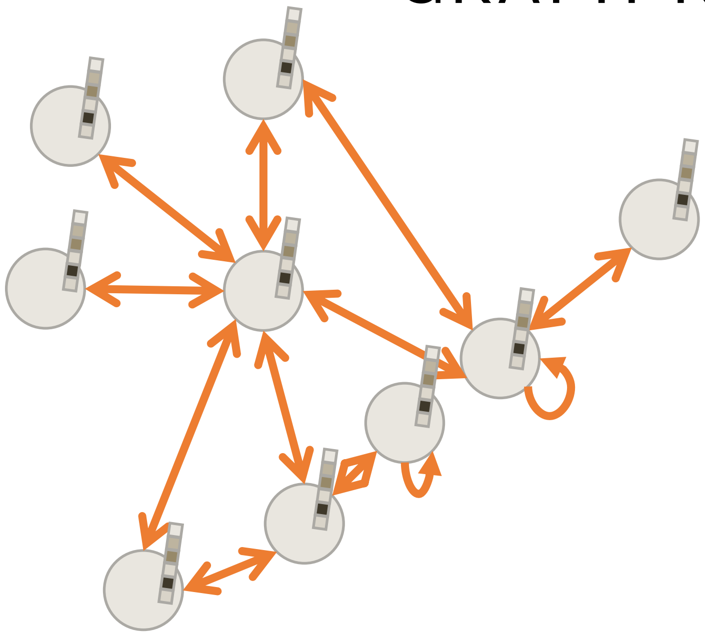
- ❑ Computational and **storage** efficiency (**~O(V + E)**)
- ❑ Fixed number of **parameters** (independent of input size)
- ❑ **Localisation** (acts on a local neighbourhood of a node)
- ❑ Specifying **different importances** to different neighbours
- ❑ Applicability to **inductive** problems.

LEARNING IN GRAPH

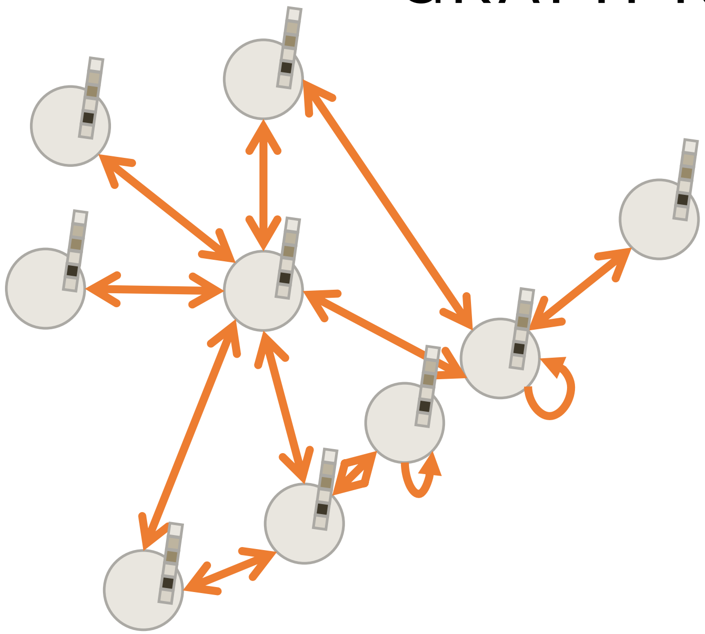
REPRESENTATION LEARNING

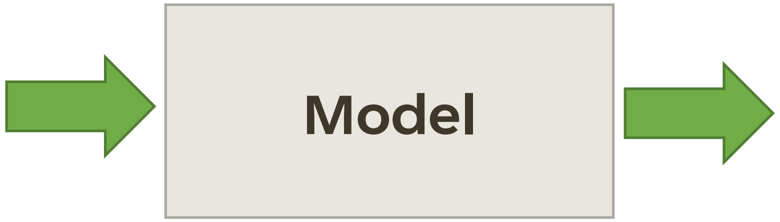


LEARNING IN
GRAPH REPRESENTATION LEARNING

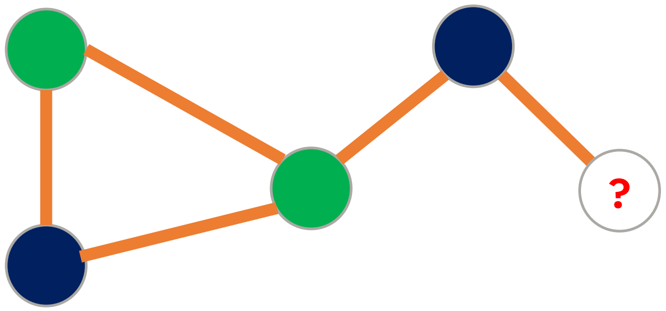


LEARNING IN GRAPH REPRESENTATION LEARNING

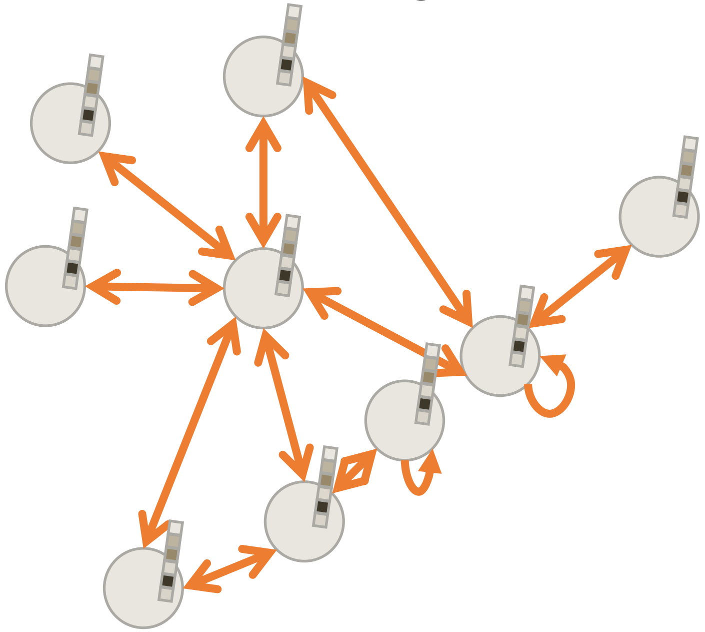


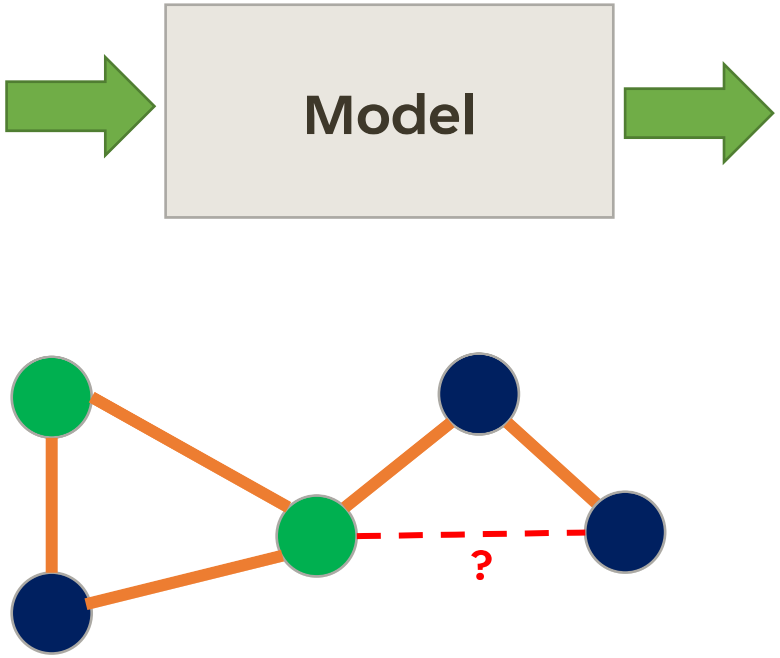


- ✓ Node Prediction
  (Node-level Prediction)



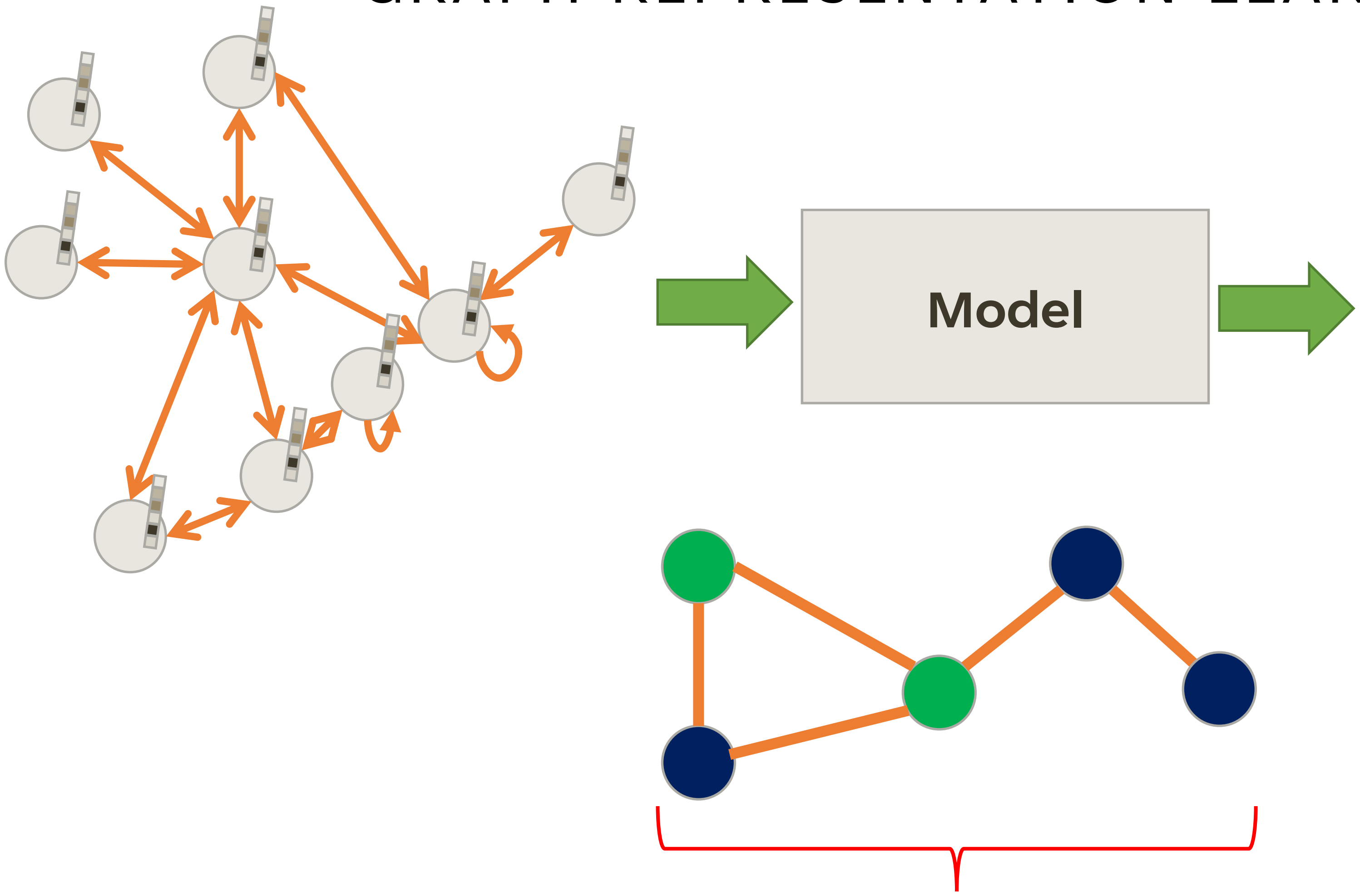
LEARNING IN GRAPH REPRESENTATION LEARNING





- ✓ Node Prediction (Node-level Prediction)
- ✓ Link Prediction (Edge-level prediction)

LEARNING IN GRAPH REPRESENTATION LEARNING



- ✓ Node Prediction (Node-level Prediction)
- ✓ Link Prediction (Edge-level prediction)
- ✓ Graph representation (Graph-level prediction)

WHAT TYPES OF PROBLEMS CAN GNNS SOLVE?

Unsupervised

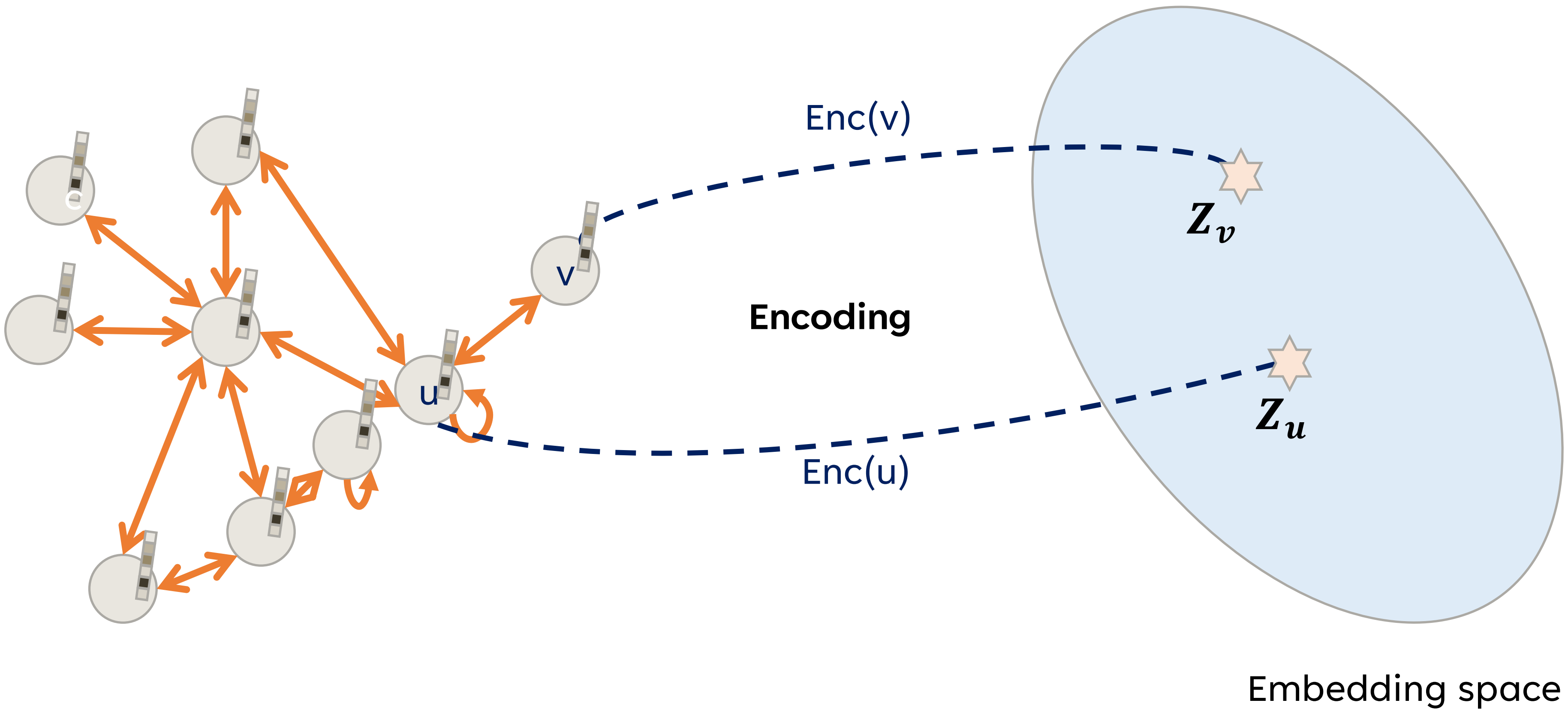
- Node, Edge, or Graph clustering
  - Use embeddings to find "similar" nodes, edges, or graphs
- Link Prediction
- Graph Generation

Supervised

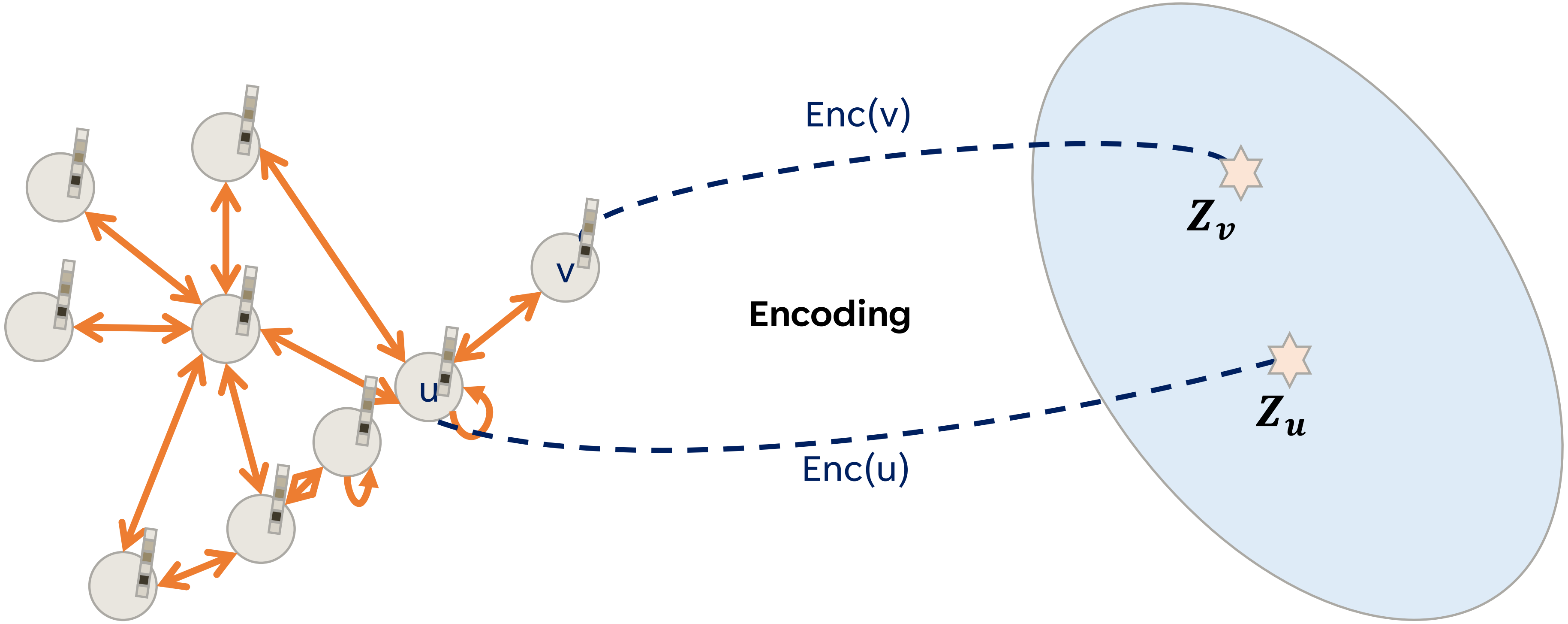
- Node, Edge, or Graph classification / regression
  - Use embeddings to predict based on known data

"A Fair Comparison of Graph Neural Networks for Graph Classification", ICLR 2020
"Revisiting Graph Neural Networks for Link Prediction" (2020)

LEARNING IN GRAPH REPRESENTATION LEARNING



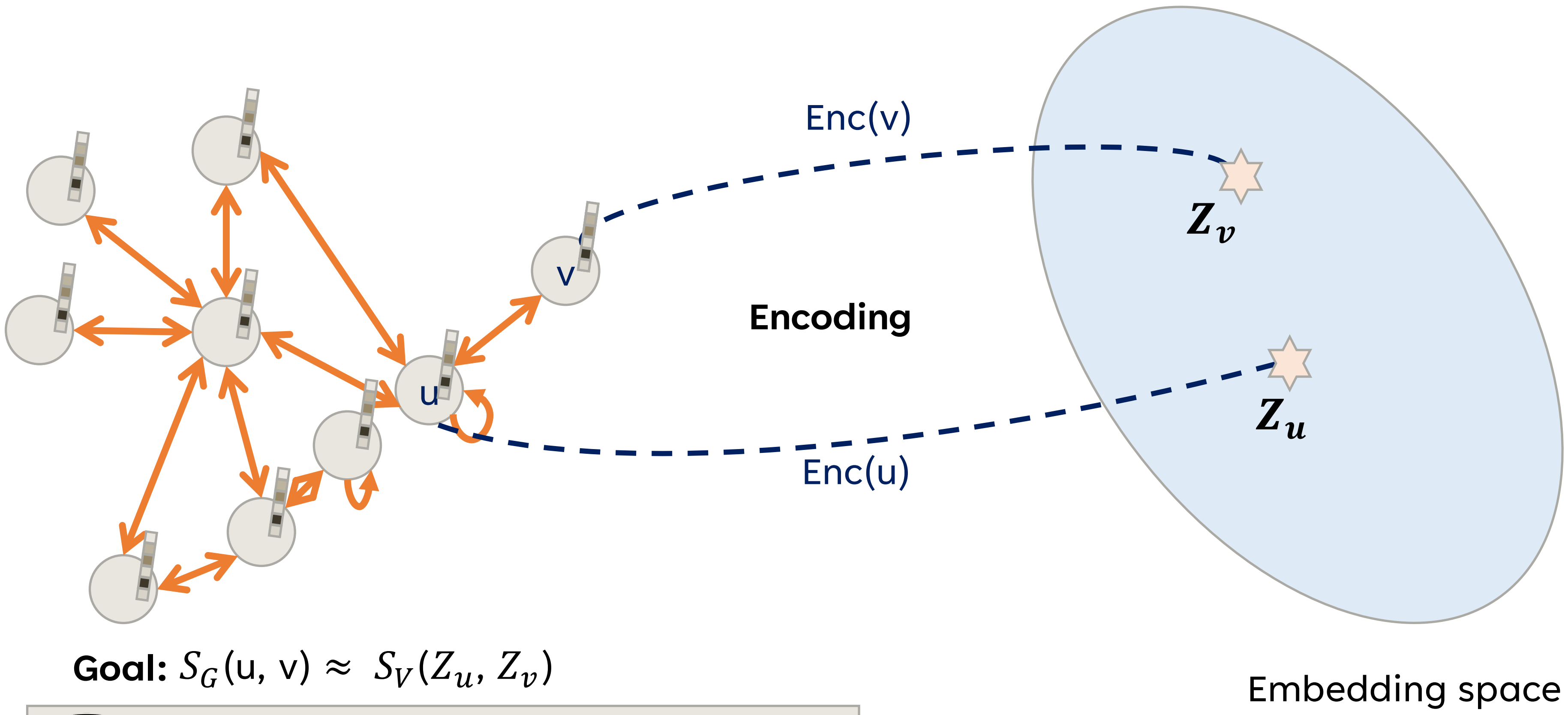
LEARNING IN GRAPH REPRESENTATION LEARNING



Goal: Similarity(u, v) ≈ Similarity(Z_u, Z_v)

S_G(u, v) ≈ $S_V(Z_u, Z_v)$

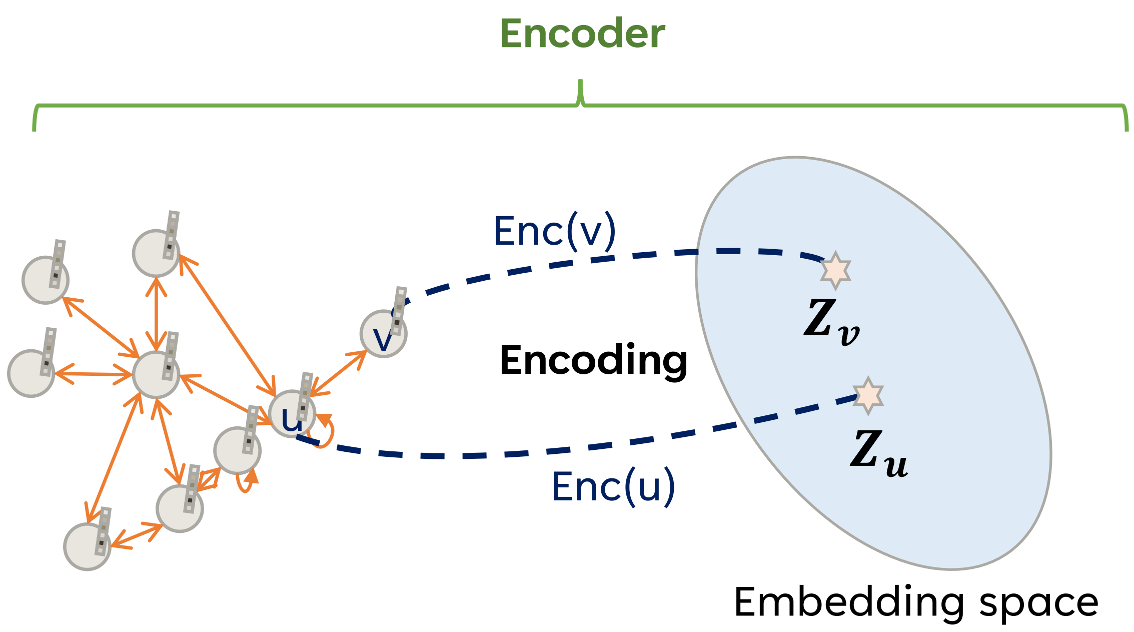
LEARNING IN GRAPH REPRESENTATION LEARNING



Goal: $S_G(\text{u, v}) \approx S_V(Z_u, Z_v)$

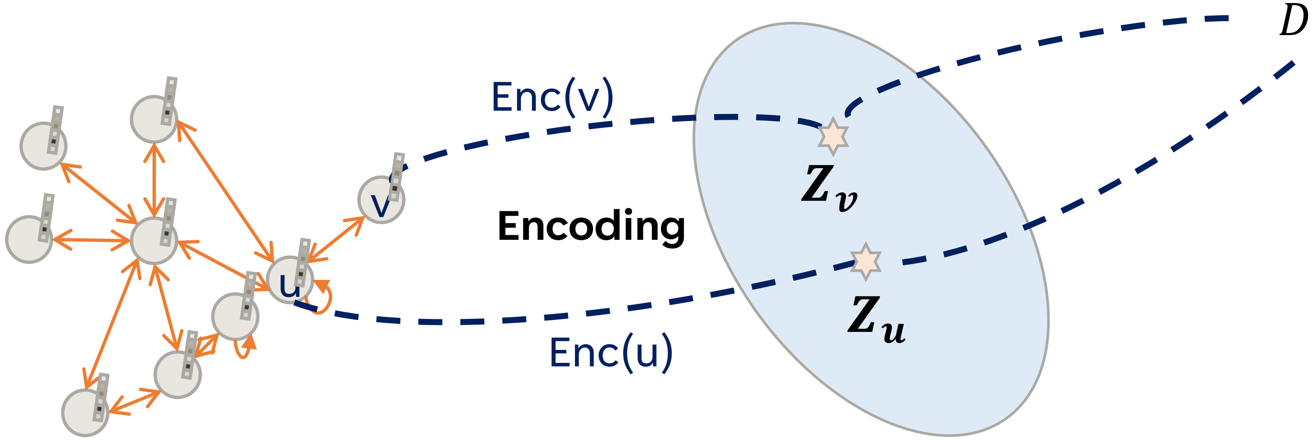
- ❑ How to perform **Encoding**?
- ❑ What is the **meaning** of **similarity** ?

HOW TO ENCODE AND DECODE?



HOW TO ENCODE AND DECODE?

Encoder | **Decoder**



Embedding space

$Dec\,(Z_u, Z_v) = How\ simmilar Z_u\ and\ Z_v\ are?$

A positive number to show the similarity

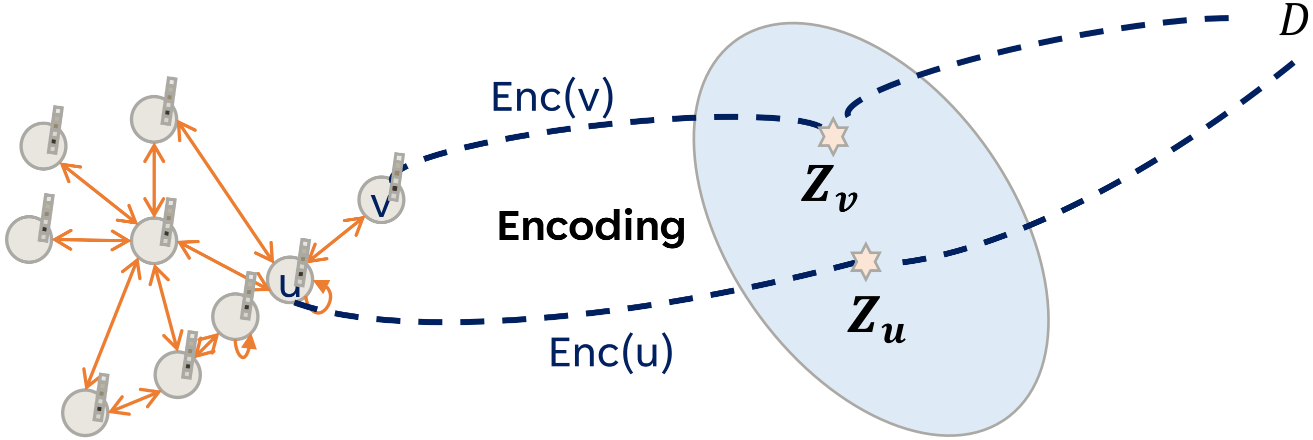
S_G(u, v) ≈ $S_V(Z_u, Z_v)$

$$\ell(z_v, z_u) = \sum_{(v,u)\in V} ||S_E(z_v, z_u) - S_G\,(v, u)||_2^2$$

HOW TO ENCODE AND DECODE?

Encoder — **Decoder**

$Dec\ (Z_u, Z_v) = How\ simmilar Z_u\ and\ Z_v\ are?$



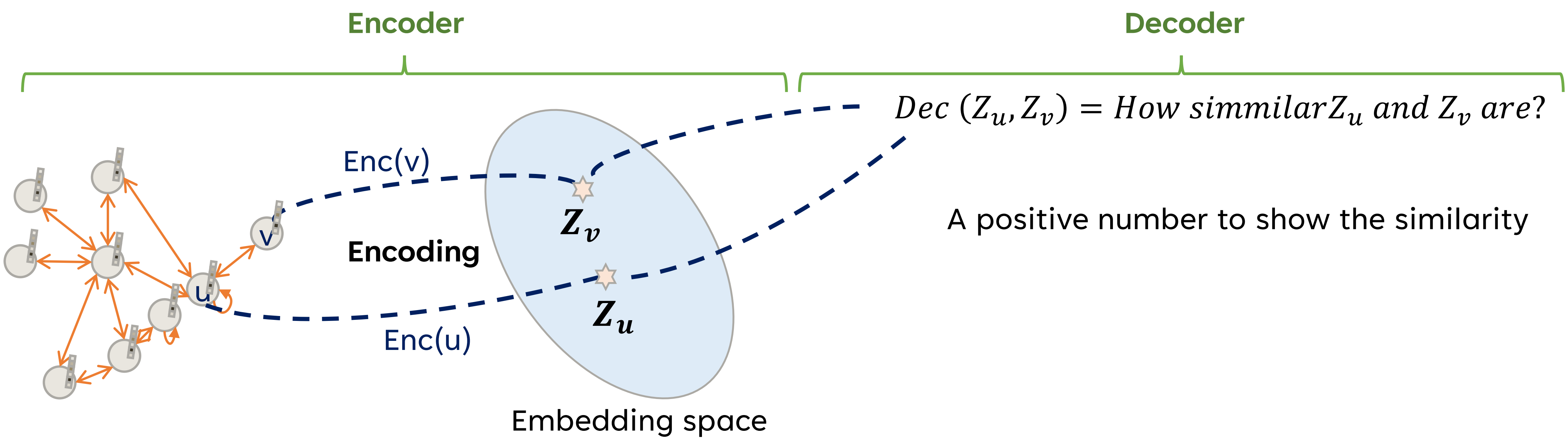
A positive number to show the similarity

Matrix factorization ------→ Inner product $Z_u^T Z_v$

Look-up table ------→ Inner product $Z_u^T Z_v$

Random Walk ------→ Decode statistic of random walk

DRAWBACKS



No parameter sharing: Computationally expensive
No semantic information: Integration of Feature nodes are difficult
Not Inductive: Cannot predict embedding for unseen data (Inherently Transudative)

DEEP VS SHALLOW

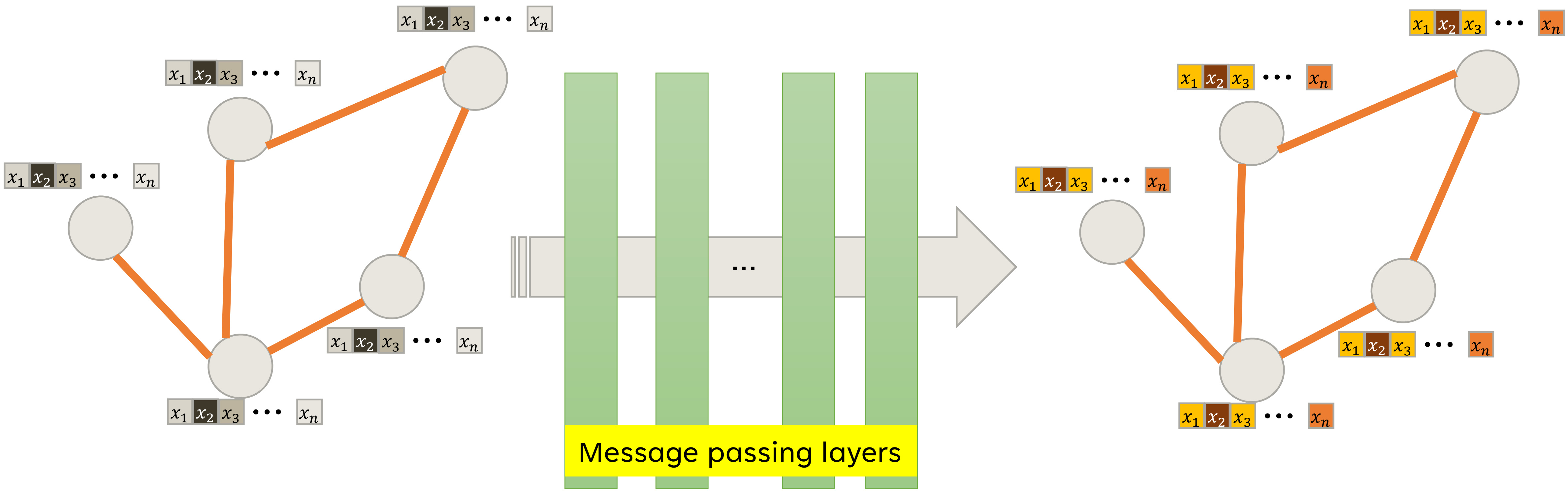
Older methods ("shallow", non-neural network models)
Deepwalk, node2vec

Generally fallen out of favor with researchers because:
- **No parameter sharing** (bad scaling, overfitting)
- **Transductive** (only work with nodes present during training)

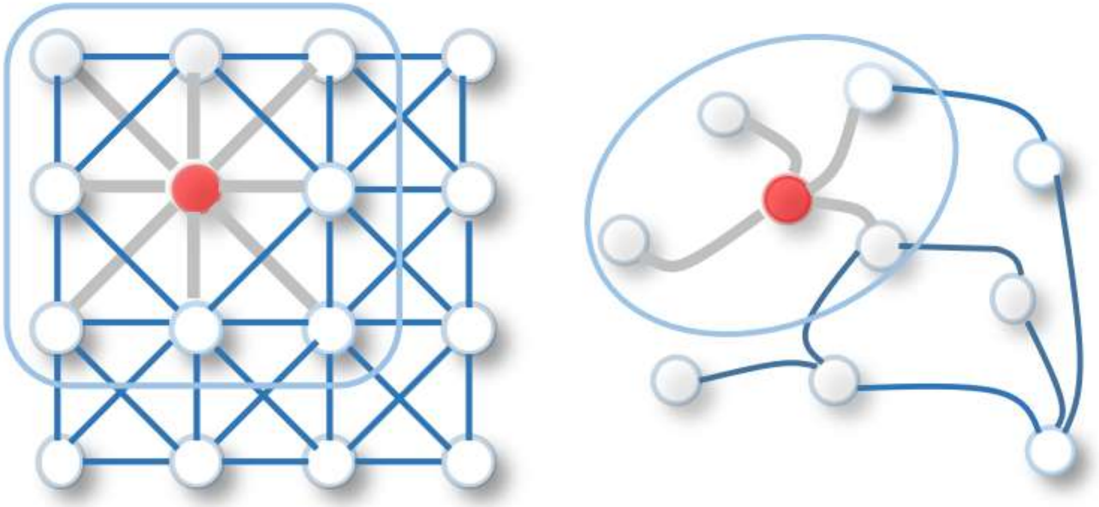
GNNs solve these problems, they can
- ✓ Share parameters
- ✓ Can generalize to inductive tasks

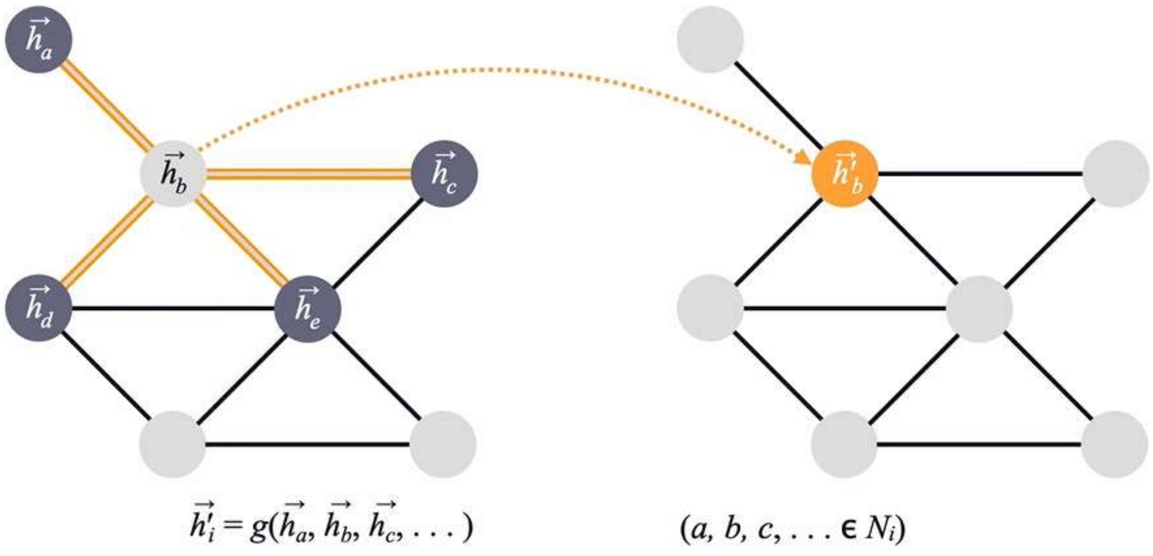
inductive and transductive!

REPRESENTATION LEARNING



GRAPH CONVOLUTIONAL NETWORK

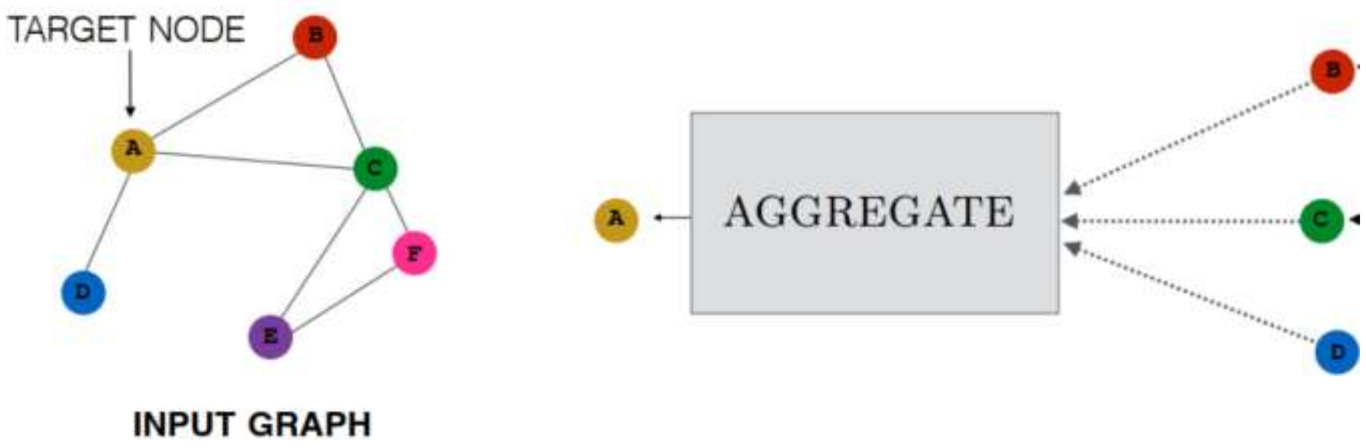




[A Comprehensive Survey on Graph Neural Networks]

UNDERSTANDING GRAPH NEURAL NETWORKS

GNNs were originally based on 2-step message passing



1. **Aggregate :**
   Pass information (the “message”) from a target node’s neighbors to the target node
2. **Update:**
   Update each node’s features based on “message” to form an embedded representation

MESSAGE PASSING

$$h_u = UPDATE\left(h_u\ , AGREGATE(\{h_v, \forall\ v\ \in N(u)\})\right)$$

h = node features / embeddings

Aggregate function operates over sets, must be permutation invariant or permutation equivariant

MESSAGE PASSING

$$h_u = UPDATE\left(h_u\ , AGREGATE(\{h_v, \forall\ v\ \in N(u)\})\right)$$

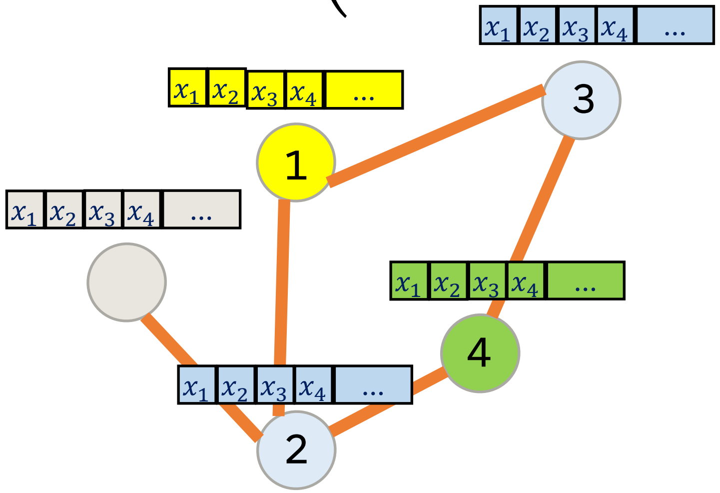
h = node features / embeddings

Aggregate function operates over sets, must be permutation invariant or permutation equivariant

$$h_u = \sigma\left(W_{self} h_u\ + W_{neigh} \sum_{v\ \in N(u)} h_v\right)$$

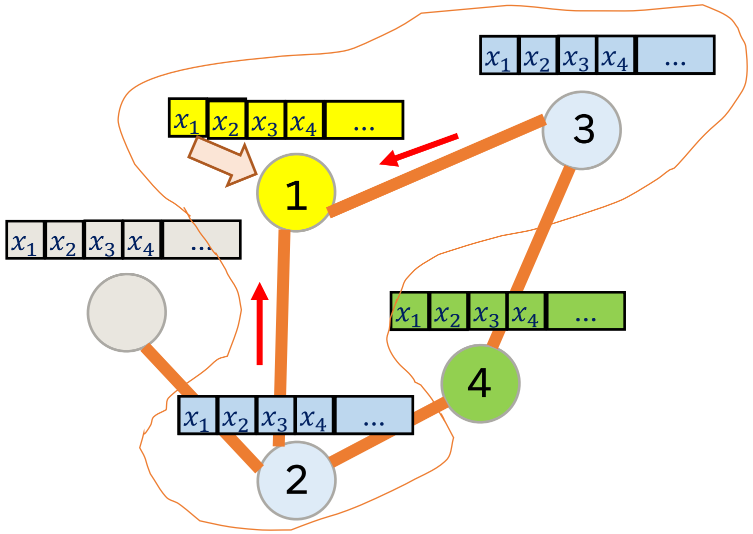
MESSAGE PASSING

$$h_u = \sigma\left(W_{self} h_u + W_{neigh} \sum_{v \in N(u)} h_v\right)$$



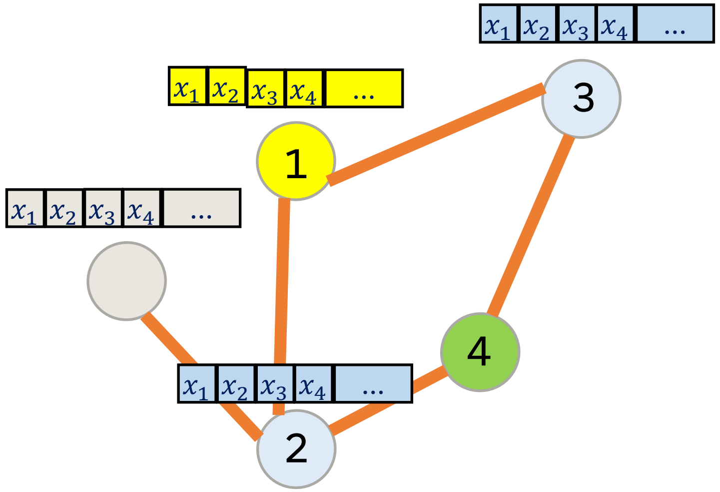
MESSAGE PASSING

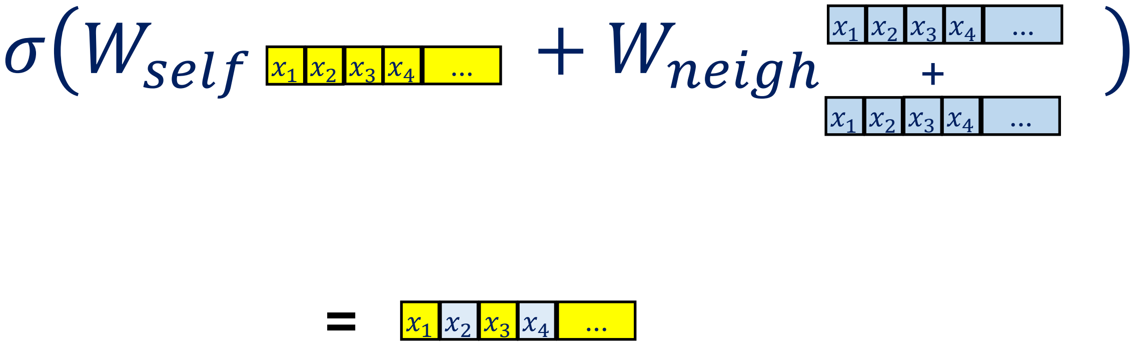
$$h_u = \sigma\left(W_{self} h_u + W_{neigh} \sum_{v \in N(u)} h_v\right)$$



MESSAGE PASSING

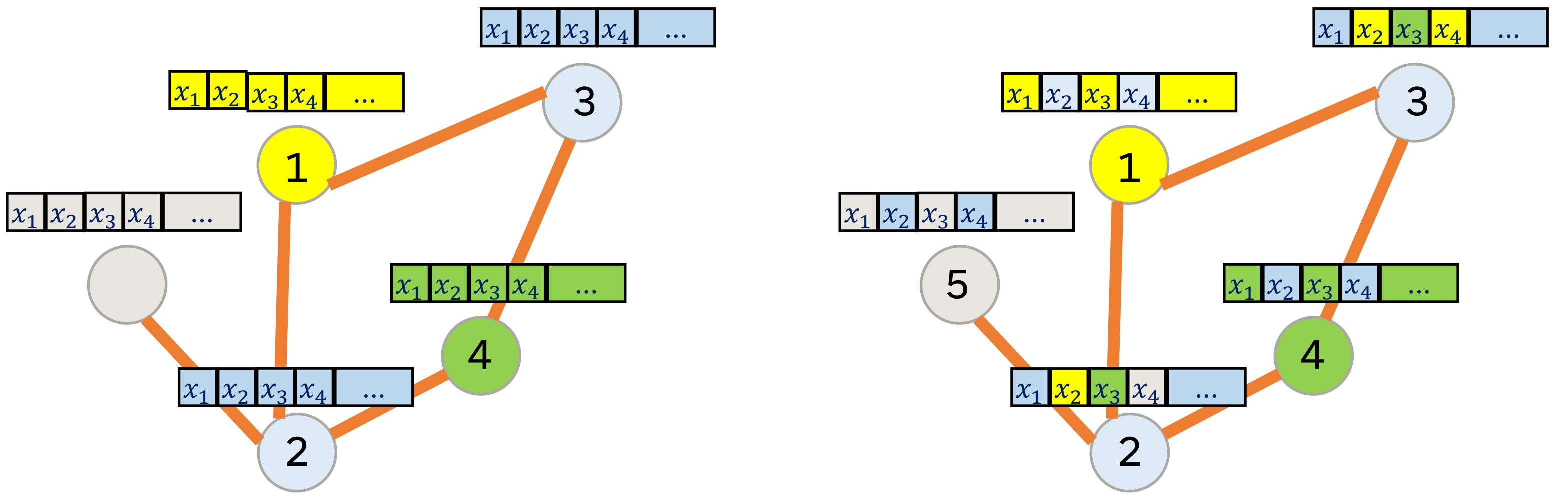
$$h_u = \sigma\left(W_{self} h_u + W_{neigh} \sum_{v \in N(u)} h_v\right)$$





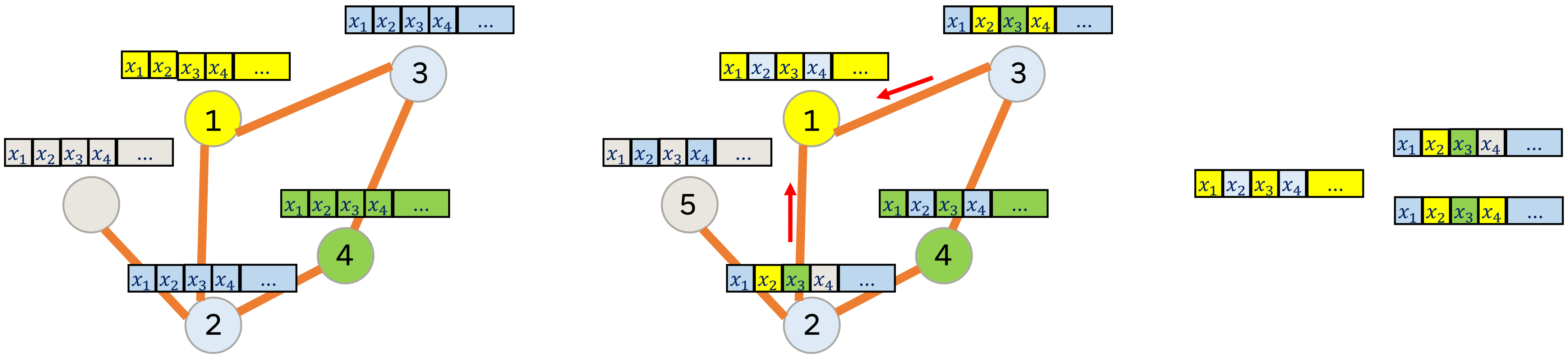
MESSAGE PASSING

$$h_u = \sigma\left(W_{self} h_u + W_{neigh} \sum_{v \in N(u)} h_v\right)$$



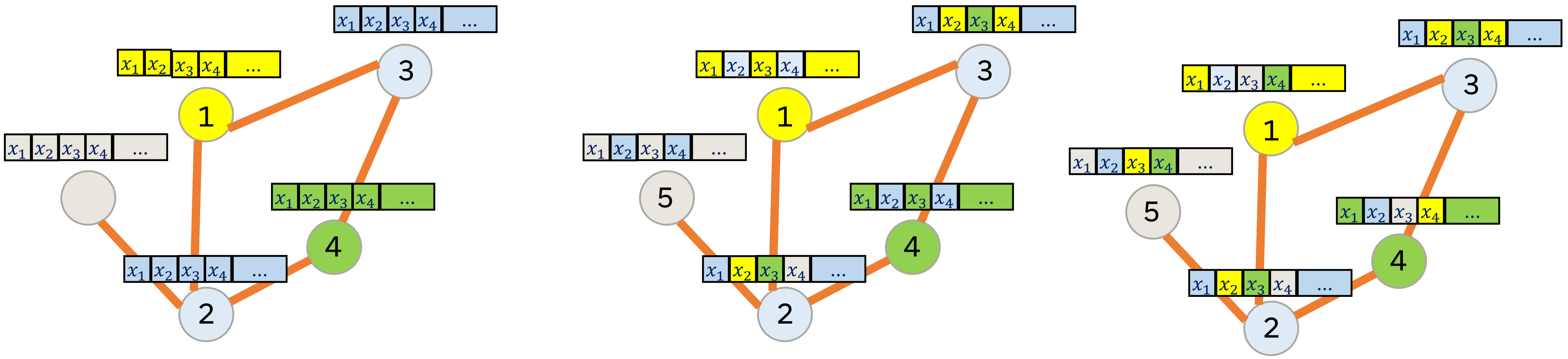
MESSAGE PASSING

$$h_u = \sigma\left(W_{self} h_u + W_{neigh} \sum_{v \in N(u)} h_v\right)$$



MESSAGE PASSING

$$h_u = \sigma\left(W_{self}h_u + W_{neigh}\sum_{v\in N(u)} h_v\right)$$

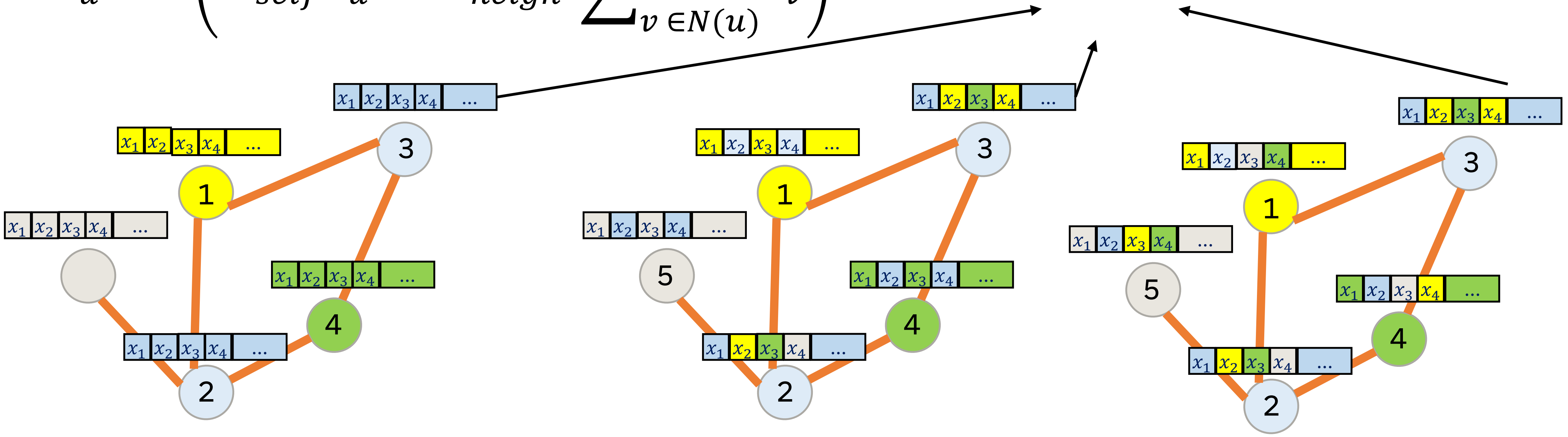


MESSAGE PASSING

$$h_u^{(k+1)} = \sigma\left(W_{self}^{(k+1)} h_u^{k} + W_{neigh}^{(k+1)} \sum_{v \in N(u)} h_v^{k}\right)$$

The dimensions can be different

$$\text{Len}(h_u^k) \neq len(h_u^{k+1})$$



✓ The **local feature aggregation** can be compared to **learnable CNN kernels:** https://distill.pub/2021/gnn-intro/

MESSAGE PASSING

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

$$h_u^{(k+1)} = \sigma\left(W_{\text{self}}^{(k+1)} h_u^k + W_{\text{neigh}}^{(k+1)} \sum_{v \in \mathcal{N}(u)} h_v^{(k)}\right)$$

- h = node features / embeddings
- k = number of hops

Each node's updated value becomes a weighting of its previous value + a weighting of its neighbor's values

The choice to sum over neighboring nodes isn't the only valid choice, other choices include mean, max, concatenation, etc.

MESSAGE PASSING

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

$$h_u^{(k+1)} = \sigma\left(W_{\text{self}}^{(k+1)} h_u^k + W_{\text{neigh}}^{(k+1)} \sum_{v \in \mathcal{N}(u)} h_v^{(k)}\right)$$

- Collapse **W**self and **W**neigh into W by adding self-loops to the adjacency matrix A

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

This method reduces message passing to relatively simple matrix multiplication

THE MEAN-POOLING UPDATE RULE

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

- **Problem**: Multiplication by A+I may increase the scale of the output features.
- ✓ **Solution**: We need to normalize appropriately:

$$H^{(k+1)} = \sigma\left(D^{-1}(A+I)H^{(k)}W^{(k+1)}\right)$$

We arrive at the mean-pooling update rule:

$$h^{(k+1)} = \sigma \sum_{j \in N_i} \frac{1}{|N_i|} W h_j^k$$

which is simple but versatile (common for inductive problems!).

GCN GRAPH CONVOLUTIONAL NETWORK

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

"Original" GNN

(Merkwirth, 2005 + Scarselli et al., 2009)

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

GCN

(Kipf + Welling, 2016)

$$\tilde{\mathbf{A}} = (\mathbf{D}+\mathbf{I})^{-\frac{1}{2}}(\mathbf{I}+\mathbf{A})(\mathbf{D}+\mathbf{I})^{-\frac{1}{2}}$$

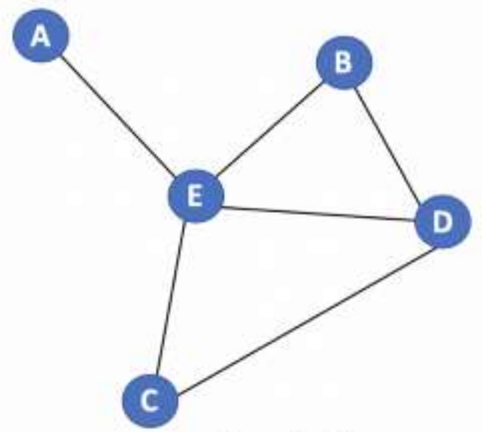
Normalizes by # of nodes in neighborhood

Node-wise, this can be written as follows:

$$h^{(k+1)} = \sigma\left(\sum_{j \in N_i} \frac{1}{\sqrt{|N_i||N_j|}} W h_j^k\right)$$

Most commonly cited GNN paper

INTUITION AND THE MATH'S BEHIND



Graph G

	A	B	C	D	E
A	0	0	0	0	1
B	0	0	0	1	1
C	0	0	0	1	1
D	0	1	1	0	1
E	1	1	1	1	0

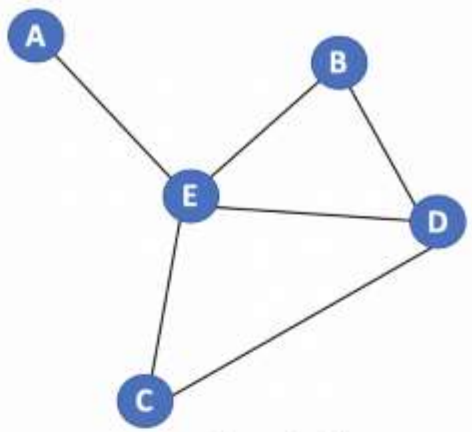
Adjacency matrix A

A	-1.1	3.2	4.2
B	0.4	5.1	-1.2
C	1.2	1.3	2.1
D	1.4	-1.2	2.5
E	1.4	2.5	4.5

Feature vector X

https://www.topbots.com/graph-convolutional-networks/

INTUITION AND THE MATH'S BEHIND



Graph G

	A	B	C	D	E
A	0	0	0	0	1
B	0	0	0	1	1
C	0	0	0	1	1
D	0	1	1	0	1
E	1	1	1	1	0

Adjacency matrix A

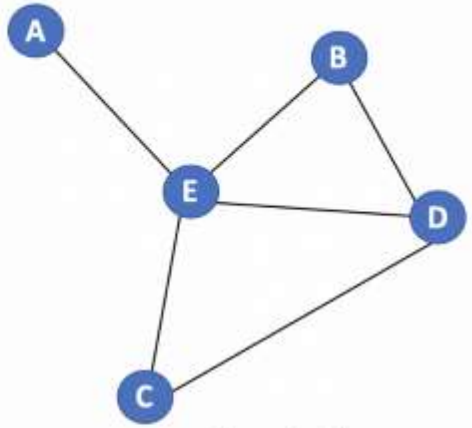
A	-1.1	3.2	4.2
B	0.4	5.1	-1.2
C	1.2	1.3	2.1
D	1.4	-1.2	2.5
E	1.4	2.5	4.5

Feature vector X

$$H^{(k+1)} = \sigma\left(AH^{(k)}W^{(k+1)}\right)$$

$$h^{(k+1)} = \sigma \sum_{j \in N_i} W h_j^k$$

INTUITION AND THE MATH'S BEHIND



Graph *G*

	A	B	C	D	E
A	0	0	0	0	1
B	0	0	0	1	1
C	0	0	0	1	1
D	0	1	1	0	1
E	1	1	1	1	0

Adjacency matrix *A*

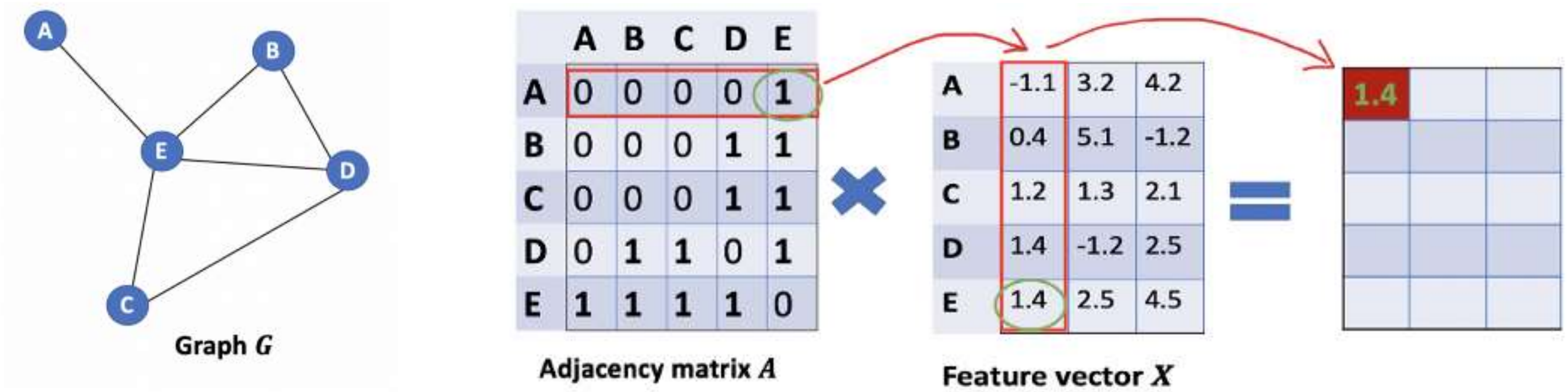
A	-1.1	3.2	4.2
B	0.4	5.1	-1.2
C	1.2	1.3	2.1
D	1.4	-1.2	2.5
E	1.4	2.5	4.5

Feature vector *X*

$$H^{(k+1)} = \sigma\left(AH^{(k)}W^{(k+1)}\right)$$

$$h^{(k+1)} = \sigma \sum_{j \in N_i} W h_j^k$$

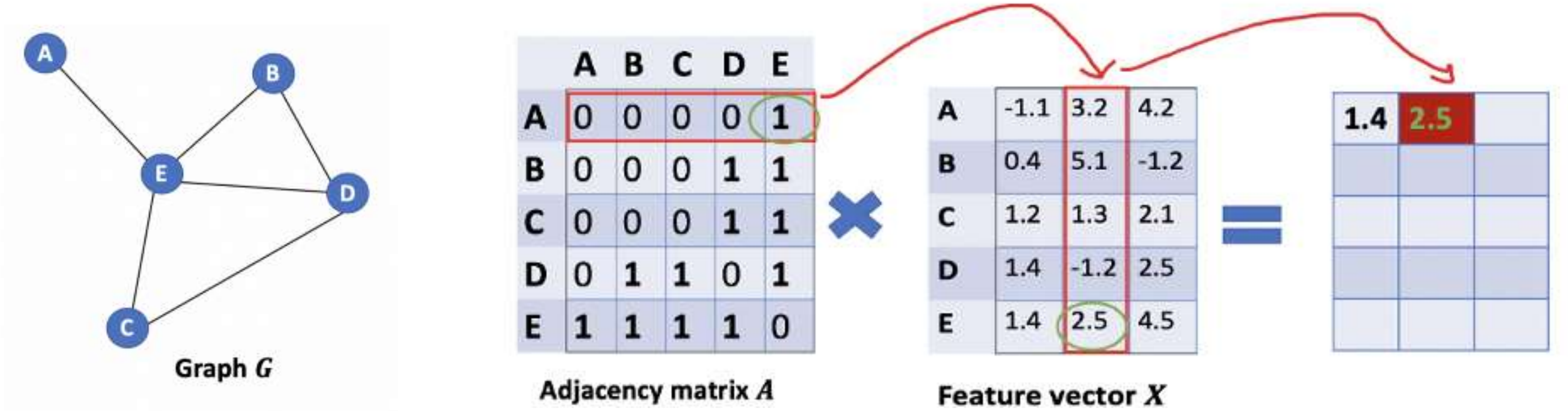
INTUITION AND THE MATH'S BEHIND



$$H^{(k+1)} = \sigma\left(AH^{(k)}W^{(k+1)}\right)$$

$$h^{(k+1)} = \sigma \sum_{j \in N_i} W h_j^k$$

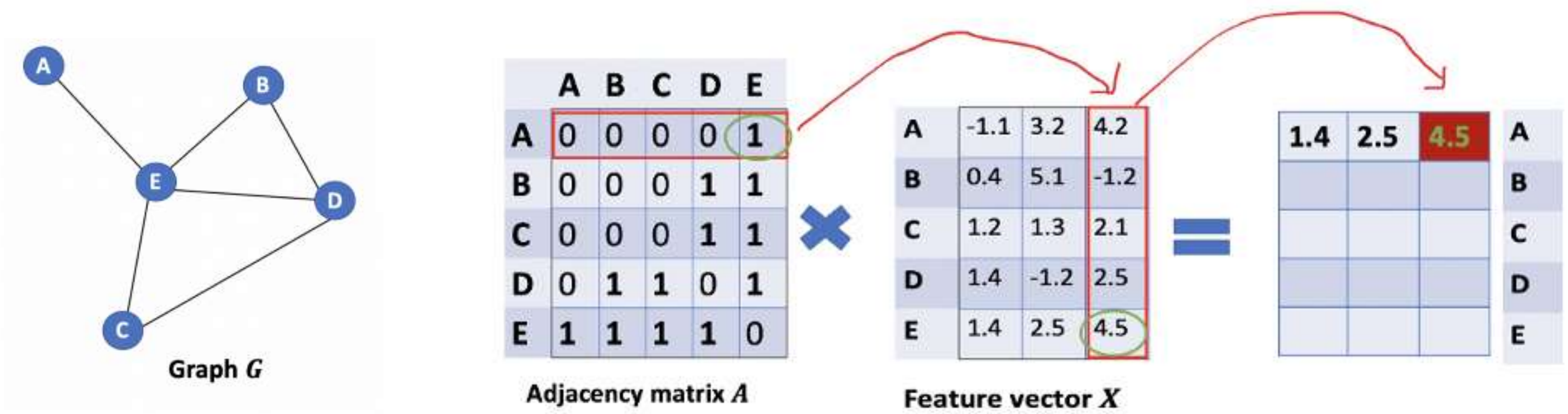
INTUITION AND THE MATH'S BEHIND



$$H^{(k+1)} = \sigma\left(AH^{(k)}W^{(k+1)}\right)$$

$$h^{(k+1)} = \sigma \sum_{j \in N_i} W h_j^k$$

INTUITION AND THE MATH'S BEHIND



$$H^{(k+1)} = \sigma\left(AH^{(k)}W^{(k+1)}\right)$$

$$h^{(k+1)} = \sigma \sum_{j \in N_i} W h_j^k$$

PROBLEMS!

1. We miss the **feature of the node itself**. For example, the first row of the result matrix should contain features of node A too.

PROBLEMS!

1. We miss the **feature of the node itself**. For example, the first row of the result matrix should contain features of node A too.

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

	A	B	C	D	E
A	0	0	0	0	1
B	0	0	0	1	1
C	0	0	0	1	1
D	0	1	1	0	1
E	1	1	1	1	0

Adjacency matrix A

+

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Identity matrix I

=

	A	B	C	D	E
A	1	0	0	0	1
B	0	1	0	1	1
C	0	0	1	1	1
D	0	1	1	1	1
E	1	1	1	1	1

New Adjacency matrix $\tilde{A}$

PROBLEMS!

1. We miss the **feature of the node itself**. For example, the first row of the result matrix should contain features of node A too.

2. Instead of sum() function, we need to take the average, or even better, the weighted average of neighbors' feature vectors. **Why don't we use the sum() function?** The reason is that when using the sum() function, high-degree nodes are likely to have huge v vectors, while low-degree nodes tend to get small aggregate vectors, which may later cause **exploding or vanishing gradients** (e.g., when using sigmoid). Besides, Neural networks seem to be **sensitive to the scale of input data**. Thus, we need to normalize these vectors to get rid of the potential issues.

PROBLEMS!

2. Instead of sum() function, we need to take the average, or even better, the weighted average of neighbors' feature vectors. **Why don't we use the sum() function?** The reason is that when using the sum() function, high-degree nodes are likely to have huge v vectors, while low-degree nodes tend to get small aggregate vectors, which may later cause **exploding or vanishing gradients** (e.g., when using sigmoid). Besides, Neural networks seem to be **sensitive to the scale of input data**. Thus, we need to normalize these vectors to get rid of the potential issues.

	A	B	C	D	E
A	1	0	0	0	1
B	0	1	0	1	1
C	0	0	1	1	1
D	0	1	1	1	1
E	1	1	1	1	1

New adjacency matrix $\widetilde{A}$

→

2	0	0	0	0
0	3	0	0	0
0	0	3	0	0
0	0	0	4	0
0	0	0	0	5

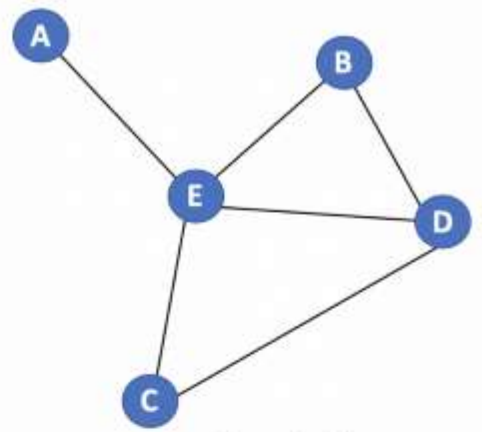
New degree matrix $\widetilde{D}$

→

1/2	0	0	0	0
0	1/3	0	0	0
0	0	1/3	0	0
0	0	0	1/4	0
0	0	0	0	1/5

$\widetilde{D}^{-1}$

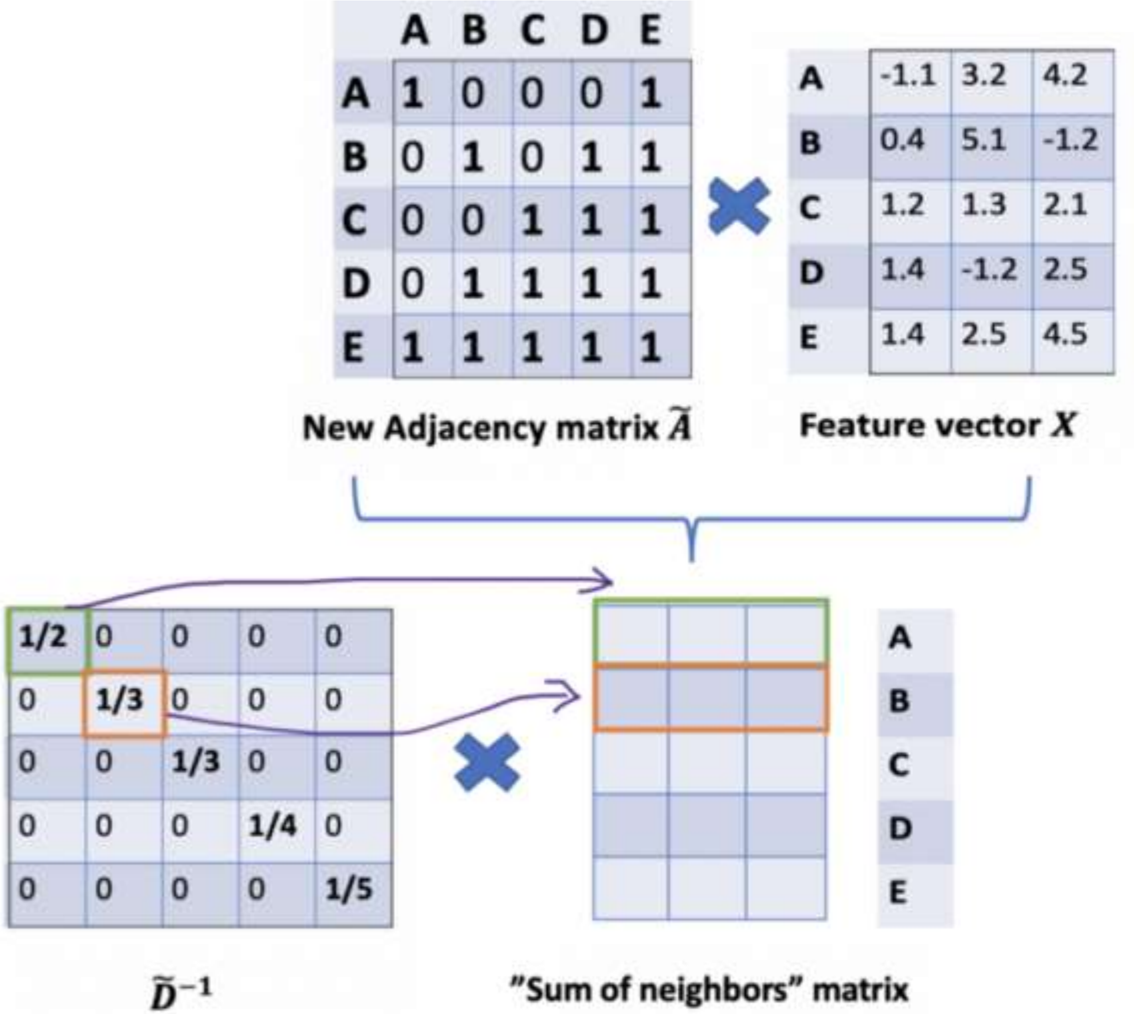
INTUITION AND THE MATH'S BEHIND



Graph G

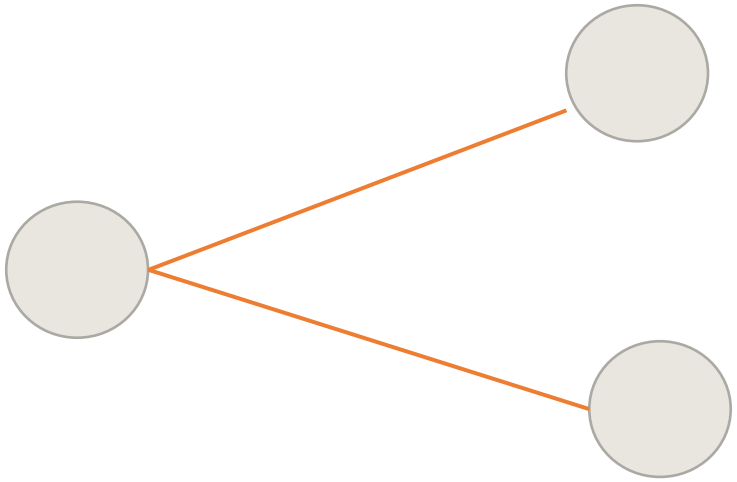
$$H^{(k+1)} = \sigma\left(D^{-1}(A+I)H^{(k)}W^{(k+1)}\right)$$

$$h^{(k+1)} = \sigma \sum_{j \in N_i} \frac{1}{|N_i|} W h_j^k$$



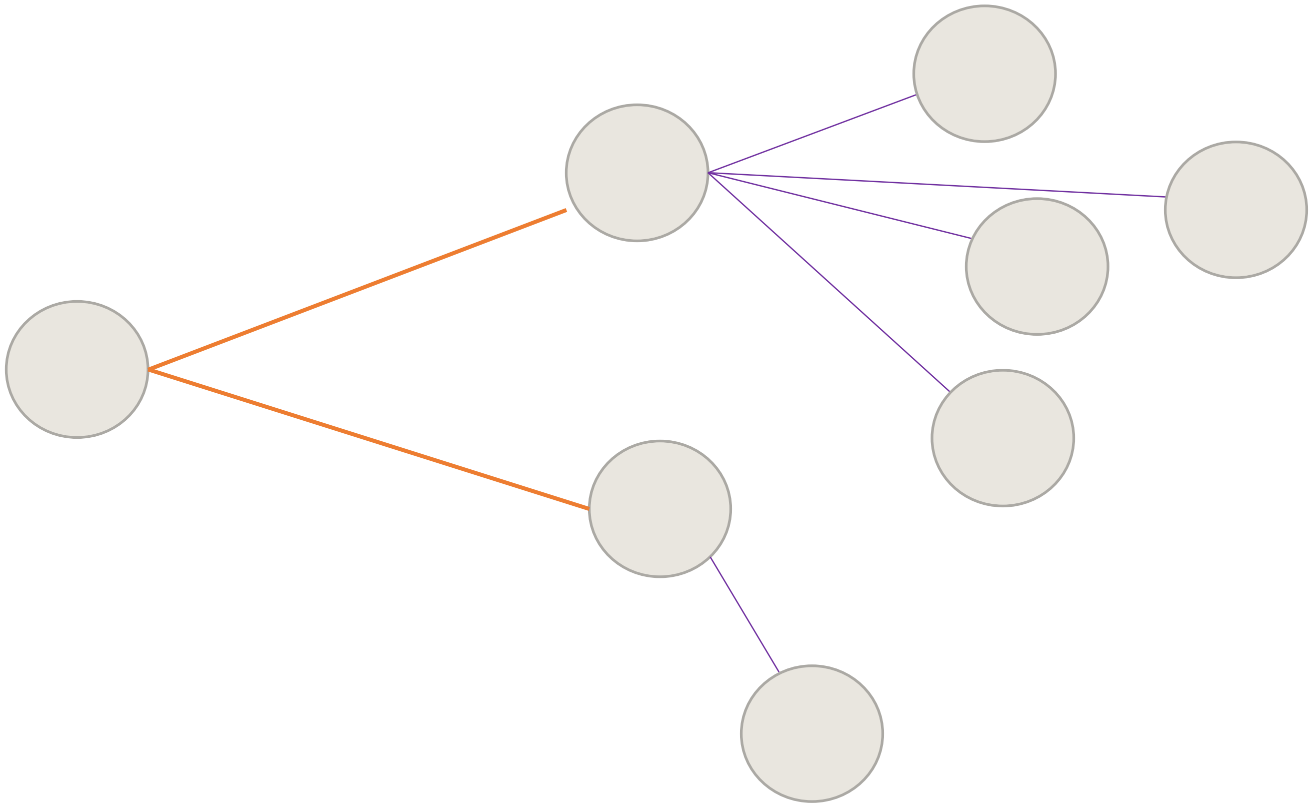
INTUITION AND THE MATH'S BEHIND

- So far, so good!

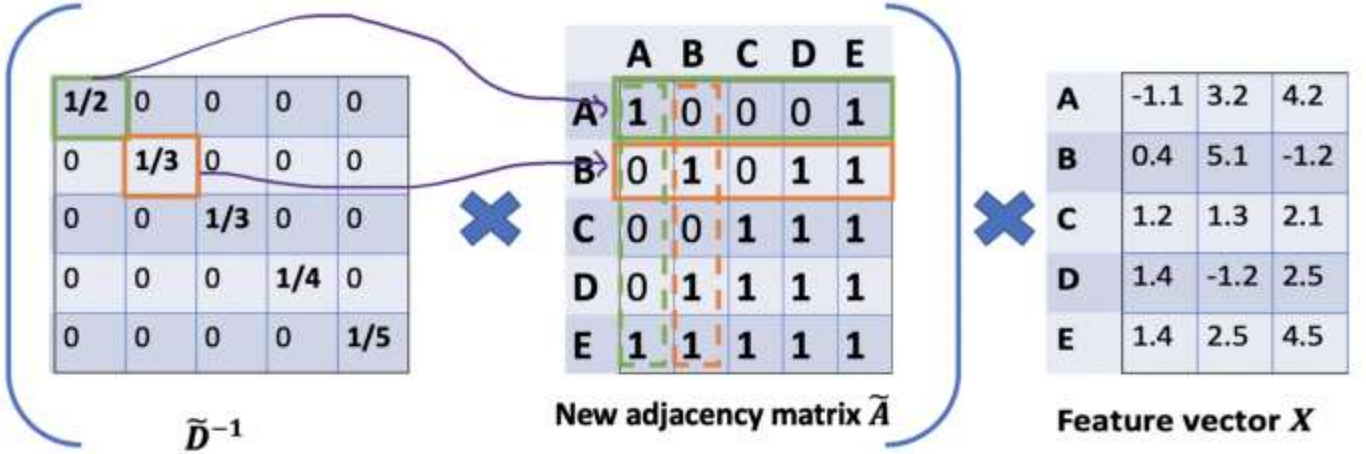


INTUITION AND THE MATH'S BEHIND

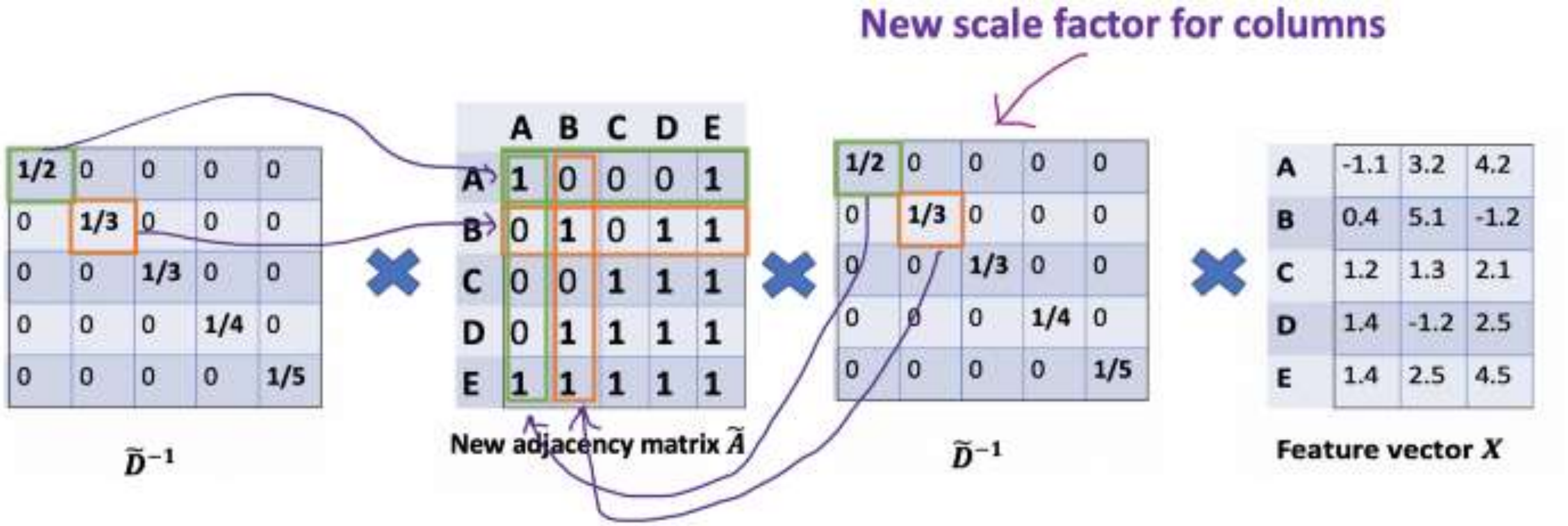
- So far, so good!
- Intuitively, it should be better if we treat high and low degree nodes differently.



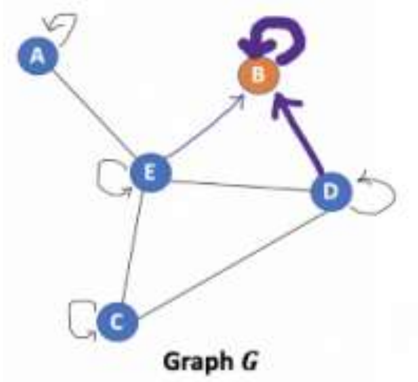
INTUITION AND THE MATH'S BEHIND



INTUITION AND THE MATH'S BEHIND



The new scaler gives us the “weighted” average. What are we doing here is to put more weights on the nodes that have low-degree and reduce the impact of high-degree nodes.



INTUITION AND THE MATH'S BEHIND

One more minor note: When using two scalers ($\tilde{D}_{ii}$ and $\tilde{D}_{jj}$), we actually normalize **twice**, one time for the row as before, and another time for the column. It would make sense if we rebalance by modifying $\tilde{D}_{ii}\tilde{D}_{jj}$ to $\sqrt{\tilde{D}_{ii}\tilde{D}_{jj}}$. In other words, instead of using $\tilde{D}^{-1}$, we use $\tilde{D}^{-1/2}$. So, we further alter the formula to $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}X$, **which is exactly used in the paper.**

2	0	0	0	0
0	**3**	0	0	0
0	0	**3**	0	0
0	0	0	**4**	0
0	0	0	0	**5**

$\tilde{D}$

1/2	0	0	0	0
0	**1/3**	0	0	0
0	0	**1/3**	0	0
0	0	0	**1/4**	0
0	0	0	0	**1/5**

$\tilde{D}^{-1}$

1/ $\sqrt{2}$	0	0	0	0
0	**1/ $\sqrt{3}$**	0	0	0
0	0	**1/ $\sqrt{3}$**	0	0
0	0	0	**1/2**	0
0	0	0	0	**1/ $\sqrt{5}$**

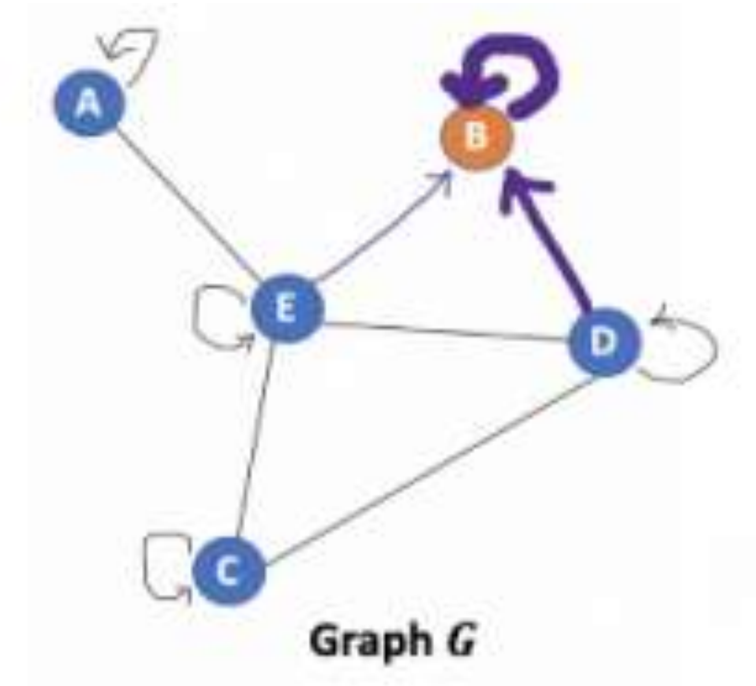
$\tilde{D}^{-1/2}$

INTUITION AND THE MATH'S BEHIND

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

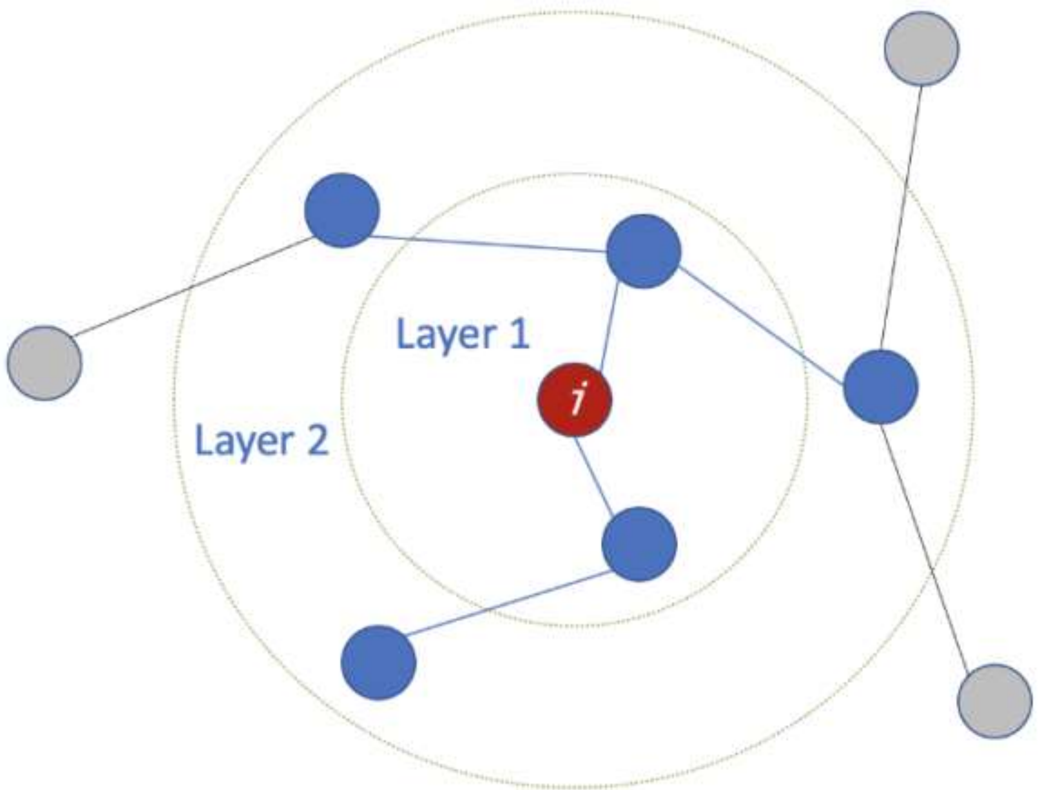
$$\tilde{\mathbf{A}} = (\mathbf{D}+\mathbf{I})^{-\frac{1}{2}}(\mathbf{I}+\mathbf{A})(\mathbf{D}+\mathbf{I})^{-\frac{1}{2}}$$

$$h^{(k+1)} = \sigma\left(\sum_{j\in N_i}\frac{1}{\sqrt{|N_i||N_j|}}Wh_j^k\right)$$

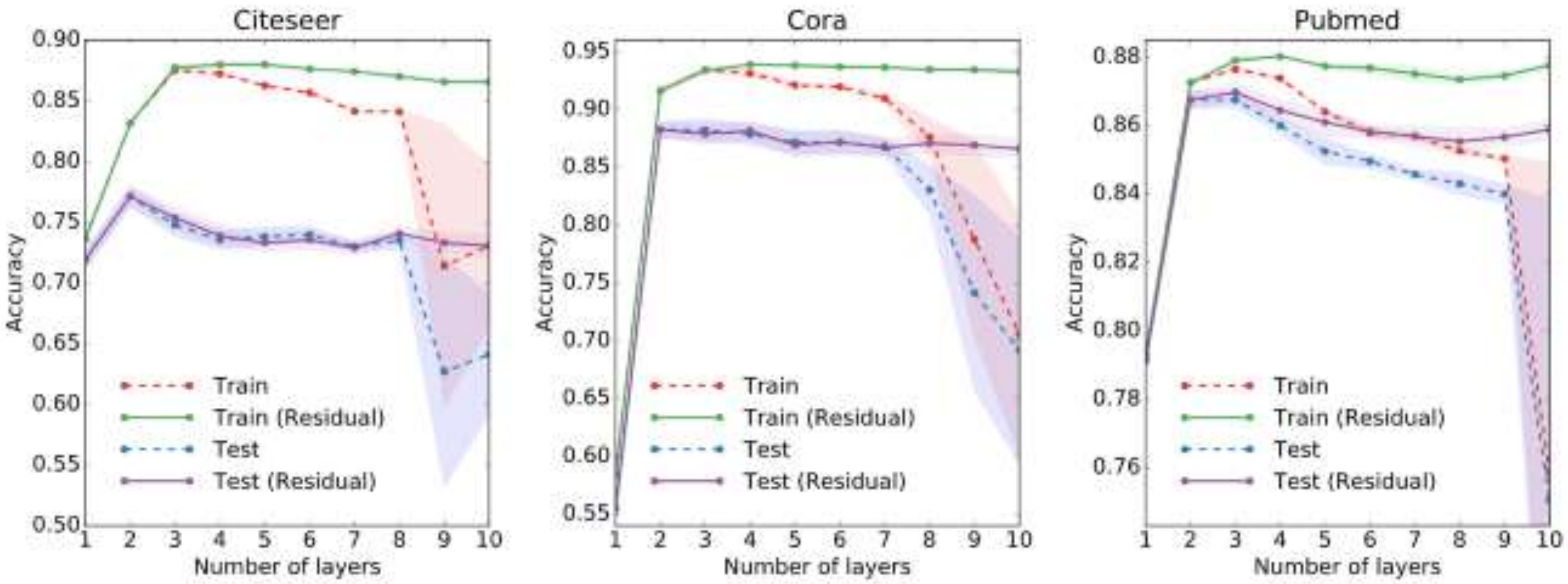


THE NUMBER OF LAYERS

- The number of layers is the farthest distance that node features can travel.
- Normally we don't want to go too far. With 6–7 hops, we almost get the entire graph which makes the aggregation less meaningful.



HOW MANY LAYERS SHOULD WE STACK THE GCN?



GNN VARIANTS

$$h_u = UPDATE\left(h_u\ , AGREGATE(\{h_v, \forall\ v\ \in N(u)\})\right)$$

Graph Convolutional Networks, Kipf and Welling [2016]

$$\mathbf{h}_v^{(k)} = \sigma\left(\mathbf{W}^{(k)} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \frac{\mathbf{h}_v}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}}\right)$$

Self-loop

Sum of normalized neighbor embeddings

Multi-Layer-Perceptron as Aggregator, Zaheer et al. [2017]

$$\mathbf{m}_{\mathcal{N}(u)} = \mathrm{MLP}_\theta\left(\sum_{v \in N(u)} \mathrm{MLP}_\phi(\mathbf{h}_v)\right)$$

Aggregated message

trainable!

Send states through a MLP

Graph Attention Networks, Veličković et al. [2017]

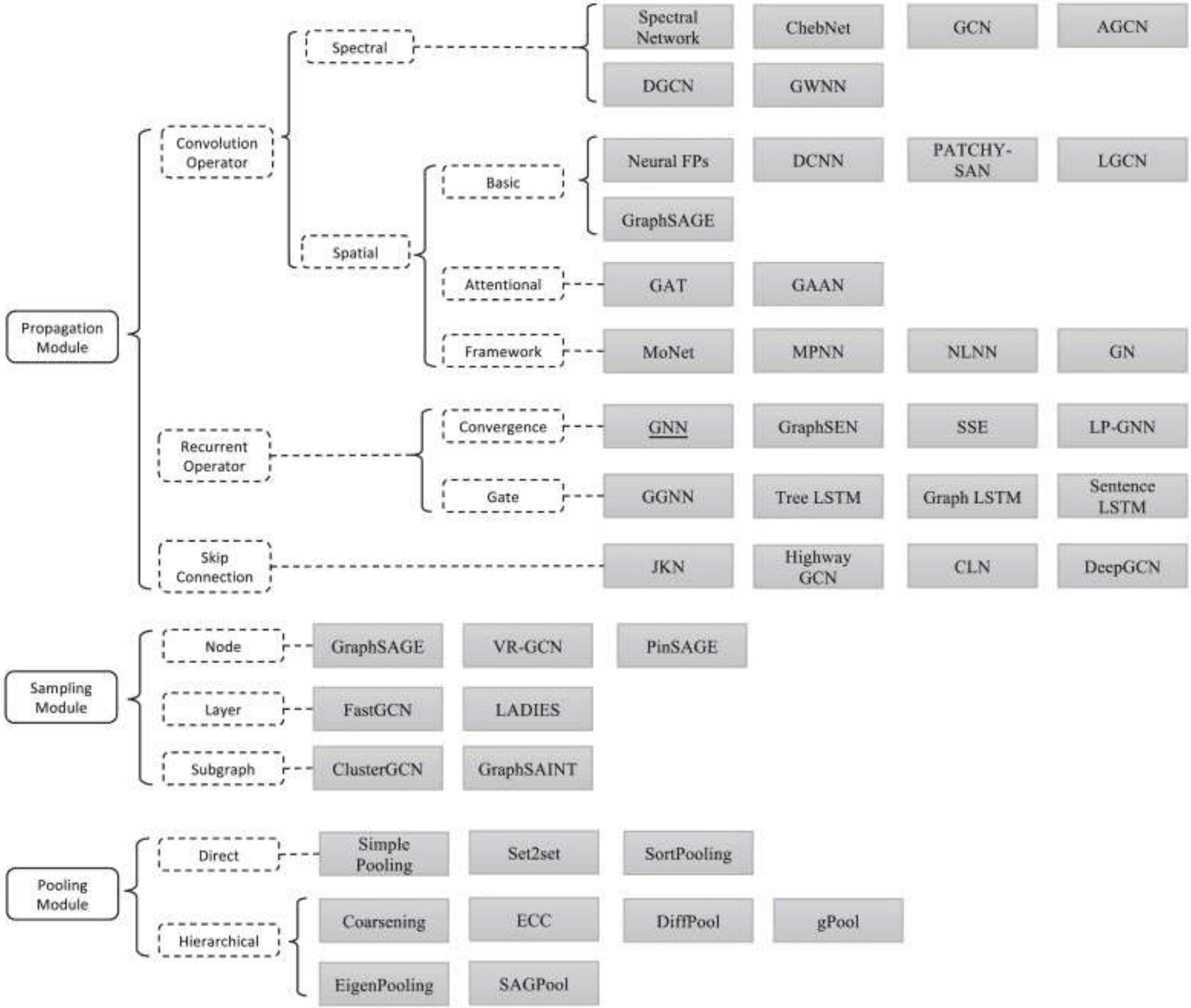
$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} \mathbf{h}_v \qquad \alpha_{u,v} = \frac{\exp\left(\mathbf{a}^\top [\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_v]\right)}{\sum_{v' \in \mathcal{N}(u)} \exp\left(\mathbf{a}^\top [\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_{v'}]\right)}$$

Attention weights

Gated Graph Neural Networks, Li et al. [2015]

$$\mathbf{h}_u^{(k)} = \mathrm{GRU}(\mathbf{h}_u^{(k-1)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)})$$

Recurrent update of the state



Source: Graph Neural Networks: A Review of Methods and Applications

Graph Representation Learning

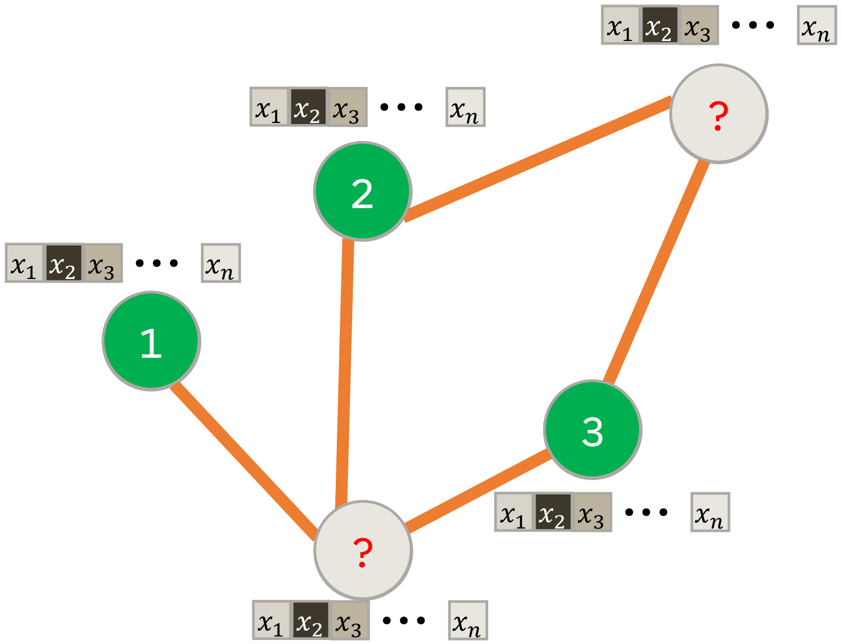
William L. Hamilton

McGill University

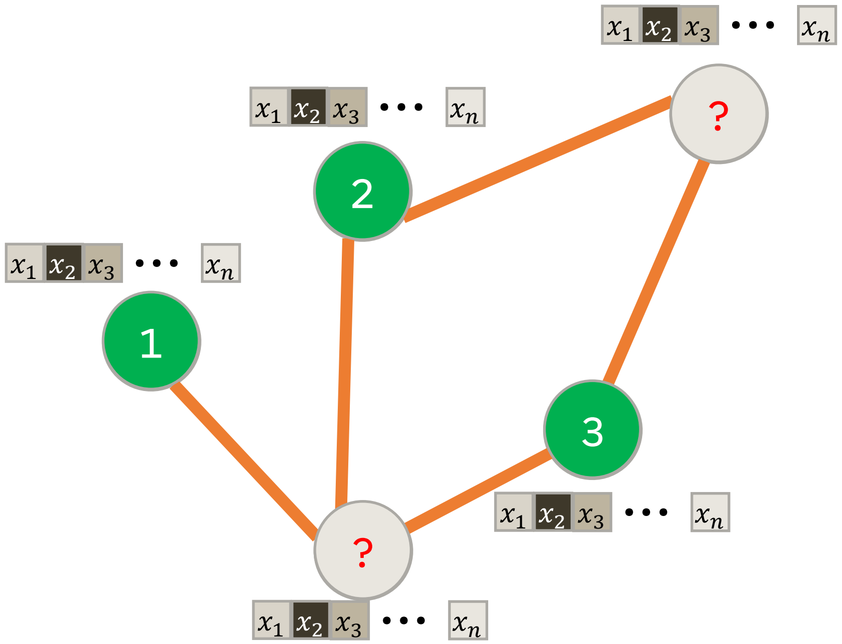
2020

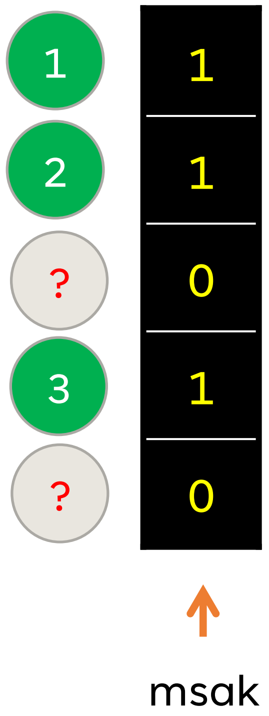
https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book.pdf

BINARY MASKS FOR NODE-LEVEL PREDICTION

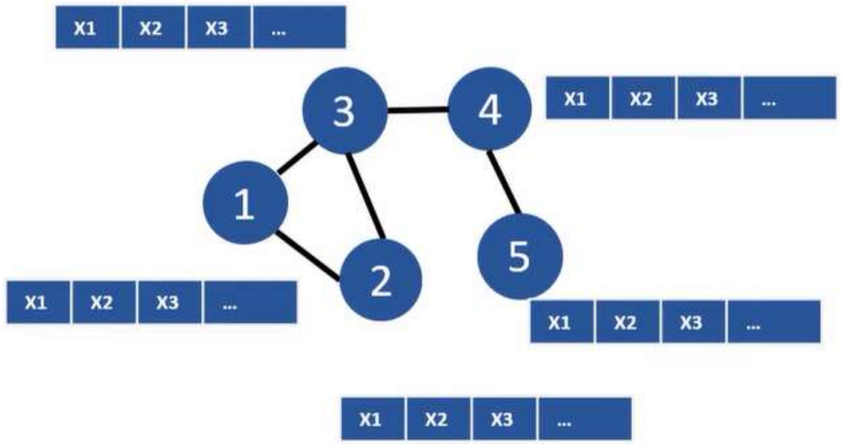


BINARY MASKS FOR NODE-LEVEL PREDICTION

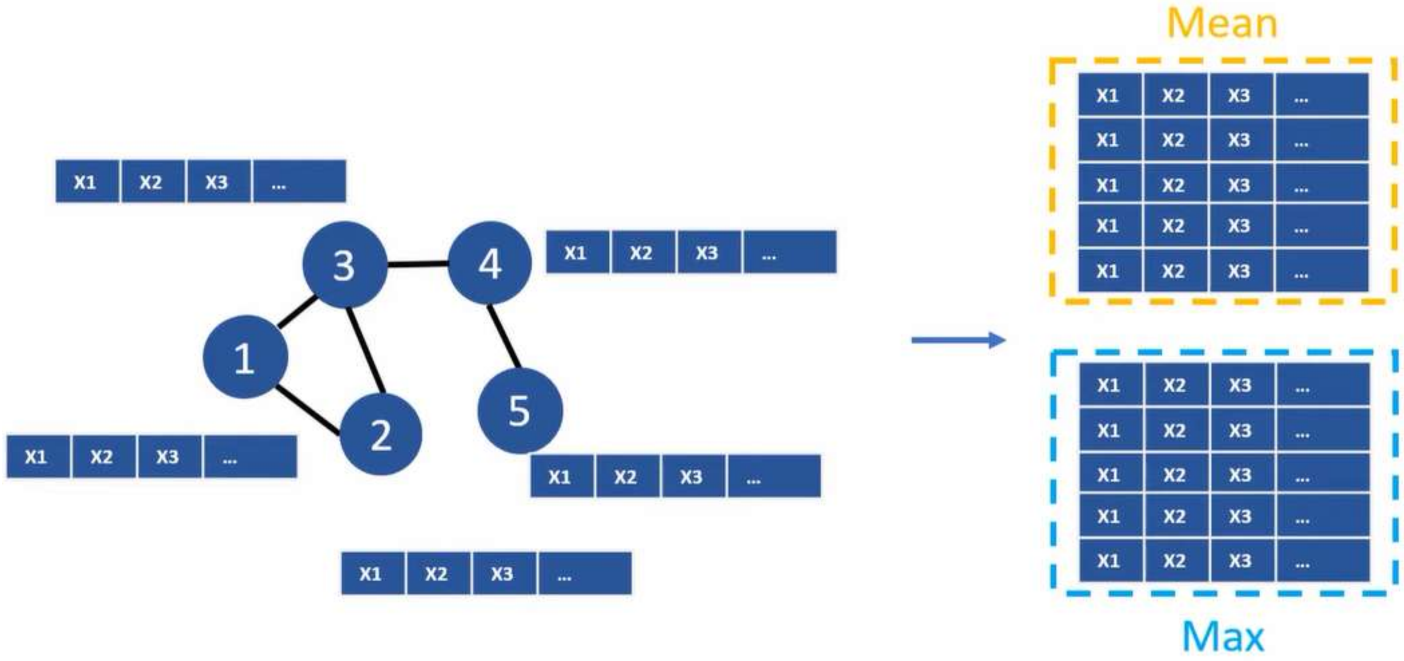




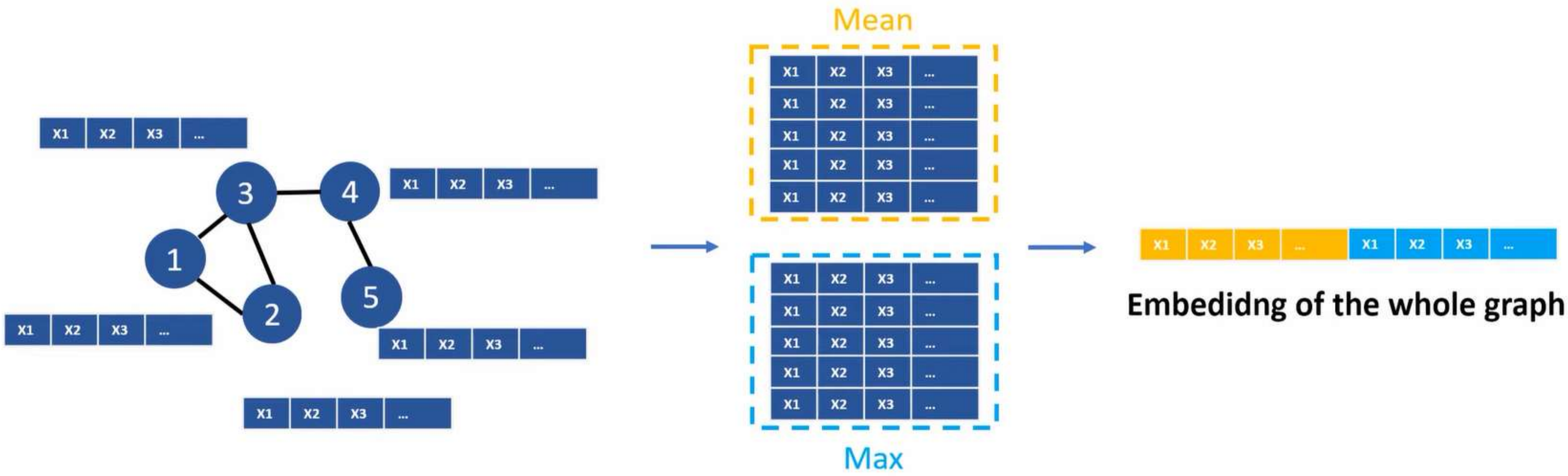
GLOBAL GRAPH POOLING



GLOBAL GRAPH POOLING



GLOBAL GRAPH POOLING



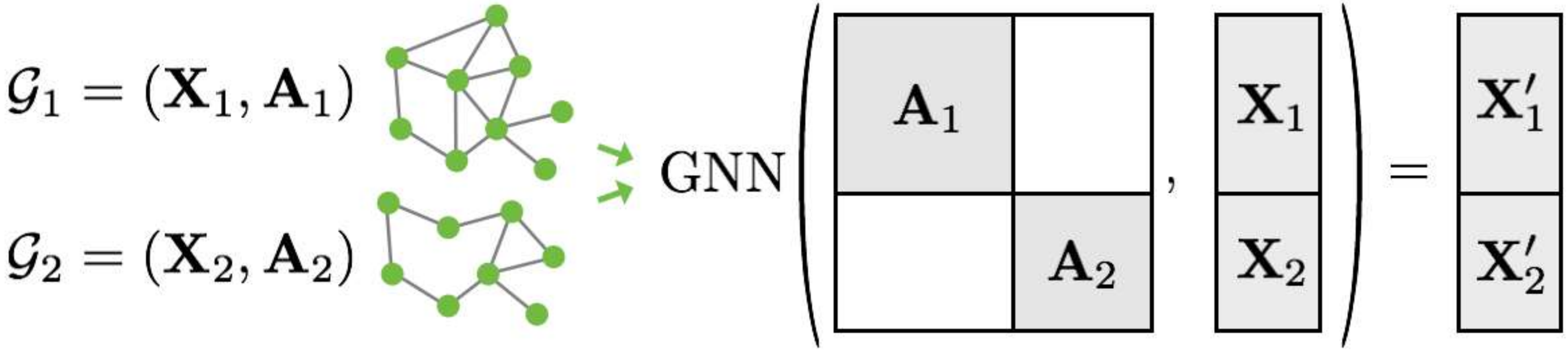
BATCHING WITH GRAPHS

In the image or language domain:

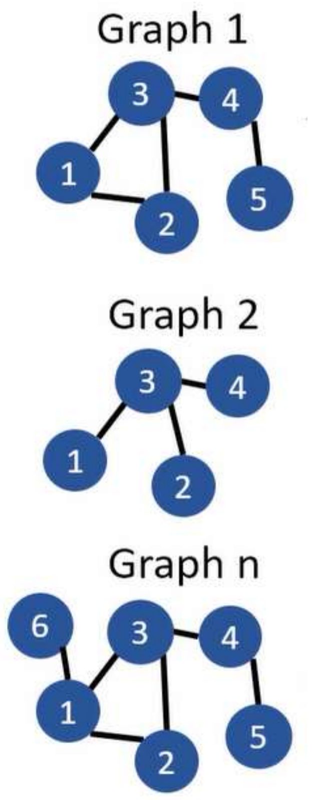
rescaling or **padding**

WHAT ABOUT Graphs?

BATCHING WITH GRAPHS

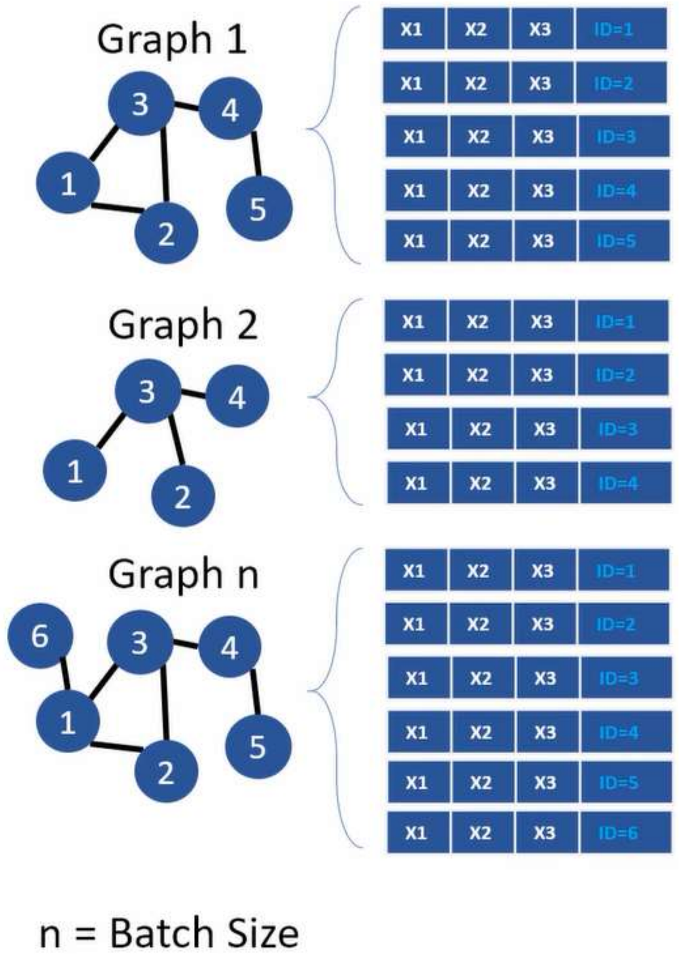


BATCHING WITH GRAPHS

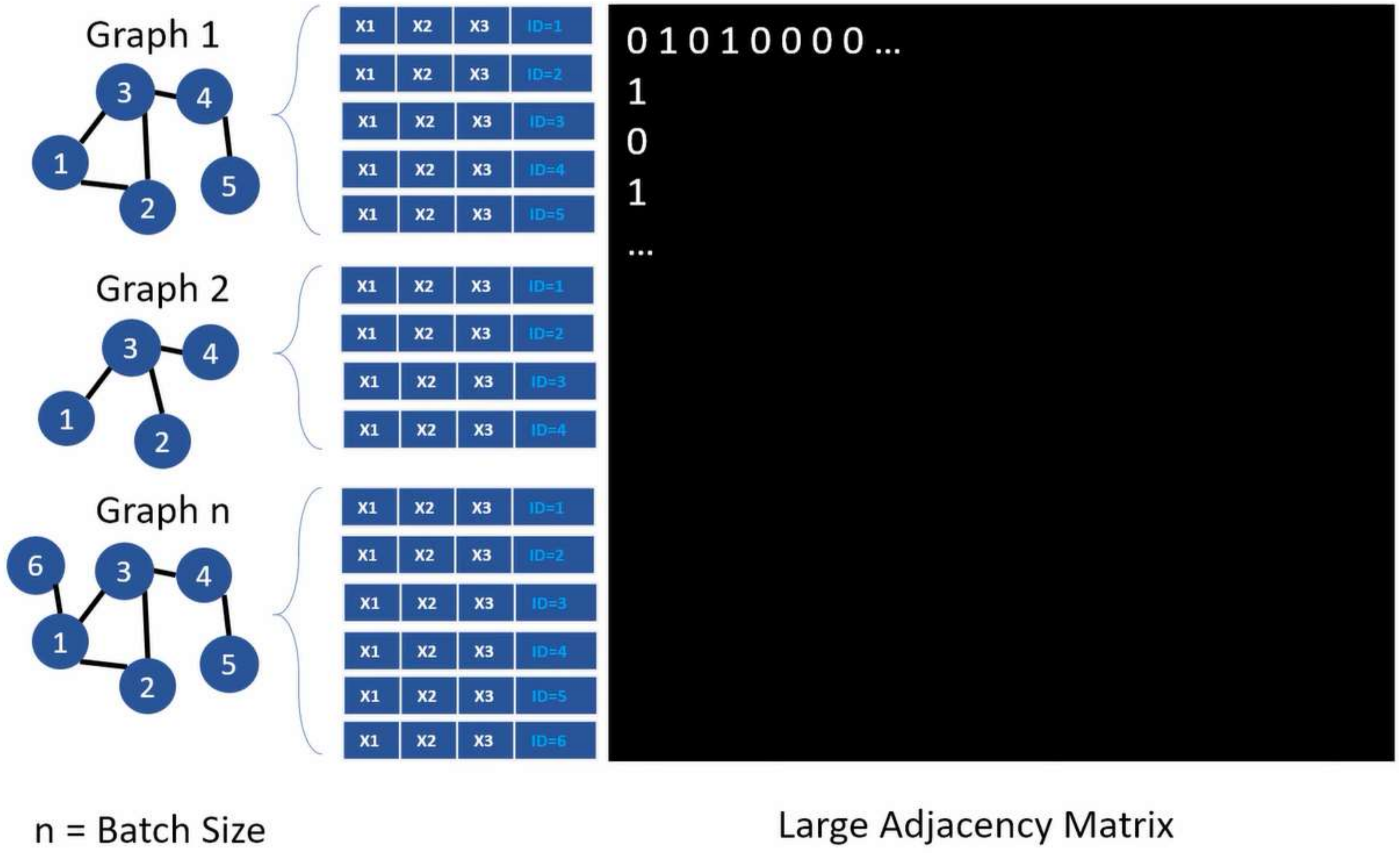


n = Batch Size

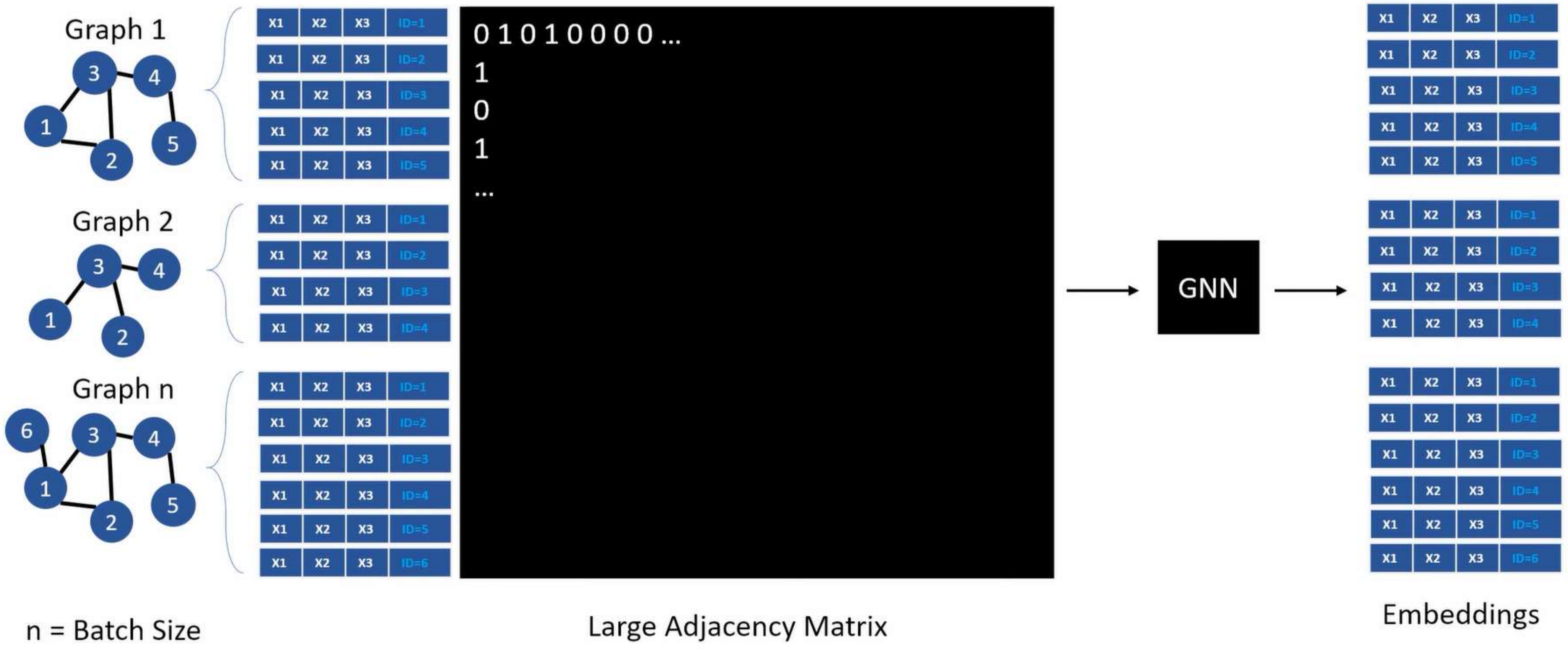
BATCHING WITH GRAPHS



BATCHING WITH GRAPHS



BATCHING WITH GRAPHS



SCALING UP GRAPH NEURAL NETWORKS TO **LARGE GRAPHS**

GRAPHS IN MODERN APPLICATIONS

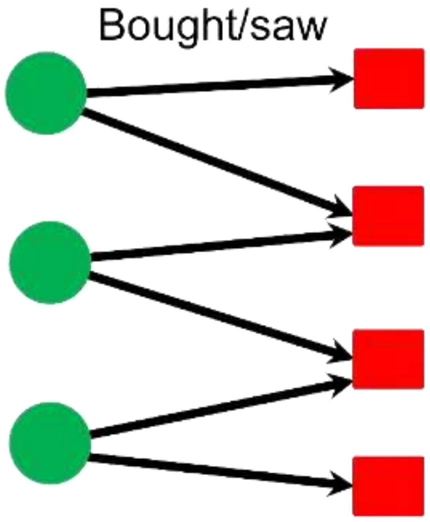
Recommender systems:

- Amazone
- YouTube
- Pinterest
- Instagram

Tasks:

- Recommend Items (Link Prediction)
- Classify users/Items (Node Classification

- **Users:** 100M ~ 1B
- **Products / Videos:** 10M~ 1B

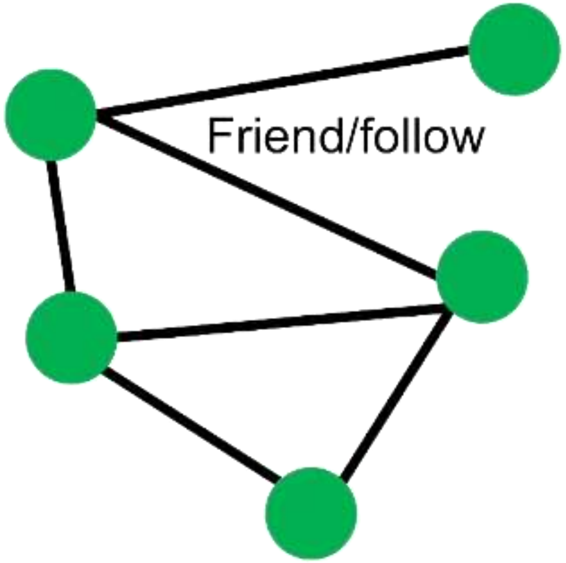


GRAPHS IN MODERN APPLICATIONS

Social Networks
- Facebook
- Twitter
- Instagram

Tasks:
- Friend Recommend(Link Prediction)
- User property recommendation (Node-Level)

❑ **Users:**
300M ~ 3B



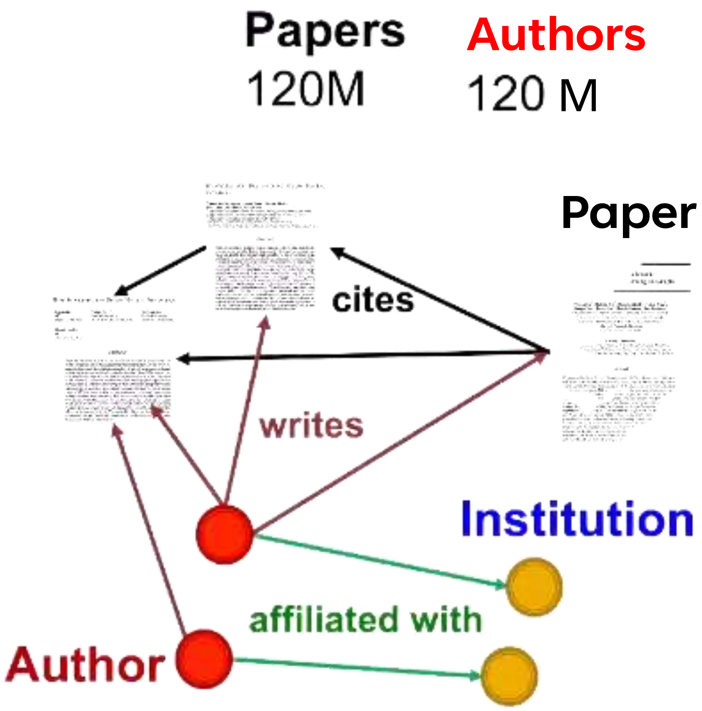
GRAPHS IN MODERN APPLICATIONS

Academic Graph

- Microsoft Academic Graph/

Tasks:

- Paper categorization (node classification)
- Author collaboration recommendation
- Paper citation recommendation (Link prediction)

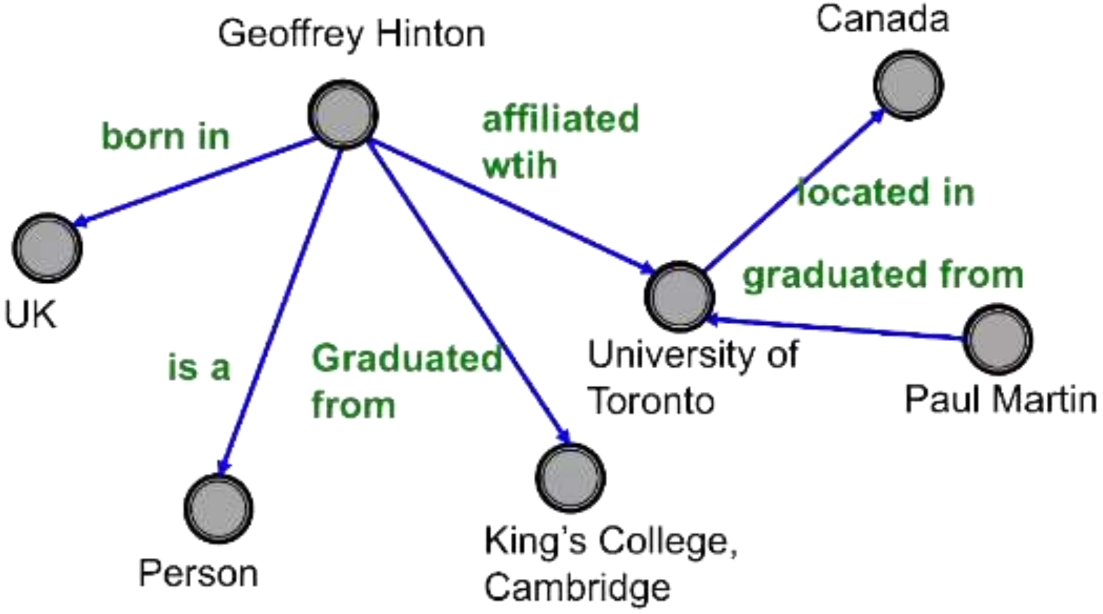


GRAPHS IN MODERN APPLICATIONS

Knowledge Graphs (KGs)
- Wikipedia
- Freebase

Tasks:
- KG completion
- Reasoning

❑ **Entities:**
80M ~ 90M



WHAT IS IN COMMON?!

- **Large-scale:**
  - #Nodes ranges from 10M to 10B
  - #edges ranges from 100M to 100B
- **Taks:**
  - **Node-level:**
    Use/Item/Paper classification
  - **Link-level:**
    Recommendation/Completion

PROBLEM!

Full-batch implementation is **not feasible** for a large graphs

Time inefficiency

- In CPU takes too much time!

Memory Limitations

- GPU memory is extremely limited
- We cannot load entire dataset into memory

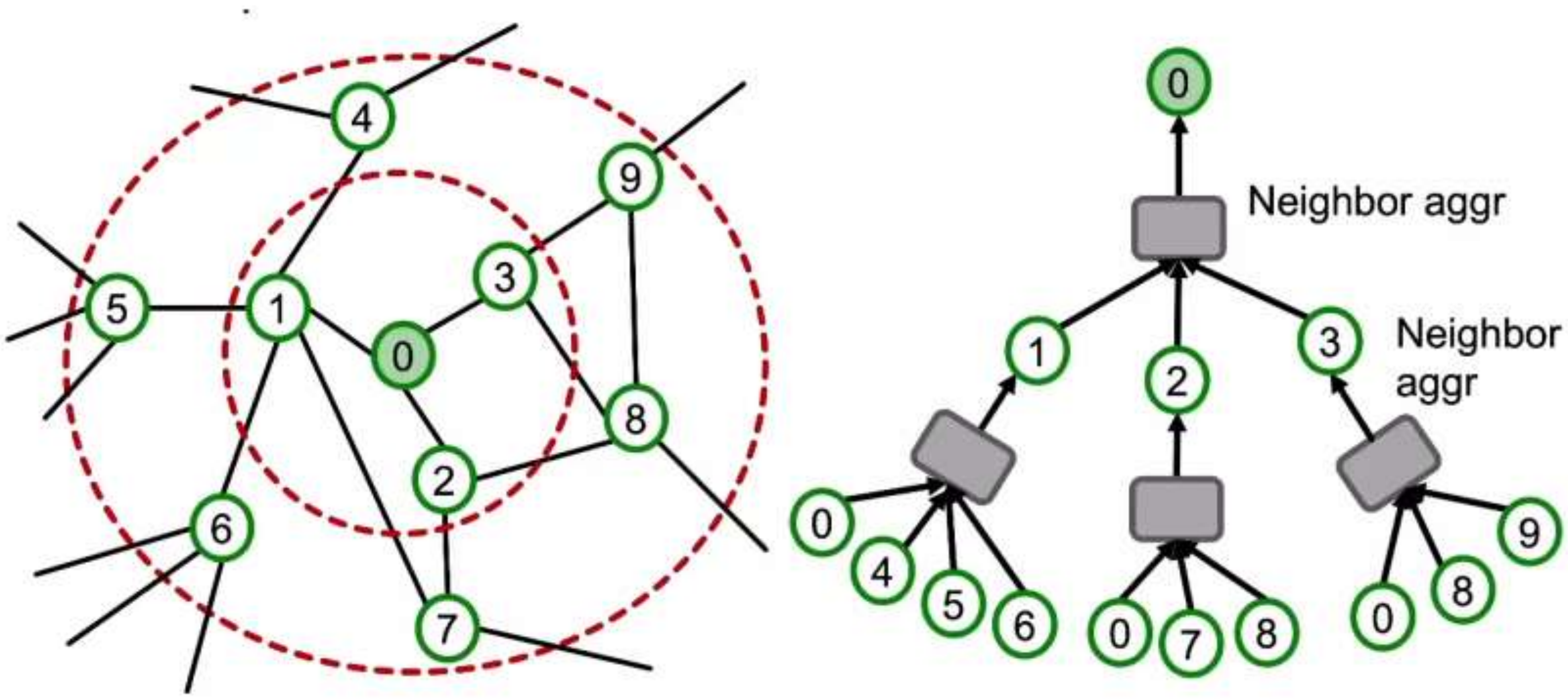
SOLUTIONS!
SOME **METHODS FOR SCALING UP GNNS**

- Perform message-passing over **small subgraphs in each mini-batch**
  - Only the subgraphs need to be loaded on a GPU at a time.
  - Neighbour Sampling [Hamilton NeuriPS 2017]
  - Cluster-GCN [Chiang et al. KDD 2019]

- **Simplifies a GNN into feature-preprocessing operation**
  - Can be efficiently performed even on a CPU
  - Simplified GCN [Wu et al. ICML2019]

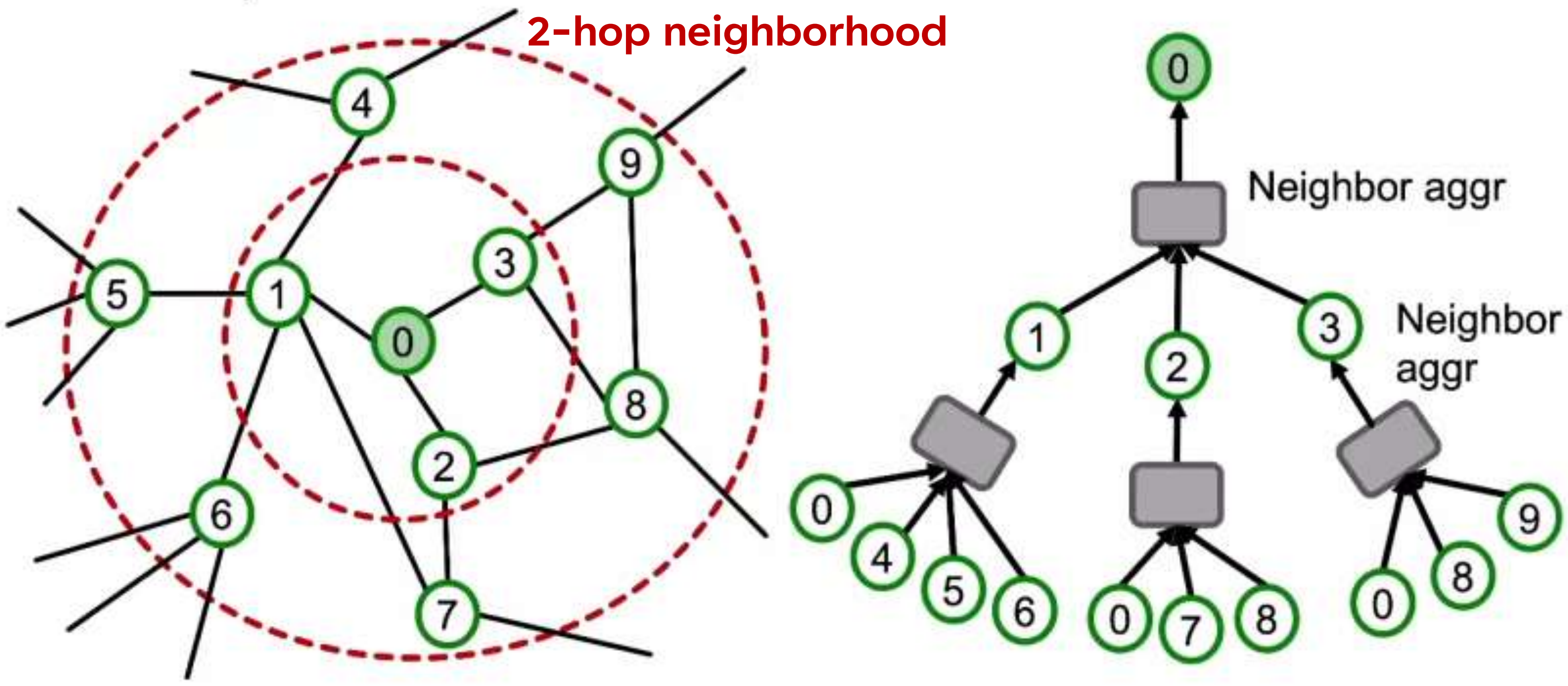
GRAPHSAGE NEIGHBOR SAMPLING

GNNs generate node embeddings via neighbour aggregation.



GRAPHSAGE NEIGHBOR SAMPLING

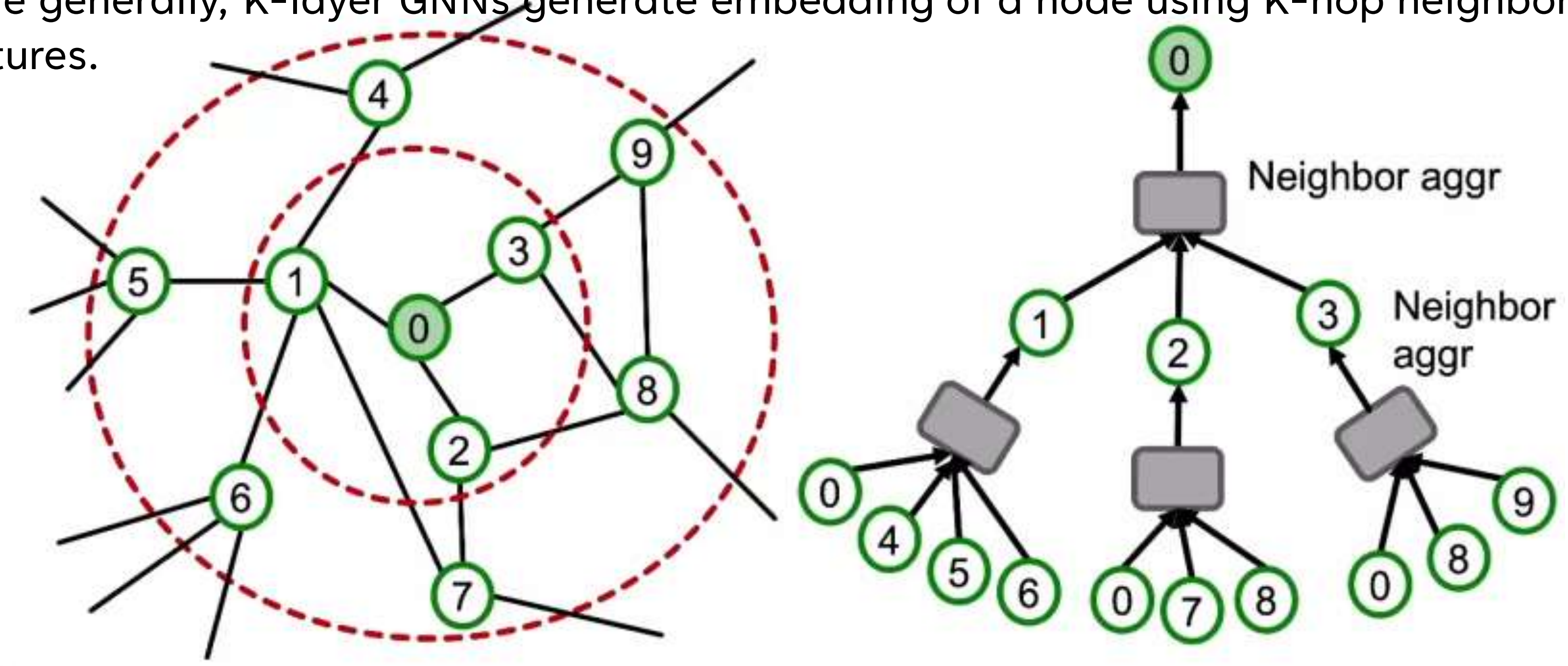
Observation: A 2-layer GNN generates embedding of node "0" using 2-hop neighborhood structure and features.



GRAPHSAGE NEIGHBOR SAMPLING

Observation: A 2-layer GNN generates embedding of node "0" using 2-hop neighborhood structure and features.

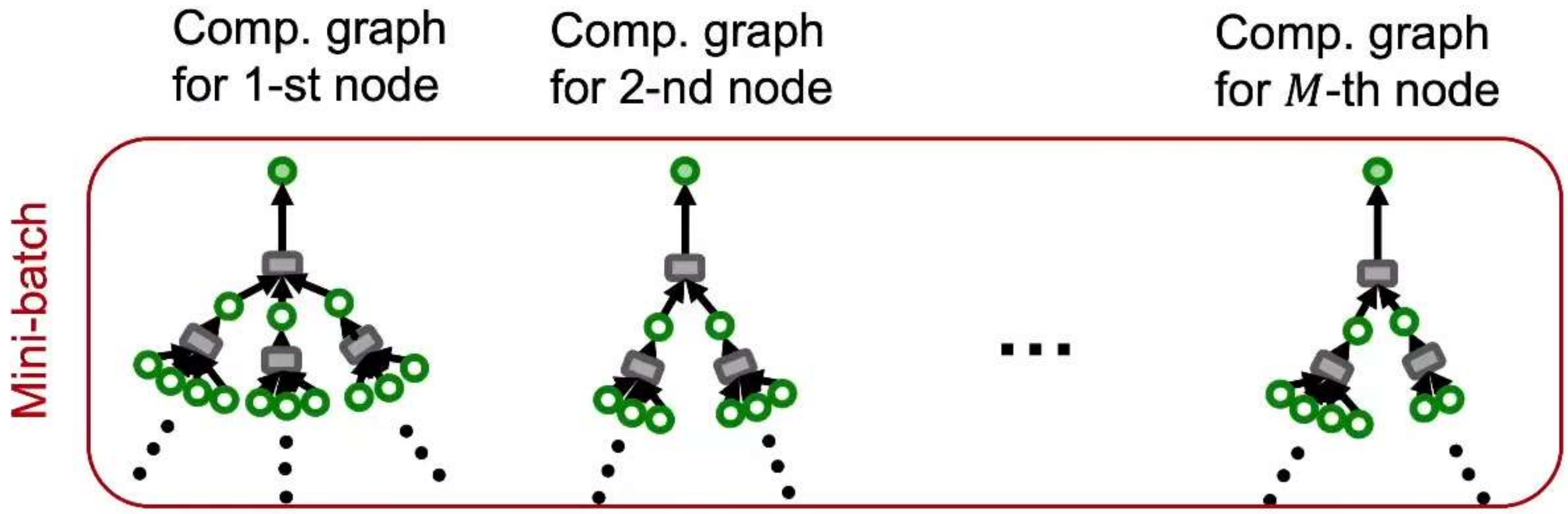
More generally, K-layer GNNs generate embedding of a node using K-hop neighborhood structure and features.



GRAPHSAGE NEIGHBOR SAMPLING

Key insight: To compute embedding of a single node, all we need is the **K-hop neighborhood** (which defines the computation graph).

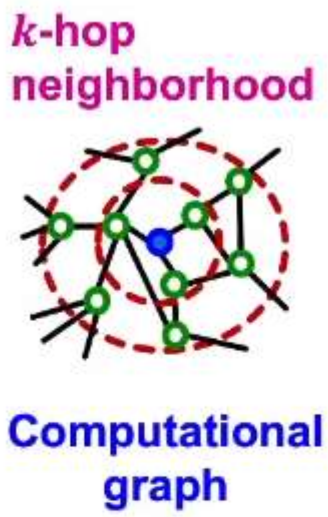
- Given a set of **M different nodes in a mini-batch**, we can generate their embeddings using M computational graphs. **Can be computed on GPU!**



STOCHASTIC TRAINING OF GNNS

We can now consider the following SGD strategy for training K-layer GNNs:

- Randomly sample M (<< N) nodes.
- For each sampled node v:
  - Get k-hop neighbourhood, and construct the computation graph.
  - Use the above to generate v's embedding.
- Compute the loss $l_{sub}(\theta)$ averaged over the M nodes.
- Perform SGD: $\theta \leftarrow \theta - \nabla l_{sub}(\theta)$

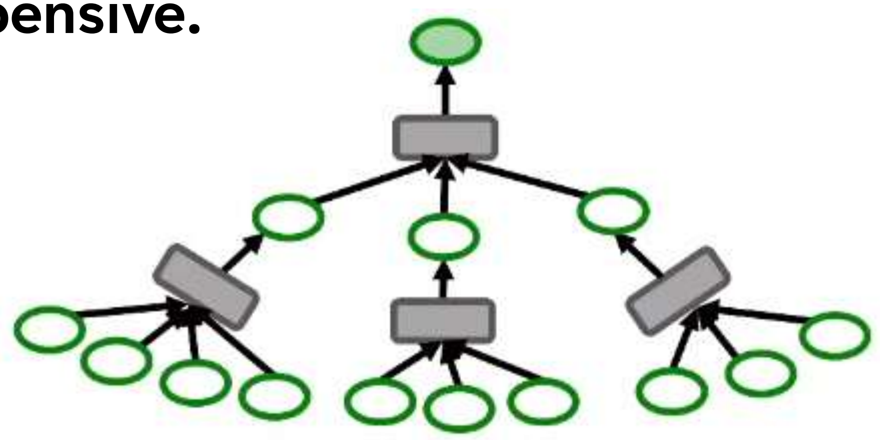


ISSUE STOCHASTIC TRAINING

- For each node, we need to get the entire K-hop neighborhood and pass it through the computation graph.

- We need to aggregate lot of information just to compute one node embedding.

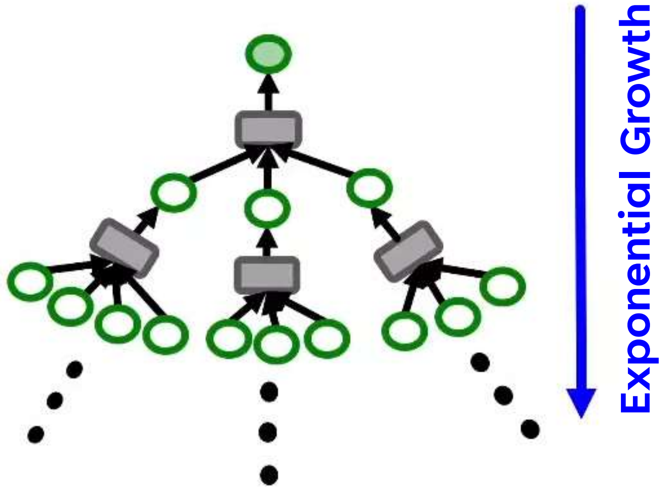
- **Computationally expensive.**

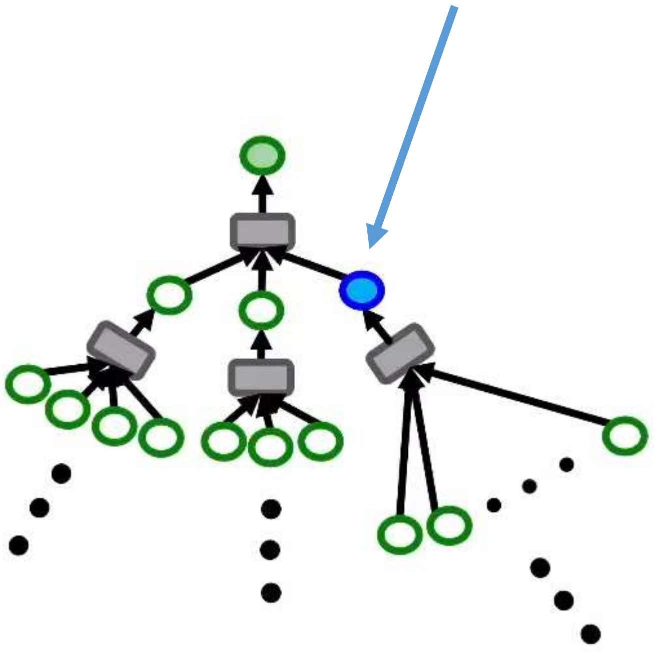


ISSUE STOCHASTIC TRAINING

More details:

- ➢ Computation graph becomes exponentially large with respect to the layer size K.
- ➢ Computation graph explodes when it hits a hub node (high-degree node).





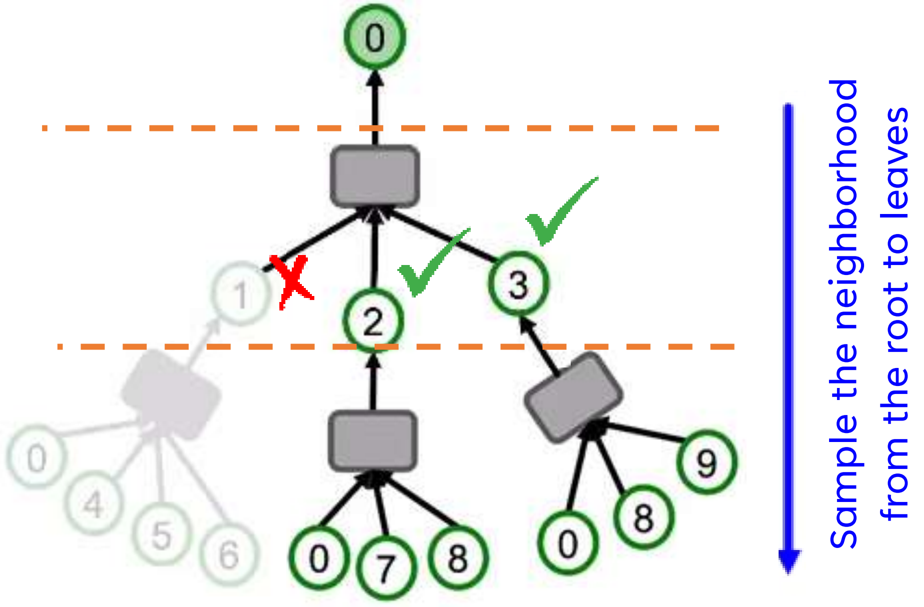
NEIGHBOR SAMPLING

Key idea: Construct the computational graph by (randomly) sampling at most H neighbours at each hop.

- Example:

1st hub neighborhood

Sample 2, 3 | Drop 1



NEIGHBOR SAMPLING

Key idea: Construct the computational graph by (randomly) sampling at most H neighbours at each hop.

❑ Example:

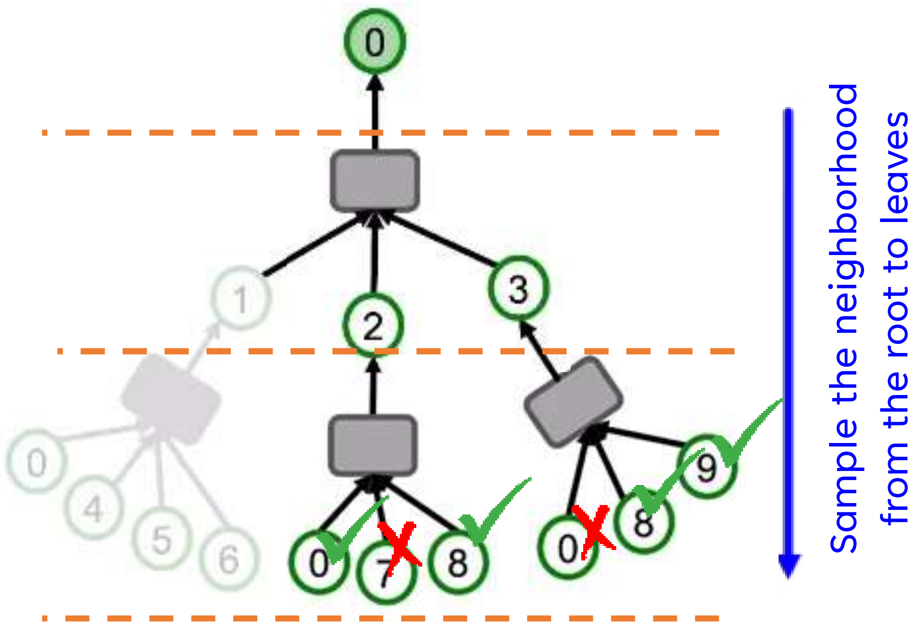
1st hub neighborhood

Sample 2, 3 | Drop 1

2nd hub neighborhood

Sample 0, 8 | Drop 7

Sample 8, 9 | Drop 0



NEIGHBOR SAMPLING

Key idea: Construct the computational graph by (randomly) sampling at most H neighbours at each hop.

- ❑ Example:

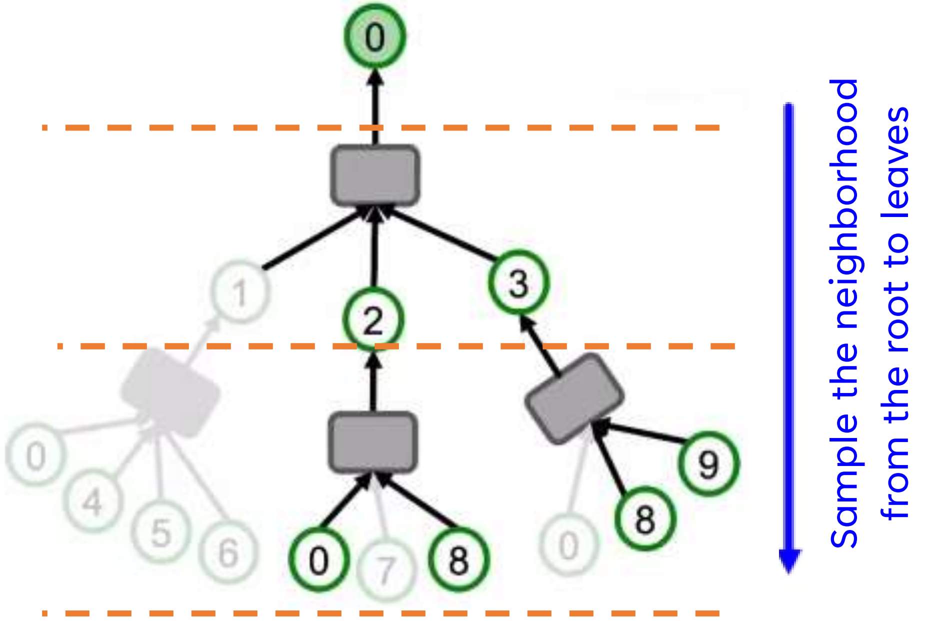
1st hub neighborhood

Sample 2, 3 | Drop 1

2nd hub neighborhood

Sample 0, 8 | Drop 7

Sample 8, 9 | Drop 0



❖ K-layer GNN will at most involve $\prod_{k=1}^{K} H_k$ leaf nodes in computation graph.

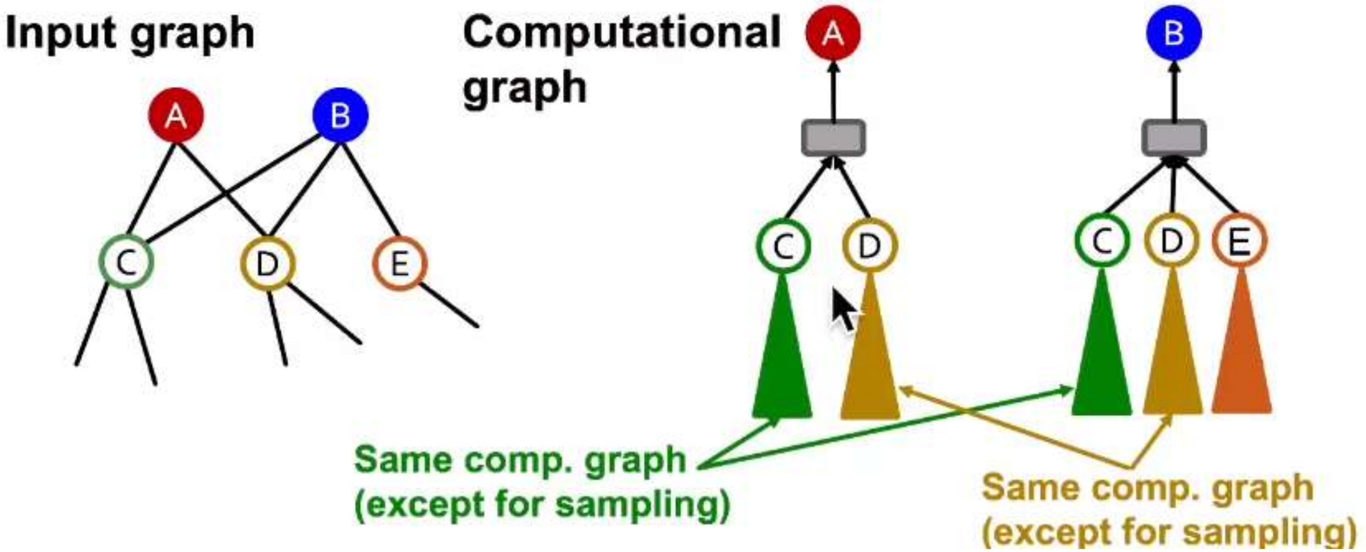
REMARKS ON NEIGHBOR SAMPLING

- **Remark 1: Trade-off in sampling number H**
  - Smaller H leads to more efficient neighbour aggregation, but results in more unstable training due to the larger variance in neighbour aggregation.

- **Remark 2: Computational time**
  - Even with neighbour sampling, the size of the computational graph is still exponential with respect to number of GNN layers K.
  - Increasing one GNN layer would make computation H times more expensive.
- **Remark 3: How to sample the nodes**
  - Random sampling: fast but many times not optimal!
  - Random walk with restart

ISSUE WITH NEIGHBOUR SAMPLING

- Issue with neighbour sampling:
  - The size of computational graph becomes exponentially large w.r.t. the #GNN layers.
  - **Computation is redundant,** especially when nodes in a mini-batch share many neighbours.



conv.SAGEConv

class **SAGEConv** (**in_channels**: Union[int, Tuple[int, int]], **out_channels**: int, **aggr**: Optional[Union[str, List[str], Aggregation]] = `'mean'`, **normalize**: bool = `False`, **root_weight**: bool = `True`, **project**: bool = `False`, **bias**: bool = `True`, ****kwargs**) [source]

Bases: `MessagePassing`

The GraphSAGE operator from the "Inductive Representation Learning on Large Graphs" paper

$$\mathbf{x}'_i = \mathbf{W}_1 \mathbf{x}_i + \mathbf{W}_2 \cdot \mathrm{mean}_{j \in \mathcal{N}(i)} \mathbf{x}_j$$

https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.SAGEConv.html#torch_geometric.nn.conv.SAGEConv

Redundancy-Free Computation for Graph Neural Networks

Zhihao Jia
Stanford University
zhihao@cs.stanford.edu

Sina Lin
Microsoft
silin@microsoft.com

Rex Ying
Stanford University
rexying@stanford.edu

Jiaxuan You
Stanford University
jiaxuan@stanford.edu

Jure Leskovec
Stanford University
jure@cs.stanford.edu

Alex Aiken
Stanford University
aiken@cs.stanford.edu

ABSTRACT

Graph Neural Networks (GNNs) are based on repeated aggregations of information from nodes' neighbors in a graph. However, because nodes share many neighbors, a naive implementation leads to repeated and inefficient aggregations and represents significant computational overhead. Here we propose *Hierarchically Aggregated computation Graphs* (HAGs), a new GNN representation technique that explicitly avoids redundancy by managing intermediate aggregation results hierarchically and eliminates repeated computations and unnecessary data transfers in GNN training and inference. HAGs perform the same computations and give the same models/accuracy as traditional GNNs, but in a much shorter time due

1 INTRODUCTION

Graph Neural Network models (GNNs) generalize deep representation learning to graph data [3, 9, 23] and have achieved state-of-the-art performance across a number of graph-based tasks, such as node classification, link prediction, and graph classification and recommender systems [8, 14, 24, 27].

GNNs are based on a recursive neighborhood aggregation scheme, where within a single layer of a GNN each node aggregates its neighbors' activations and uses the aggregated value to update its own activation [23]. Such updated activations are then recursively propagated multiple times (multiple layers). In the end, every node in a GNN collects information from other nodes that are in its k-

One approach to solve the redundancy problem!

https://dl.acm.org/doi/pdf/10.1145/3394486.3403142

CLUSTER-GCN: REVIEW FULL-BATCH GNN

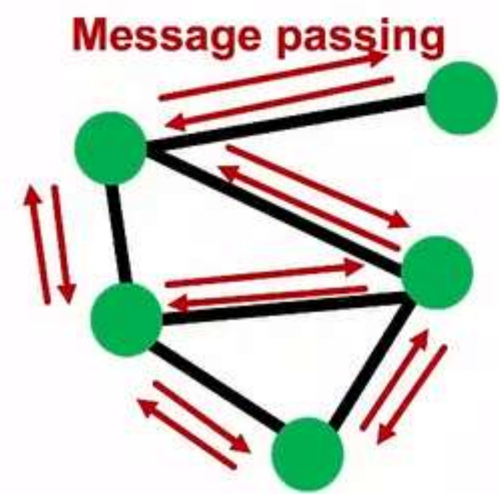
- In full-batch GNN implementation, all the node embeddings are updated together using embeddings of the previous layer

Update for all $v \in V$

Message

$$h_v^{(\ell)} = COMBINE\left(h_v^{(\ell-1)}, AGGR\left(\left\{h_u^{(\ell-1)}\right\}_{u \in N(v)}\right)\right)$$

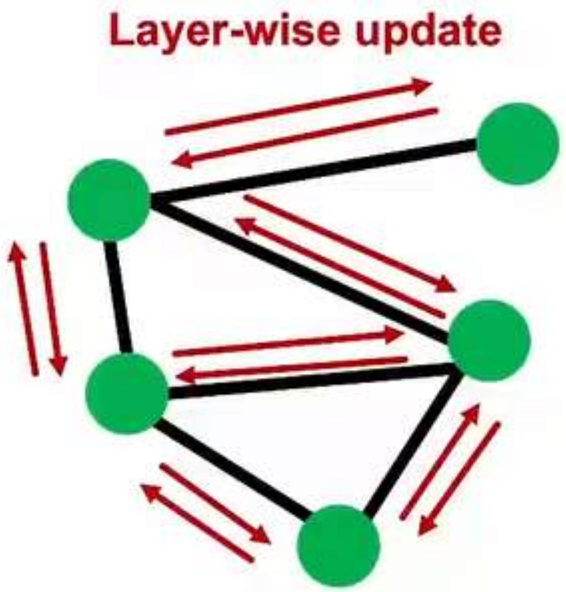
- In each layer, only 2*#(edges) messages need to be computed.
- For K-layer GNN, only 2K*#(edges) messages need to be computed.
- GNN's entire computation is only linear in #(edges) and #(GNN layers). **Fast!**



CLUSTER-GCN: INSIGHT FROM FULL-BATCH GNN

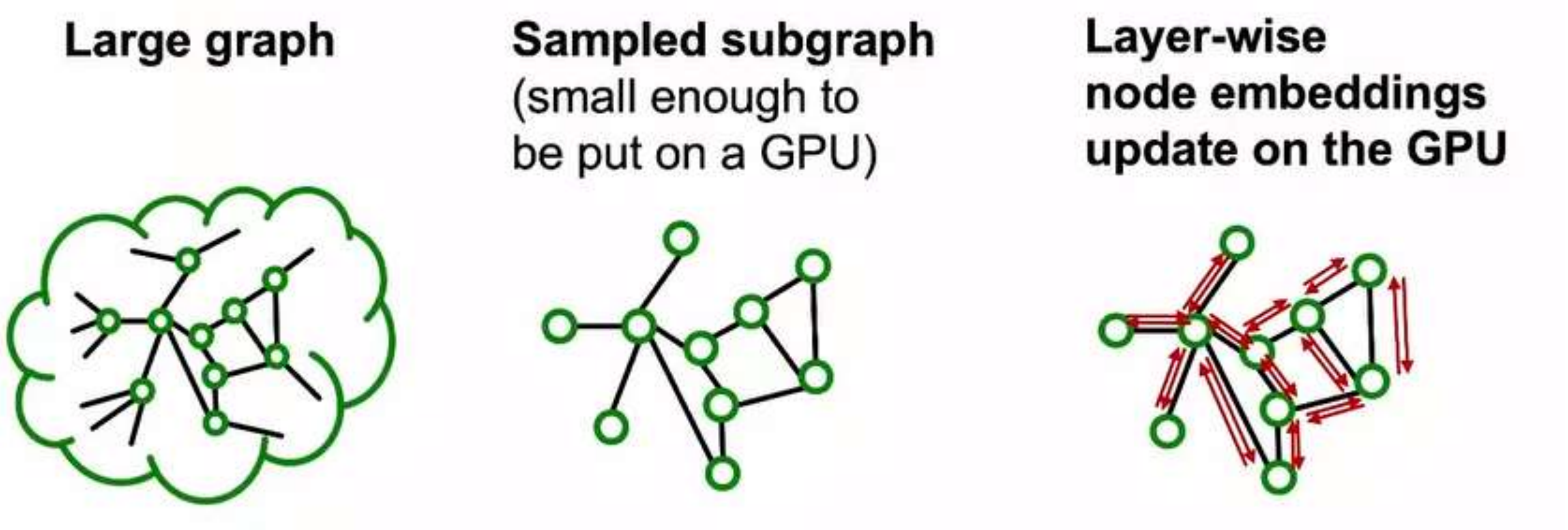
- The **layer-wise** node embedding update allows the re-use of embeddings from the previous layer.

- This significantly **reduces the computational redundancy of neighbour sampling**.
  - Of course, the layer-wise update is not feasible for a large graph due to limited GPU memory.



CLUSTER-GCN: SUB-GRAPH SAMPLING

- ✓ **Key idea:** We can sample a small subgraph of the large graph and then perform the efficient layer-wise node embeddings update over the subgraph.



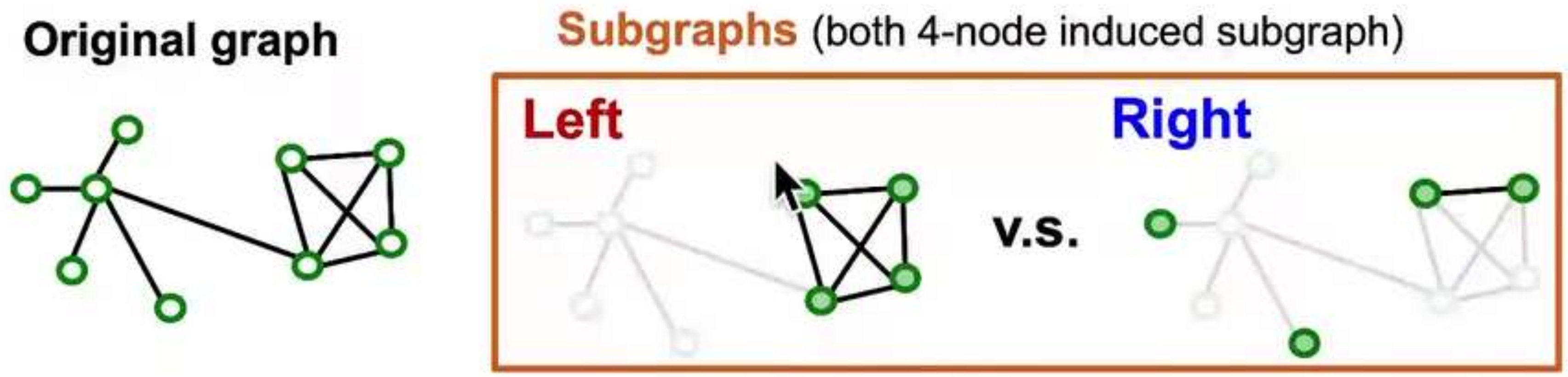
CLUSTER-GCN: SUB-GRAPH SAMPLING

Key question: What subgraphs are good for training GNNs?

- Recall: GNN performs node embedding by passing messages via the edges.
  - Subgraphs **should retain edge connectivity structure of the original graph as much as possible**.
  - This way, the GNN over the subgraph generates embeddings closer to the GNN over the original graph.

CLUSTER-GCN: SUB-GRAPH SAMPLING

Which subgraph is good for training GNN?



- **Left subgraph:**
  retains the essential community structure among the 4 nodes → Good ✓
- **Right subgraph:**
  drops many connectivity patterns, even leading to isolated nodes → Bad ✗

CLUSTER-GCN: EXPLOITING COMMUNITY STRUCTURE

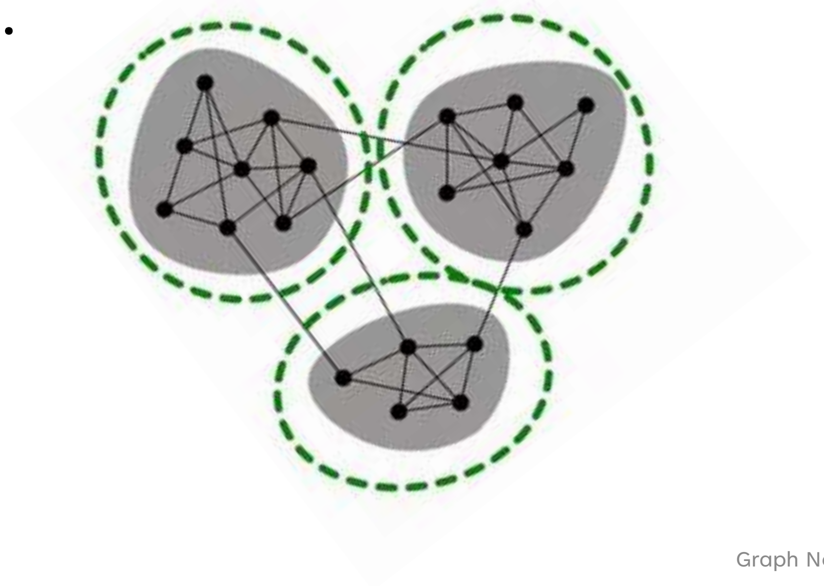
Real-world graph exhibits community structure

➢ A large graph can be decomposed into many small communities.

Key insight [Chiang et al. KDD 2019]**:**

❑ Sample a community as a subgraph.

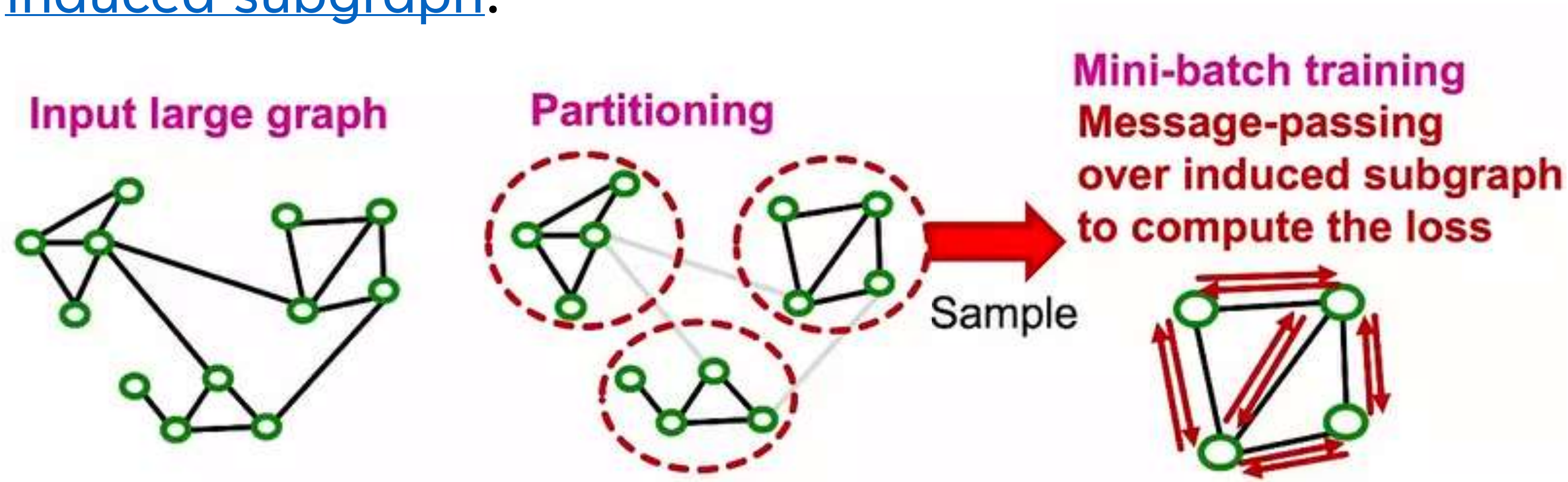
❑ Each subgraph retains essential local connectivity pattern of the original graph.



CLUSTER-GCN: OVERVIEW

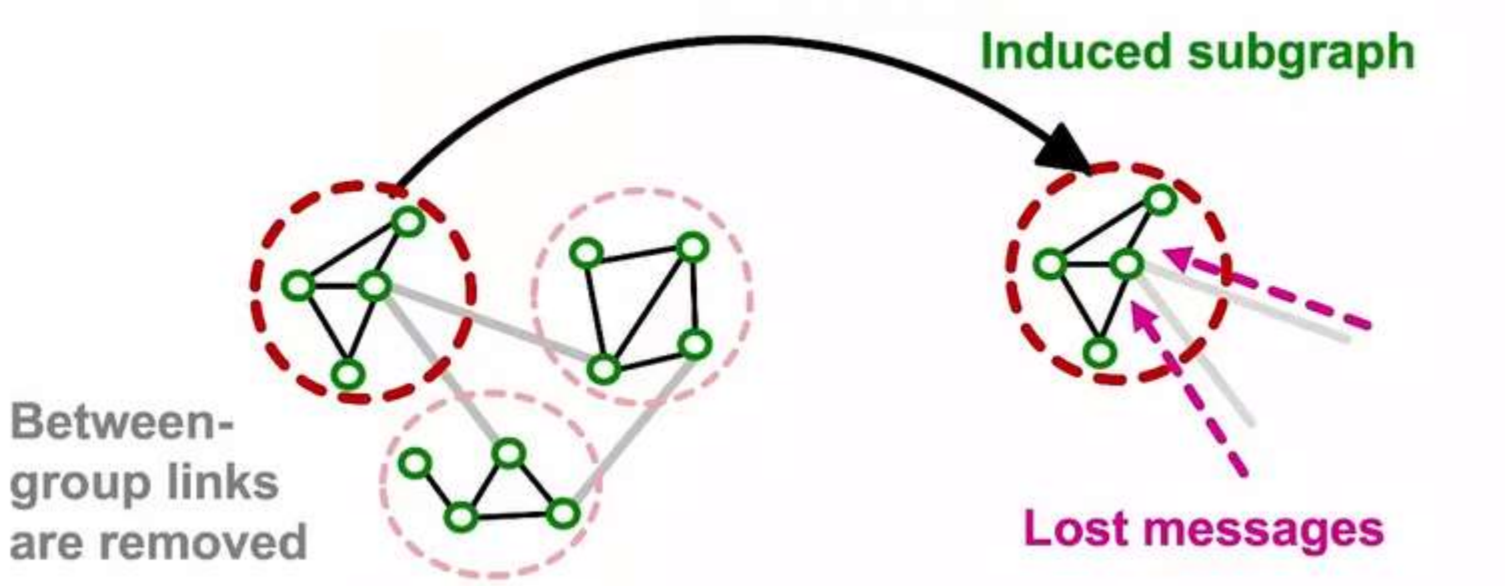
Cluster-GCN consists of two steps: It is a **Vanilla cluster-GCN**

1. **Pre-processing**:
   Given a large graph, partition it into groups of nodes (i.e., subgraphs).
2. **Mini-batch training**:
   Sample one node group at a time. Apply GNN's message passing over the induced subgraph.



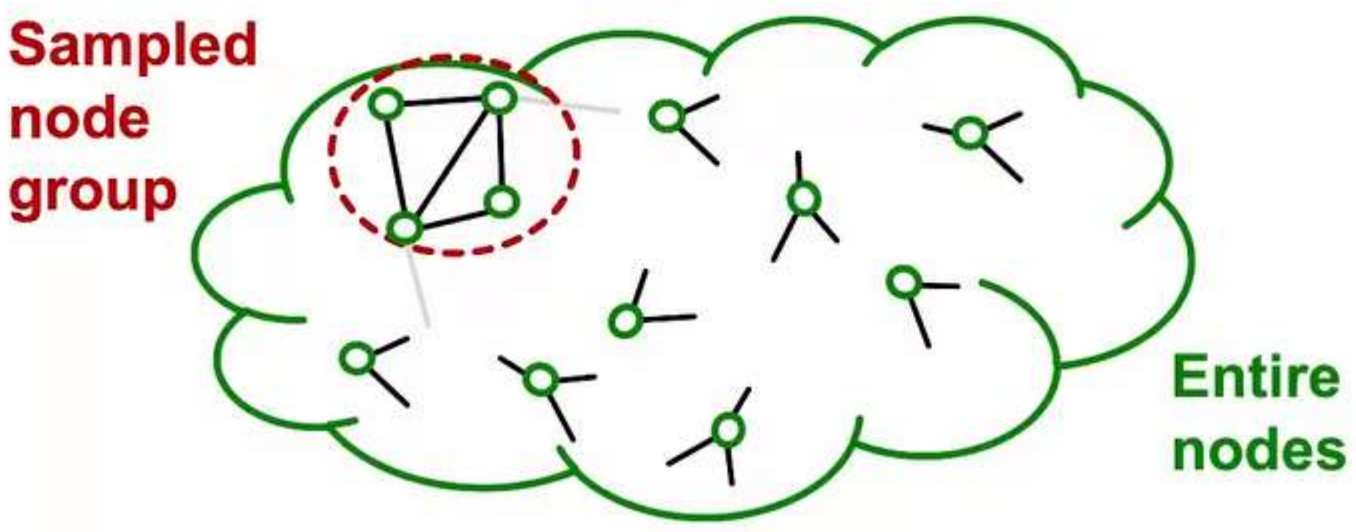
CLUSTER-GCN: ISSUES(1)

- The induced subgraph **removes** between-group links.
- As a result, messages from other groups will be lost during message passing, which could hurt the GNN's performance.



CLUSTER-GCN: ISSUES(2)

- Graph community detection algorithm puts similar nodes together in the same group.
- Sampled node group tends to only cover the small-concentrated portion of the entire data.



ADVANCED CLUSTER-GCN: ISSUES(3)

Sampled nodes are not diverse enough to be represent the graph structure:

- ❑ As a result, the gradient averaged over the sampled nodes, $\frac{1}{|V_c|}\sum_{v \epsilon V_c} \nabla l_v(\theta)$, becomes unreliable.
  - Fluctuates a lot from a node group to another.
  - In other words, the gradient has high variance.
- ❑ Leads to slow convergence of SGD

ADVANCED CLUSTER-GCN

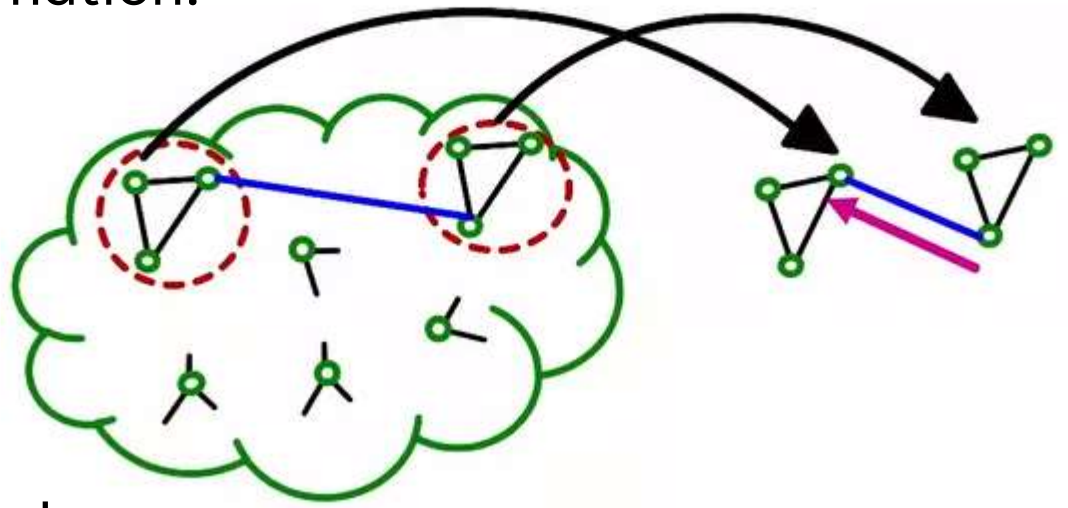
✓ **Solution:** Aggregate multiple node groups per mini-batch.

❑ Partition the graph into **relatively-small groups of nodes**.
❑ For each mini-batch:
1. Sample and aggregate multiple node groups.
2. Construct the induced subgraph of the aggregated node group.
3. The rest is the same as vanilla Cluster-GCN (compute node embeddings and the loss, update parameters)

ADVANCED CLUSTER-GCN

Why does the solution work?

- ❑ Sampling multiple node groups
  - Makes the sampled nodes more representative of the entire nodes.
  - Leads to less variance in gradient estimation.



- ❑ The induced subgraph over aggregated node groups
  - Includes between-group edges
  - Message can flow across groups.

GRAPHSAGE VS CLUSTER-GCN

- Cluster-GCN is more computationally efficient than neighbour sampling, especially when #(GNN layers) is large.

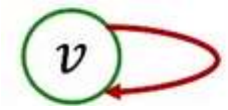
- But Cluster-GCN leads to systematically biased gradient estimates (due to missing cross-community edges)

SIMPLIFYING GNNS

- We start from Graph Convolutional Network (GCN) [Kipf & Welling ICLR 2017].
- We simplify GCN by **removing the non-linear activation** from the GCN [Wu et all. ICML 2019].
  - *Wu et al.* demonstrated that the performance on benchmark is not much lower by the simplification.
- Simplified GCN turns out to be extremely scalable by the model design.

SIMPLIFYING GNNS: RECALL MEAN-POOL IN GCN

- **Given:** Graph $G = (V, E)$ with input node features X_v for $v \in V$, where E includes the self-loop:
  - $(v, v) \in E$ for all $v \in V$.



- Set input node embeddings: $h_v^{(0)} = X_v \; for \; v \in V$
- For $k \in \{O, \ldots, K - 1\}$:
  - For all $v \in V$, aggregate neighbouring information as

$$h_v^{(k+1)} = \mathrm{ReLU}\left(\boldsymbol{W_k} \frac{1}{|N(v)|} \sum_{u \in N(v)} h_u^{(k)}\right)$$

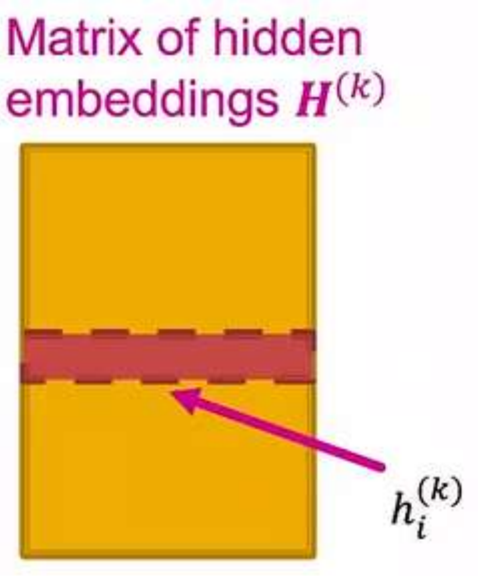
Trainable weight matrices (i.e., what we learn)

Mean-pooling

- Final node embedding: $Z_v = h_v^{(k)}$

SIMPLIFYING GNNS: RECALL MATRIX FORMULATION OF GCN

GCN aggregations can be formulated as matrix vector product:



- Let $\boldsymbol{H}^{(k)} = [h_1^{(k)} \ldots h_{|v|}^{(k)}]^T$
- Let $\boldsymbol{A}$ be the adjacency matrix (w/ self-loop)
- Then: $\sum_{u \in N(v)} h_u^{(k)} = A_{v,:} \boldsymbol{H}^{(k)}$
- Let $\boldsymbol{D}$ be diagonal matrix where

$$D_{v,v} = Deg(v) = |N(v)|$$

- The inverse of D: D^{-1} is also diagonal: $D_{v,v}^{-1} = 1/|N(v)|$
- **Therefore,**

$$\frac{1}{|N(v)|} \sum_{u \in N(v)} h_u^{(k)} \Rightarrow \boldsymbol{H}^{(l+1)} = \boldsymbol{D}^{-1} \boldsymbol{A} \boldsymbol{H}^{(l)}$$

SIMPLIFYING GNNS: RECALL MATRIX FORMULATION OF GCN

GCN's neighbour aggregation:

$$h_v^{(k+1)} = \mathrm{ReLU}\left(\boldsymbol{W_k} \frac{1}{|N(v)|} \sum_{u \in N(v)} h_u^{(k)}\right)$$

In matrix form:

$$\boldsymbol{H}^{(k+1)} = \mathrm{ReLU}\left(\tilde{\boldsymbol{A}} \boldsymbol{H}^{(k)} \boldsymbol{W}_{\boldsymbol{k}}^{\mathrm{T}}\right)$$

where $\tilde{A} = D^{-1}A$

Note: The original GCN uses re-normalized version: $\tilde{A} = D^{-1/2} A\, D^{-1/2}$

- Empirically, this version of $\tilde{A}$ often gives better performance than $D^{-1}A$

SIMPLIFYING GNNS

Simplify GCN by removing ReLU non-linearity:

$$H^{(k+1)} = \widetilde{A} H^{(k)} W_k^{\mathrm{T}}$$

The final node embedding matrix is given as

$$H^{(K)} = \widetilde{A}\, H^{(K-1)}\, W_{K-1}^{\mathrm{T}}$$

$$= \widetilde{A}(\widetilde{A} H^{(K-2)} W_{K-2}^{T}) W_{K-1}^{\mathrm{T}}$$

$$\cdots = \widetilde{A}(\widetilde{A}(\cdots(\widetilde{A}\, H^{(0)}\, W_0^{\mathrm{T}}) \cdots) W_{K-2}^{\mathrm{T}}) W_{K-1}^{\mathrm{T}}$$

$$= \widetilde{A}^K\ X\ (W_0^{\mathrm{T}} \cdots W_{K-1}^{\mathrm{T}})$$

Composition of linear transformation is still linear!

$$= \widetilde{A}^K X W^{\mathrm{T}} \quad \text{where } W \equiv W_{K-1} \cdots W_0$$

SIMPLIFYING GNNS

- Removing ReLU significantly simplifies GCN!

$$H^{(K)} = \tilde{A}^K X W^T$$

- Notice $\tilde{\boldsymbol{A}}^{\boldsymbol{K}}\boldsymbol{X}$ does not contain any learnable parameters; hence, **it can be pre-computed**.
  - Efficiently computable as a sequence of sparse-matrix vector products:
  - Do $X \leftarrow \tilde{A}X$ for K times.

SIMPLIFYING GNNS

- Let $\widetilde{\boldsymbol{X}} = \widetilde{\boldsymbol{A}}^{\boldsymbol{K}}\boldsymbol{X}$ be pre-computed matrix.

Simplified GCN's final embedding is

$$H^{(K)} = \tilde{X}W^{T}$$

- It's just **a linear transformation of pre-computed matrix**!
- Back to the node embedding form:

$$h_v^{(K)} = \boldsymbol{W}\,\boxed{\widetilde{\boldsymbol{X}}_v}$$

Pre-computed feature vector for node v

- Embedding of node v only depends on its own (pre-processed) feature!

SIMPLIFYING GNNS

- Once $\tilde{X}$ is pre-computed, embeddings of M nodes can be generated in time linear in M:
  - Given M nodes $\{v_1, v_2, \dots, v_M\}$, their embeddings are

- $h_{v_1}^{(K)} = \boldsymbol{W}\widetilde{\boldsymbol{X}}_{v_1}$,
- $h_{v_2}^{(K)} = \boldsymbol{W}\widetilde{\boldsymbol{X}}_{v_2}$,
- ...
- $h_{v_M}^{(K)} = \boldsymbol{W}\widetilde{\boldsymbol{X}}_{v_M}$.

SIMPLIFYING GNNS

In summary, simplified GCN consists of **two steps**:

- **Pre-processing step**:
  - Pre-compute $\widetilde{\boldsymbol{X}} = \widetilde{\boldsymbol{A}}^K \boldsymbol{X}$. Can be done on CPU.
- **Mini-batch training step**:
  - For each mini-batch, randomly-sample M nodes $\{v_1, v_2, \dots, v_M\}$.
  - Compute their embeddings by
    - $h_{v_1}^{(K)} = \boldsymbol{W}\widetilde{\boldsymbol{X}}_{v_1}, h_{v_2}^{(K)} = \boldsymbol{W}\widetilde{\boldsymbol{X}}_{v_2}, \dots, h_{v_M}^{(K)} = \boldsymbol{W}\widetilde{\boldsymbol{X}}_{v_M}$
  - Use the embeddings to make prediction and compute the loss averaged over the M data points.
  - Perform SGD parameter update.

COMPARISON WITH OTHER MODELS

- **Compared to neighbour sampling:**
  - Simplified GCN generates node embeddings much more efficiently (no need to construct the giant computational graph for each node).

- **Compared to Cluster-GCN:**
  - Mini-batch nodes of simplified GCN can be sampled completely randomly from the entire nodes (no need to sample from multiple groups as Cluster-GCN does)
  - Leads to lower SGD variance during training.

- But the model is much less expressive.

COMPARISON WITH OTHER MODELS

Compared to the original GN models, simplified GCN's expressive power is limited due to the lack of non-linearity in generating node embeddings.

COMPARISON WITH OTHER MODELS

Compared to the original GN models, simplified GCN's expressive power is limited due to the lack of non-linearity in generating node embeddings.

Why the performance is good?
https://youtu.be/iTRW9Gh7yKI?list=PLoROMvodv4rPLKxIpqhjhPgdQy7imNkDn&t=880

latest

Search docs

INSTALL PYG

Installation

GET STARTED

GINConv	... Powerful are Graph Neural Networks?" paper
`GINEConv`	The modified `GINConv` operator from the "Strategies for Pre-training Graph Neural Networks" paper
`ARMAConv`	The ARMA graph convolutional operator from the "Graph Neural Networks with Convolutional ARMA Filters" paper
`SGConv`	The simple graph convolutional operator from the "Simplifying Graph Convolutional Networks" paper
`SSGConv`	The simple spectral graph convolutional operator from the "Simple Spectral Graph Convolution" paper
`APPNP`	The approximate personalized propagation of neural predictions layer from the "Predict then Propagate: Graph Neural Networks meet Personalized PageRank" paper
	The graph neural network operator from the ...

https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html

conv.SGConv

class **SGConv** (**in_channels**: int, **out_channels**: int, **K**: int = **1**, **cached**: bool = **False**, **add_self_loops**: bool = **True**, **bias**: bool = **True**, ****kwargs**) [source]

Bases: `MessagePassing`

The simple graph convolutional operator from the "Simplifying Graph Convolutional Networks" paper

$$\mathbf{X}' = \left(\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\right)^K \mathbf{X}\mathbf{\Theta},$$

where $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ denotes the adjacency matrix with inserted self-loops and $\hat{D}_{ii} = \sum_{j=0} \hat{A}_{ij}$ its diagonal degree matrix. The adjacency matrix can include other values than `1` representing edge weights via the optional `edge_weight` tensor.

https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.SGConv.html#torch_geometric.nn.conv.SGConv

EXAMPLE

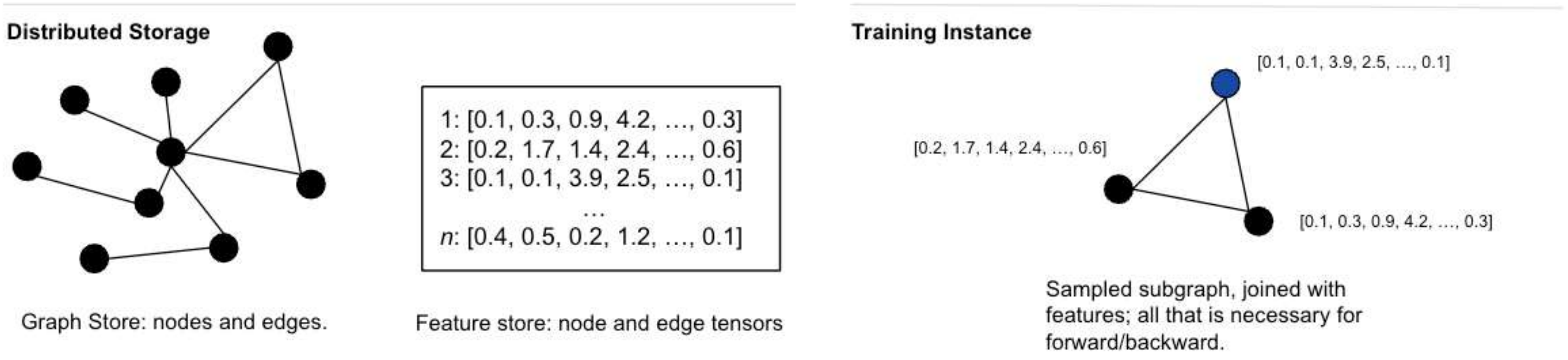
```
class Net(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = SGConv(dataset.num_features, dataset.num_classes, K=2,
                            cached=True)

    def forward(self):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        return F.log_softmax(x, dim=1)
```

https://github.com/pyg-team/pytorch_geometric/blob/master/examples/sgc.py

SCALING UP GNNS VIA REMOTE BACKENDS

- **Using key-value and graph database:**
  - Documentation: https://pytorch-geometric.readthedocs.io/en/latest/advanced/remote.html
  - Example: https://github.com/pyg-team/pytorch_geometric/tree/master/examples/kuzu/papers_100M



EDGE FEATURES

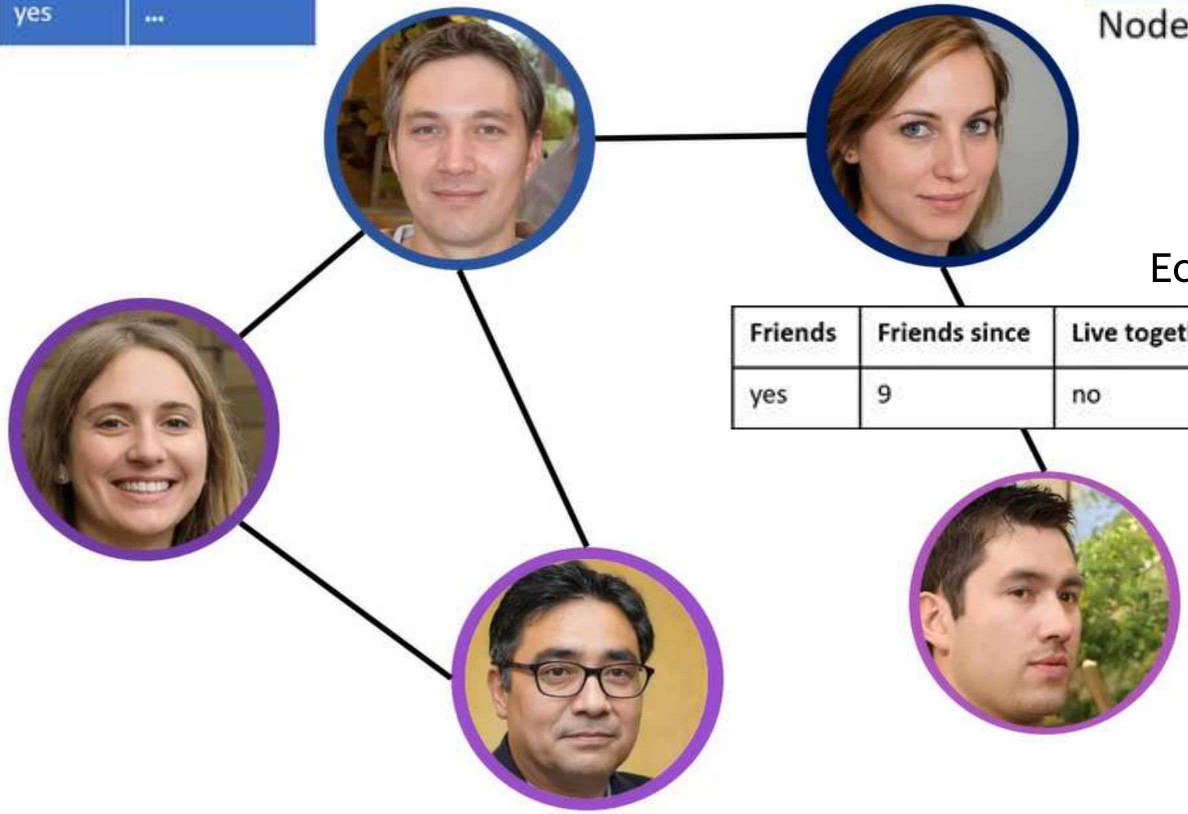
WHY ARE EDGE FEATURES ARE IMPORTANT?

Age	Weight	Smokes	...
39	79	yes	...

Node feature

Age	Weight	Smokes	...
31	65	no	...

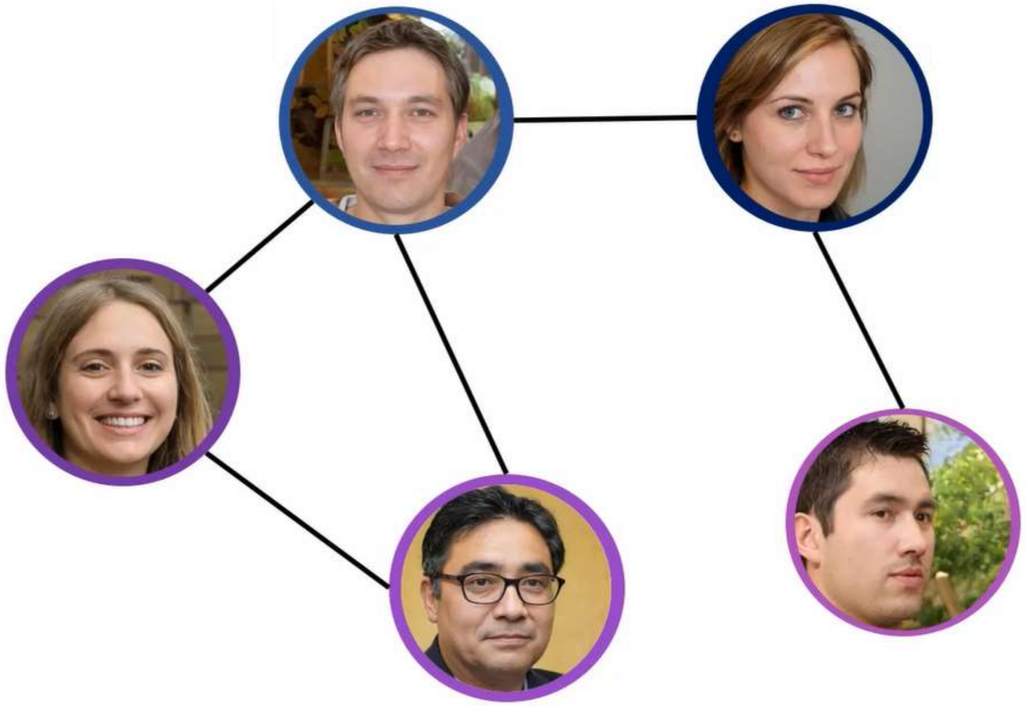
Node feature



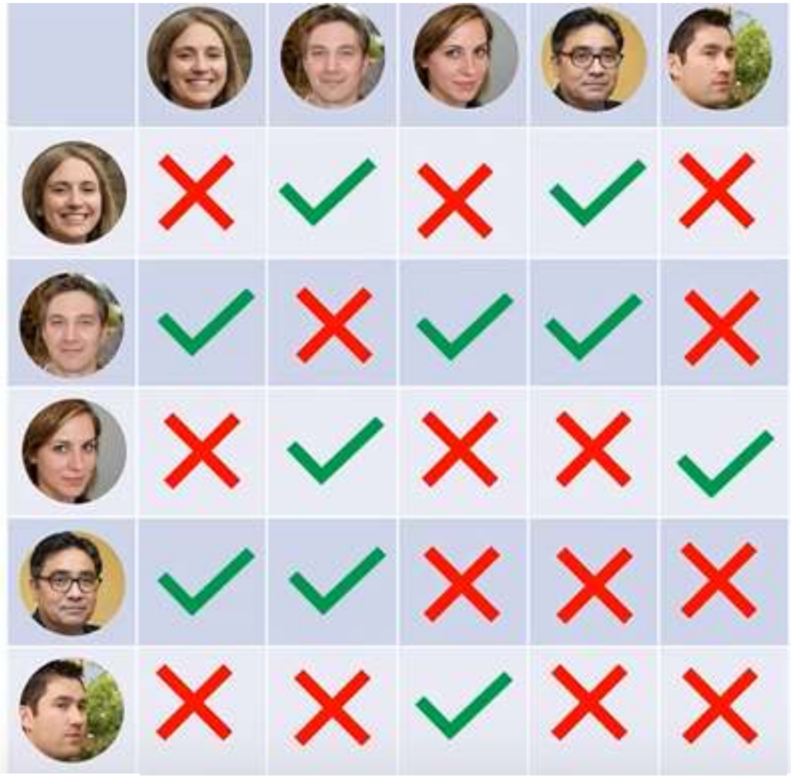
Edge feature

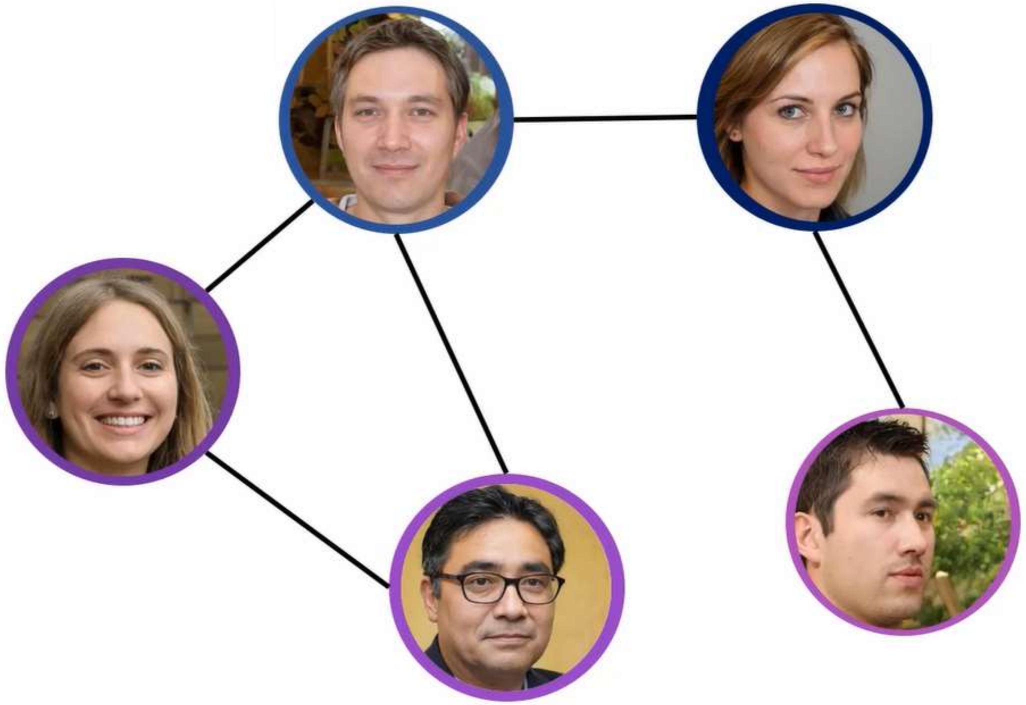
Friends	Friends since	Live together	...
yes	9	no	...

WHY ARE EDGE FEATURES ARE IMPORTANT?



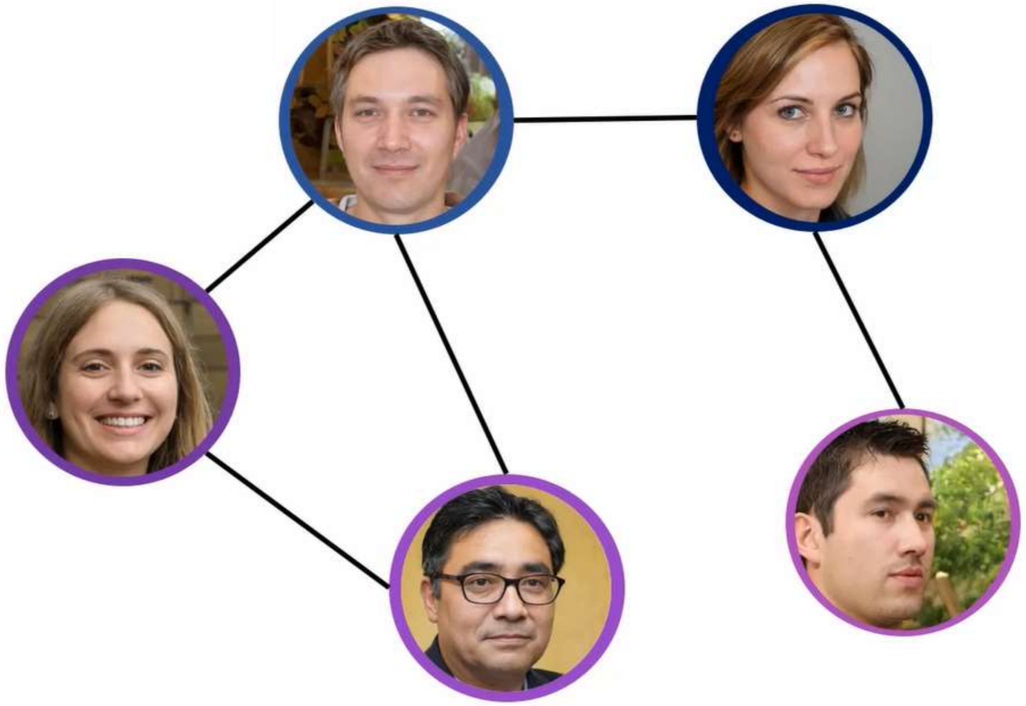
WHY ARE EDGE FEATURES ARE IMPORTANT?



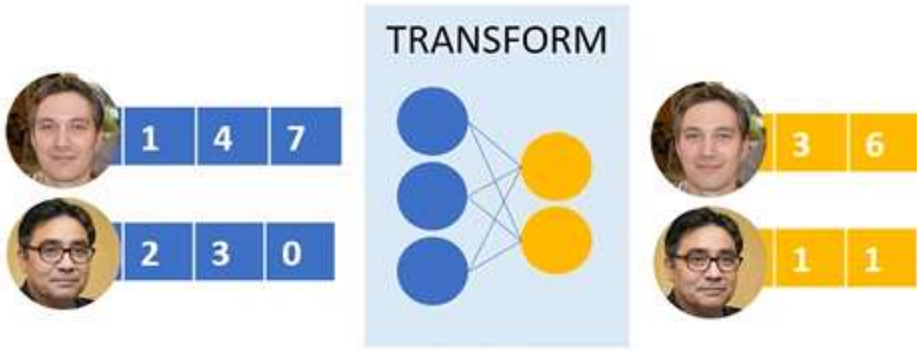


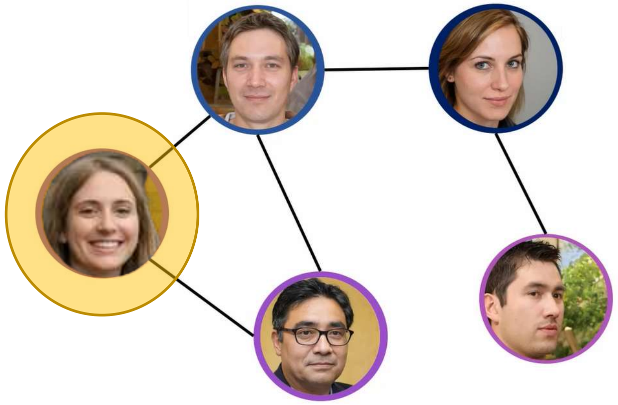
WHY ARE EDGE FEATURES ARE IMPORTANT?

	0	1	0	1	0
	1	0	1	1	0
	0	1	0	0	1
	1	1	0	0	0
	0	0	1	0	0

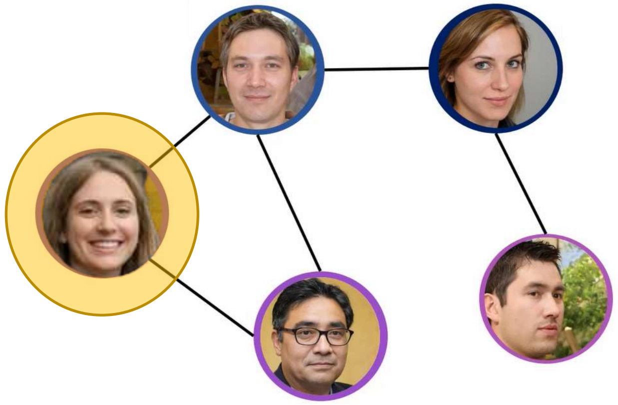


THE GENERAL PROCESS IN GNNS

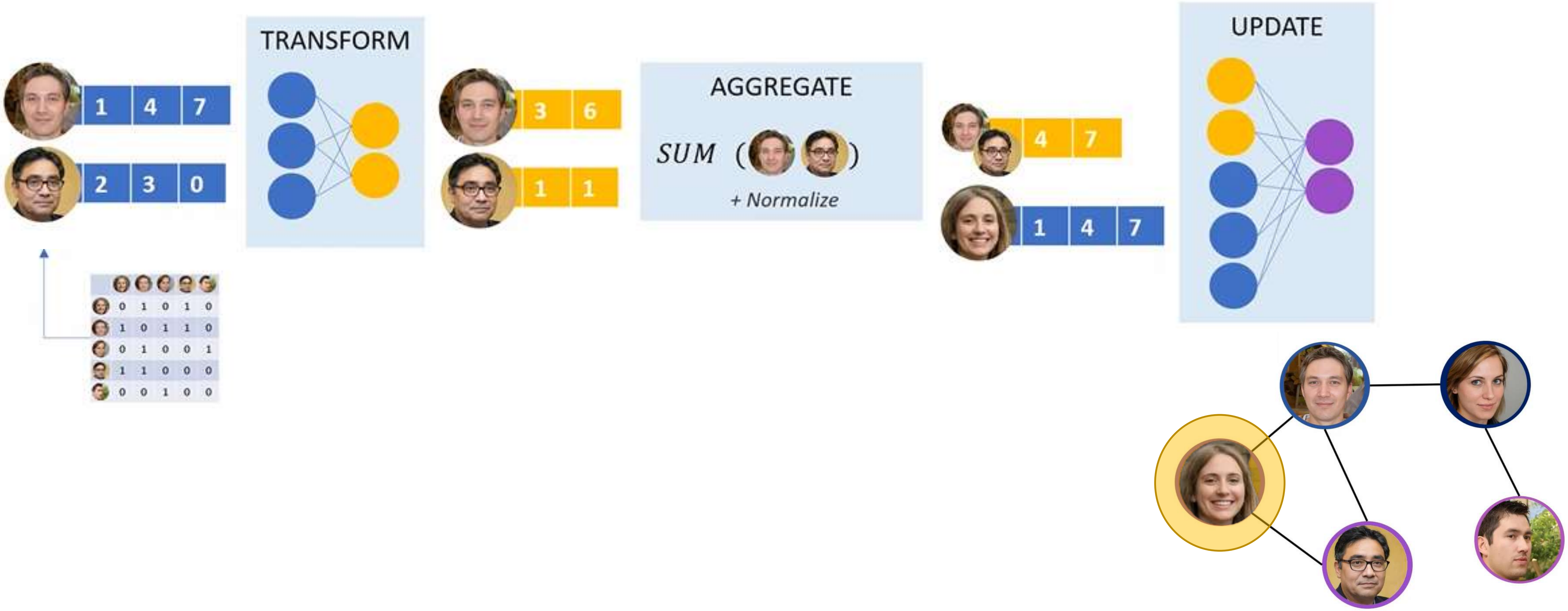




THE GENERAL PROCESS IN GNNS



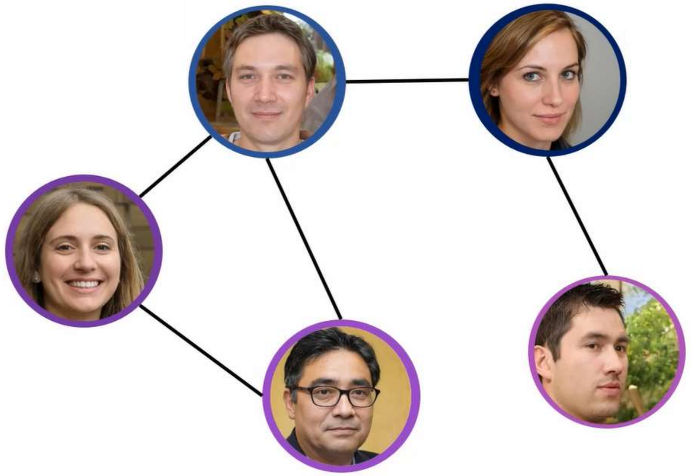
THE GENERAL PROCESS IN GNNS



USING EDGE WEIGHT

	0	1	0	1	0
	1	0	1	1	0
	0	1	0	0	1
	1	1	0	0	0
	0	0	1	0	0

	0	0.4	0	0.5	0
	0.9	0	1	1	0
	0	1	0	0	0.5
	1	0.7	0	0	0
	0	0	0.1	0	0

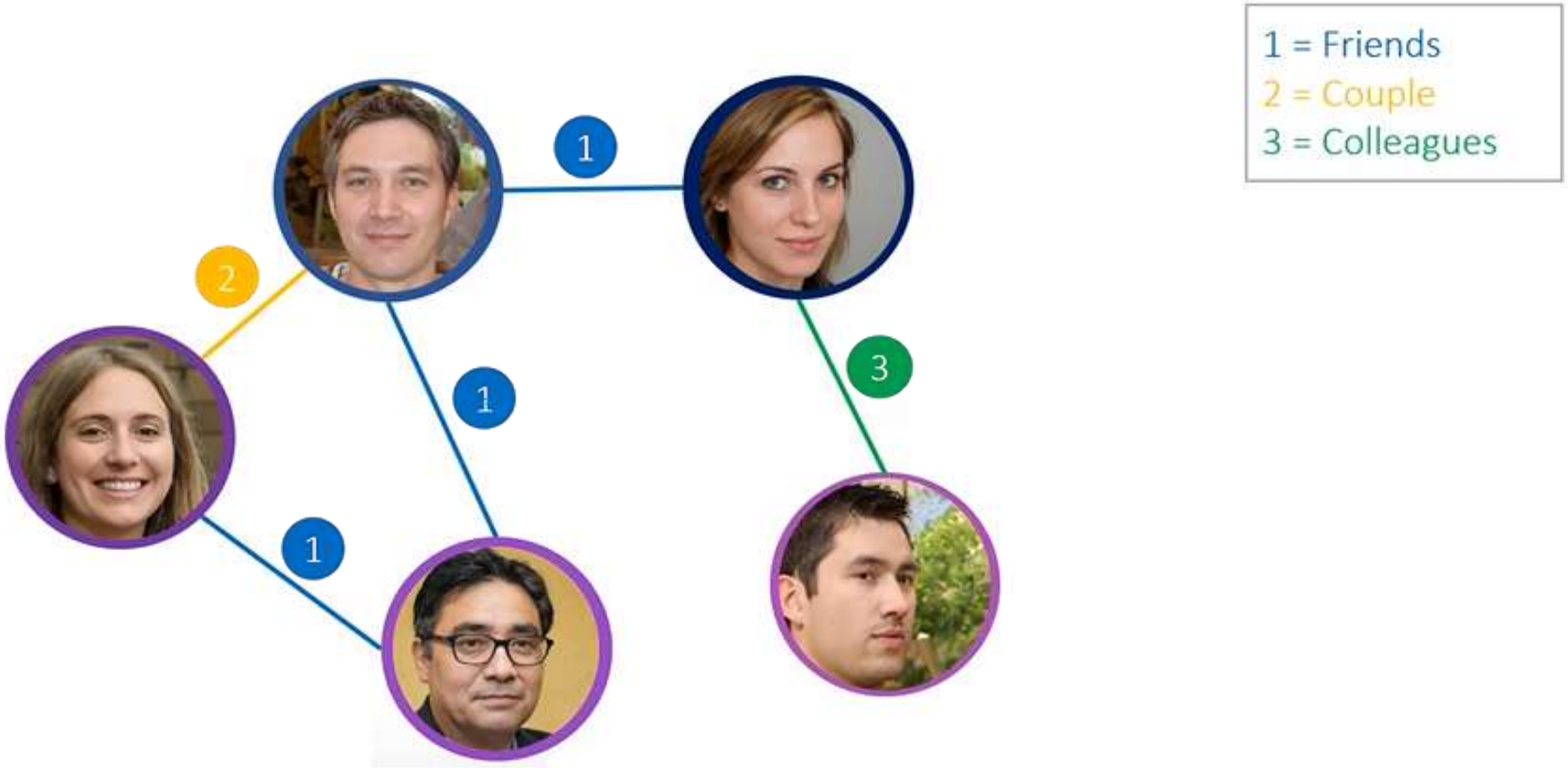


Node Features/embeddings

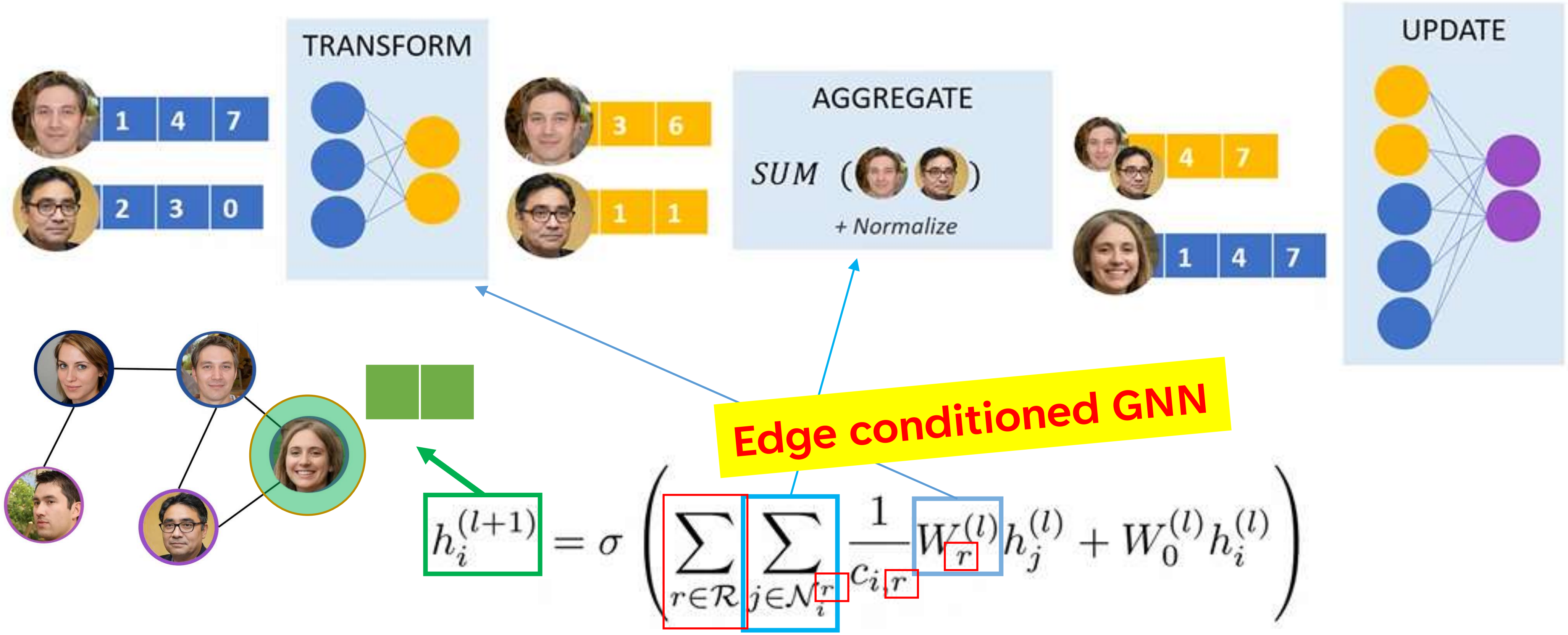
$$\mathbf{H}^{(k+1)} = \sigma\left(\tilde{\mathbf{A}}\mathbf{H}^{(k)}\mathbf{W}^{(k+1)}\right)$$

$$\tilde{\mathbf{A}} = (\mathbf{D}+\mathbf{I})^{-\frac{1}{2}}(\mathbf{I}+\mathbf{A})(\mathbf{D}+\mathbf{I})^{-\frac{1}{2}}$$

DIFFERENT EDGE TYPES



DIFFERENT EDGE TYPES – RELATIONAL GCN

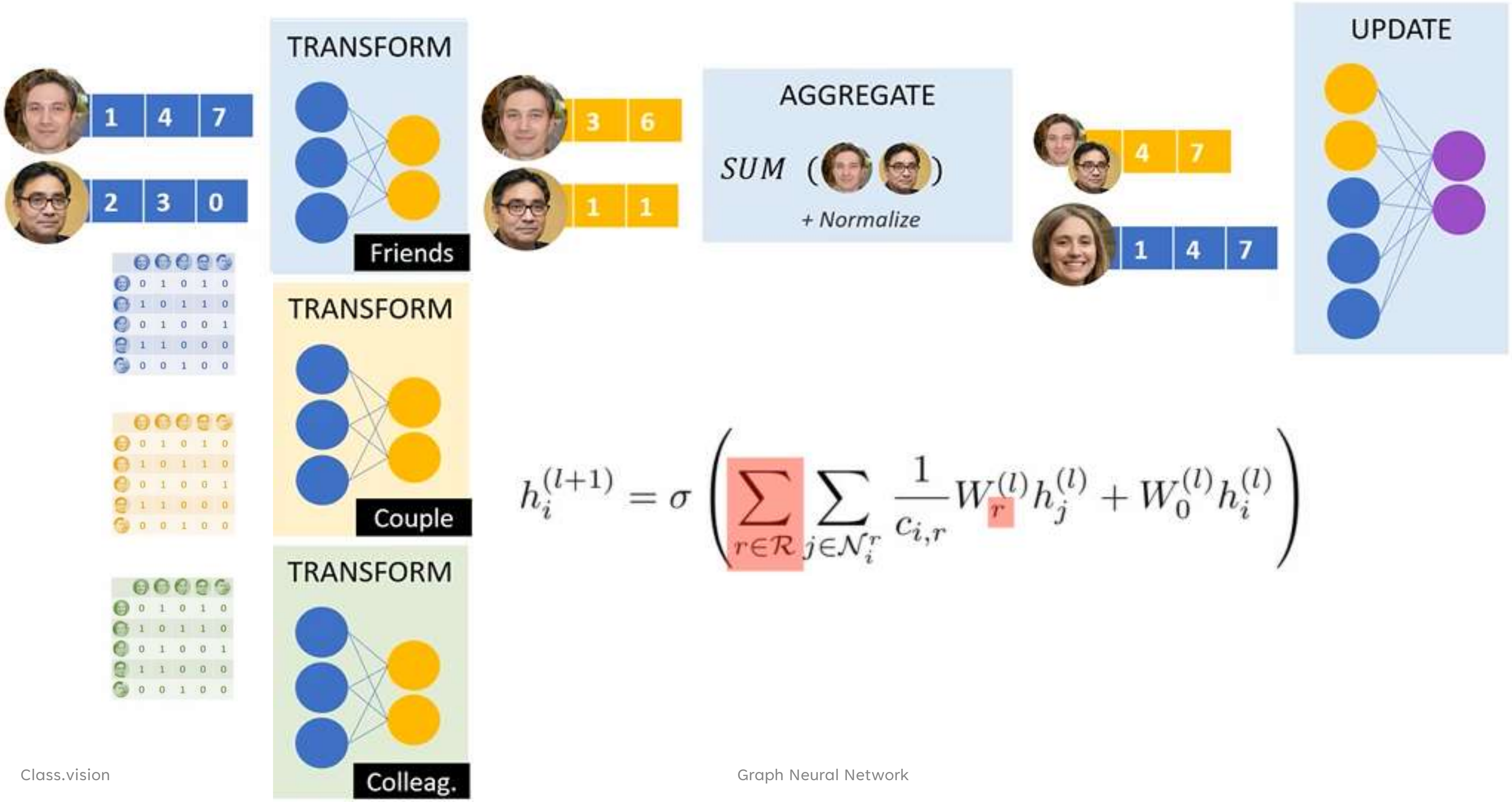


$$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right)$$

Relational GCN

Modelling Relational Data with Graph Convolutional Networks, Schlichtkrull et al.

DIFFERENT EDGE TYPES – RELATIONAL GCN



$$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right)$$

`APPNP`	The approximate personalized propagation of neural predictions layer from the "Predict then Propagate: Graph Neural Networks meet Personalized PageRank" paper
`MFConv`	The graph neural network operator from the "Convolutional Networks on Graphs for Learning Molecular Fingerprints" paper
`RGCNConv`	The relational graph convolutional operator from the "Modeling Relational Data with Graph Convolutional Networks" paper
`FastRGCNConv`	See `RGCNConv`.
`CuGraphRGCNConv`	The relational graph convolutional operator from the "Modeling Relational Data with Graph Convolutional Networks" paper.

https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html

conv.RGCNConv

class **RGCNConv** (**in_channels**: Union[int, Tuple[int, int]], **out_channels**: int, **num_relations**: int, **num_bases**: Optional[int] = None, **num_blocks**: Optional[int] = None, **aggr**: str = 'mean', **root_weight**: bool = True, **is_sorted**: bool = False, **bias**: bool = True, **kwargs) [source]

Bases: `MessagePassing`

The relational graph convolutional operator from the "Modeling Relational Data with Graph Convolutional Networks" paper

$$\mathbf{x}'_i = \mathbf{\Theta}_{\textrm{root}} \cdot \mathbf{x}_i + \sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_r(i)} \frac{1}{|\mathcal{N}_r(i)|} \mathbf{\Theta}_r \cdot \mathbf{x}_j,$$

where $\mathcal{R}$ denotes the set of relations, *i.e.* edge types. Edge type needs to be a one-dimensional `torch.long` tensor which stores a relation identifier $\in \{0, \ldots, |\mathcal{R}| - 1\}$ for each edge.

https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.RGCNConv.html#torch_geometric.nn.conv.RGCNConv

conv.FastRGCNConv

class **FastRGCNConv** (**in_channels**: Union[int, Tuple[int, int]], **out_channels**: int, **num_relations**: int, **num_bases**: Optional[int] = None, **num_blocks**: Optional[int] = None, **aggr**: str = 'mean', **root_weight**: bool = True, **is_sorted**: bool = False, **bias**: bool = True, **kwargs) [source]

Bases: `RGCNConv`

See `RGCNConv`.

forward (**x**: Union[Tensor, None, Tuple[Optional[Tensor], Tensor]], **edge_index**: Union[Tensor, SparseTensor], **edge_type**: Optional[Tensor] = None) [source]

Runs the forward pass of the module.

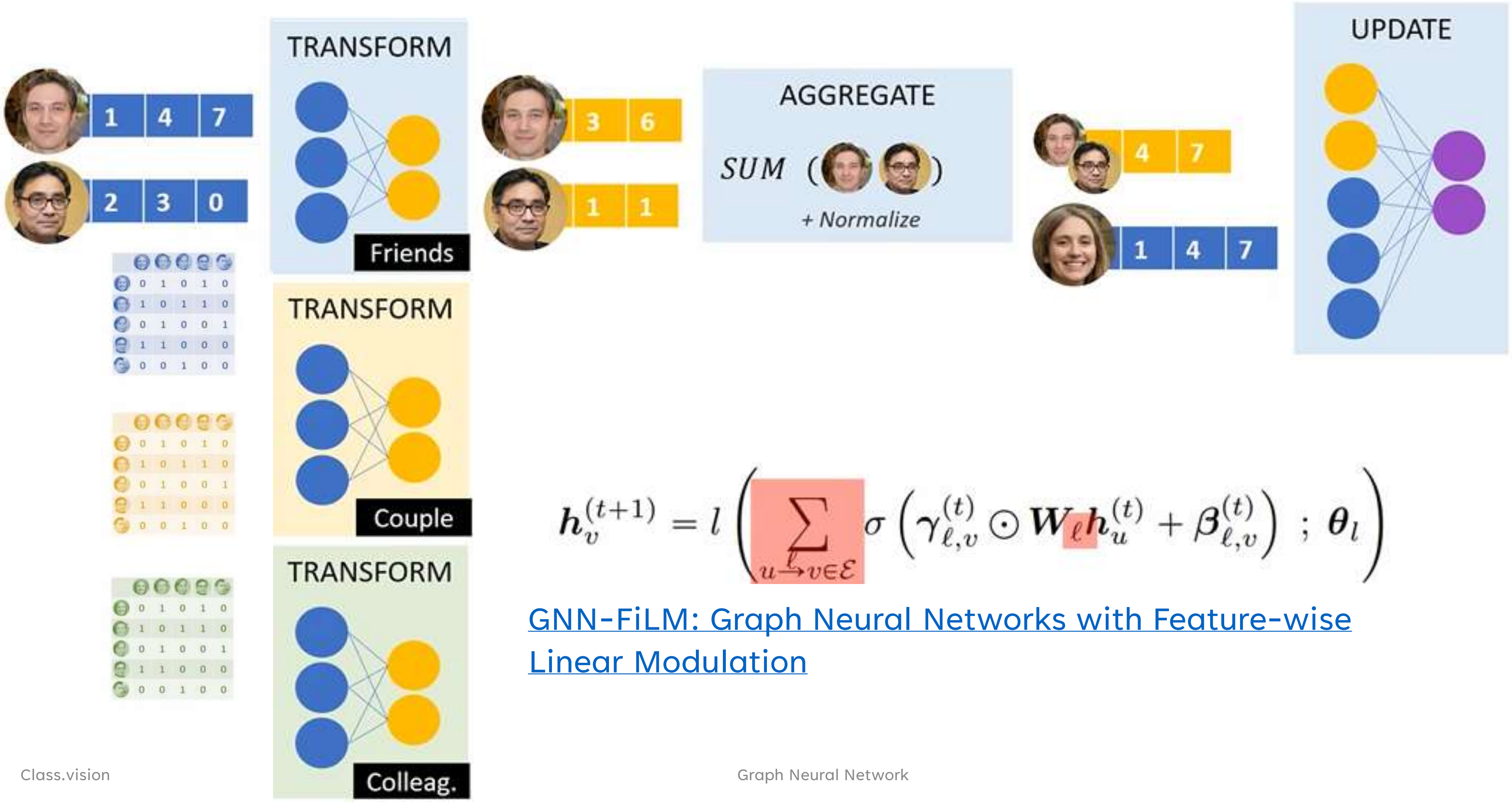
PARAMETERS

- **x** (*torch.Tensor or tuple, optional*) – The input node features. Can be either a `[num_nodes, in_channels]` node feature matrix, or an optional one-dimensional node index tensor (in which case input features

https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.FastRGCNConv.html#torch_geometric.nn.conv.FastRGCNConv

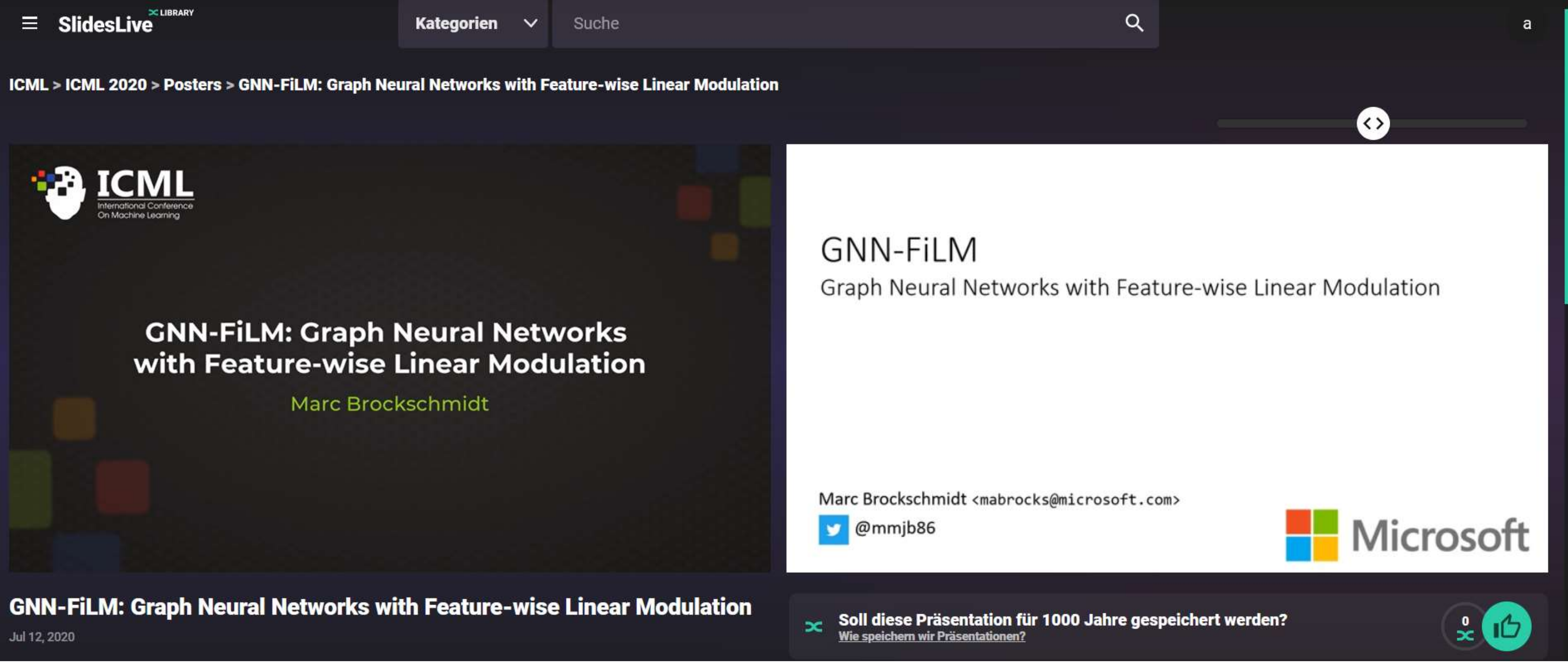
Example: https://github.com/pyg-team/pytorch_geometric/blob/master/examples/rgcn.py

DIFFERENT EDGE TYPES – GNN FILM



$$h_v^{(t+1)} = l\left(\sum_{u \xrightarrow{\ell} v \in \mathcal{E}} \sigma\left(\gamma_{\ell,v}^{(t)} \odot \mathbf{W}_\ell \mathbf{h}_u^{(t)} + \boldsymbol{\beta}_{\ell,v}^{(t)}\right) ; \boldsymbol{\theta}_l\right)$$

[GNN-FiLM: Graph Neural Networks with Feature-wise Linear Modulation]



https://slideslive.com/38927627/gnnfilm-graph-neural-networks-with-featurewise-linear-modulation?ref=recommended

`WLConv`	The Weisfeiler Lehman operator from the "A Reduction of a Graph to a Canonical Form and an Algebra Arising During this Reduction" paper, which iteratively refines node colorings:
`WLConvContinuous`	The Weisfeiler Lehman operator from the "Wasserstein Weisfeiler-Lehman Graph Kernels" paper.
`FiLMConv`	The FiLM graph convolutional operator from the "GNN-FiLM: Graph Neural Networks with Feature-wise Linear Modulation" paper
`SuperGATConv`	The self-supervised graph attentional operator from the "How to Find Your Friendly Neighborhood: Graph Attention Design with Self-Supervision" paper
`FAConv`	The Frequency Adaptive Graph Convolution operator from the "Beyond Low-Frequency Information in Graph Convolutional Networks" paper

https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html

conv.FiLMConv

class **FiLMConv** (**in_channels**: Union[int, Tuple[int, int]], **out_channels**: int, **num_relations**: int = 1, **nn**: Optional[Callable] = None, **act**: Optional[Callable] = ReLU(), **aggr**: str = 'mean', ****kwargs**) [source]

Bases: `MessagePassing`

The FiLM graph convolutional operator from the "GNN-FiLM: Graph Neural Networks with Feature-wise Linear Modulation" paper

$$\mathbf{x}'_i = \sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}(i)} \sigma \left(\boldsymbol{\gamma}_{r,i} \odot \mathbf{W}_r \mathbf{x}_j + \boldsymbol{\beta}_{r,i} \right)$$

where $\boldsymbol{\beta}_{r,i}, \boldsymbol{\gamma}_{r,i} = g(\mathbf{x}_i)$ with g being a single linear layer by default. Self-loops are automatically added to the input graph and represented as its own relation type.

https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.FiLMConv.html#torch_geometric.nn.conv.FiLMConv

DIFFERENT EDGE TYPES- OTHER VARIANTS

$$\text{GGNN: } A' = GRU(A,\ \boldsymbol{W}_1 \cdot B + \boldsymbol{W}_2 \cdot C + \boldsymbol{W}_1 \cdot D)$$

$$\text{R-GCN: } A' = \sigma(\boldsymbol{W}_{\circlearrowleft} \cdot A + \boldsymbol{W}_1 \cdot B + \boldsymbol{W}_2 \cdot C + \boldsymbol{W}_1 \cdot D)$$

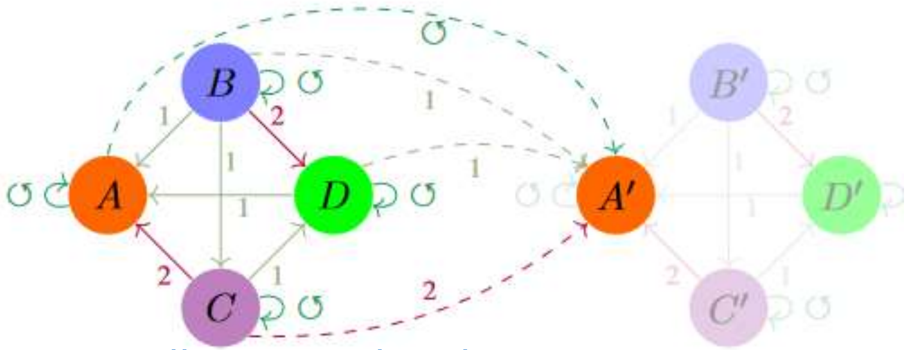
$$\text{R-GAT: } A' = \sigma((\boldsymbol{a}_{A'})_{A \overset{\circlearrowleft}{\to} A} \cdot \boldsymbol{W}_{\circlearrowleft} \cdot A + (\boldsymbol{a}_{A'})_{B \overset{1}{\to} A} \cdot \boldsymbol{W}_1 \cdot B + (\boldsymbol{a}_{A'})_{C \overset{2}{\to} A} \cdot \boldsymbol{W}_2 \cdot C + (\boldsymbol{a}_{A'})_{D \overset{1}{\to} A} \cdot \boldsymbol{W}_1 \cdot D)$$

$$\text{R-GIN: } A' = \sigma(MLP_{\circlearrowleft}(A) + MLP_1(B) + MLP_2(C) + MLP_1(D))$$

$$\text{GNN-MLP: } A' = \sigma(MLP_{\circlearrowleft}(A \| A) + MLP_1(B \| A) + MLP_2(C \| A) + MLP_1(D \| A))$$

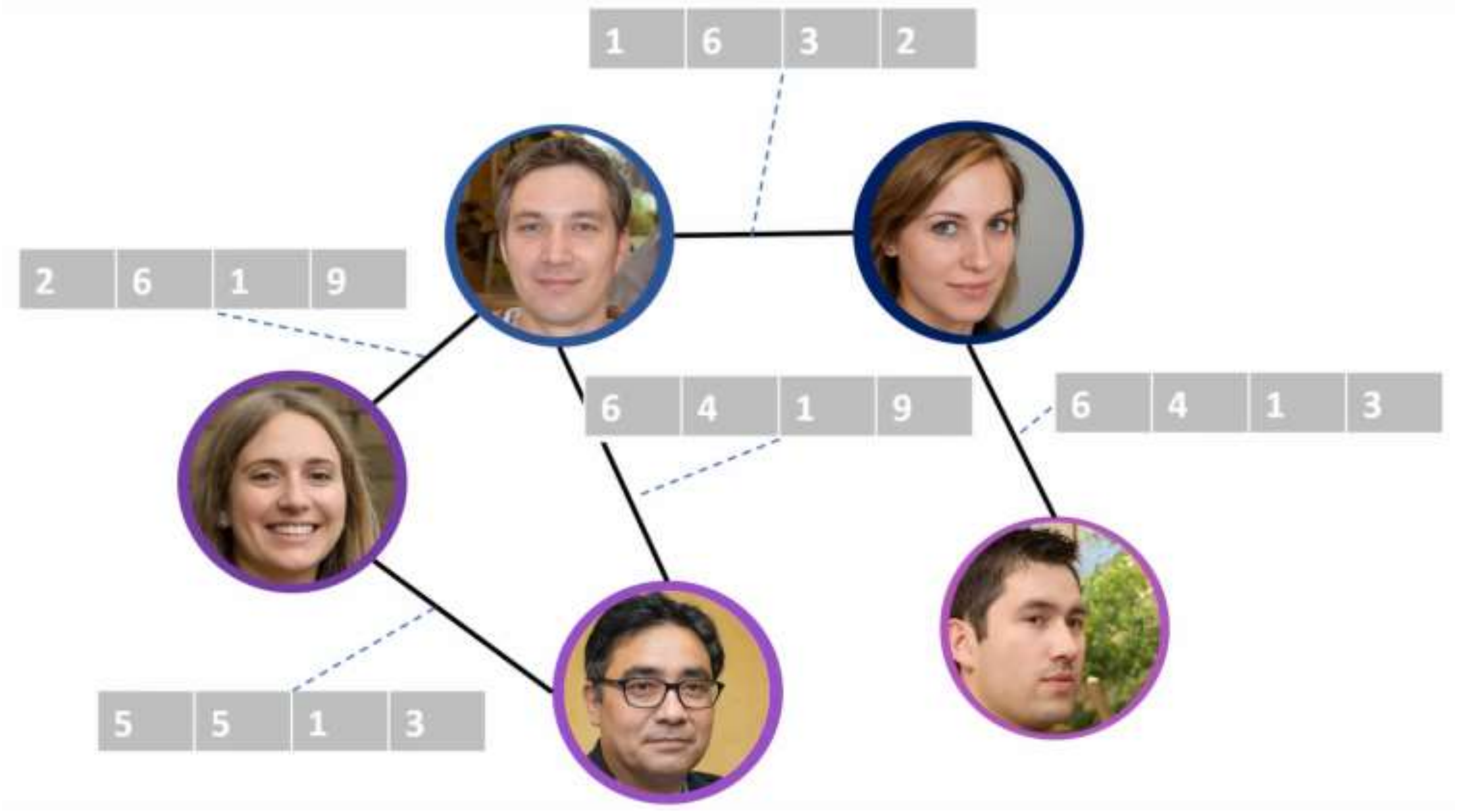
$$\text{RGDCN: } A' = \sigma(\boldsymbol{W}_{\circlearrowleft,A} \cdot A + \boldsymbol{W}_{1,A} \cdot B + \boldsymbol{W}_{2,A} \cdot C + \boldsymbol{W}_{1,A} \cdot D)$$

$$\text{GNN-FiLM: } A' = \sigma(\boldsymbol{\beta}_{\circlearrowleft,A} + \boldsymbol{\gamma}_{\circlearrowleft,A} \odot \boldsymbol{W}_{\circlearrowleft} \cdot A + \boldsymbol{\beta}_{1,A} + \boldsymbol{\gamma}_{1,A} \odot \boldsymbol{W}_1 \cdot B + \boldsymbol{\beta}_{2,A} + \boldsymbol{\gamma}_{2,A} \odot \boldsymbol{W}_2 \cdot C + \boldsymbol{\beta}_{1,A} + \boldsymbol{\gamma}_{1,A} \odot \boldsymbol{W}_1 \cdot D)$$



https://arxiv.org/pdf/1906.12192.pdf

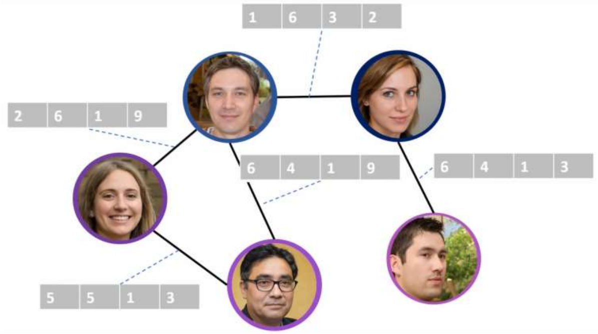
MULTIDIMENSIONAL EDGE FEATURES



MULTIDIMENSIONAL EDGE FEATURES

$$h_v = \gamma \left(x_v, \bigoplus_{w \in N(v)} \phi(x_v, x_w, e_{wv}) \right)$$

UPDATE — AGGREGATE — TRANSFORM



MULTIDIMENSIONAL EDGE FEATURES: MP-GNN

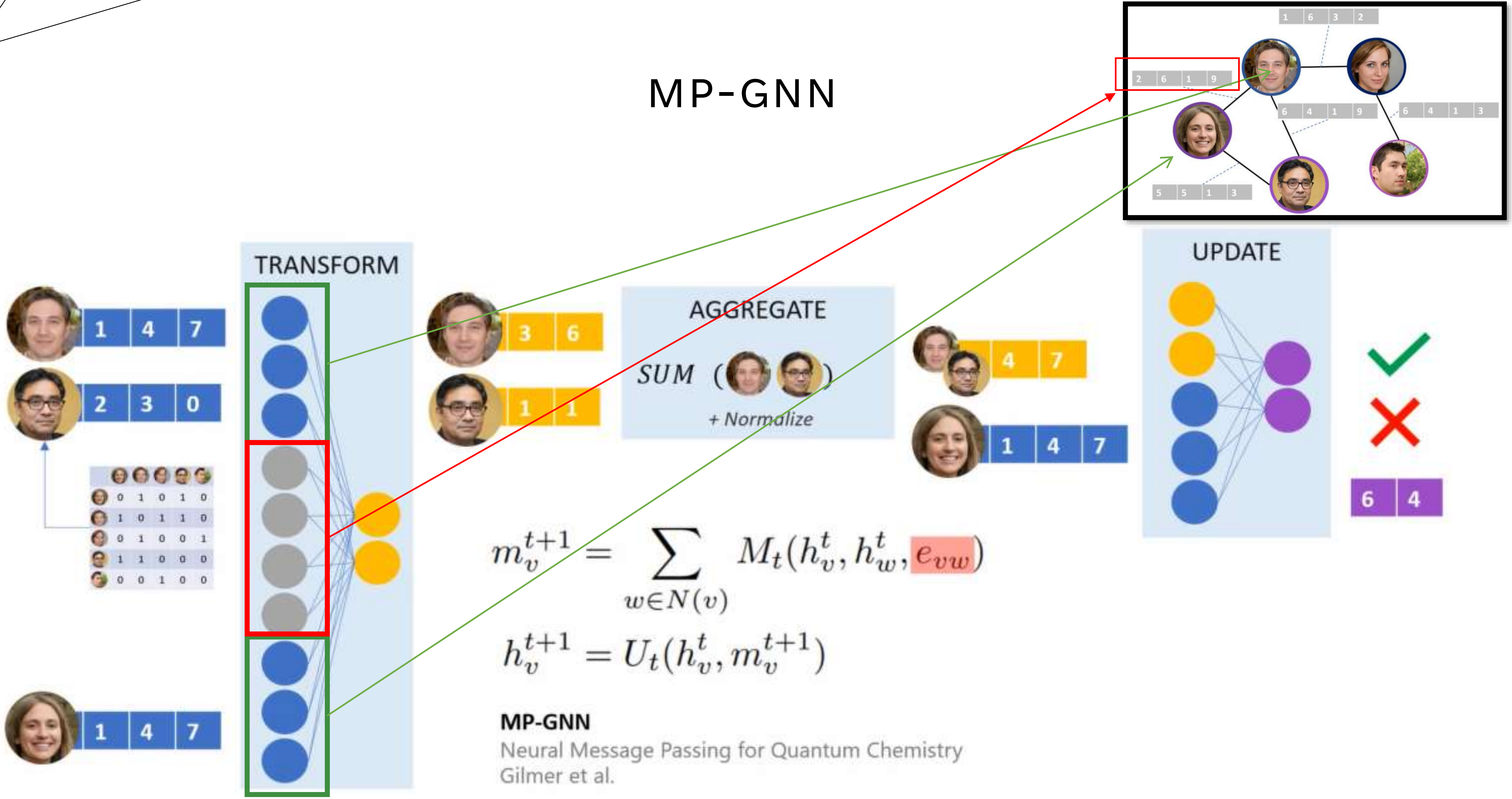
$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

MP-GNN

Neural Message Passing for Quantum Chemistry
Gilmer et al.

MP-GNN



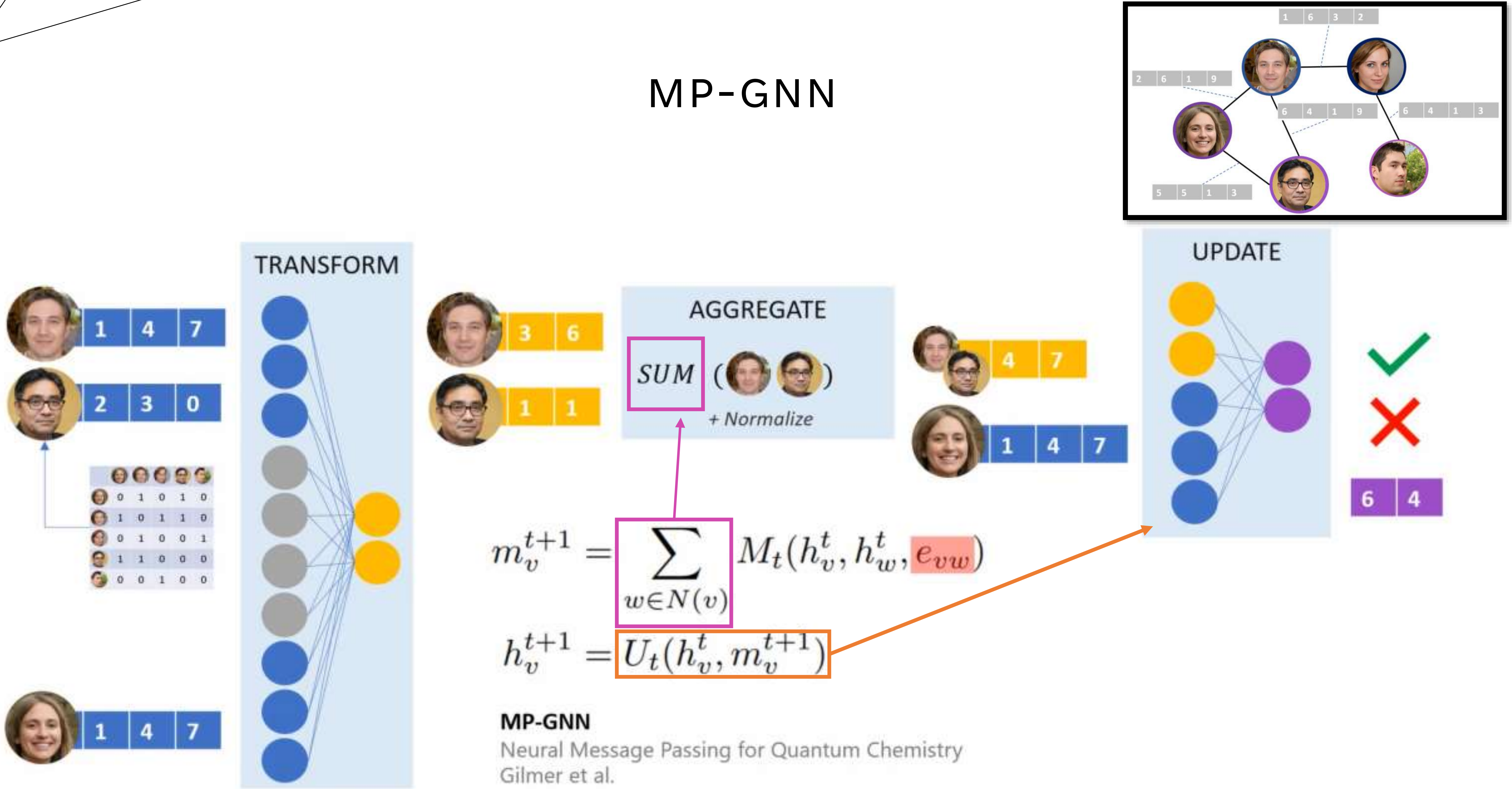
$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

MP-GNN

Neural Message Passing for Quantum Chemistry
Gilmer et al.

MP-GNN



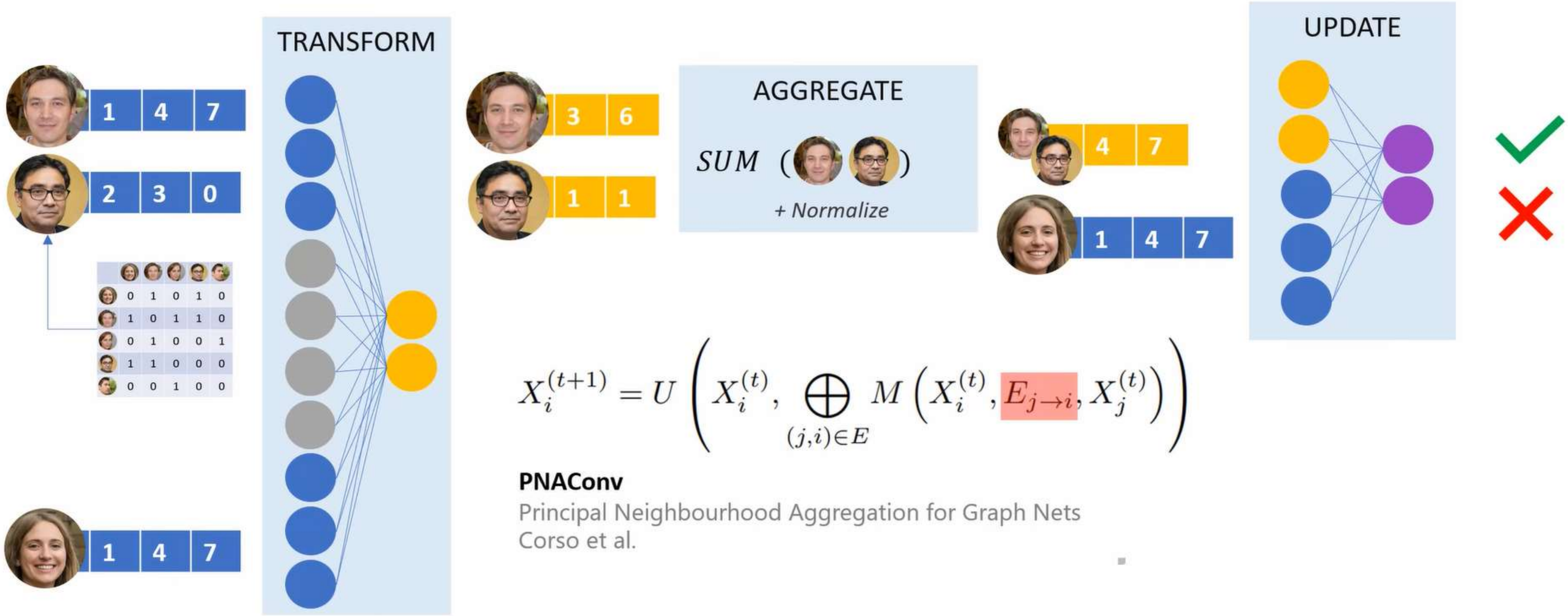
$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

MP-GNN

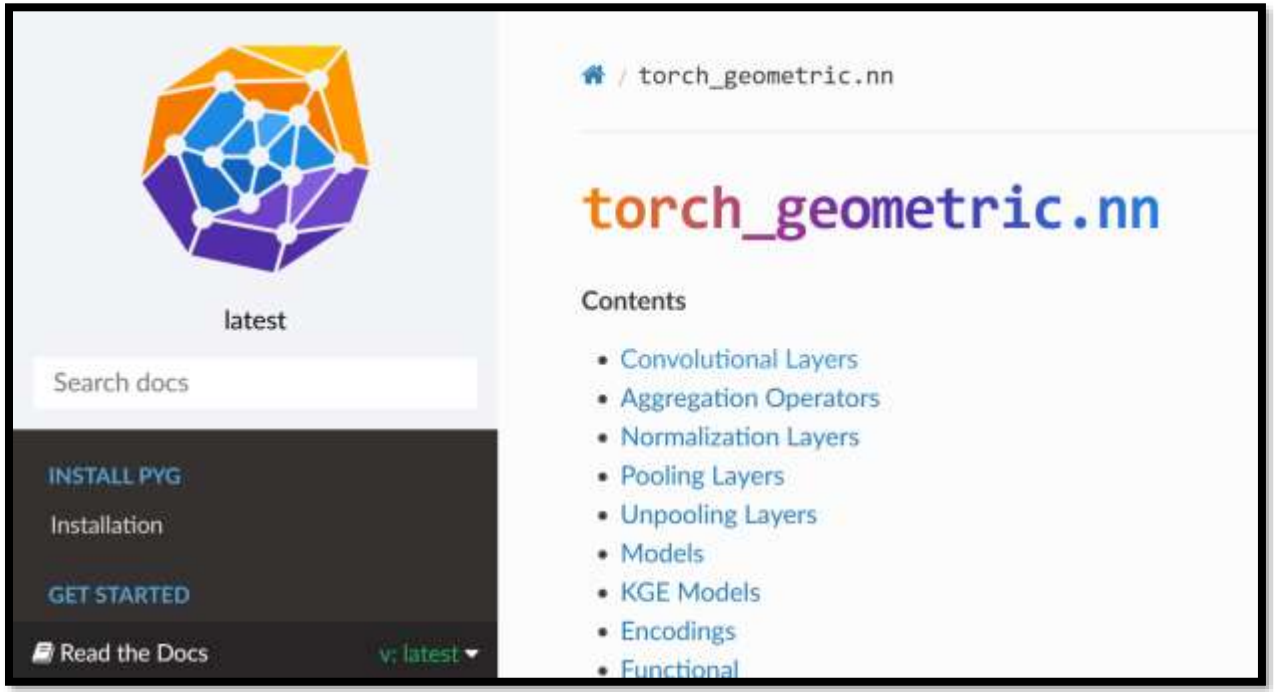
Neural Message Passing for Quantum Chemistry
Gilmer et al.

MULTIDIMENSIONAL EDGE FEATURES: PNACONV



MULTIDIMENSIONAL EDGE FEATURES
OTHER EXAMPLES

USING EDGE FEATURES IN PYTORCH GEOMETRIC



USING EDGE FEATURES IN PYTORCH GEOMETRIC

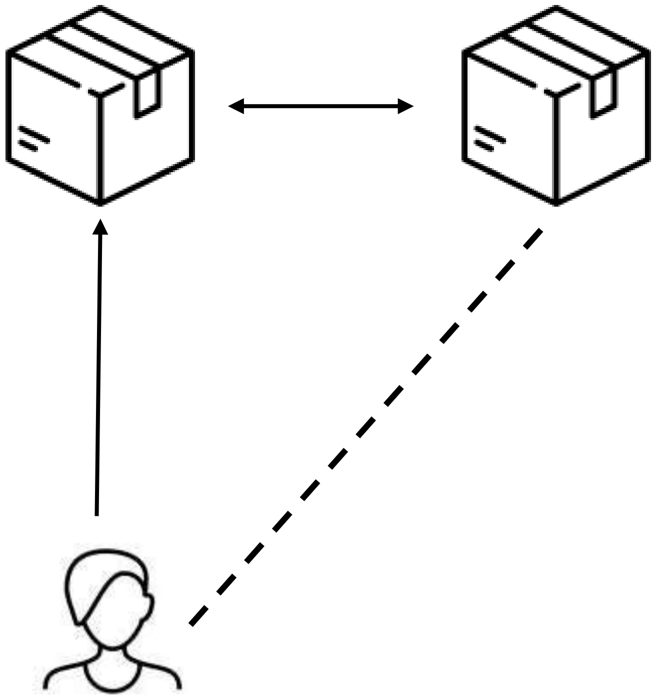
PyTorch geometric

- **edge_weight** → GNN Layer can use weight values on the adjacency matrix
- **edge_type** → GNN Layer can use different edge types / relations
- **edge_attr** → GNN Layer can use edge features

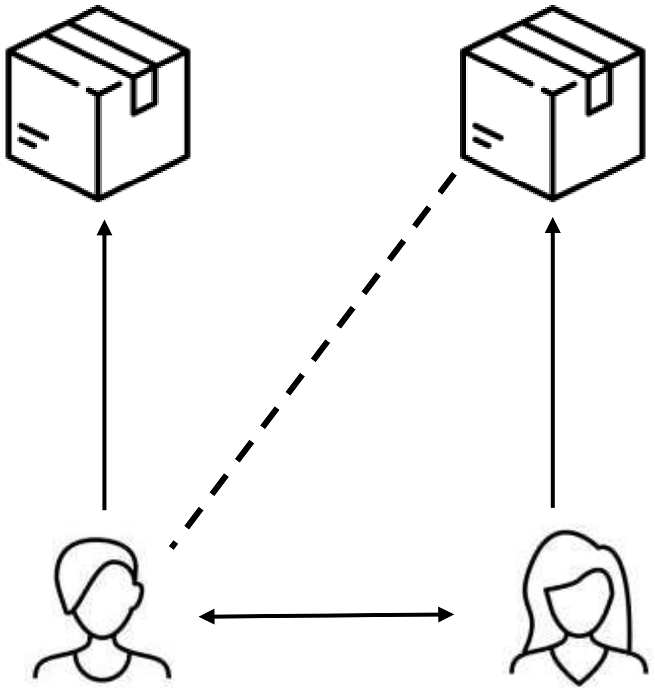
```
forward ( x: Union[Tensor, None, Tuple[Optional[Tensor], Tensor]], edge_index: Union[Tensor, SparseTensor], edge_type: Optional[Tensor] = None )    [source]
```

LINK PREDICTION AND GRAPH AUTOENCODER

WHAT IS A RECOMMENDER SYSTEM?

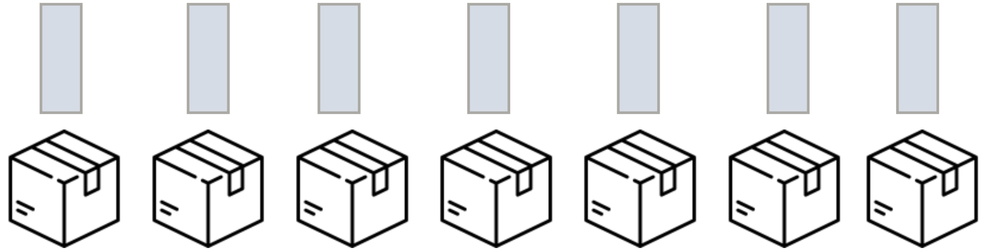


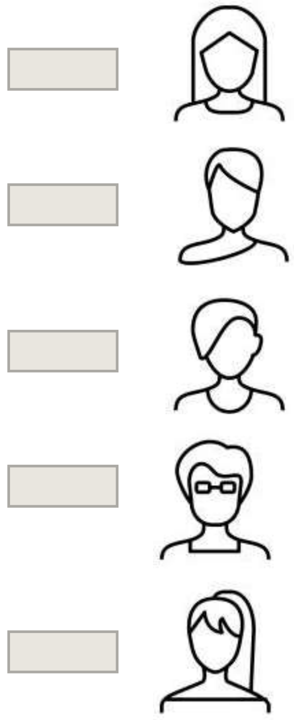
Content-based filtering



Collaborative filtering

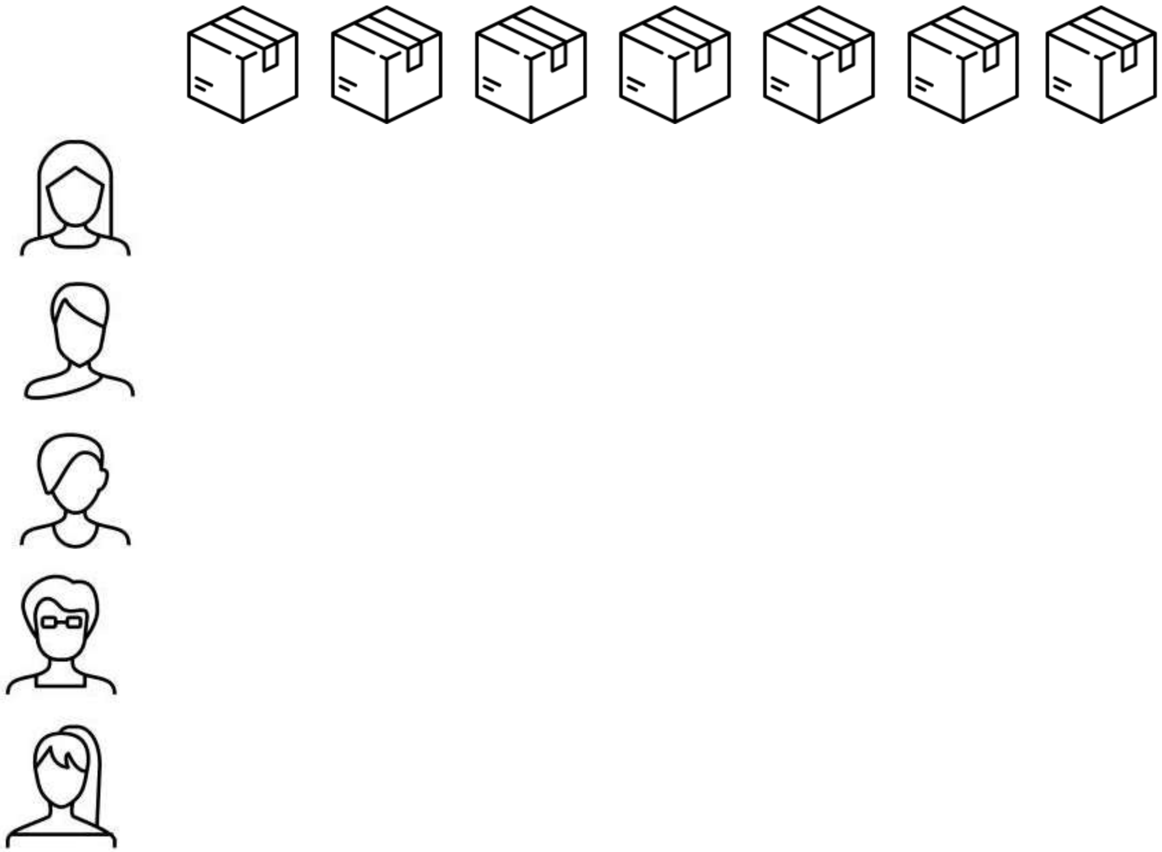
COLLABORATIVE FILTERING



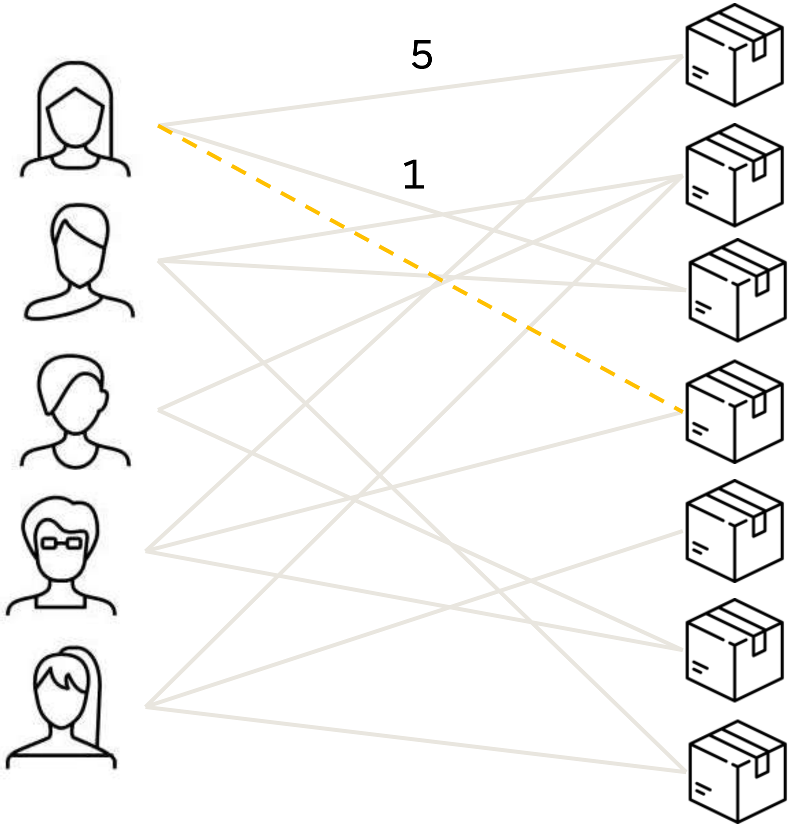


5		1				
	4		5			1
	3				5	
1			4		5	
	2			4		1

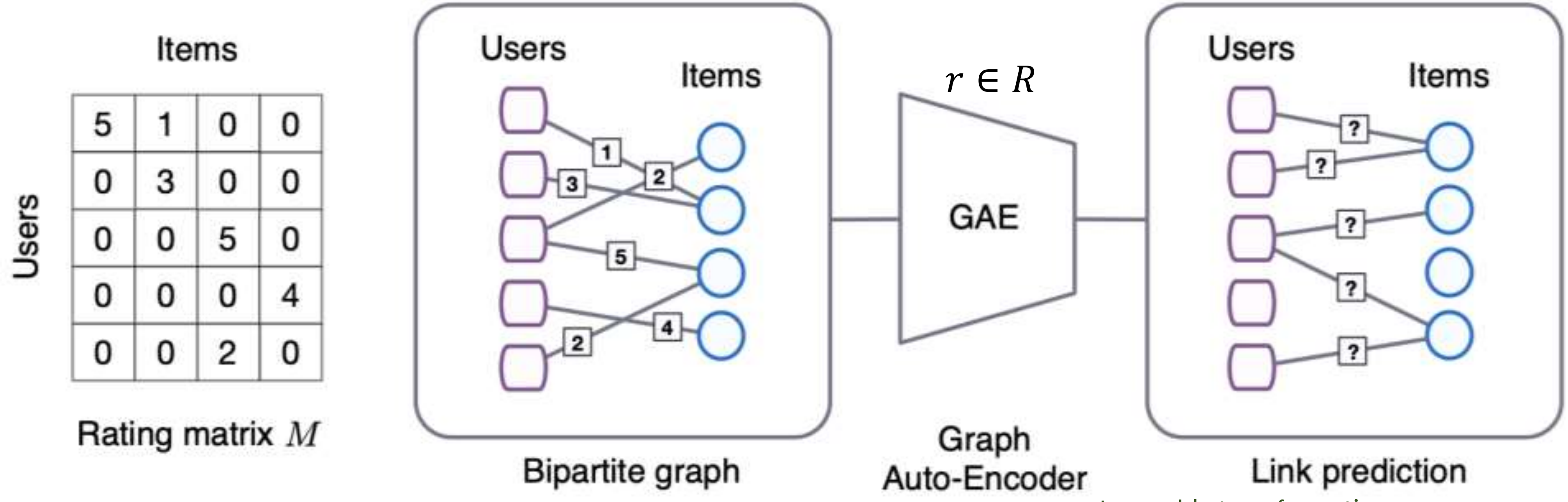
BIPARTITE GRAPH



BIPARTITE GRAPH



GRAPH CONVOLUTIONAL MATRIX COMPLETION



Learnable transformation

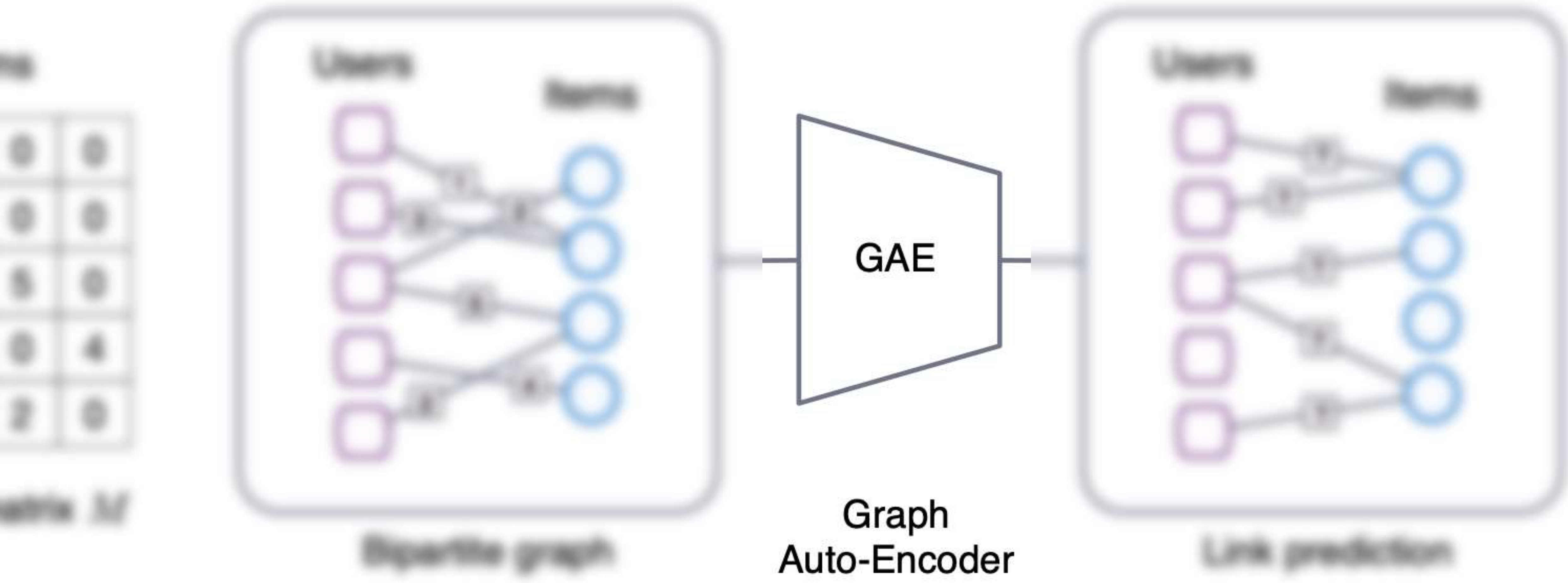
users

items

SoftMax
Which edge type

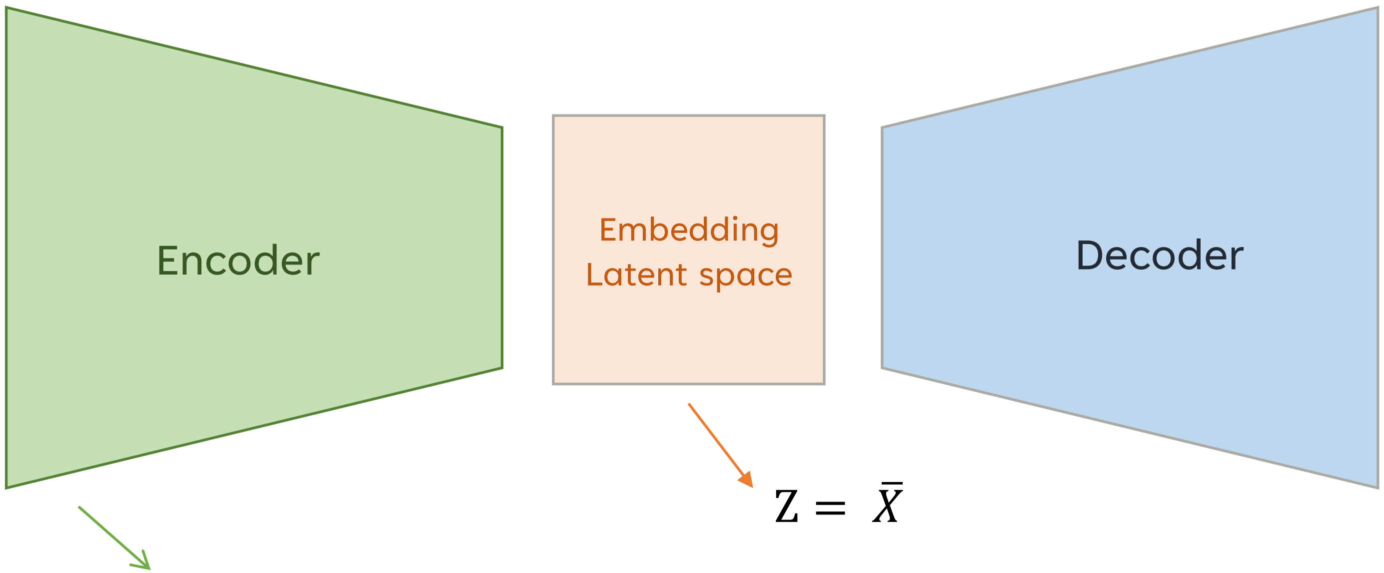
$$p(\check{M}_{ij} = r) = \frac{e^{u_i^T Q_r v_j}}{\sum_{s \in R} e^{u_i^T Q_s v_j}}$$

Graph Convolutional Matrix Completion
Rianne van den Berg, Thomas N. Kipf, Max Welling 2017



Graph Convolutional Matrix Completion
Rianne van den Berg, Thomas N. Kipf, Max Welling 2017

GRAPH AUTOENCODERS (GAE)



$Z = \bar{X}$

A graph convolutional Neural Network produces a low dimensional embedding representation

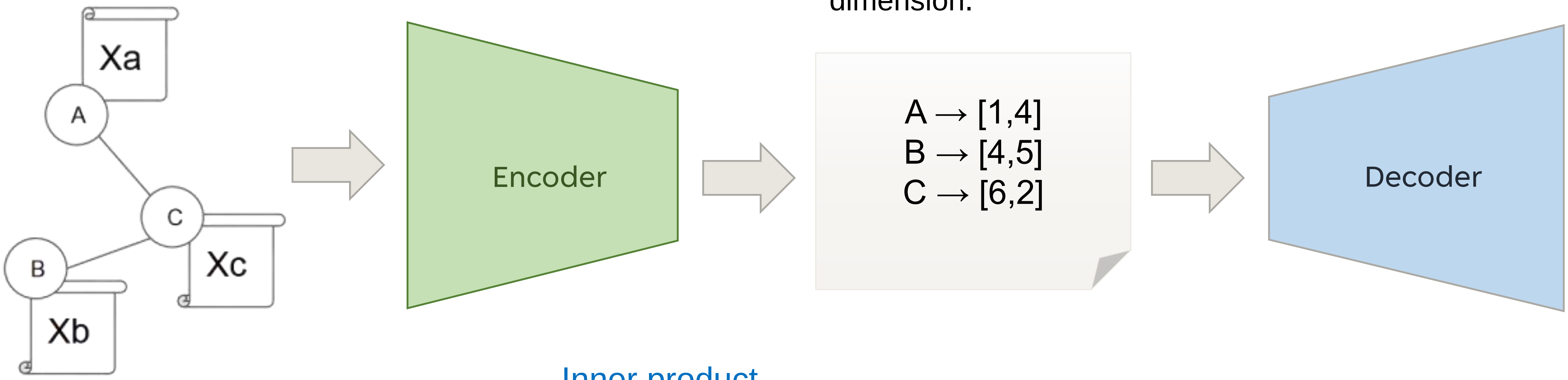
$$\bar{X} = GCN(A, X) = ReLU\left(\tilde{A} X W_0\right)$$

With $\tilde{A} = D^{-1/2} A D^{-1/2}$

GRAPH AUTOENCODERS (GAE)

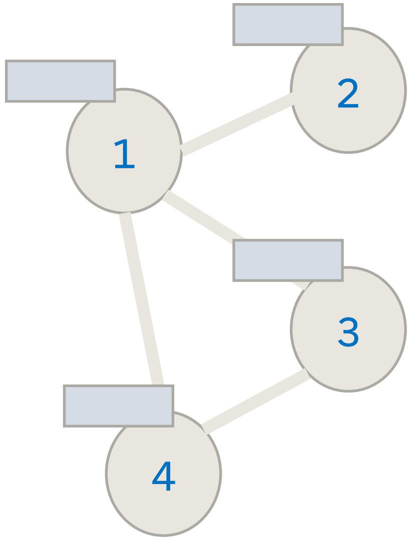
Node embedding in a latent space with two dimension.

Reconstruct
The input graph



Inner product
Between latent variable Z

WHY INNER PRODUCT?



1	2.4	8.1	0.3
2	0.7	0.6	0.2
3	0.3	9.2	1.2
4	2.1	1.8	0.8

Z

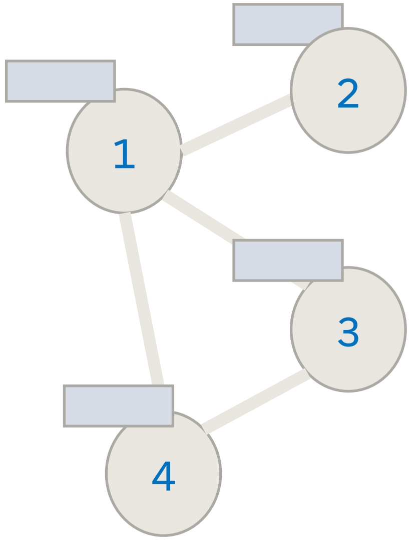
1	2	3	4
2.4	0.7	0.3	2.1
8.1	0.6	9.2	1.8
0.3	0.2	1.2	0.8

Z^T

$$\hat{\mathbf{A}} = \sigma\left(\mathbf{Z}\mathbf{Z}^\top\right), \quad \text{with} \quad \mathbf{Z} = \mathrm{GCN}(\mathbf{X}, \mathbf{A})$$

Variational Graph Auto-Encoders, 2016
https://arxiv.org/abs/1611.07308

WHY INNER PRODUCT?



1	2.4	8.1	0.3
2	0.7	0.6	0.2
3	0.3	9.2	1.2
4	2.1	1.8	0.8

Z 4×3

1	2	3	4
2.4	0.7	0.3	2.1
8.1	0.6	9.2	1.8
0.3	0.2	1.2	0.8

Z^T 3×4

	1	2	3	4
1	?	?	?	?
2	?	?	?	?
3	?	?	?	?
4	?	?	?	?

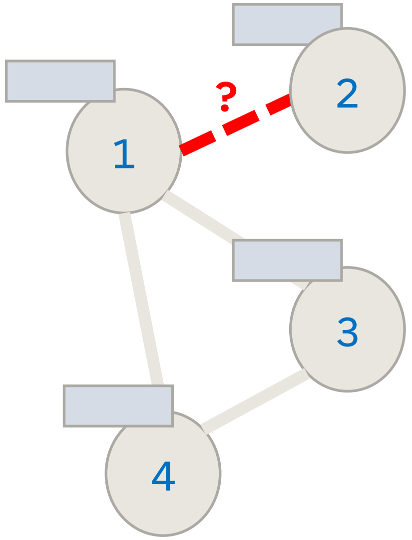
Adjacency

4×4

$$\hat{\mathbf{A}} = \sigma(\mathbf{Z}\mathbf{Z}^\top), \quad \text{with} \quad \mathbf{Z} = \mathrm{GCN}(\mathbf{X}, \mathbf{A})$$

Variational Graph Auto-Encoders, 2016
https://arxiv.org/abs/1611.07308

WHY INNER PRODUCT?



1	2.4	8.1	0.3
2	0.7	0.6	0.2
3	0.3	9.2	1.2
4	2.1	1.8	0.8

Z 4×3

1	2	3	4
2.4	0.7	0.3	2.1
8.1	0.6	9.2	1.8
0.3	0.2	1.2	0.8

Z^T 3×4

	1	2	3	4
1	?	?	?	?
2	?	?	?	?
3	?	?	?	?
4	?	?	?	?

Adjacency

4×4

$$\hat{\mathbf{A}} = \sigma\left(\mathbf{Z}\mathbf{Z}^\top\right), \quad \text{with} \quad \mathbf{Z} = \mathrm{GCN}(\mathbf{X}, \mathbf{A})$$

Variational Graph Auto-Encoders, 2016
https://arxiv.org/abs/1611.07308

HETEROGENEOUS & KNOWLEDGE GRAPH EMBEDDING

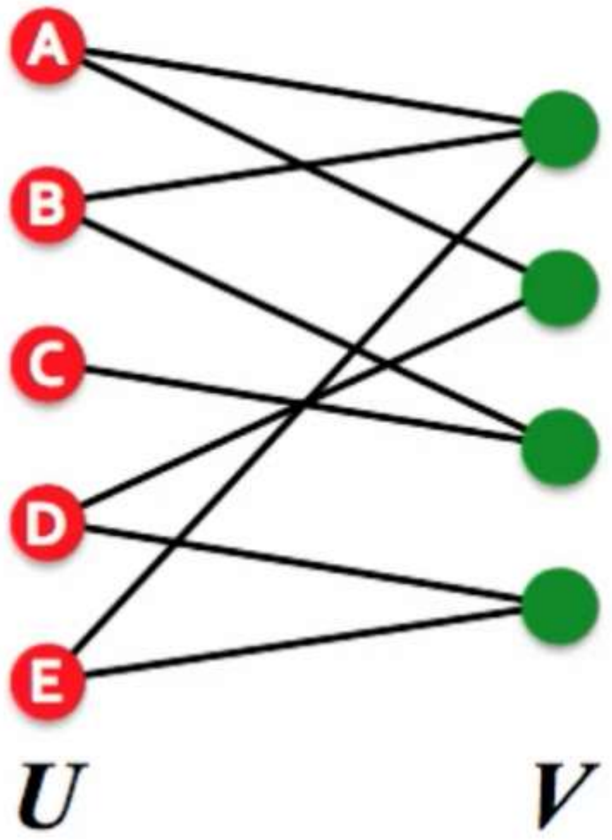
HETEROGENEOUS GRAPHS

- A heterogeneous graph is defined as

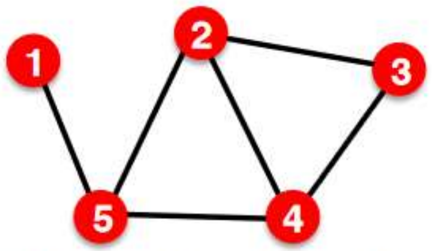
$$G = (V, E, R, T)$$

  - Nodes with node types $v_i \in V$
  - Edges with relation types $(v_i, \mathrm{r}, v_j) \in E$
  - Node type $T(v_i)$
  - Relation type $r \in R$

BIPARTITE GRAPH

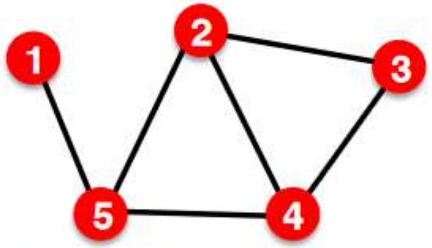


SETTING UP LINK PREDICTION

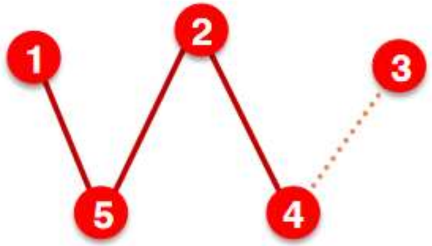


The original graph

SETTING UP LINK PREDICTION

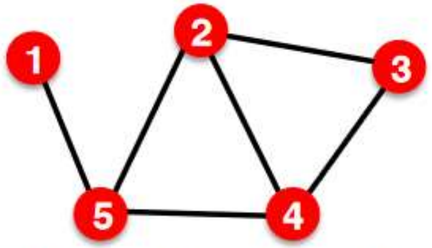


The original graph

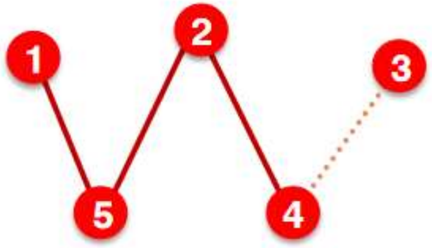


(1) At training time:
Use **training message edges** to predict **training supervision edges**

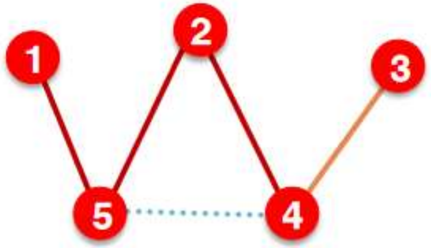
SETTING UP LINK PREDICTION



The original graph

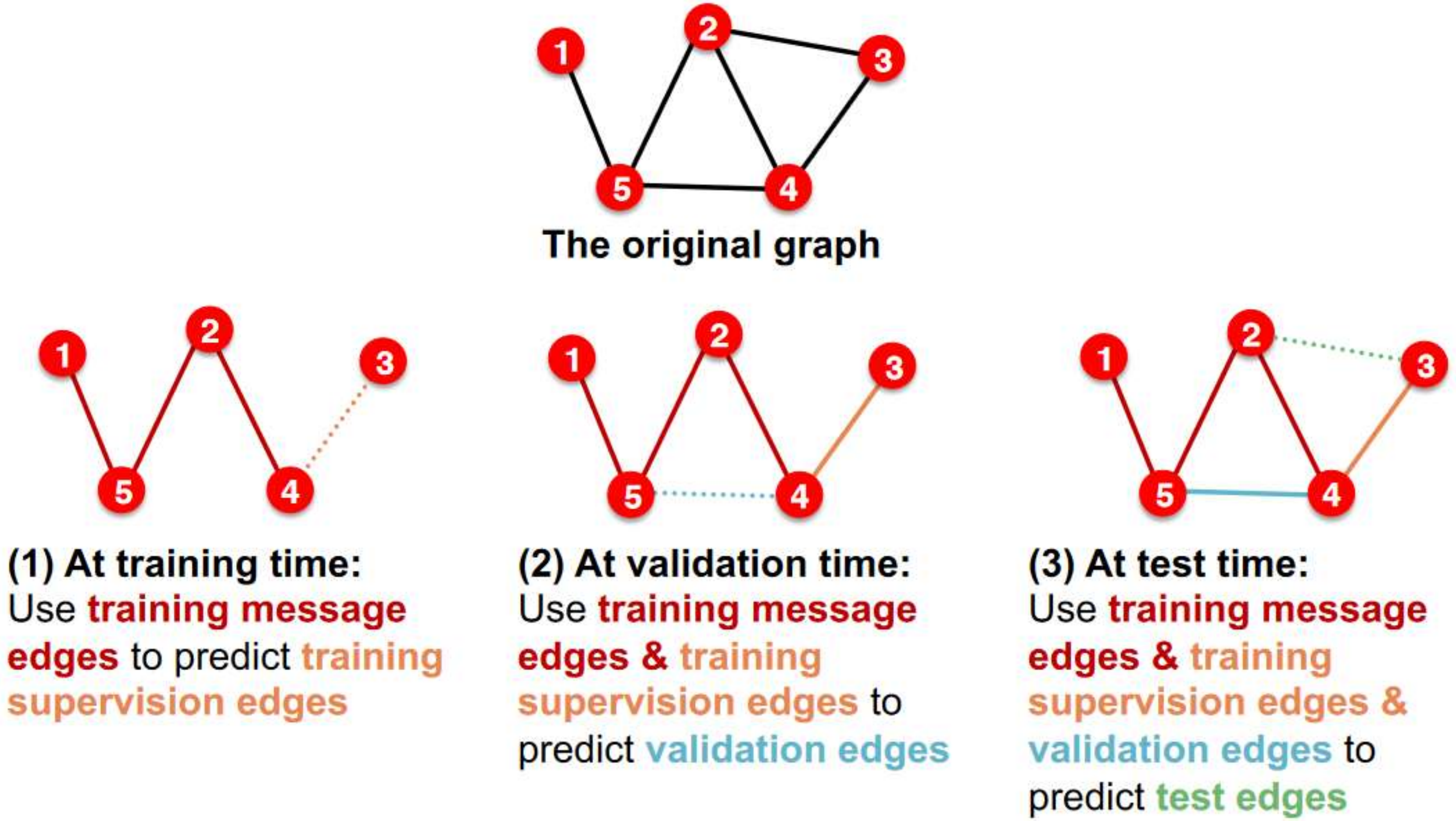


(1) At training time:
Use **training message edges** to predict **training supervision edges**



(2) At validation time:
Use **training message edges & training supervision edges** to predict **validation edges**

SETTING UP LINK PREDICTION



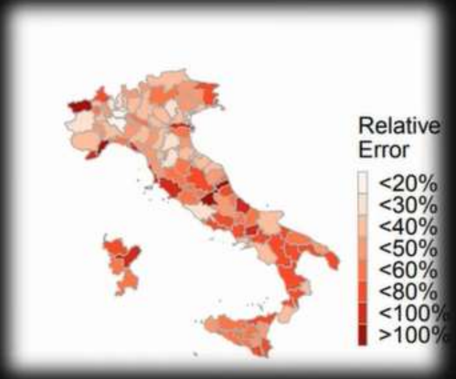
SPATIO-TEMPORAL GRAPH NEURAL NETWORKS

Source: DCRNN paper

Traffic Forecasting

Source: DCRNN paper

Traffic Forecasting

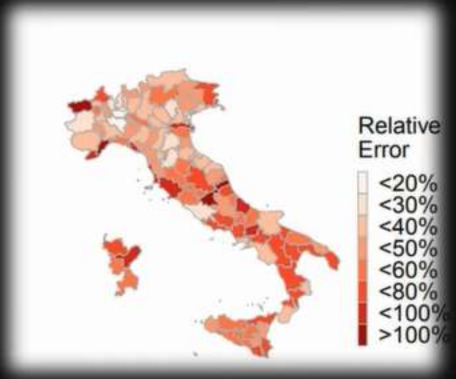


Source: Transfer GNN for Pandemic forecasting

Epidemics (Covid Predictions)

Source: DCRNN paper

Traffic Forecasting



Source: Transfer GNN for Pandemic forecasting

Epidemics (Covid Predictions)

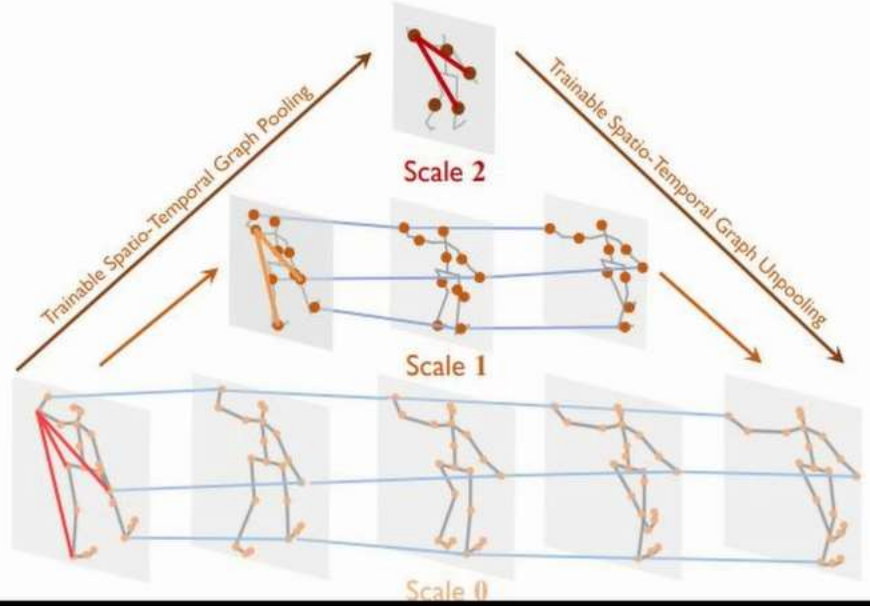
Source: mediapipe

Motion Classification

Multiscale Spatio-Temporal Graph Neural Networks for 3D Skeleton-Based Motion Prediction

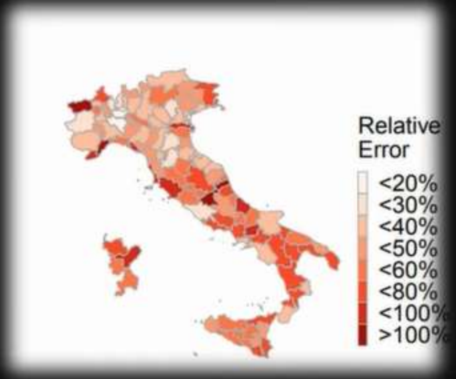
Maosen Li, *Student Member, IEEE,* Siheng Chen, *Member, IEEE,* Yangheng Zhao, Ya Zhang, *Member, IEEE,* Yanfeng Wang, and Qi Tian, *Fellow, IEEE*

***Abstract*—We propose a multiscale spatio-temporal graph neural network (MST-GNN) to predict the future 3D skeleton-based human poses in an action-category-agnostic manner. The core of MST-GNN is a multiscale spatio-temporal graph that explicitly models the relations in motions at various spatial and temporal scales. Different from many previous hierarchical structures, our multiscale spatio-temporal graph is built in a *data-adaptive fashion*, which captures nonphysical, yet motion-based relations. The key module of MST-GNN is a multiscale spatio-temporal graph computational unit (MST-GCU) based on the trainable graph structure. MST-GCU embeds underlying features at individual scales and then fuses features across scales to obtain a comprehensive representation. The overall architecture of MST-GNN follows an encoder-decoder framework, where the encoder consists of a sequence of MST-GCUs to learn the spatial and**



Source: DCRNN paper

Traffic Forecasting



Source: Transfer GNN for Pandemic forecasting

Epidemics (Covid Predictions)

Source: mediapipe

Motion Classification

Source: GCLSTM, Simeunovic et al.

Power Systems Forecasting

TIME VARYING GRAPH

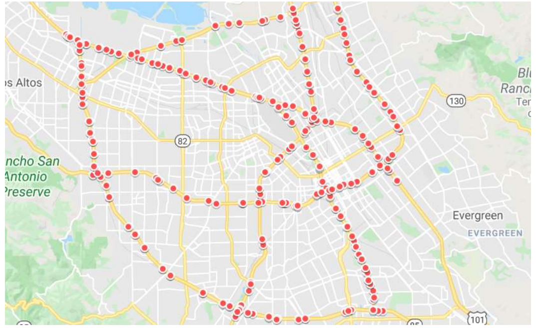
$$G(V, E, X_V, X_E)$$

Static structure, static features

$$G(V, E, X_V(t), X_E(t))$$

Static structure, time-varying features

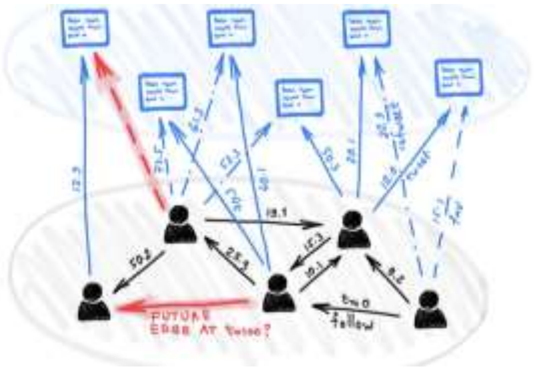
Spatio-temporal graph



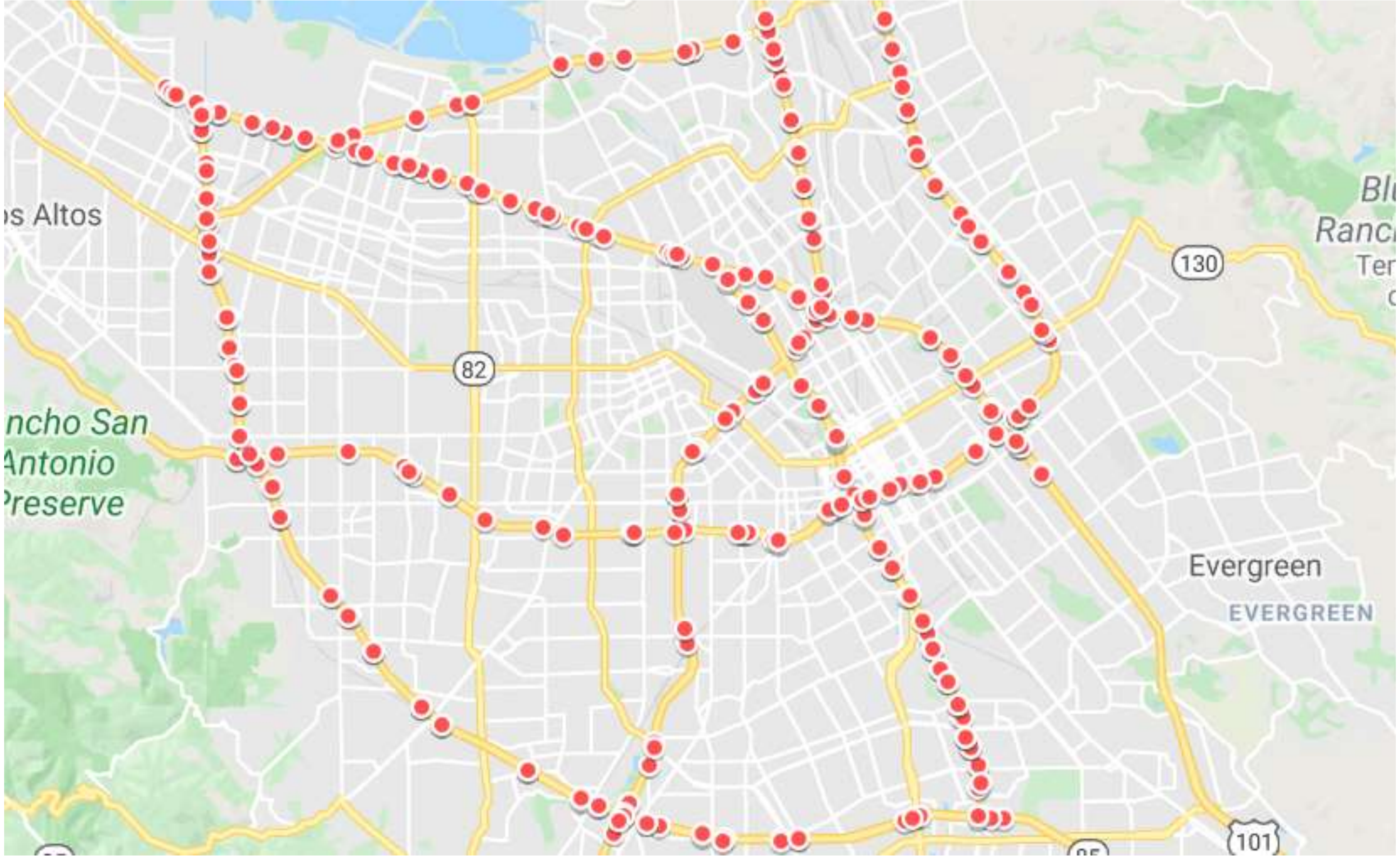
$$G(V(t), E(t), X_V(t), X_E(t))$$

Time-varying structure, time-varying features

Dynamic graph



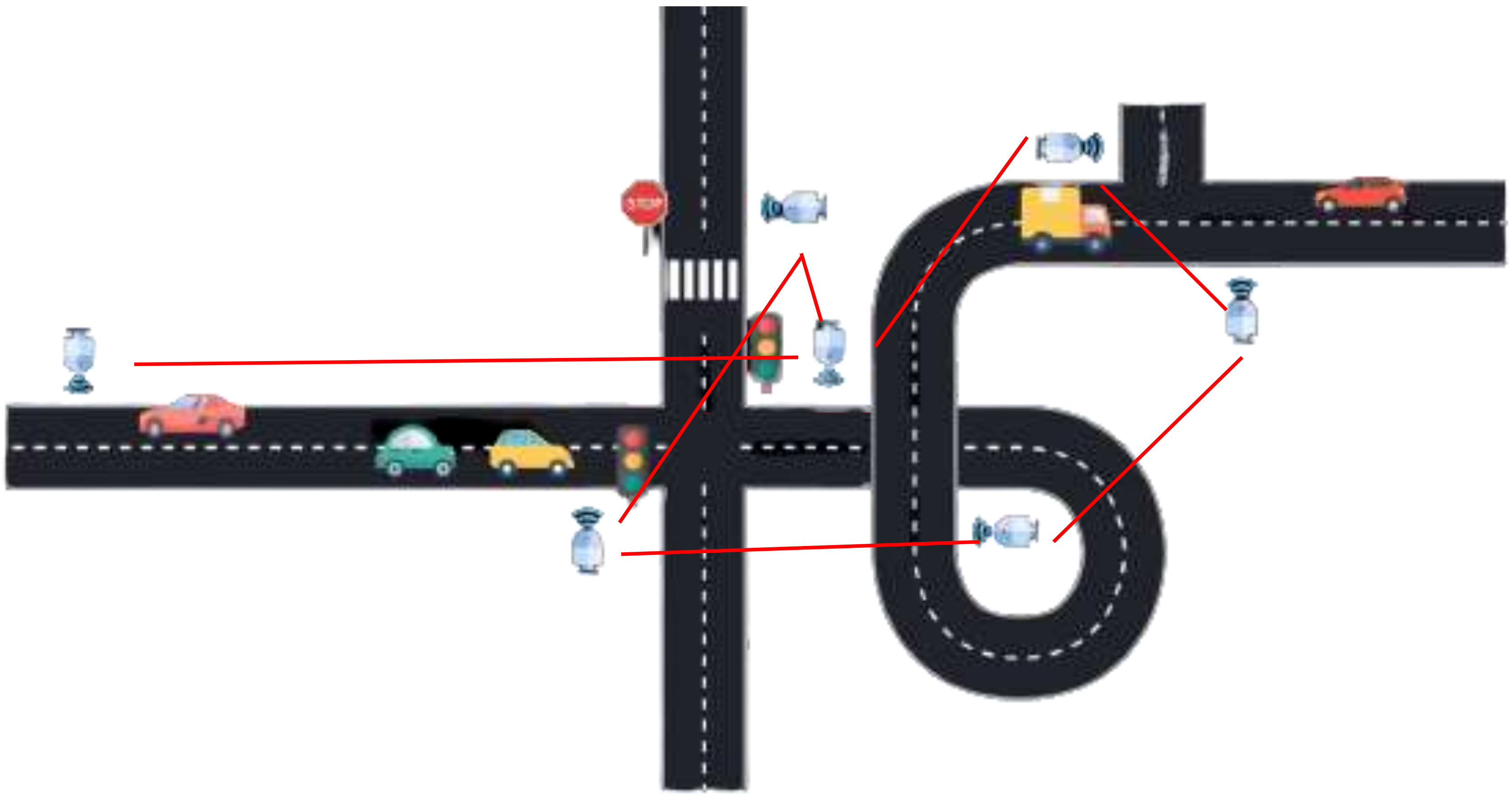
HOW DO WE DEAL WITH GRAPHS WITH STATIC STRUCTURE AND TIME-VARYING FEATURES?



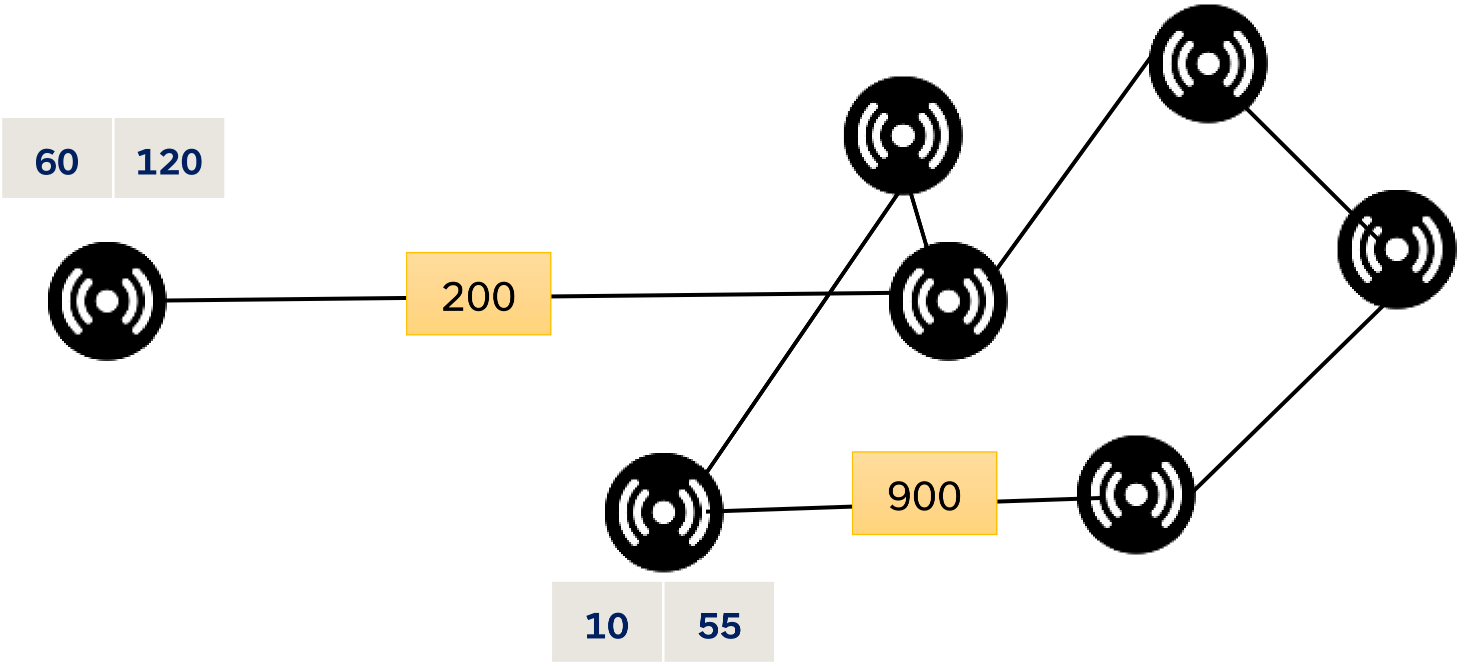
TRAFFIC FORECASTING

TRAFFIC FORECASTING

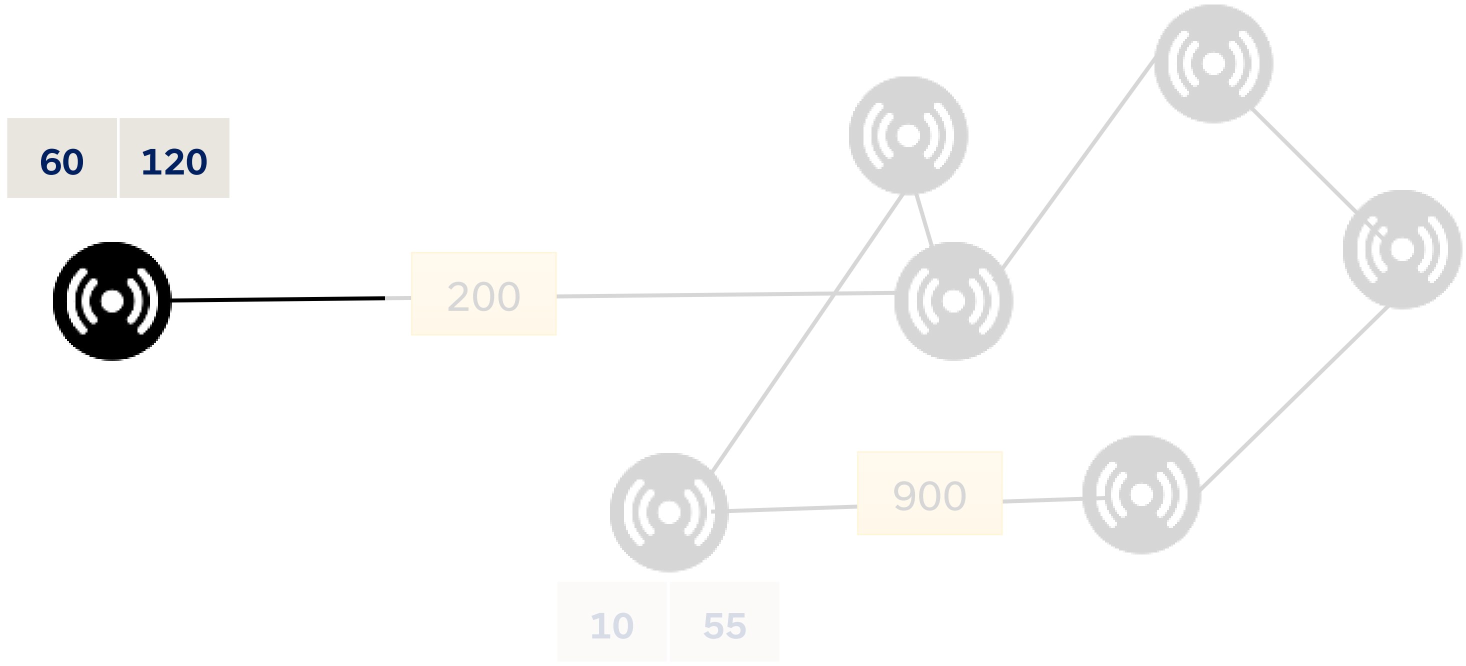
TRAFFIC FORECASTING



TRAFFIC FORECASTING



TIME SERIES

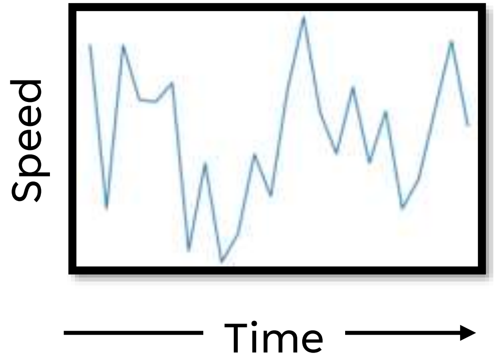


TIME SERIES

TIME SERIES

X_{N_1,t_1}	60	120
X_{N_1,t_2}	65	130
X_{N_1,t_3}	50	100
	…	
X_{N_1,t_T}	40	60

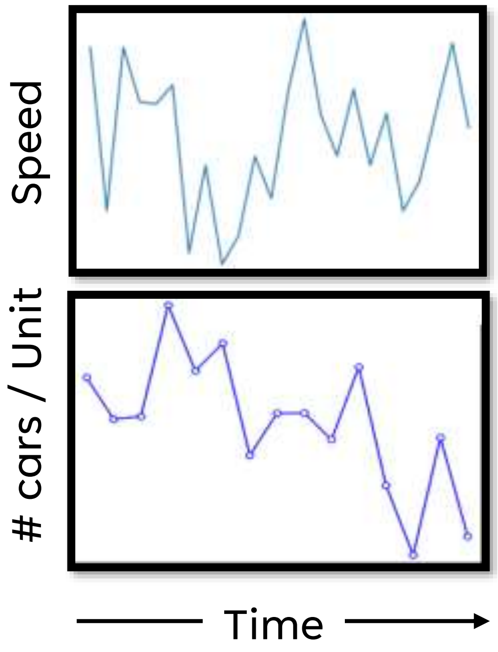
Time ↓



TIME SERIES

X_{N_1,t_1}	60	120
X_{N_1,t_2}	65	130
X_{N_1,t_3}	50	100
…		
X_{N_1,t_T}	40	60

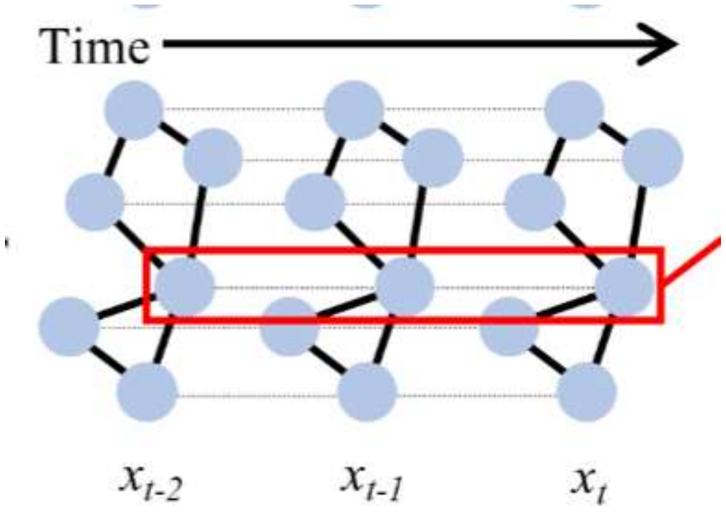
Time ↓



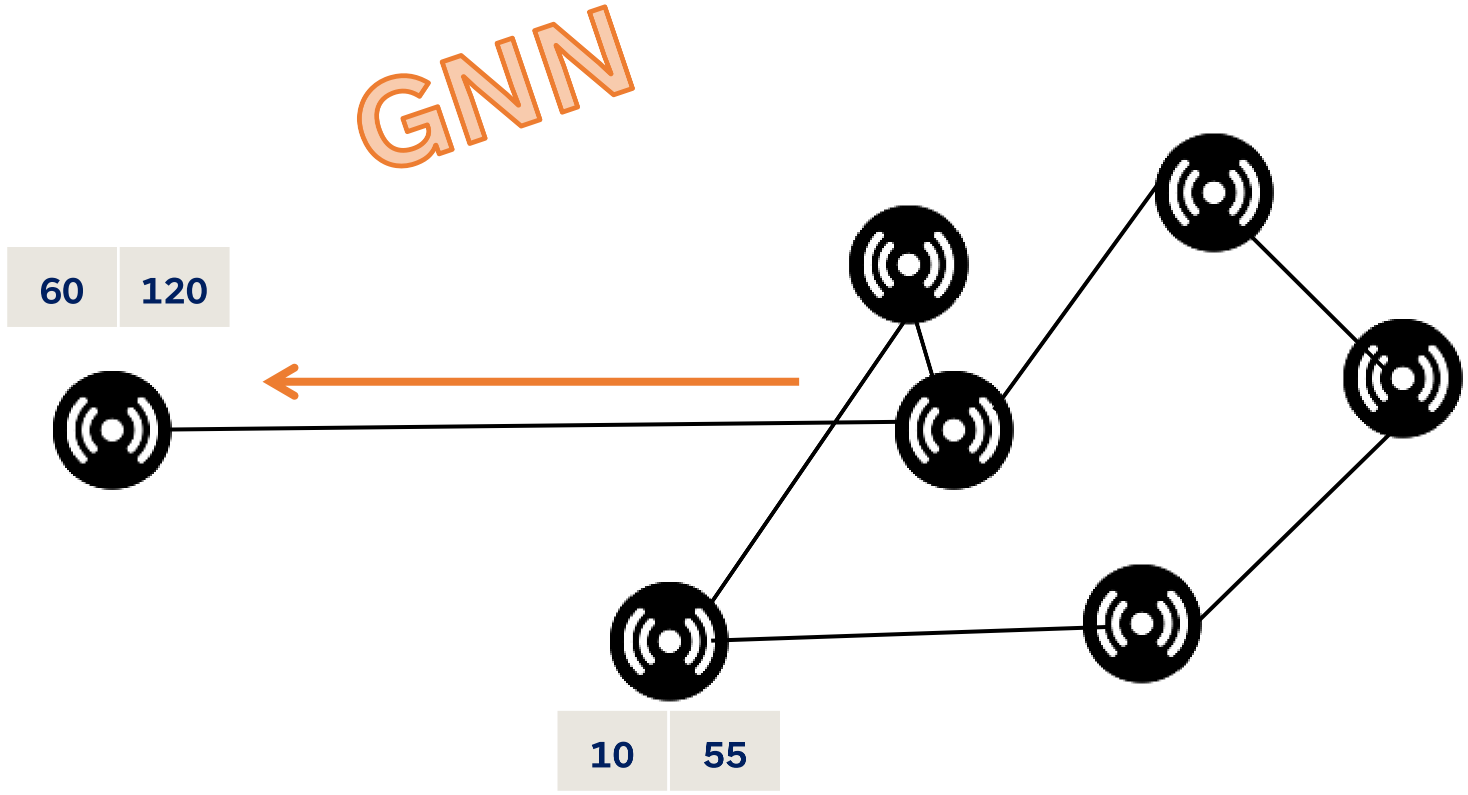
THERE ARE SEVERAL EXISTING MODELS FOR TIME SERIES FORECASTING

- Basic models
  - ARMA-type models (ARMA, VARIMAX, etc.)
    - Basically multi-linear regression over time
    - Requires “stationary” generating process

- Neural network-based models
  - Recurrent neural networks (LSTM, GRU)
  - Temporal convolutions (see 2016 paper)
  - Temporal attention (see 2019 paper)



SPATIAL



STGNNS ARE FAIRLY STRAIGHTFORWARD TO IMPLEMENT, HERE IS AN EXAMPLE IN PSEUDOCODE

```python
class STGNN():
    """Processes a sequence of graph data to produce a spatio-temporal embedding
    to be used for regression, classification, clustering, etc.

    """
    def __init__(self):
        # spatial block can be any standard GNN from the literature
        self.spatial_block = GNN()

        # temporal block can be any method for learning over sequences of data
        ## temporal convolution, temporal attention, etc.
        self.temporal_block = TemporalConv()
        self.fc = torch.nn.Linear(F_in, F_out)
```

STGNNS ARE FAIRLY STRAIGHTFORWARD TO IMPLEMENT, HERE IS AN EXAMPLE IN PSEUDOCODE

```python
def forward(self, X, A):
    """
    Args:
    X (array): matrix of node features, X.shape = (B, N, F, T)
    A (array): adjacency matrix (potentially sparse), defines graph structure,
    if non-sparse A.shape = (N, N)

    where
    B = batch size for batch training
    N = number of nodes in the graph
    F = number of features per node
    T = number of previous timesteps we consider
    """
    tmp = self.temporal_block(X)
    tmp = self.spatial_block(tmp, A)
    tmp = self.temporal_block(tmp)
    tmp = self.fc(tmp)

    return tmp
```

PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models

Benedek Rozemberczki*
AstraZeneca
United Kingdom
benedek.rozemberczki@astrazeneca.com

Paul Scherer
University of Cambridge
United Kingdom
pms69@cam.ac.uk

Yixuan He
University of Oxford
United Kingdom
yixuan.he@stats.ox.ac.uk

George Panagopoulos
École Polytechnique
France
george.panagopoulos@polytechnique.edu

Alexander Riedel
Ernst-Abbe University for Applied Sciences
Germany
alexander.riedel@eah-jena.de

Maria Astefanoaei
IT University of Copenhagen
Denmark
msia@itu.dk

Oliver Kiss
Central European University
Hungary
kiss_oliver@phd.ceu.edu

Ferenc Beres
ELKH SZTAKI
Hungary
beres@sztaki.hu

Guzmán López
Tryolabs
Uruguay
guzman@tryolabs.com

Nicolas Collignon
Pedal Me
United Kingdom
nicolas@pedalme.co.uk

Rik Sarkar
The University of Edinburgh
United Kingdom
rsarkar@inf.ed.ac.uk

ABSTRACT

We present PyTorch Geometric Temporal a deep learning framework combining state-of-the-art machine learning algorithms for

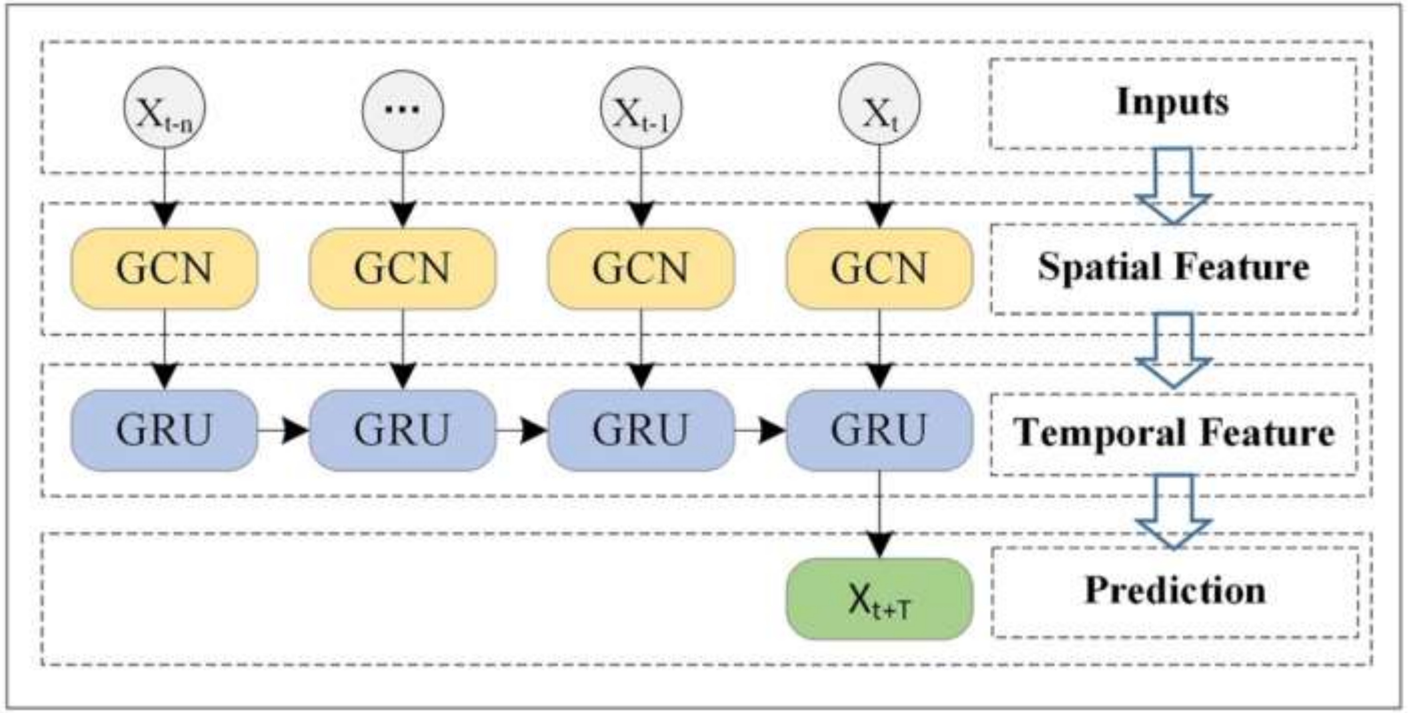
ACM Reference Format:

Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Beres, Guzmán López, Nicolas Collignon, and Rik Sarkar. 2021. PyTorch Geometric Tempo-

Model	Temporal Layer	GNN Layer	Proximity Order	Multi Type
DCRNN [32]	GRU	DiffConv	Higher	False
GConvGRU [51]	GRU	Chebyshev	Lower	False
GConvLSTM [51]	LSTM	Chebyshev	Lower	False
GC-LSTM [10]	LSTM	Chebyshev	Lower	True
DyGrAE [54, 55]	LSTM	GGCN	Higher	False
LRGCN [31]	LSTM	RGCN	Lower	False
EGCN-H [39]	GRU	GCN	Lower	False
EGCN-O [39]	LSTM	GCN	Lower	False
T-GCN [65]	GRU	GCN	Lower	False
A3T-GCN [68]	GRU	GCN	Lower	False
AGCRN [4]	GRU	Chebyshev	Higher	False
MPNN LSTM [38]	LSTM	GCN	Lower	False
STGCN [63]	Attention	Chebyshev	Higher	False
ASTGCN [22]	Attention	Chebyshev	Higher	False
MSTGCN [22]	Attention	Chebyshev	Higher	False
GMAN [66]	Attention	Custom	Lower	False
MTGNN [61]	Attention	Custom	Higher	False
AAGCN [52]	Attention	Custom	Higher	False

Model	Temporal Layer	GNN Layer	Proximity Order	Multi Type
DCRNN [32]	GRU	DiffConv	Higher	False
GConvGRU [51]	GRU	Chebyshev	Lower	False
GConvLSTM [51]	LSTM	Chebyshev	Lower	False
GC-LSTM [10]	LSTM	Chebyshev	Lower	True
DyGrAE [54, 55]	LSTM	GGCN	Higher	False
LRGCN [31]	LSTM	RGCN	Lower	False
EGCN-H [39]	GRU	GCN	Lower	False
EGCN-O [39]	LSTM	GCN	Lower	False
T-GCN [65]	GRU	GCN	Lower	False
A3T-GCN [68]	GRU	GCN	Lower	False
AGCRN [4]	GRU	Chebyshev	Higher	False
MPNN LSTM [38]	LSTM	GCN	Lower	False
STGCN [63]	Attention	Chebyshev	Higher	False
ASTGCN [22]	Attention	Chebyshev	Higher	False
MSTGCN [22]	Attention	Chebyshev	Higher	False
GMAN [66]	Attention	Custom	Lower	False
MTGNN [61]	Attention	Custom	Higher	False
AAGCN [52]	Attention	Custom	Higher	False

T-GCN:A TEMPORAL GRAPH CONVOLUTIONAL NETWORK FOR TRAFFIC PREDICTION



Model	Temporal Layer	GNN Layer	Proximity Order	Multi Type
DCRNN [32]	GRU	DiffConv	Higher	False
GConvGRU [51]	GRU	Chebyshev	Lower	False
GConvLSTM [51]	LSTM	Chebyshev	Lower	False
GC-LSTM [10]	LSTM	Chebyshev	Lower	True
DyGrAE [54, 55]	LSTM	GGCN	Higher	False
LRGCN [31]	LSTM	RGCN	Lower	False
EGCN-H [39]	GRU	GCN	Lower	False
EGCN-O [39]	LSTM	GCN	Lower	False
T-GCN [65]	GRU	GCN	Lower	False
A3T-GCN [68]	GRU	GCN	Lower	False
AGCRN [4]	GRU	Chebyshev	Higher	False
MPNN LSTM [38]	LSTM	GCN	Lower	False
STGCN [63]	Attention	Chebyshev	Higher	False
ASTGCN [22]	Attention	Chebyshev	Higher	False
MSTGCN [22]	Attention	Chebyshev	Higher	False
GMAN [66]	Attention	Custom	Lower	False
MTGNN [61]	Attention	Custom	Higher	False
AAGCN [52]	Attention	Custom	Higher	False

T-GCN: A Temporal Graph ConvolutionalNetwork for Traffic Prediction, Zhao et all

PYTORCH GEOMETRIC TEMPORAL

- ✓ StaticGraphTemporalSignal
- ✓ DynamicGraphTemporalSignal
- ✓ DynamicGraphStaticSignal

Spatiotemporal Signal Splitting

PyTorch Geometric Temporal

Temporal GNN Layers

Datasets

https://pytorch-geometric-temporal.readthedocs.io/en/latest/index.html

THANK YOU

Alireza Akhavanpour

https://Class.vision

SOURCES

CS224W: Machine Learning with Graphs

https://web.stanford.edu/class/cs224w/

Intro to graph neural networks (ML Tech Talks)

https://www.youtube.com/watch?v=8owQBFAHw7E&t=253s

Introduction to graph neural networks (made easy!)

https://www.youtube.com/watch?v=cka4Fa4TTI4

https://www.topbots.com/graph-convolutional-networks/

How to use edge features in Graph Neural Networks (and PyTorch Geometric)

https://www.youtube.com/watch?v=mdWQYYapvR8&list=PLV8yxwGOxvvoNkzPfCx2i8an--Tkt7O8Z&index=5