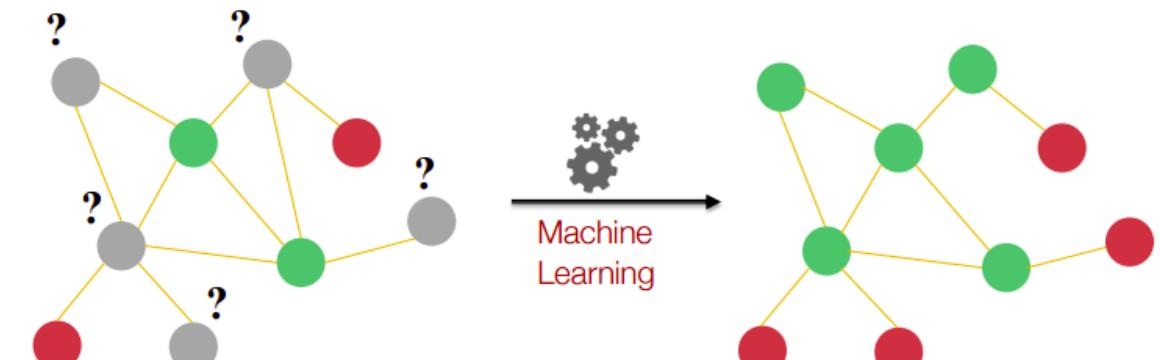
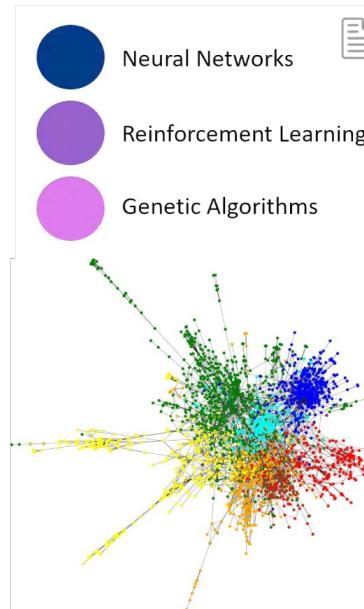
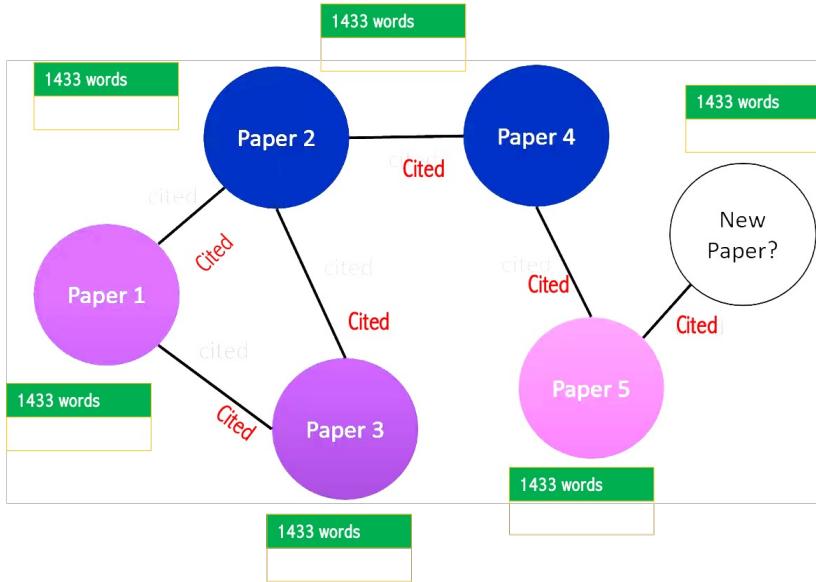


Introduction to Graph Neural Network

(Node Classification: Cora Citation Dataset)



[Data & Code](#)

Vinh Dinh Nguyen
PhD in Computer Science

Outline



Objective

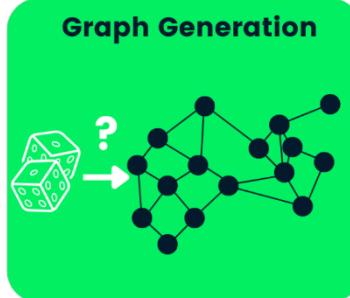
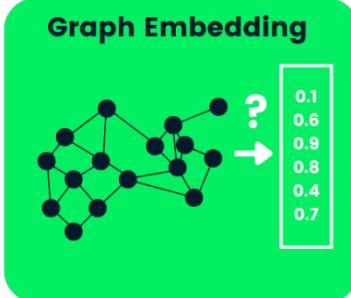
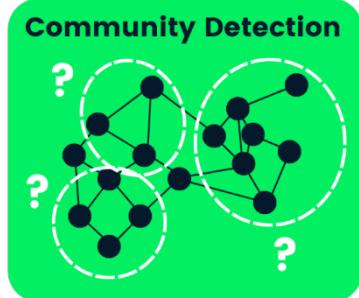
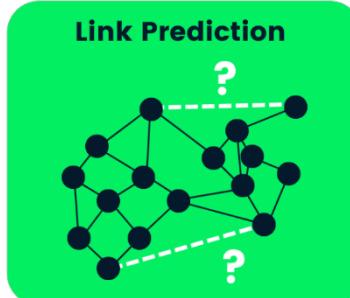
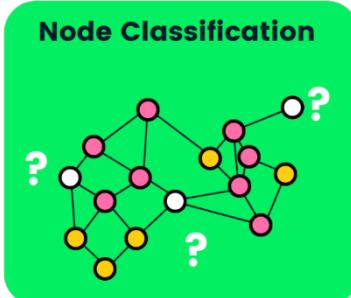
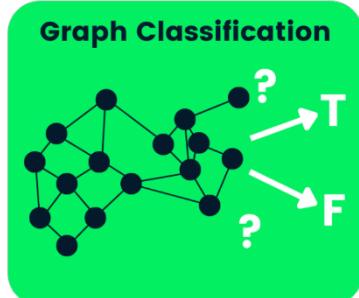
Introduction to Graph Data

Graph Data with Neural Network

Node Classification Problem: Cora Citation Dataset

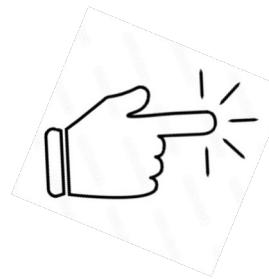
Summary

Objective



- 1 • What is Graph Data Around Us
- 2 • Understand Graph Neural Network
- 3 • Understand Graph Convolutional Neural Network
- 4 • Node Classification with Cora Citation Dataset

Outline



Objective

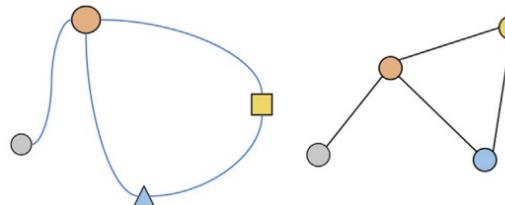
Introduction to Graph Data

Graph Data with Neural Network

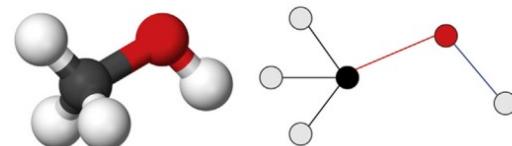
Node Classification Problem: Cora Citation Dataset

Summary

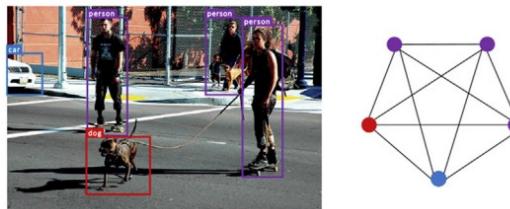
Applications of GNNs



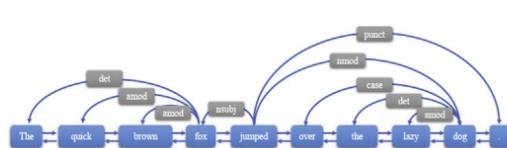
(a) Physics



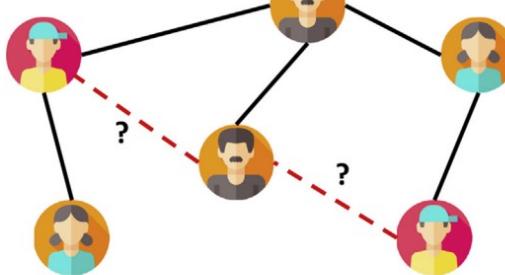
(b) Molecule



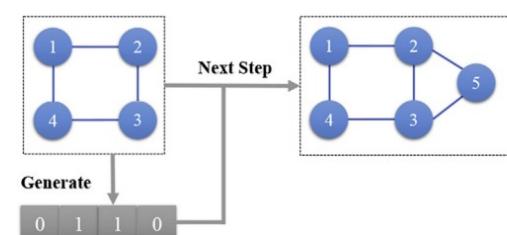
(c) Image



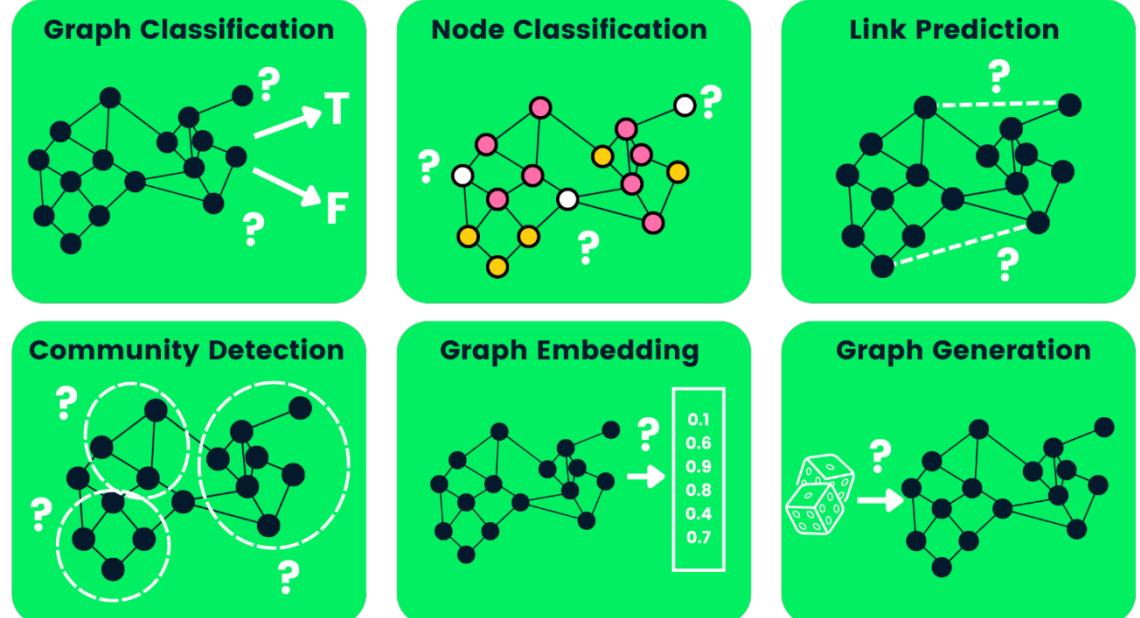
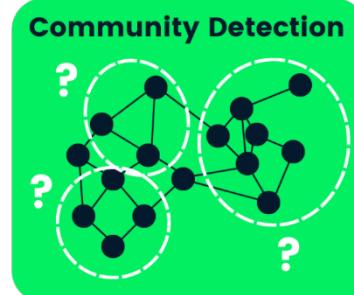
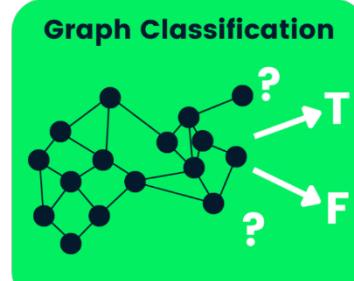
(d) Text



(e) Social Network



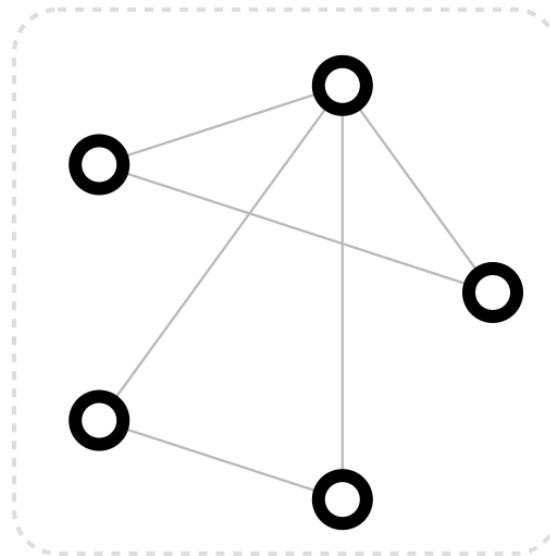
(f) Generation



Icons made by Freepik from Flaticon

Graph Definition

A graph represents the relations (edges) between a collection of entities (nodes).

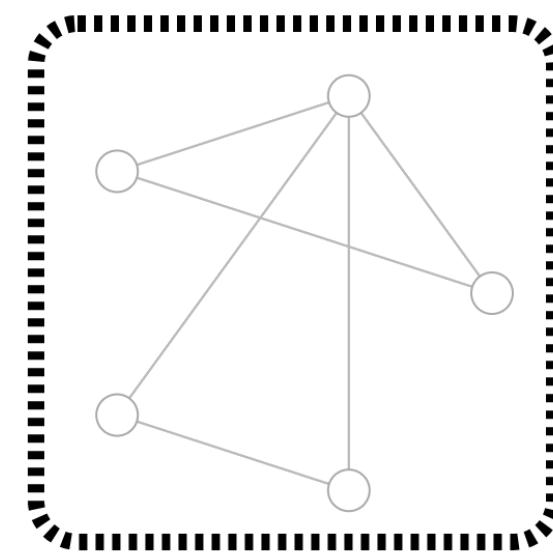


V Vertex (or node) attributes

e.g., node identity, number of neighbors

E Edge (or link) attributes and directions

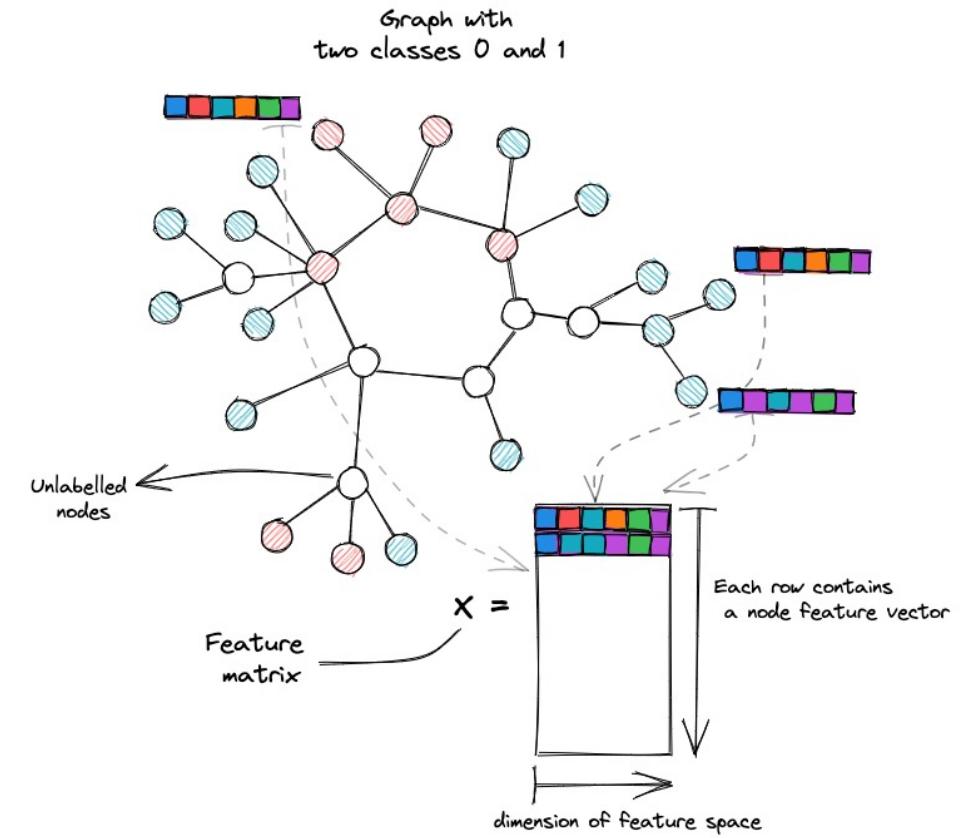
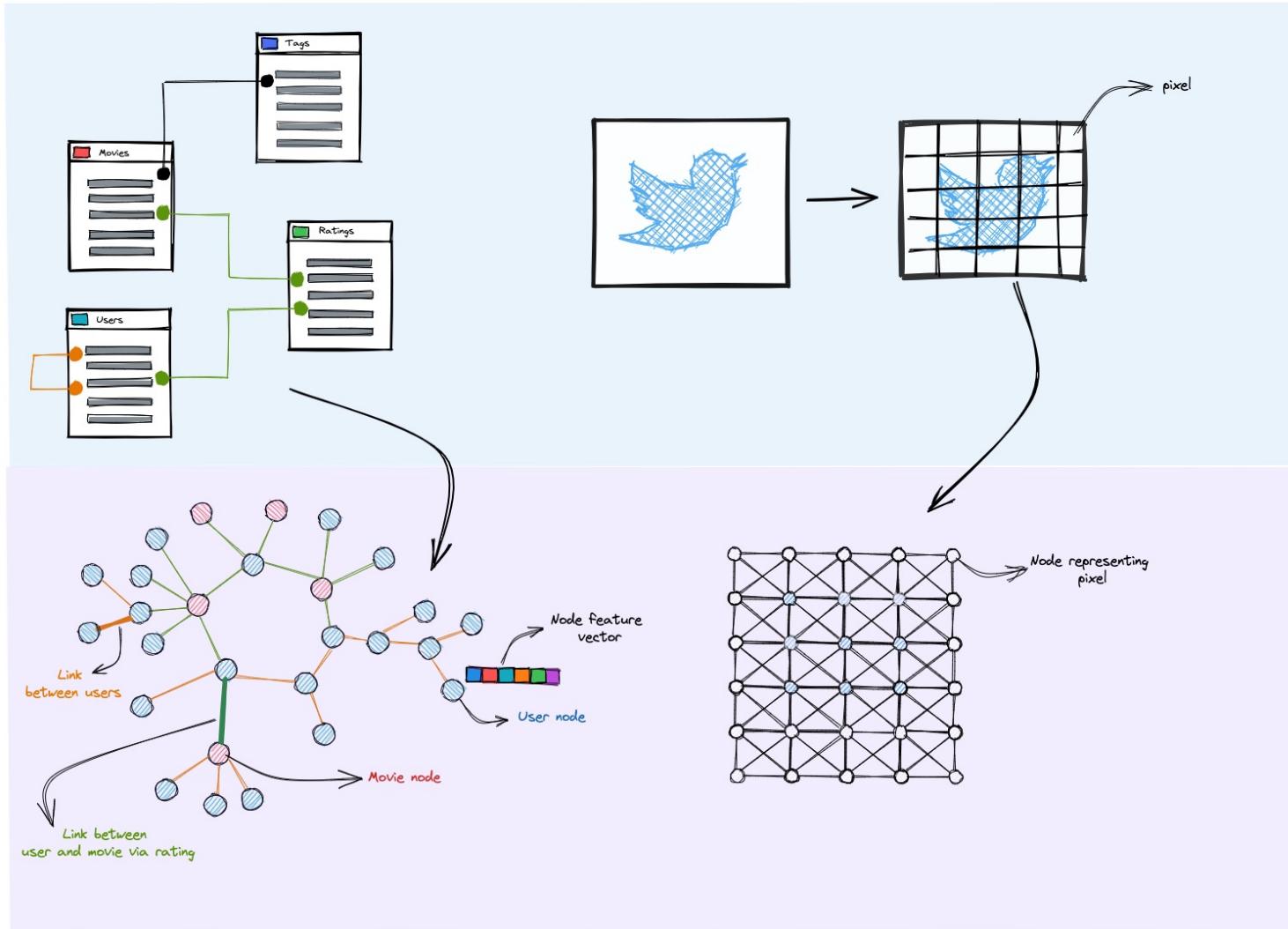
e.g., edge identity, edge weight



U Global (or master node) attributes

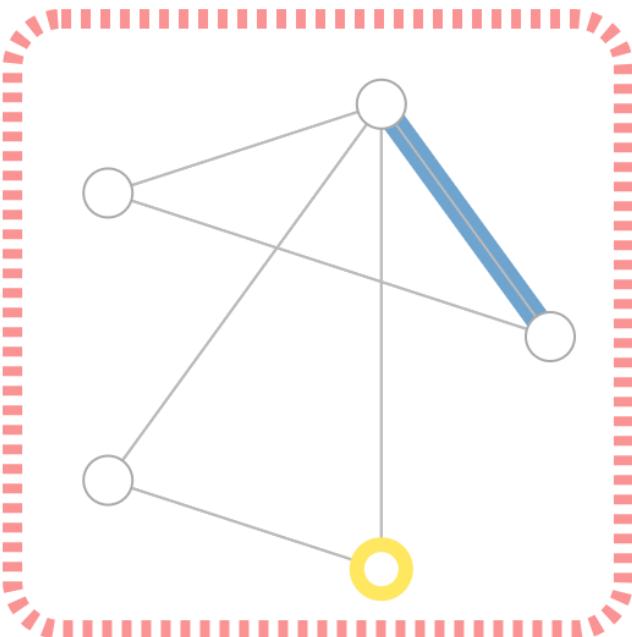
e.g., number of nodes, longest path

Graphs are everywhere



Graph Definition

To further describe each node, edge or the entire graph, we can store information in each of these pieces of the graph



Vertex (or node) embedding



Edge (or link) attributes and embedding



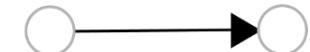
Global (or master node) embedding



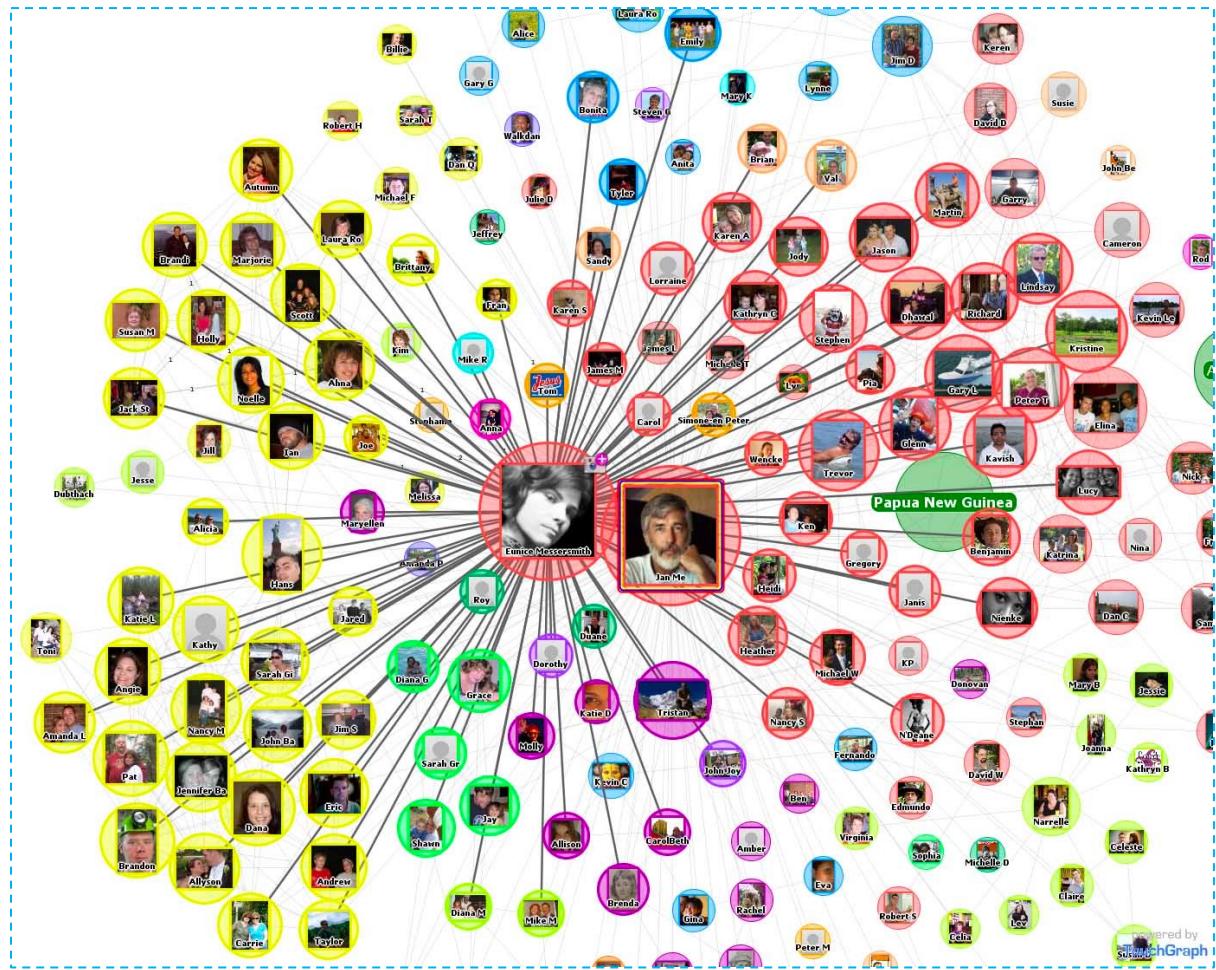
Undirected edge



Directed edge



Graphs and where to find them



Social networks

Two types of data that you might not think could be modeled as graphs:



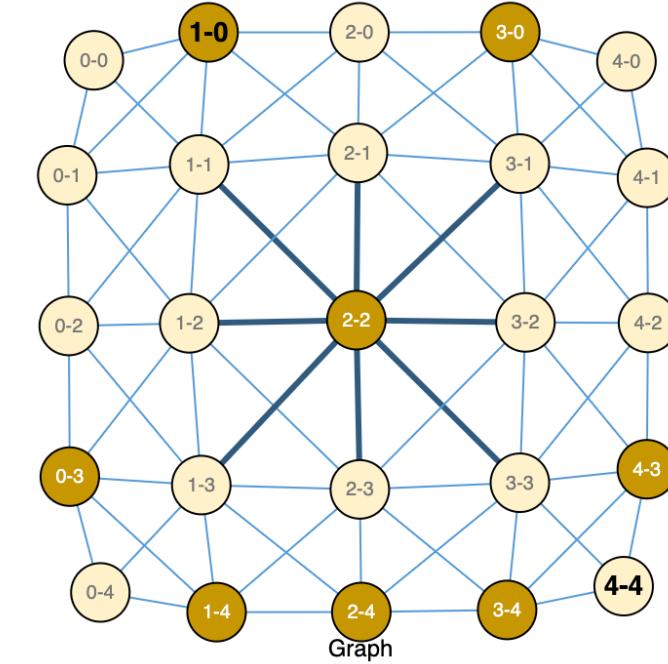
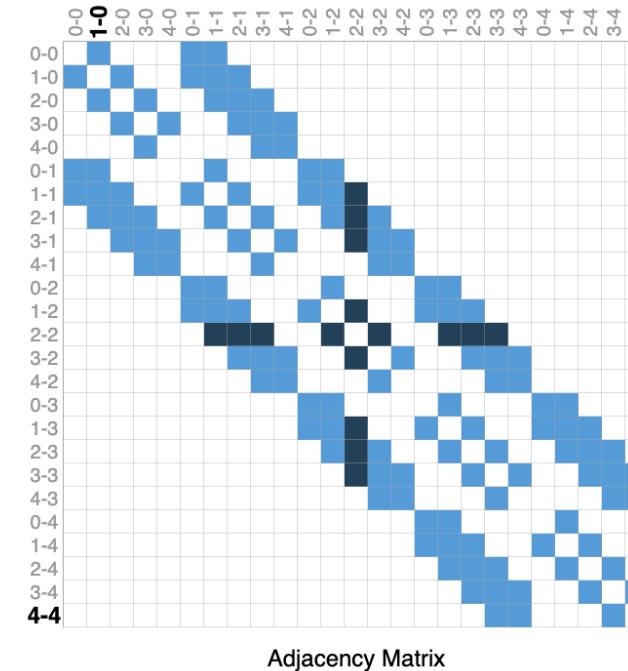
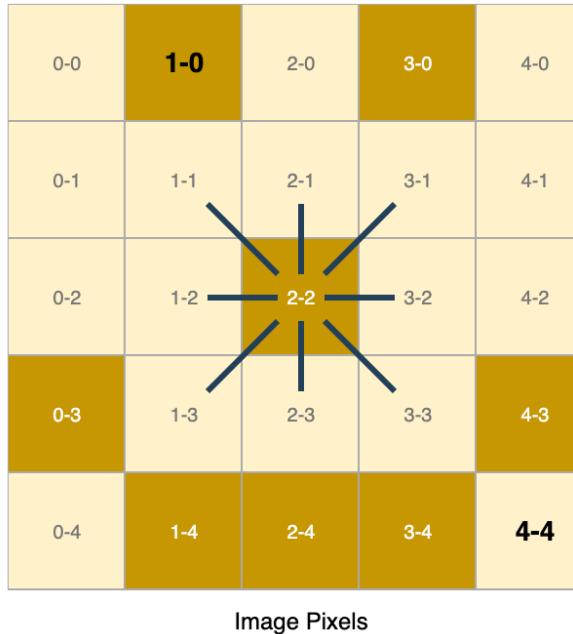
Image



Graphs are all around us

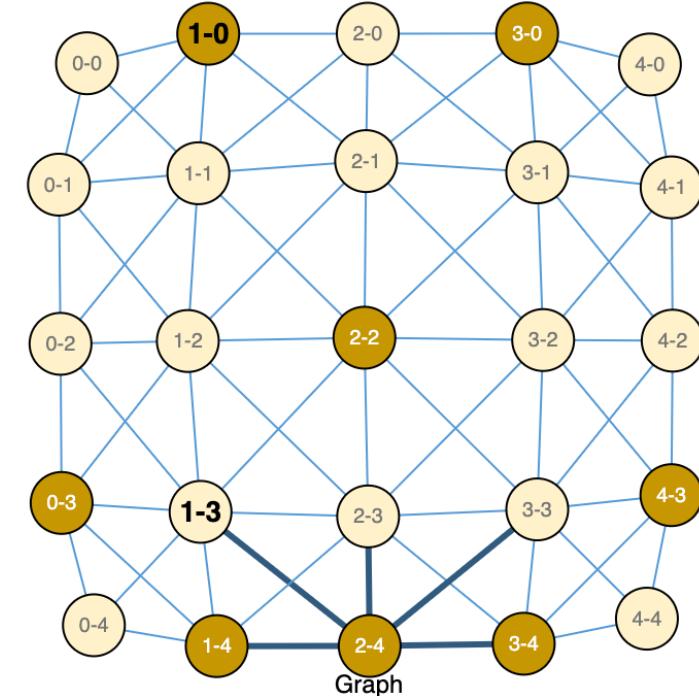
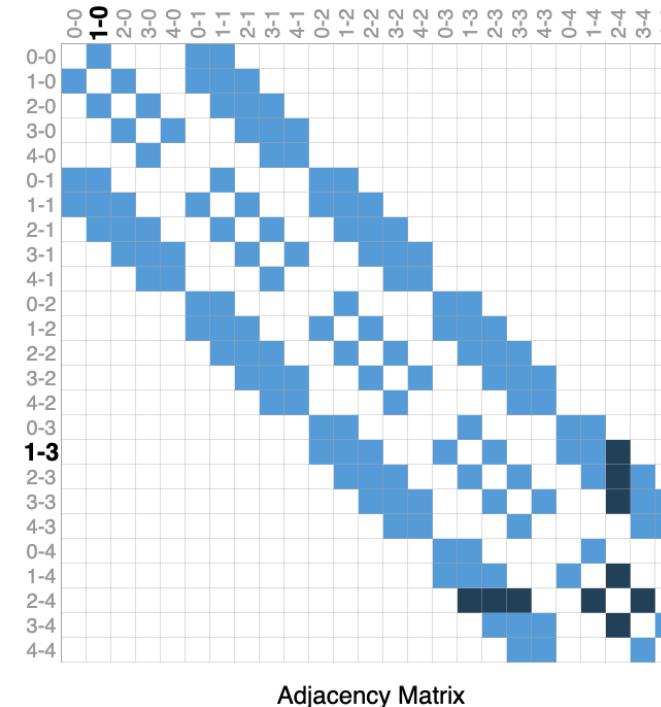
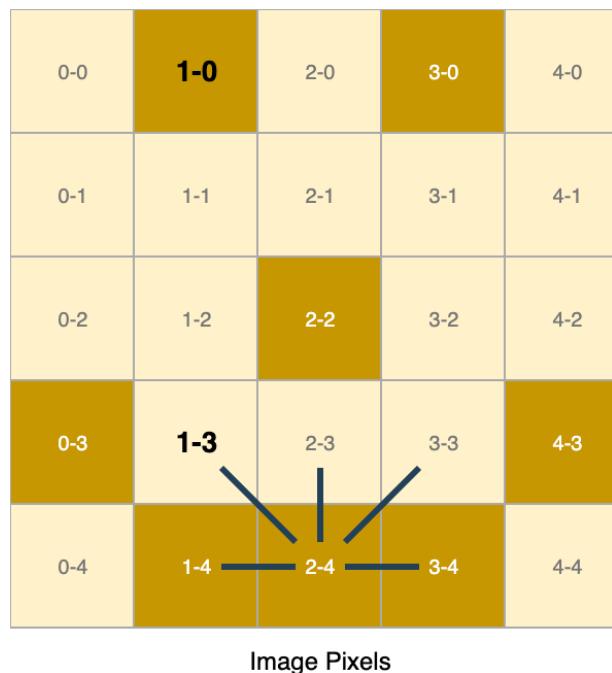
Text

Image as Graphs



Another way to think of images is as graphs with regular structure, where each pixel represents a node and is connected via an edge to adjacent pixels. Each non-border pixel has exactly 8 neighbors, and the information stored at each node is a 3-dimensional vector representing the RGB value of the pixel.

Image as Graphs

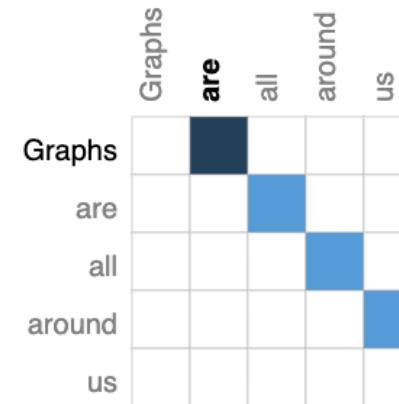
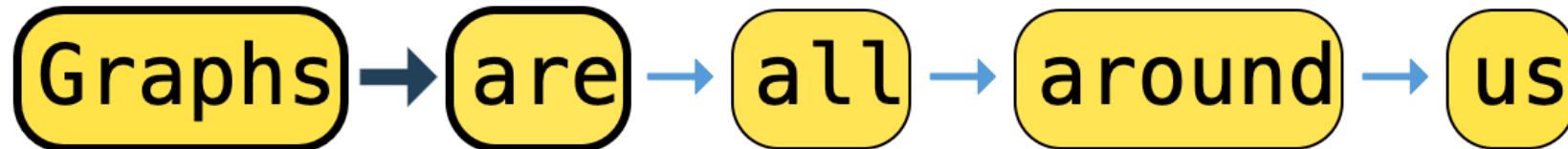


Click on an image pixel to toggle its value, and see how the graph representation changes.



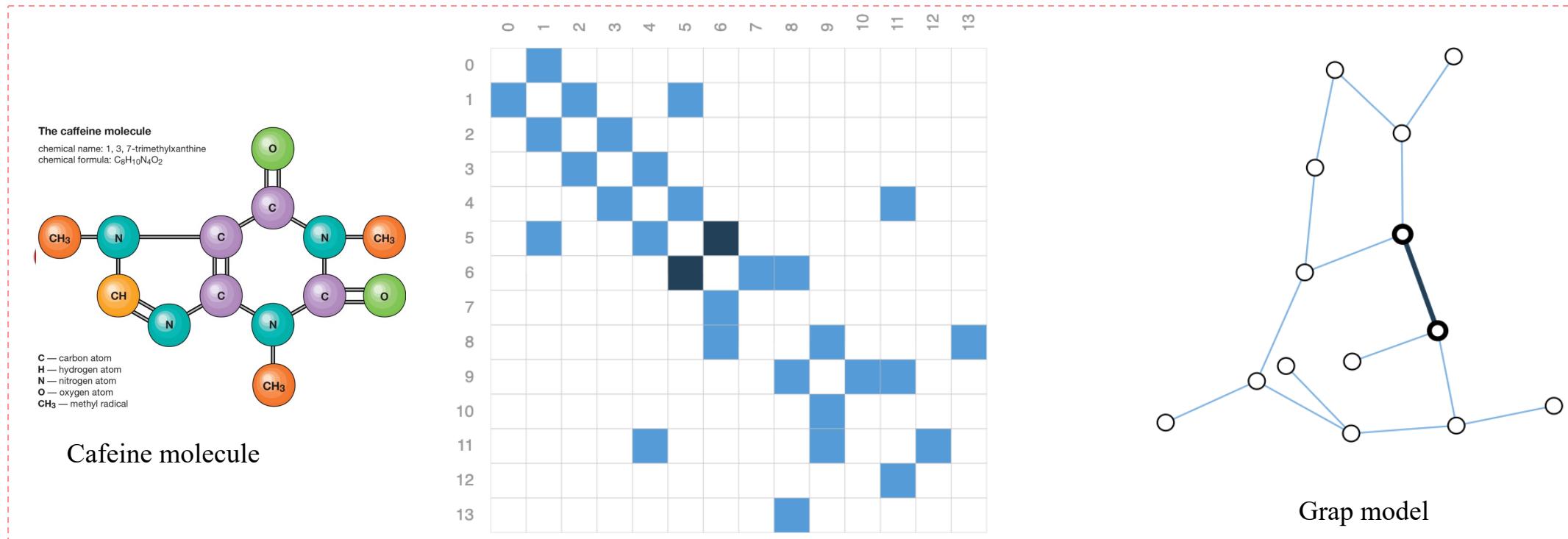
Another way to think of images is as graphs with regular structure, where each pixel represents a node and is connected via an edge to adjacent pixels. Each non-border pixel has exactly 8 neighbors, and the information stored at each node is a 3-dimensional vector representing the RGB value of the pixel.

Text as Graphs



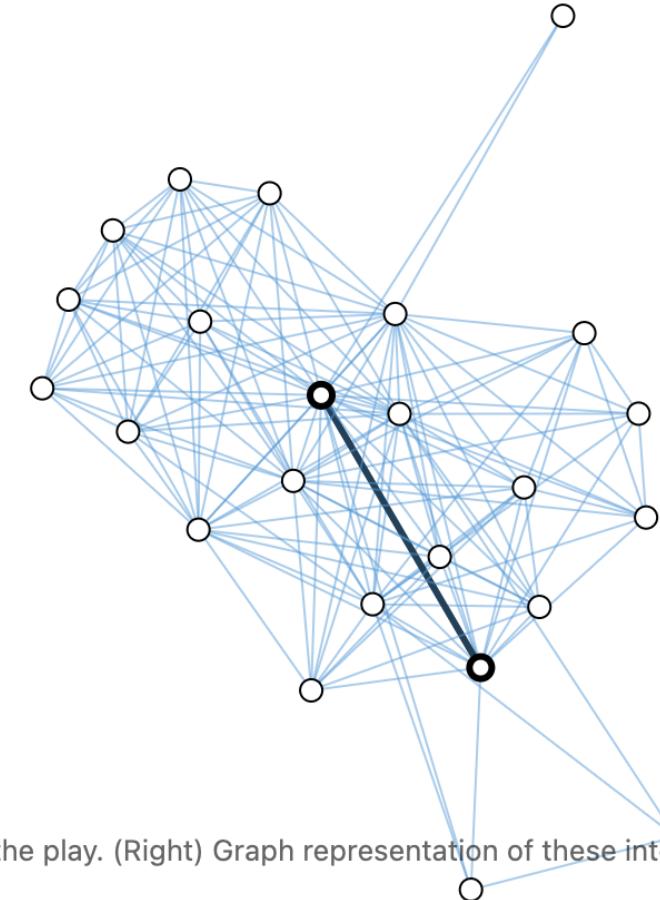
We can digitize text by associating indices to each character, word, or token, and representing text as a sequence of these indices. This creates a simple directed graph, where each character or index is a node and is connected via an edge to the node that follows it.

Other Graph Data



Molecules as graphs

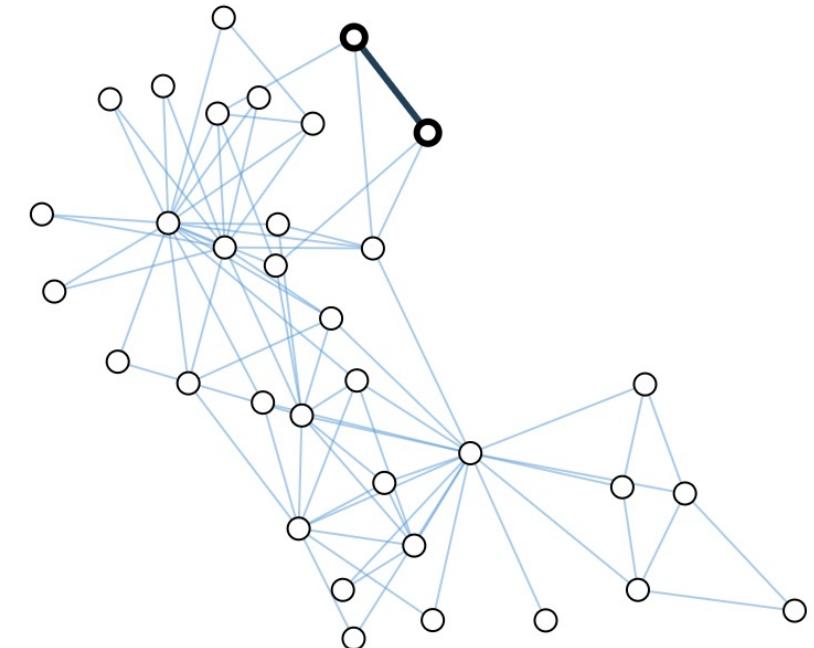
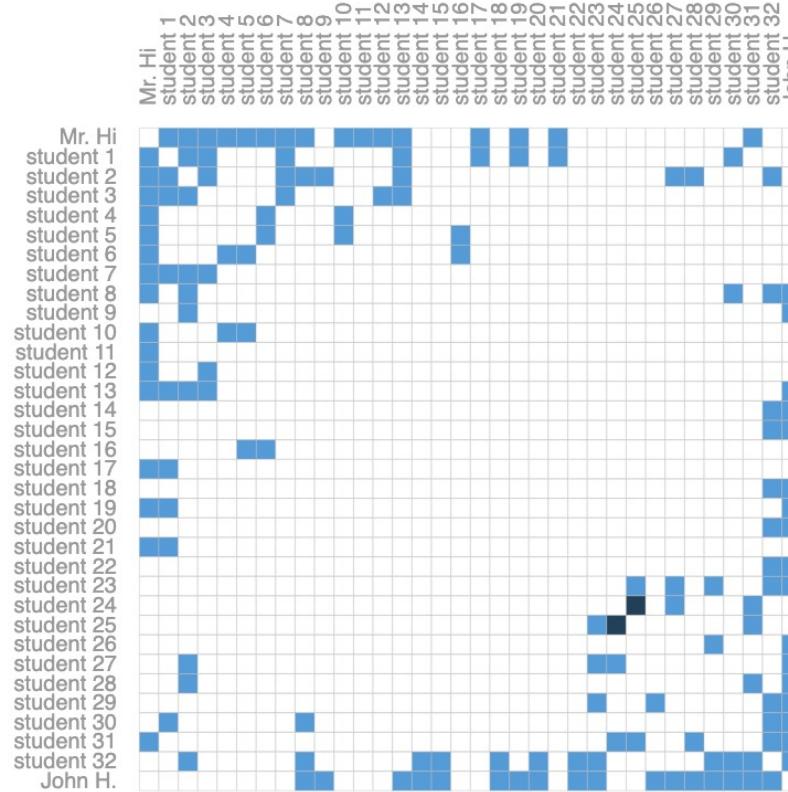
Other Graph Data



(Left) Image of a scene from the play "Othello". (Center) Adjacency matrix of the interaction between characters in the play. (Right) Graph representation of these interactions.

Unlike image and text data, social networks do not have identical adjacency matrices.

Other Graph Data



(Left) Image of karate tournament. (Center) Adjacency matrix of the interaction between people in a karate club. (Right) Graph representation of these interactions.

Unlike image and text data, social networks do not have identical adjacency matrices.

Other Graph Data

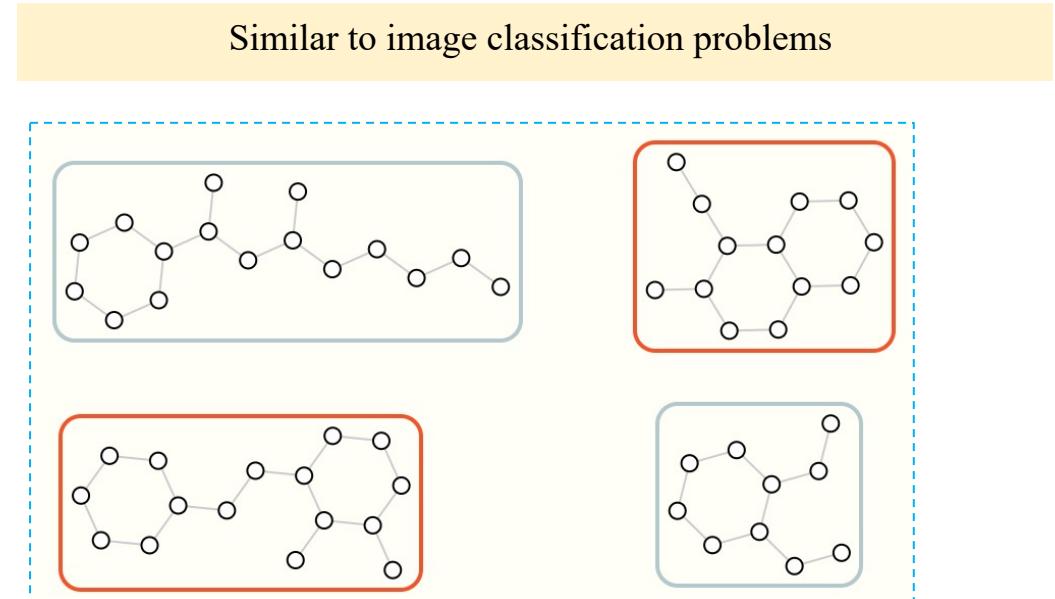
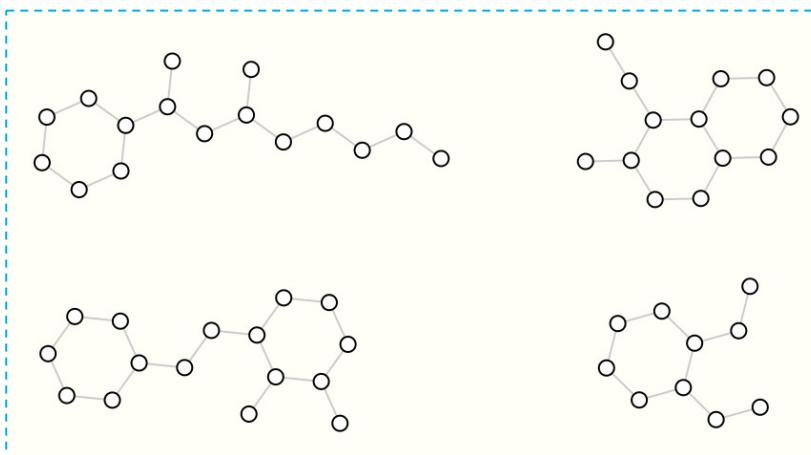
Dataset	Domain	Edges per node (degree)					
		graphs	nodes	edges	min	mean	max
karate club	Social network	1	34	78		4.5	17
qm9	Small molecules	134k	≤ 9	≤ 26	1	2	5
Cora	Citation network	1	23,166	91,500	1	7.8	379
Wikipedia links, English	Knowledge graph	1	12M	378M		62.24	1M



Summary statistics on graphs found in the real world. Numbers are dependent on featurization decisions.

Tasks on Graph Data

Graph-level task



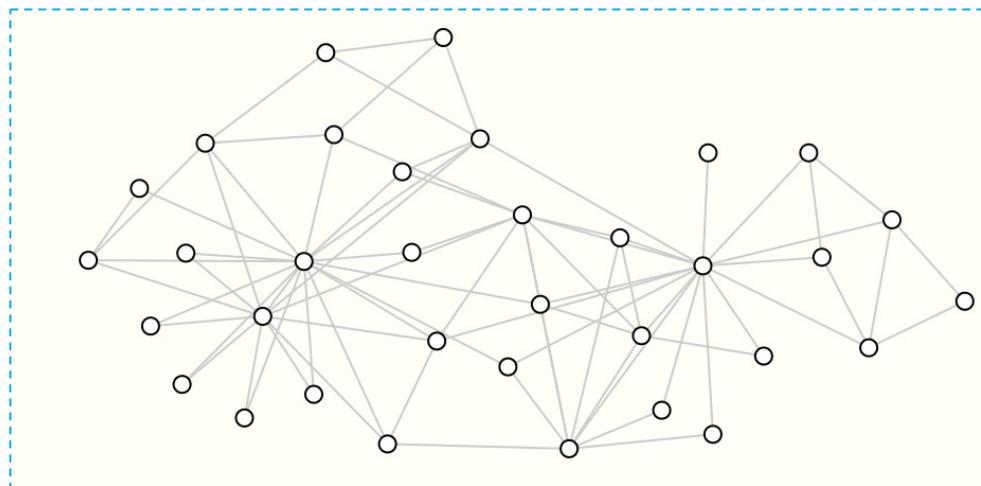
Output: labels for each graph, (e.g., "does the graph contain two rings?")



In a graph-level task, our goal is to predict the property of an entire graph. For example, for a molecule represented as a graph, we might want to predict what the molecule smells like, or whether it will bind to a receptor implicated in a disease.

Tasks on Graph Data

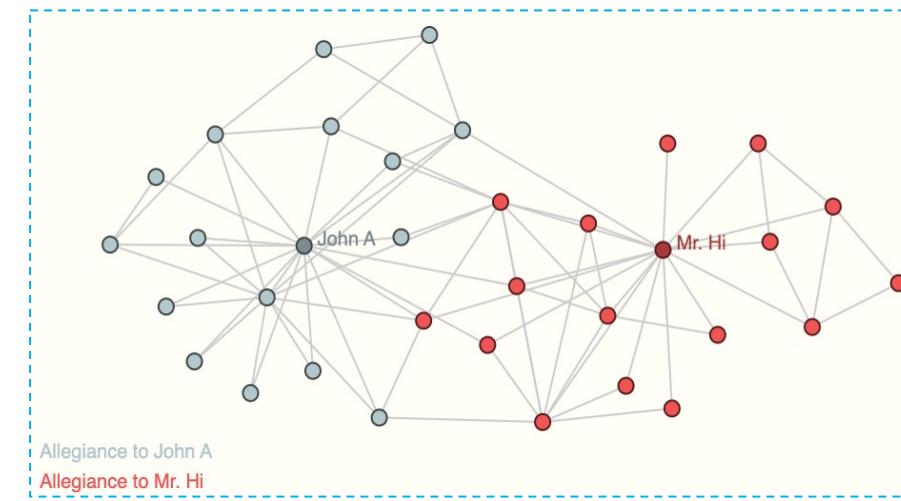
Node-level task



Input: graph with unlabeled nodes



Similar to image segmentation problems



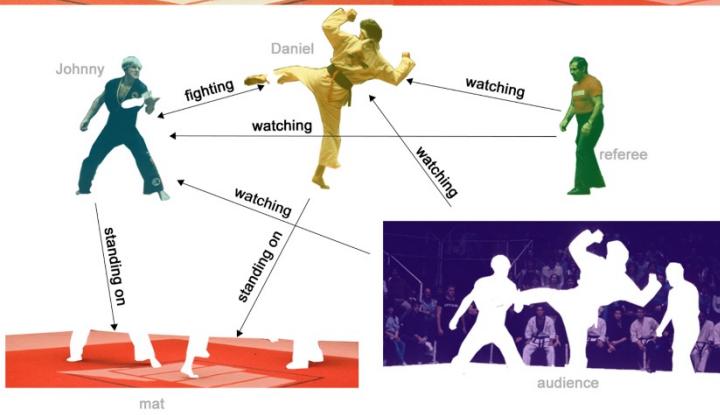
Output: graph node labels



Node-level tasks are concerned with predicting the identity or role of each node within a graph.

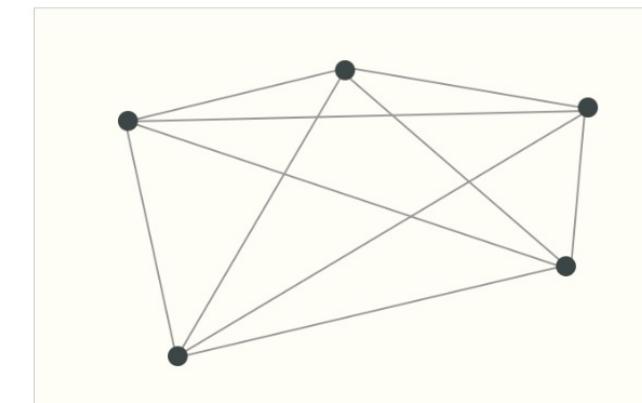
Tasks on Graph Data

Edge-level task

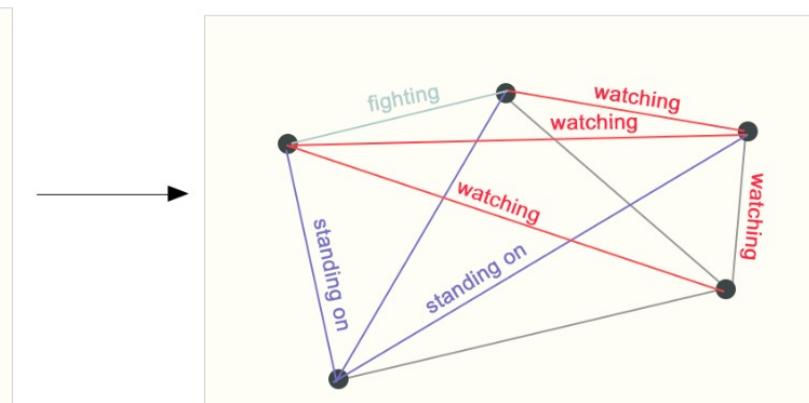


Similar to image scene understanding

Edge-level tasks are concerned with predicting the relation for each pair node within a graph.



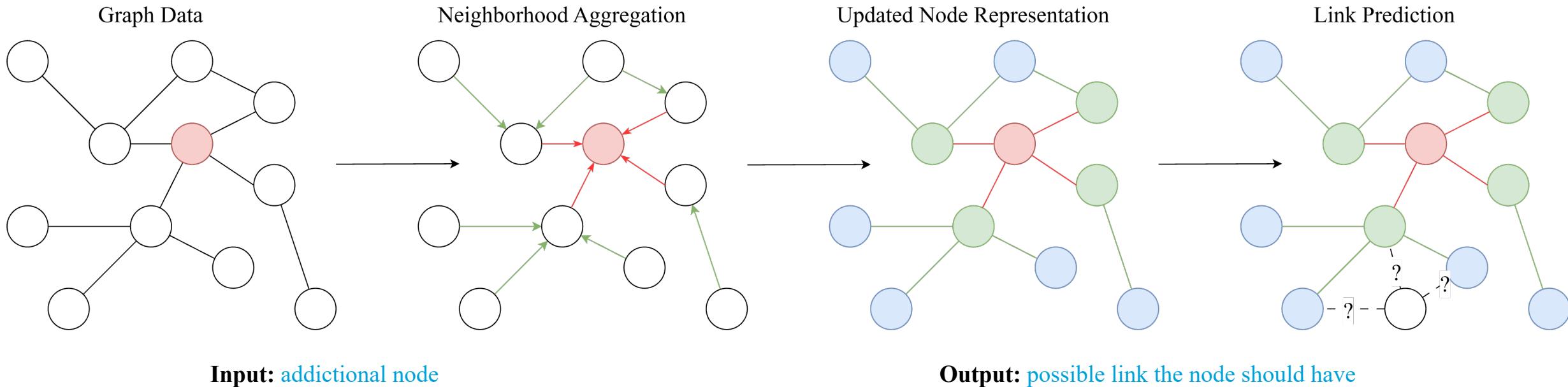
Input: fully connected graph, unlabeled edges



Output: labels for edges

Tasks on Graph Data

Edge-level task

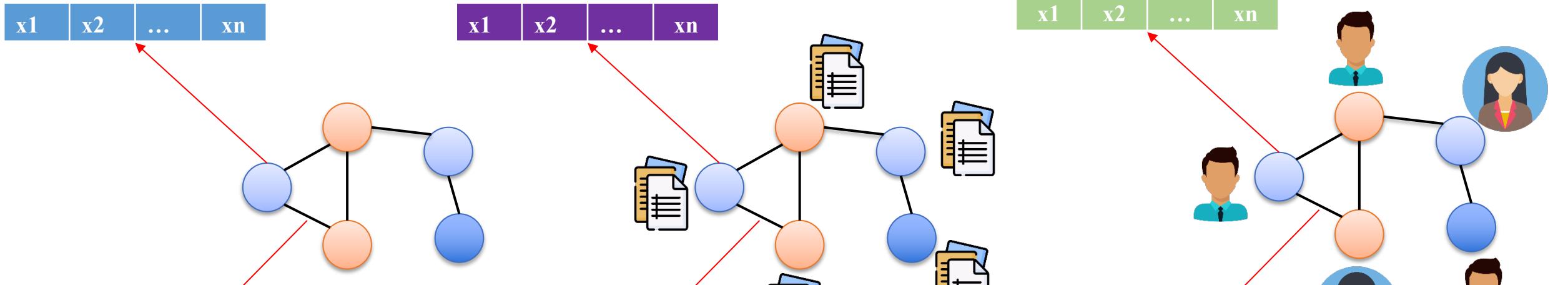


The edge-level task aims to predict where the link is placed or whether the link can be trimmed to simplify the graph.

A Simple Graph

Node information

x1		x2		...		xn
----	--	----	--	-----	--	----



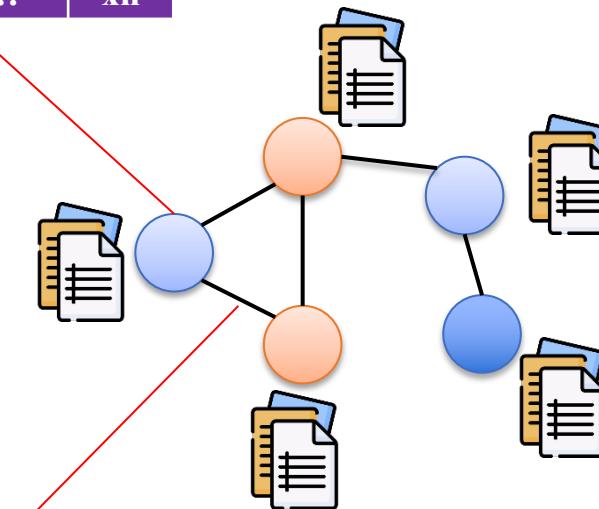
Edge information

Paper Content

x1		x2		...		xn
----	--	----	--	-----	--	----



Paper citation

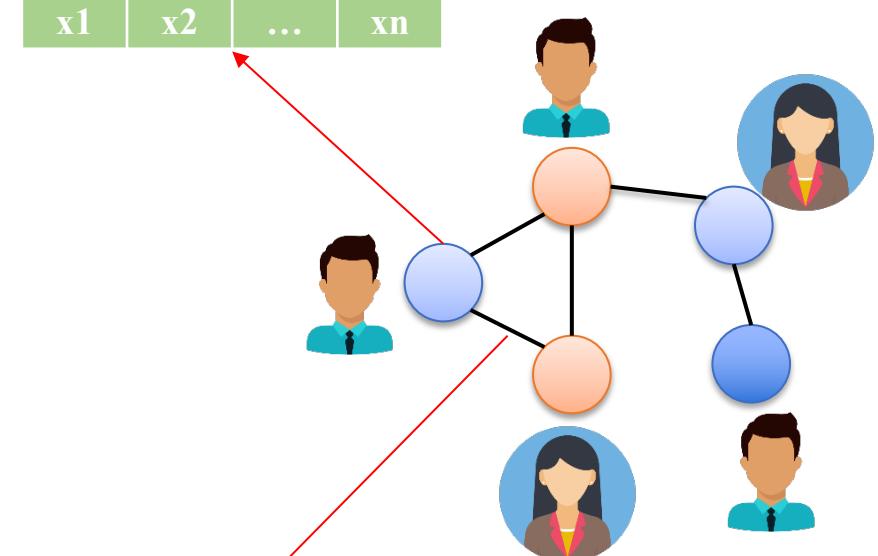


Personal Information

x1		x2		...		xn
----	--	----	--	-----	--	----

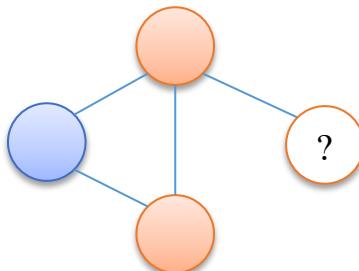


Relationship



Example

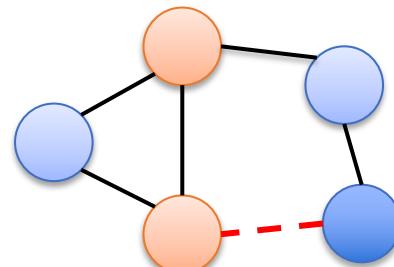
Node level prediction



Does this student smoke?
(unlabeled node)

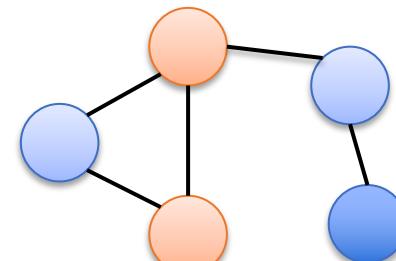


Edge level predictions
(Link prediction)



Next Youtube Video?

Graph Level Predictions



Is this molecule a suitable drug?



Outline

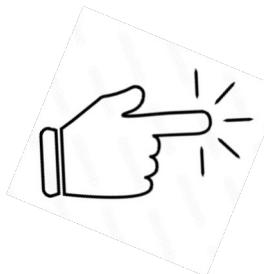
Objective

Introduction to Graph Data

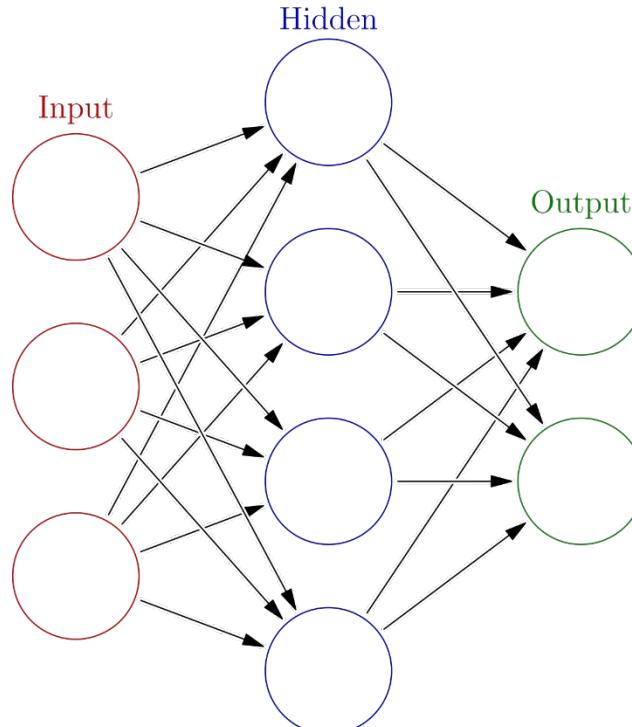
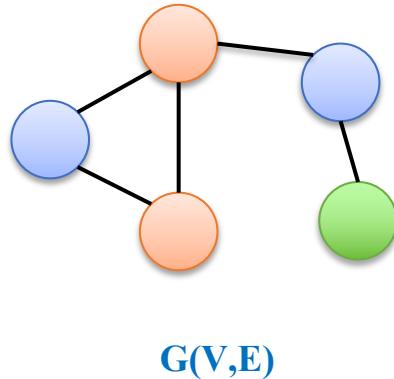
Graph Data with Neural Network

Node Classification Problem: Cora Citation Dataset

Summary

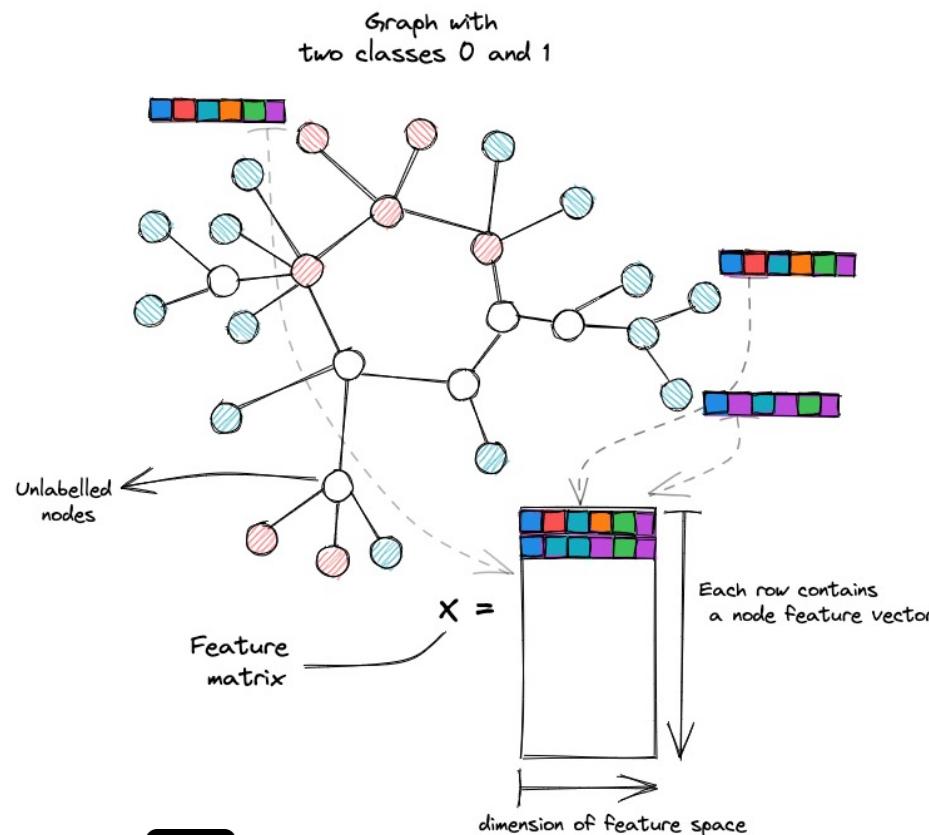


Graph Data with Neural Network

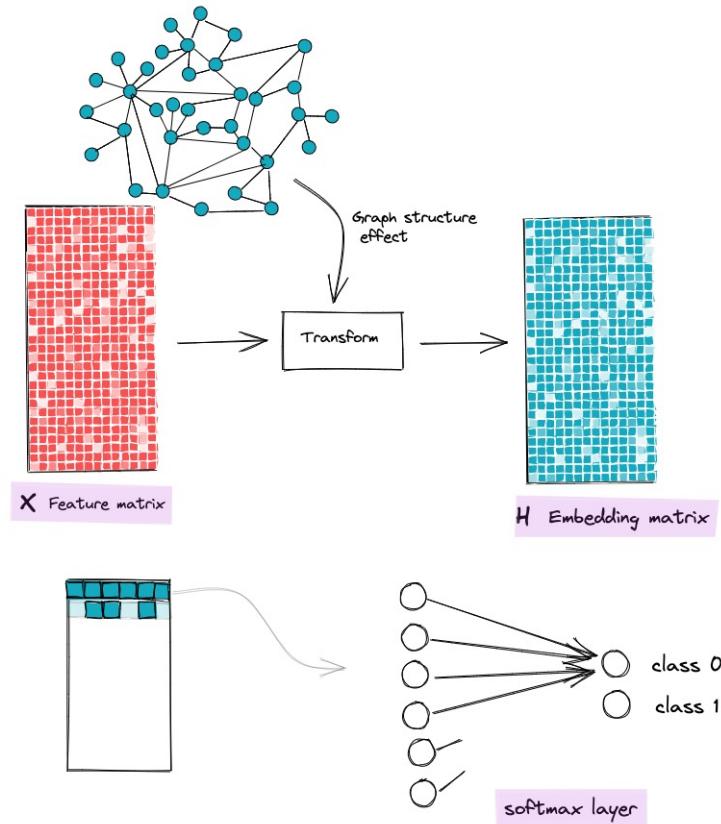


So, how do we go about solving these different graph tasks with neural networks? The first step is to think about how we will represent graphs to be compatible with neural networks.

Graph Data with Neural Network

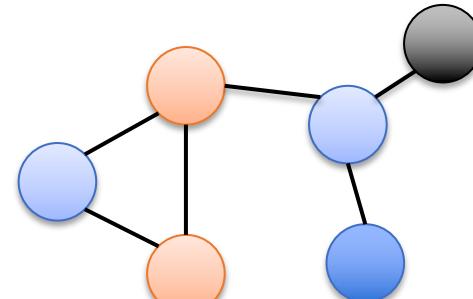
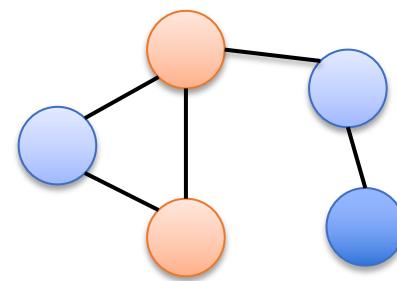


The goal of GNN is to transform **node features** to **features that are aware of the graph structure**



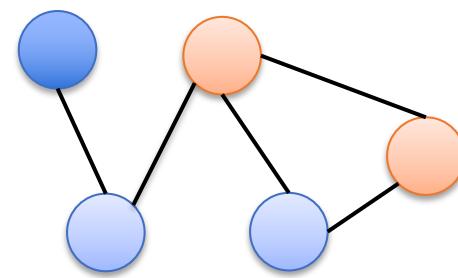
Problem with Graph Data

Difference in Size and Shape



How to handle arbitrary input graph shape?

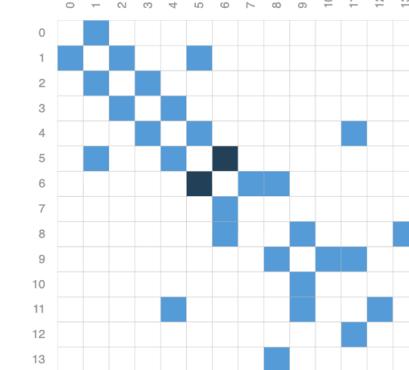
Isomorphism



Permutation invariant

Graph G	Graph H	An isomorphism between G and H
a b c d e f g h i j	1 2 3 4 5 6 7 8	$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

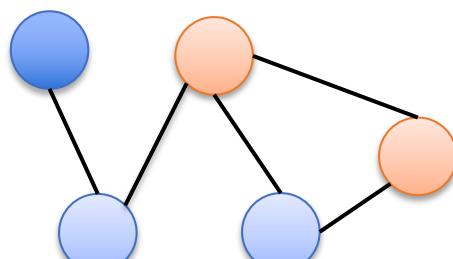
Cannot use adjacency matrix as an input



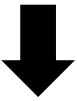
Problem with Graph Data



Isomorphism

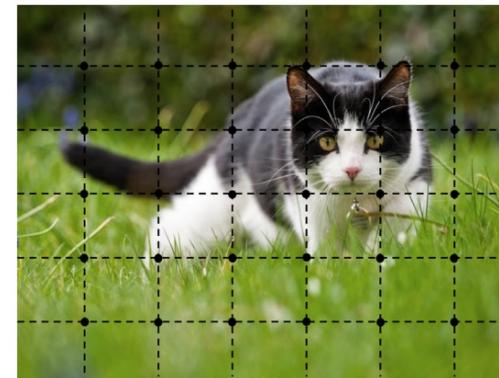
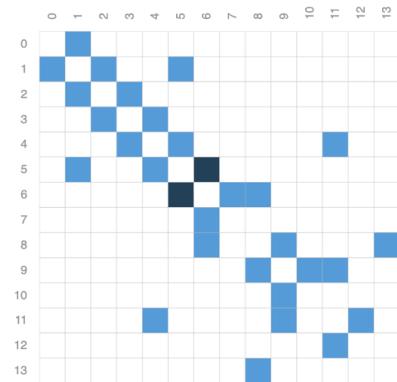


Permutation invariant

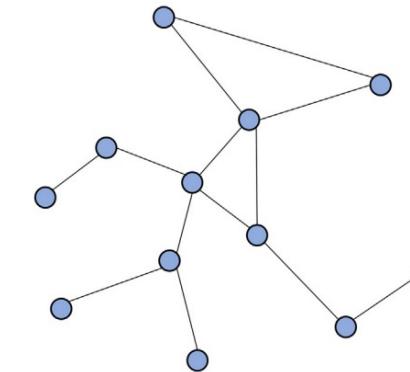


Graph G	Graph H	An isomorphism between G and H
		$\begin{aligned}f(a) &= 1 \\ f(b) &= 6 \\ f(c) &= 8 \\ f(d) &= 3 \\ f(g) &= 5 \\ f(h) &= 2 \\ f(i) &= 4 \\ f(j) &= 7\end{aligned}$

Cannot use adjacency matrix as an input



Euclidean space



Non-Euclidean space

Geometric Deep Learning

Information between nodes

Fundamental Idea of GNNs

Learning a neural networks suitable representation of graph data

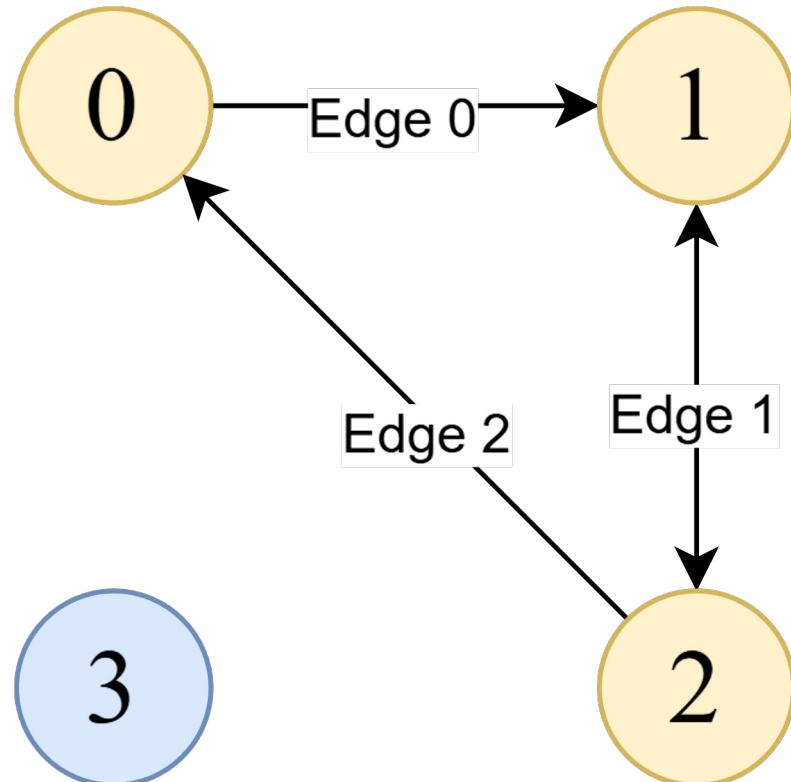
Representation learning

Sparse Graph Representation



Why Adjacency Lists?

$O(n_{edges})$ vs. $O(n^2)$



Adjacency Matrix	Adjacency List
$O(n^2)$ memory is used, which is constant	$O(n_{edges})$ memory use depends upon the number of edges in a graph
Slow to iterate over all edges	Fast to iterate over all edges
Slow in adding or deleting a node (can take $O(n^2)$ for complex operations)	Fast in terms of adding and deleting a node in a graph
Easy to implement (uses a 2D matrix as the data structure)	Hard to implement (uses vector or linked list as the data structure)

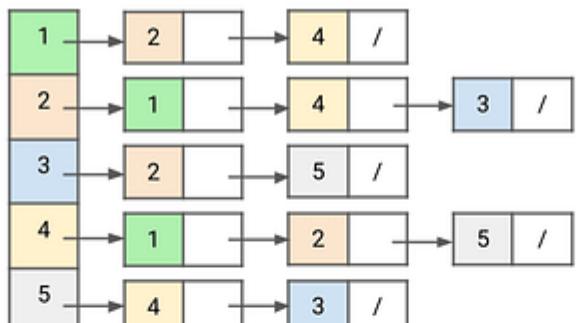
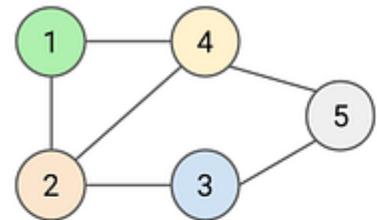
Nodes:
[1, 1, 1, 0]

Adjacency List:
[[0, 1], [1, 2], [2, 0]]

Edges:
[0, 1, 0]

Global:
1

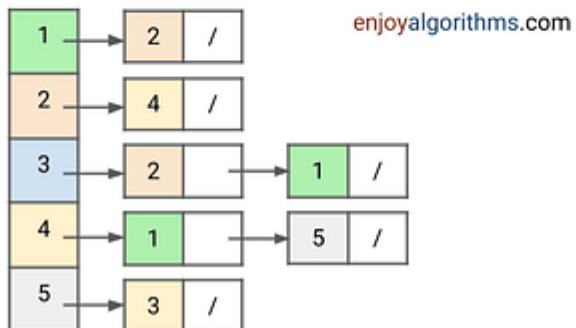
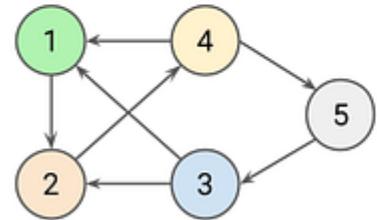
Sparse Graph Representation



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

Adjacency Matrix Representation

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



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

GNN take input from 4 tensors

Node Position:
[num_nodes, num_dimensions]

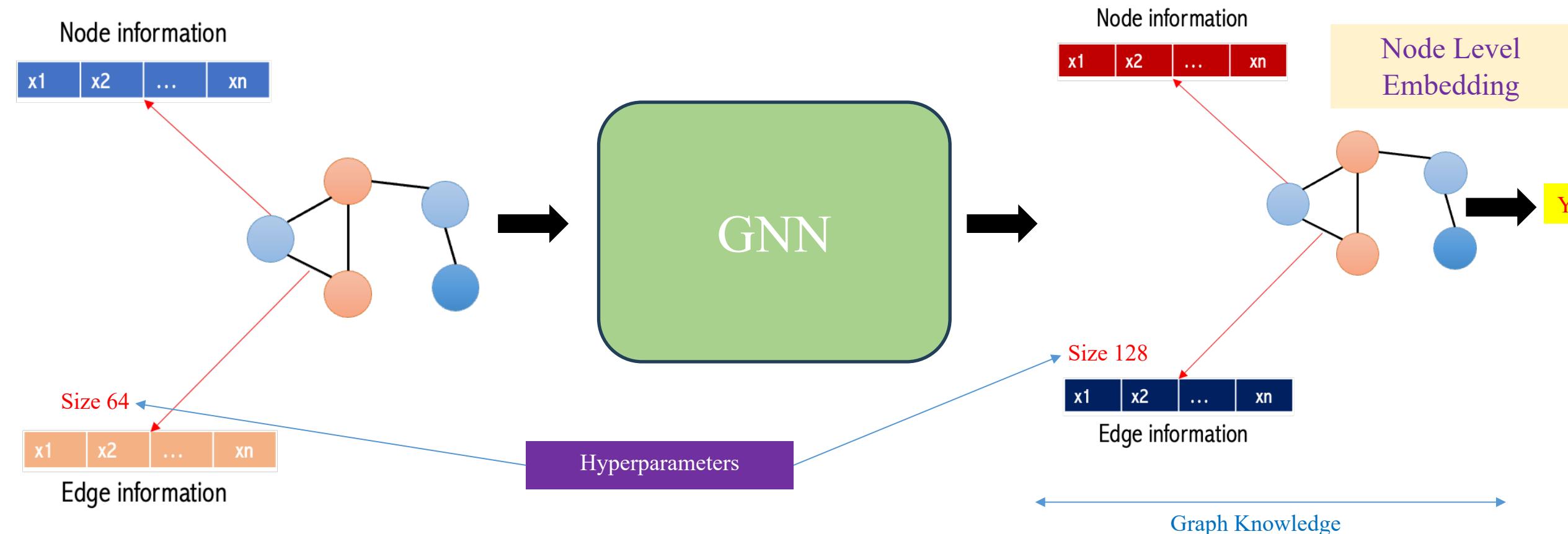
Node Features:
[num_nodes, num_node_features]

Edge Position:
[2, num_edges]

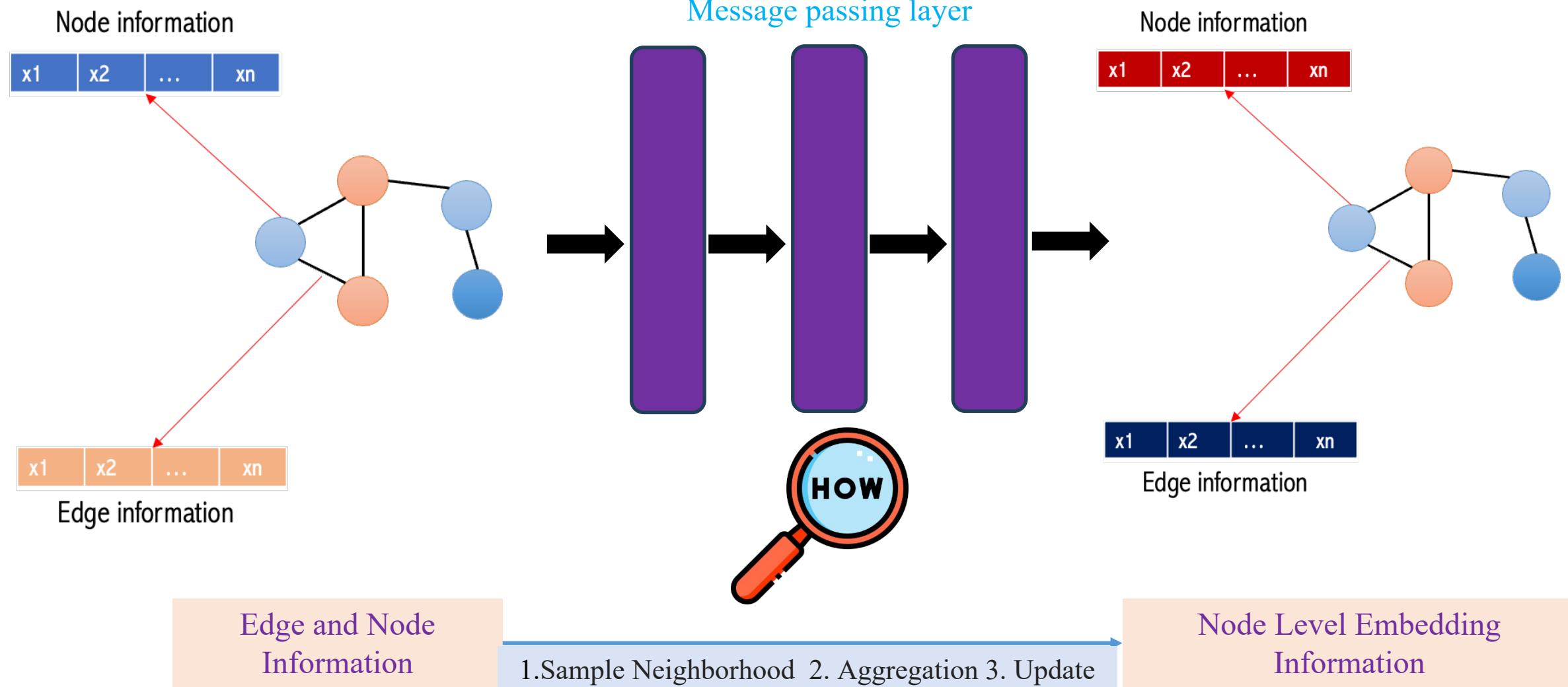
Edge Features:
[num_edges, num_edge_features]

Fundamental Idea of GNNs

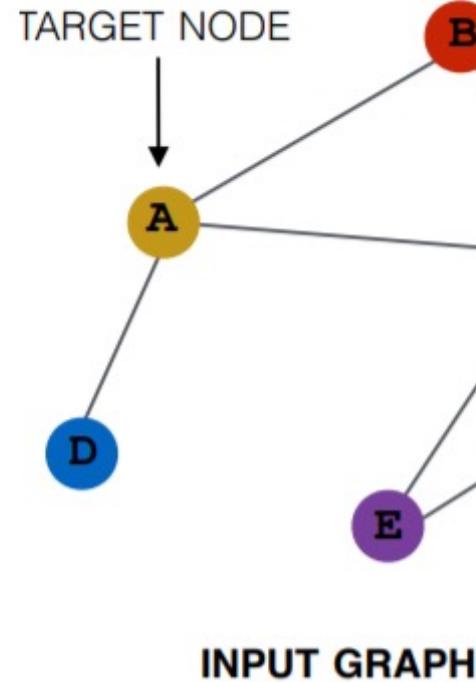
Learning a for neural network suitable representation of graph data
<<Representation Learning>>



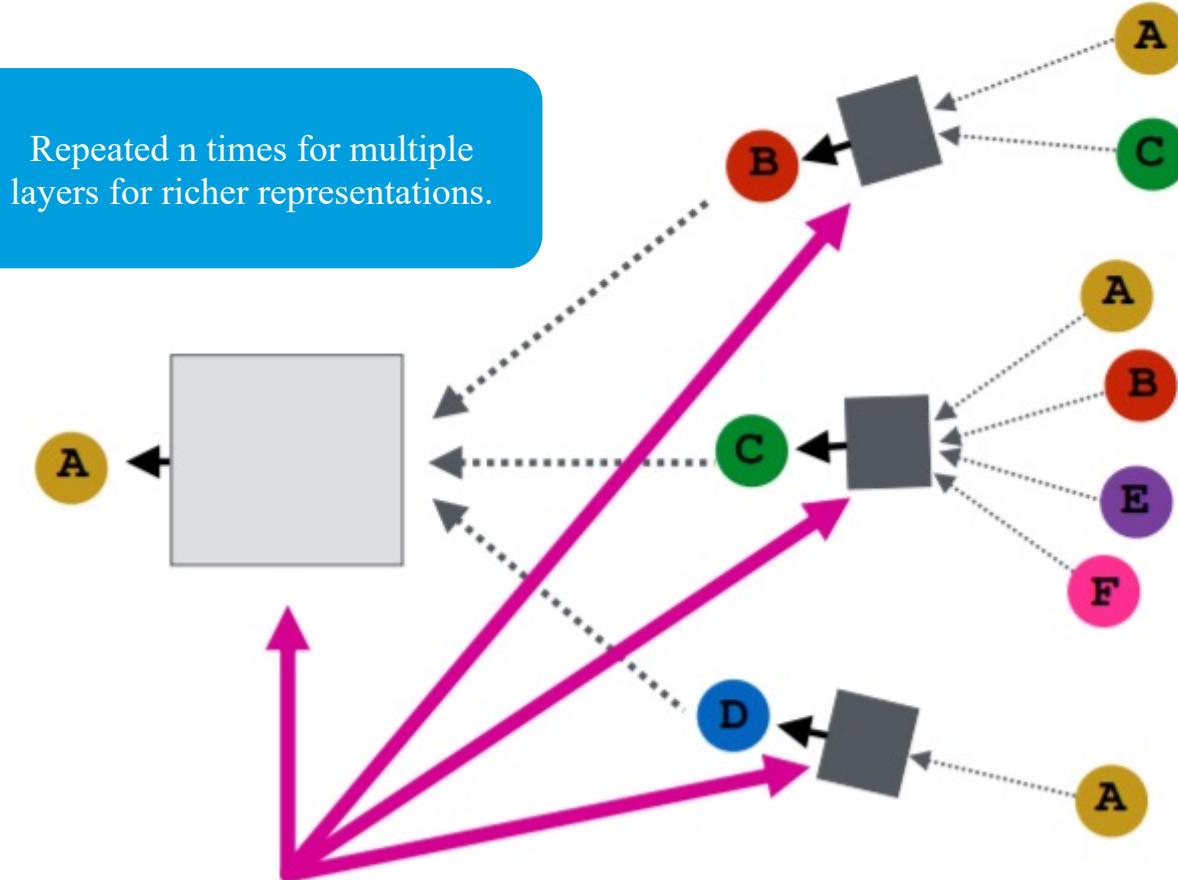
How do Graph Neural Network Work?



How do Graph Neural Network Work?



Repeated n times for multiple layers for richer representations.

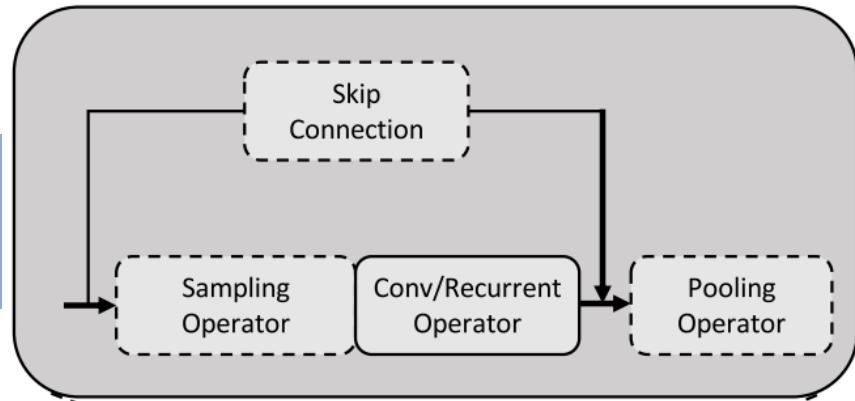


Neural networks

Each node gathers features from its neighbors.

How do Graph Neural Network Work?

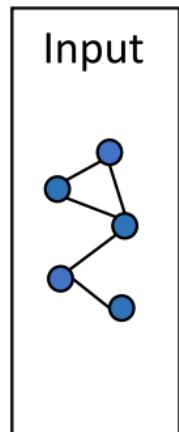
Sampling when graph contains a very large number of nodes/edges.



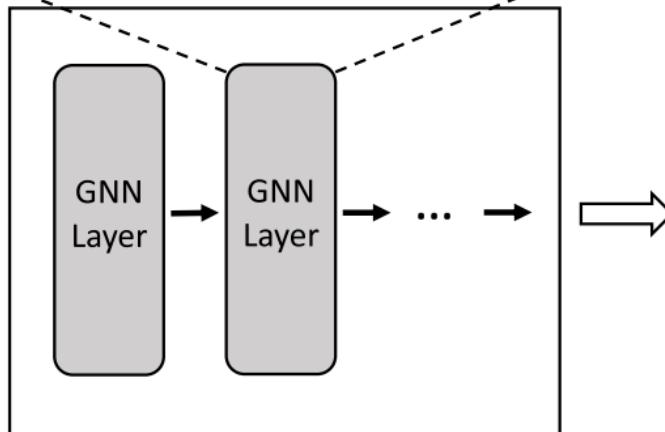
Sampling when graph contains a very large number of nodes/edges.

Pool features if needed for graph-level tasks

Find graph structure, set node/edge features

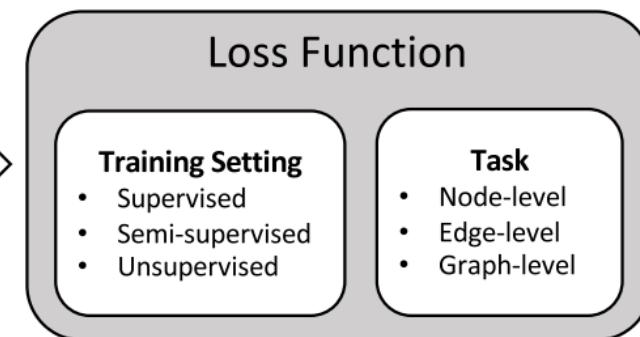


GNN layers apply message passing / convolution



Output
Node Embedding
Edge Embedding
Graph Embedding
...

Obtain final embeddings, apply a readout network



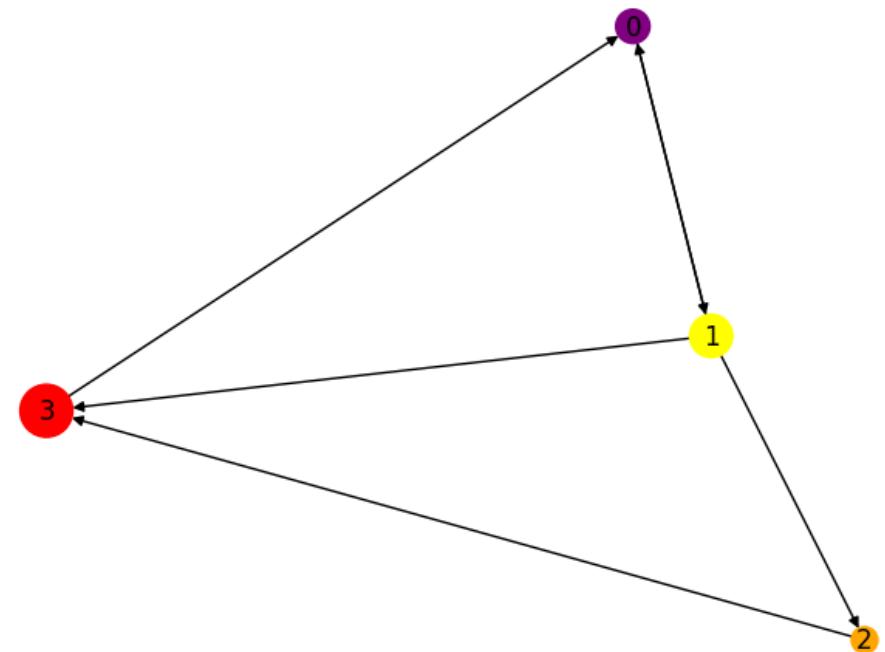
Create a graph using NetworkX

```
import networkx as nx
H = nx.DiGraph()

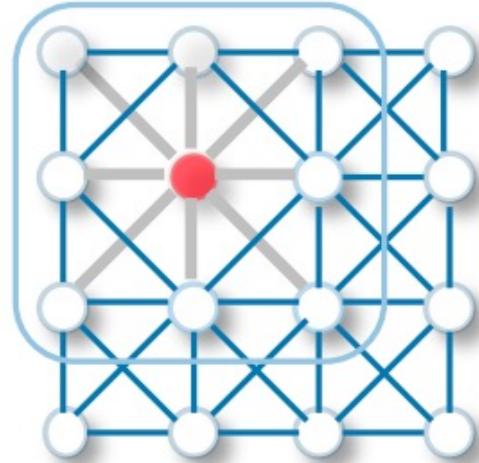
#adding nodes
H.add_nodes_from([
    (0, {"color": "purple", "size": 250}),
    (1, {"color": "yellow", "size": 400}),
    (2, {"color": "orange", "size": 150}),
    (3, {"color": "red", "size": 600})
])

#adding edges
H.add_edges_from([
    (0, 1), (1, 2), (1, 0), (1, 3), (2, 3), (3,0)
])
|_
node_colors = nx.get_node_attributes(H, "color").values()
colors = list(node_colors)
node_sizes = nx.get_node_attributes(H, "size").values()
sizes = list(node_sizes)

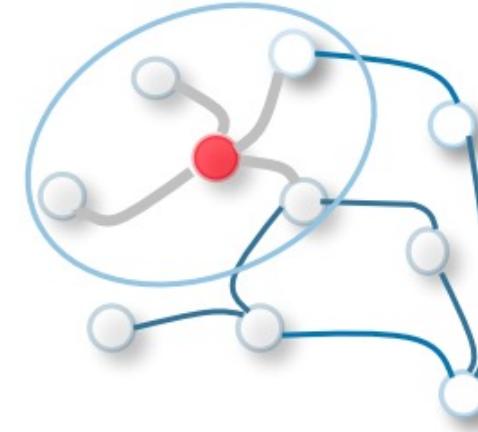
#Plotting Graph
nx.draw(H, with_labels=True, node_color=colors, node_size=sizes)
```



CNN Vs. GNN: Messsage Passing

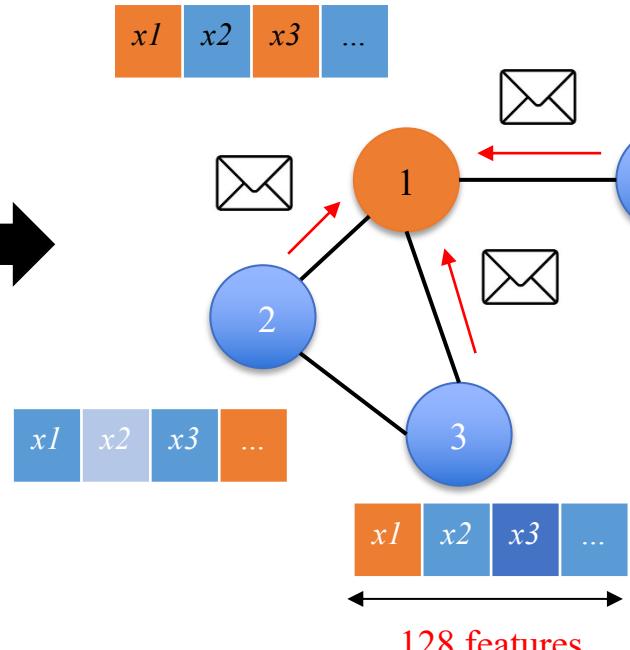
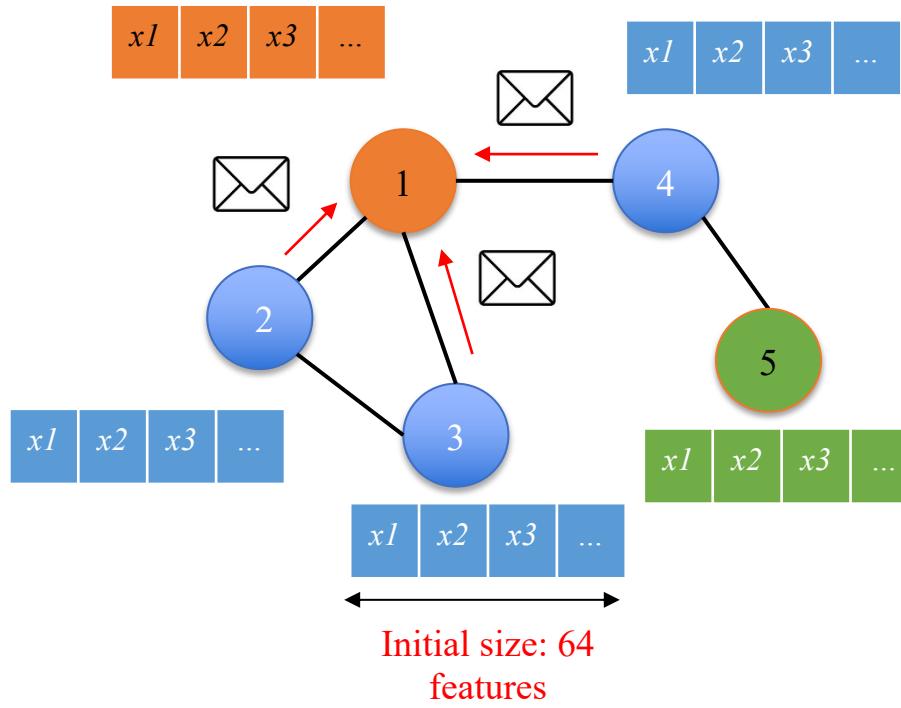


2D Convolution. Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes the weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size

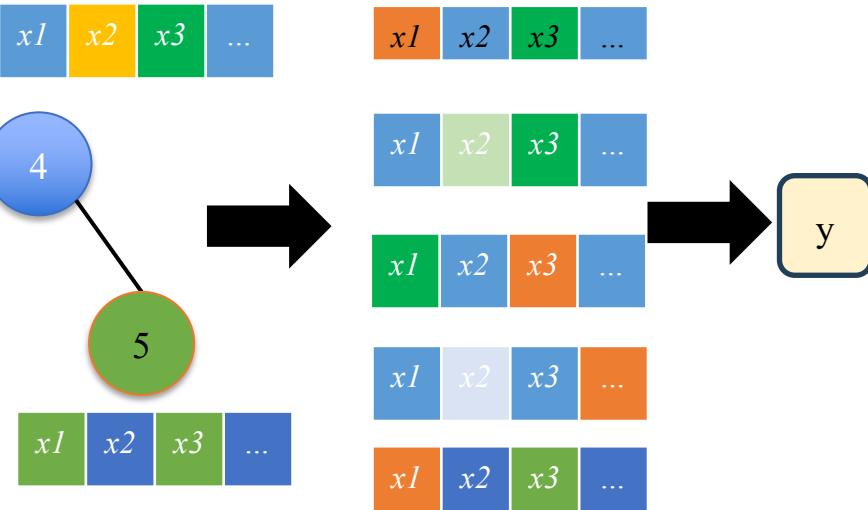


Graph Convolution. To get a hidden representation of the red node, one simple solution of the graph convolutional operation is to take the average value of the node features of the red node along with its neighbors. Different from image data, the neighbors of a node are unordered and variable in size

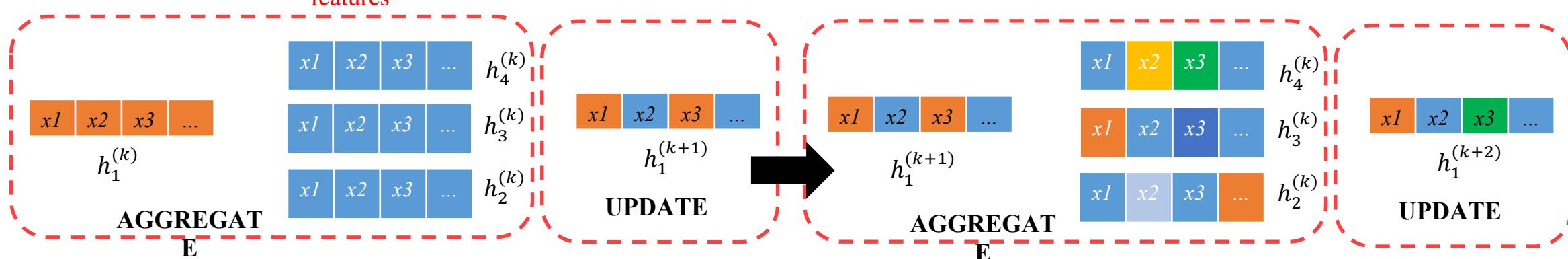
Message Passing: Behind the Scene



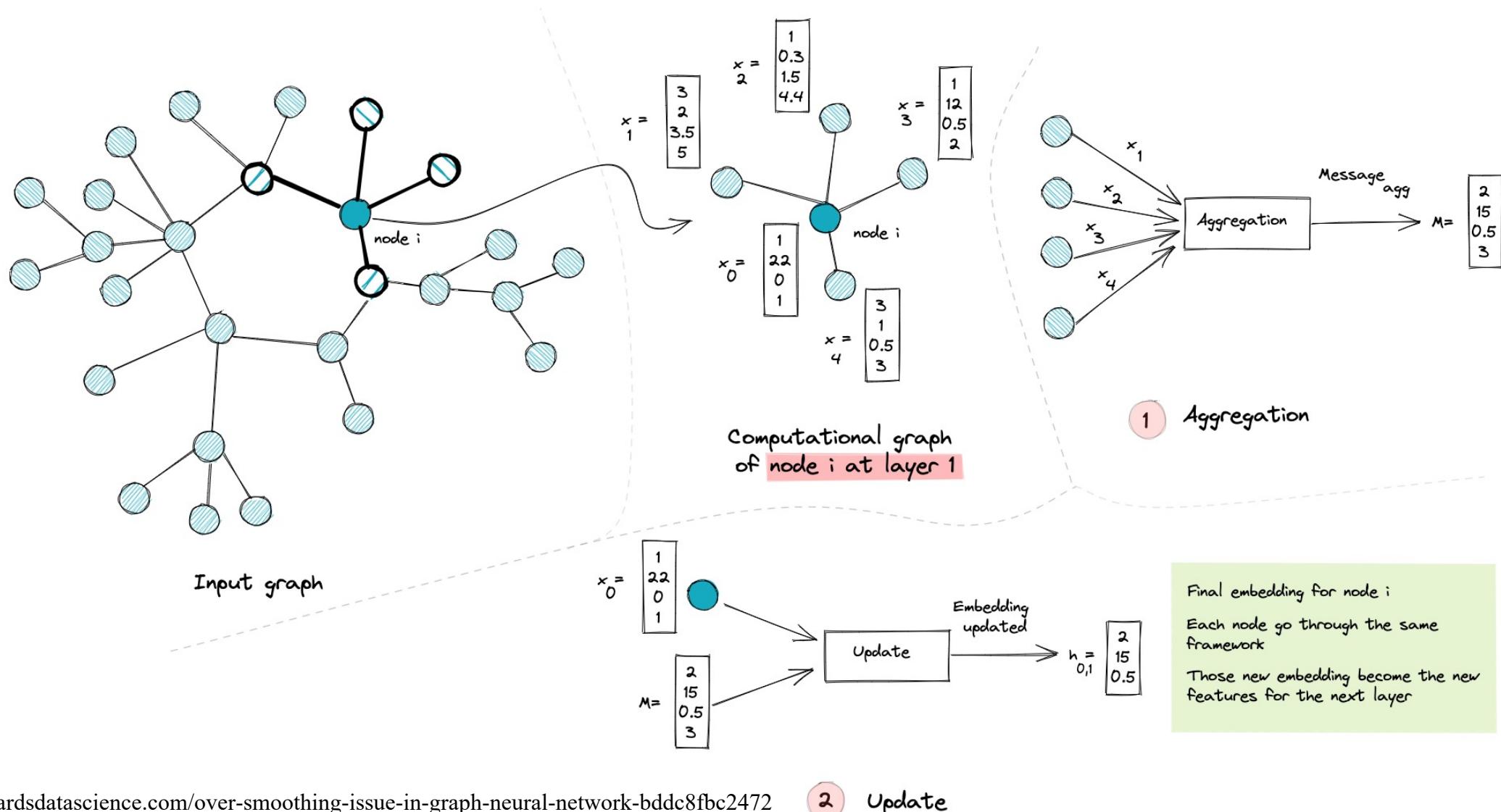
The local feature aggregation \neq CNN Kernel



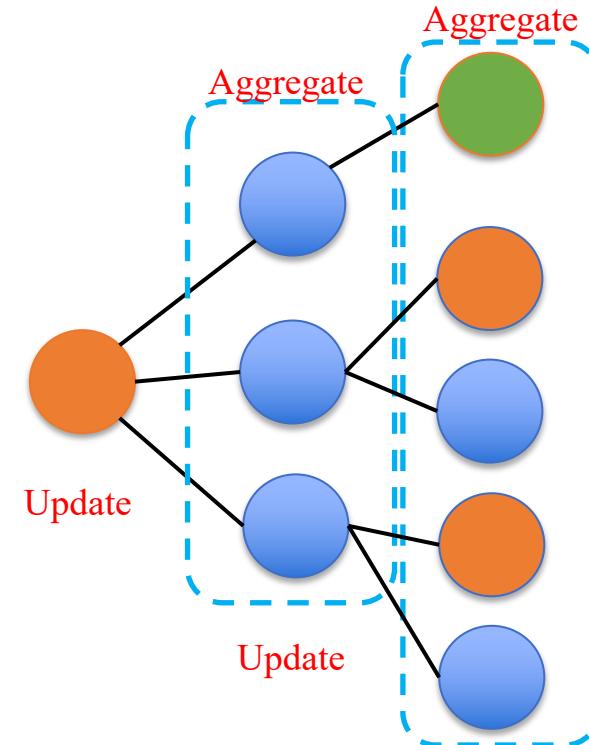
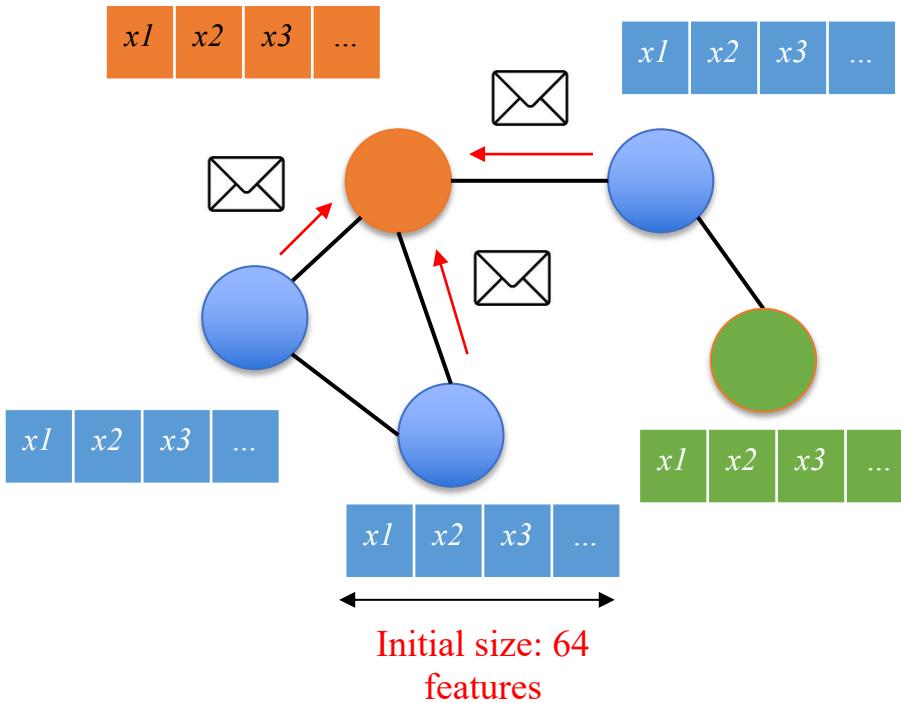
Structural and feature



Graph: Example

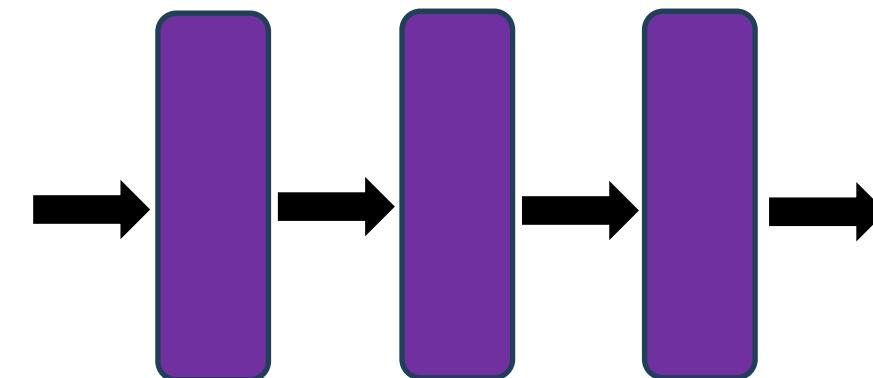


Computation Graph Representation



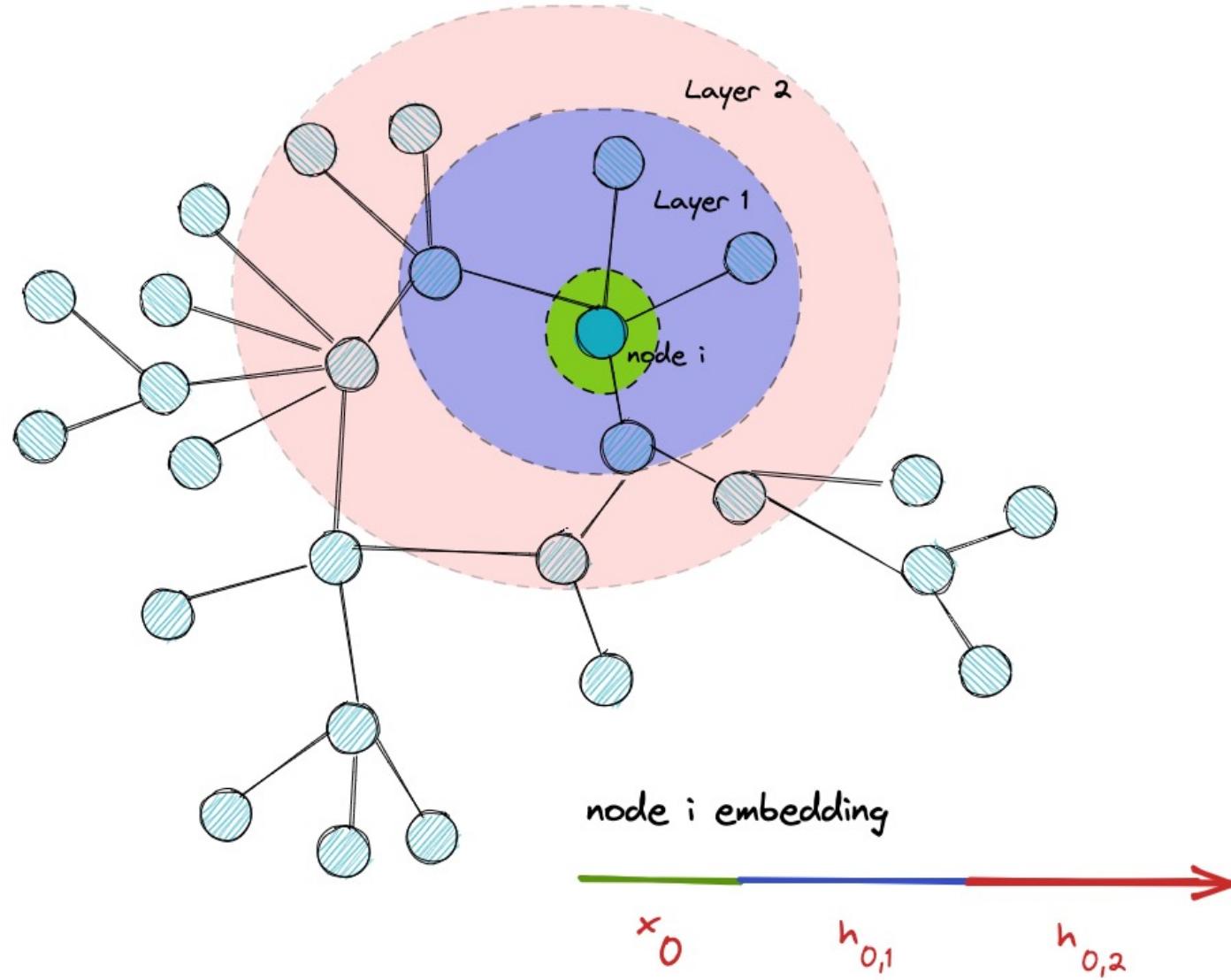
Computational Graph
for Node v

The number of layers
defined by how many
neighborhood nodes

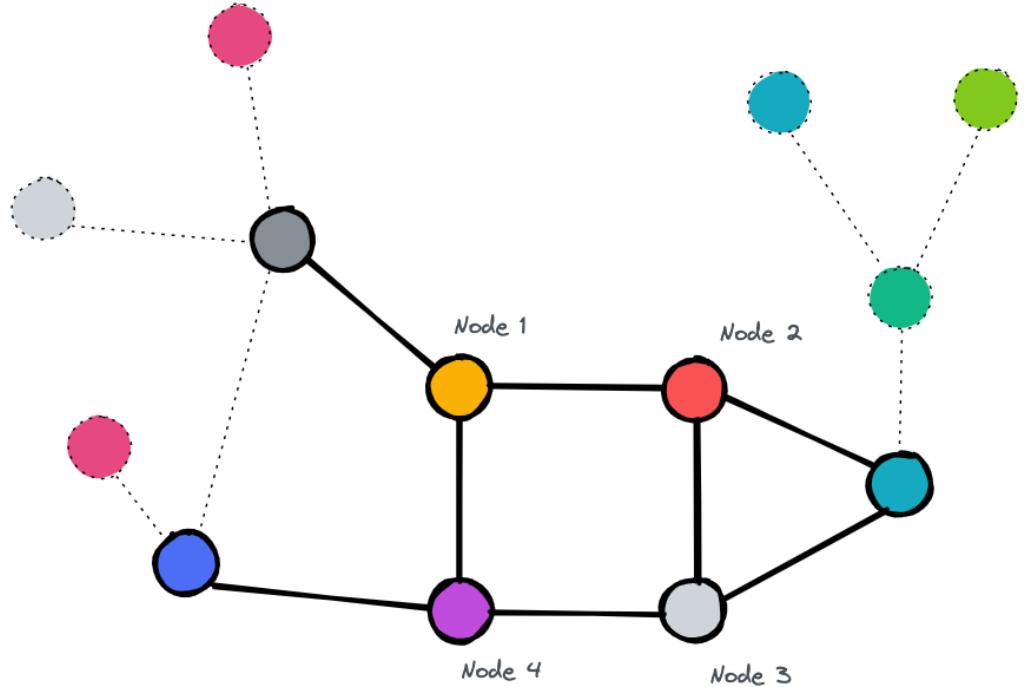


The number of MP-Layer
is a hyper parameters

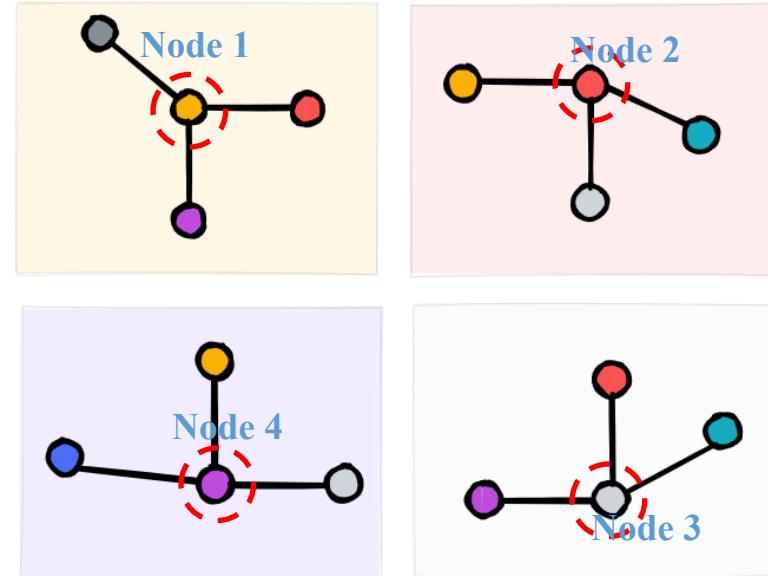
Graph: Example



Over-smoothing in GNN



Input graph



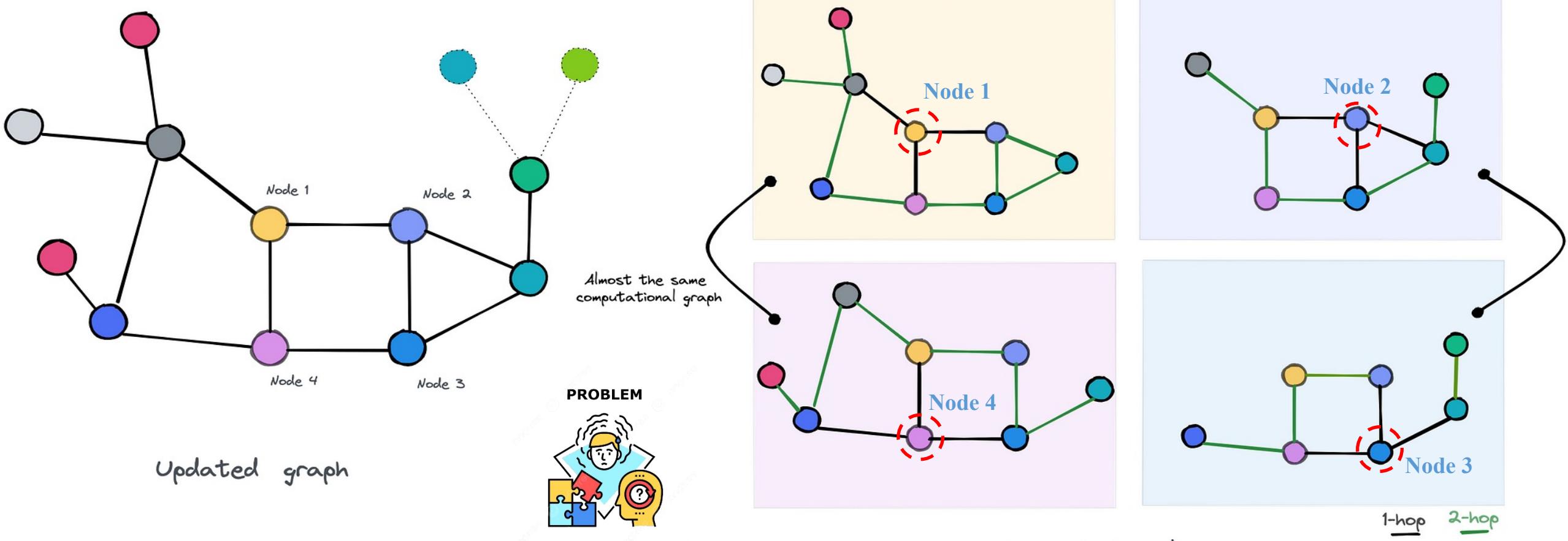
Computational graphs
at layer 1



Node 2 and node 3 have almost access to the same information -> We can predict that their embeddings will be slightly similar.

Node 1 and Node 4, they interact with each other but have different neighbors -> We may predict that their new embeddings will be different.

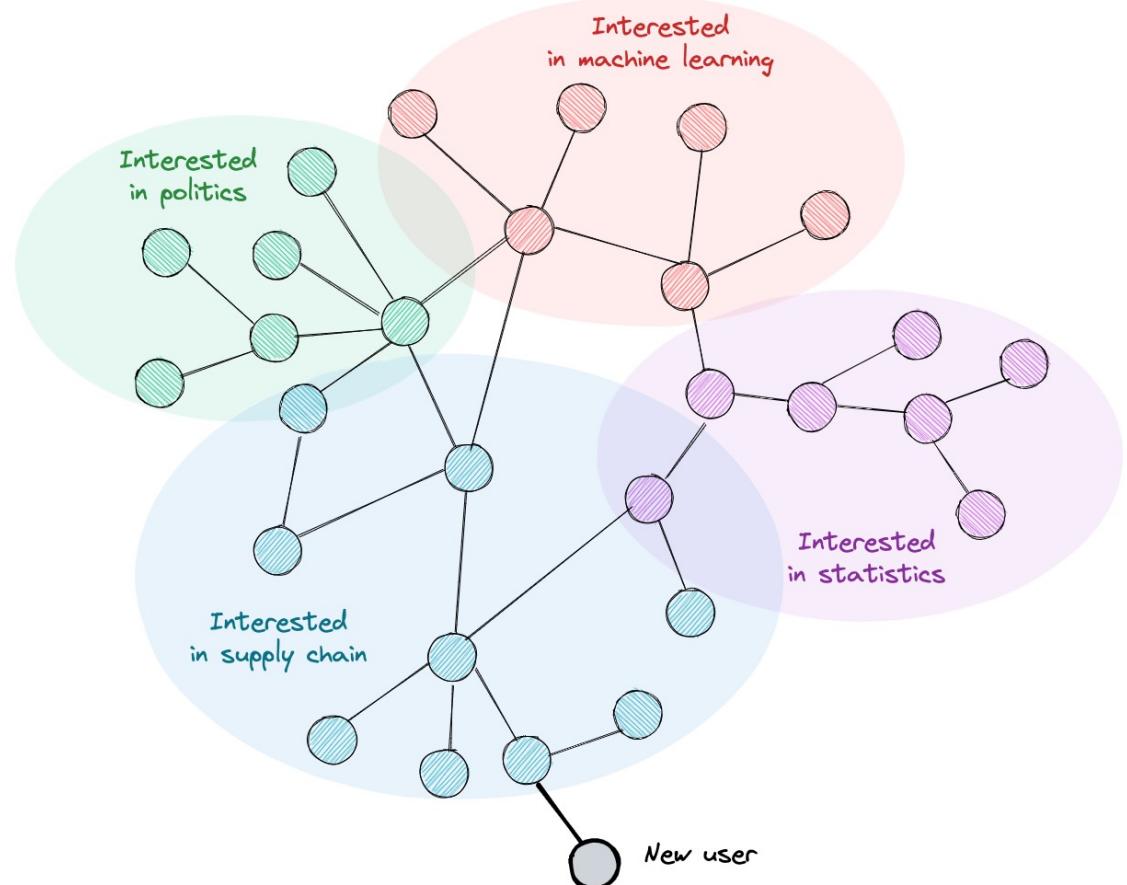
Over-smoothing in GNN



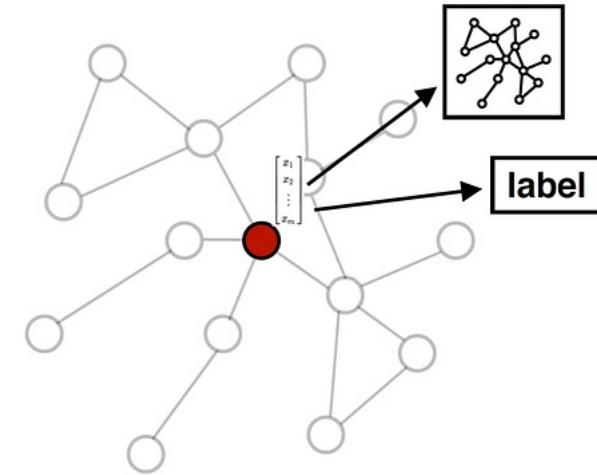
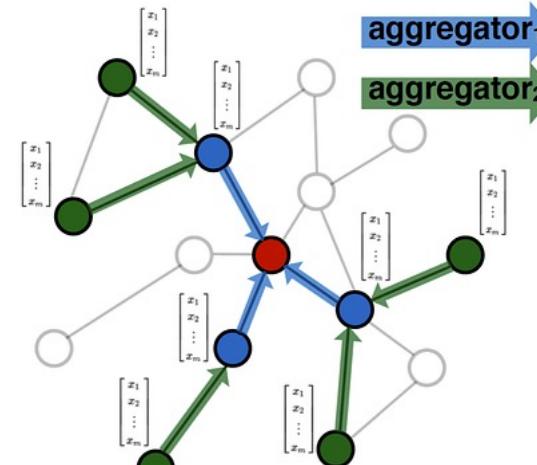
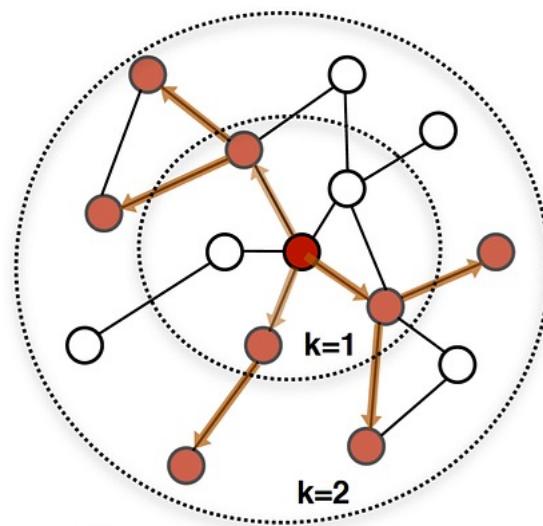
The computational graphs of nodes 1,4, and 2,3 are almost the same respectively

Over-smoothing in GNN

Graph Neural Networks, or GNNs, are really good at working with data that is organized in a graph structure. But sometimes, when we add more layers to a GNN architecture, it doesn't work as well as we would like. This is called over-smoothing.

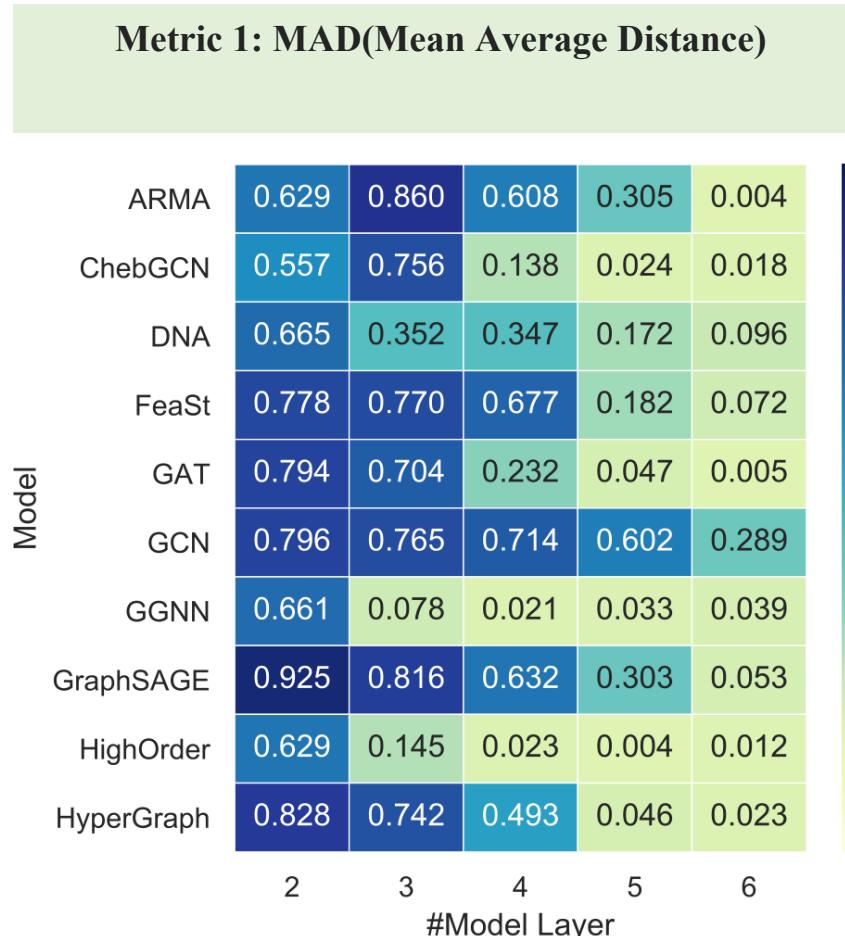


Why over-smoothing happens?



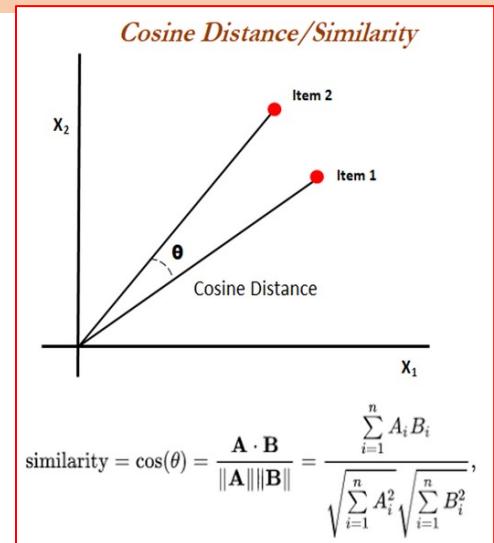
Reason 1 → More number of layers (depth)
 Reason 2 → Default nature of GNNs

How to detect over-smoothing?



MAD computes the Mean Average Distance (MAD) between node representations (embeddings) in the graph

$$D_{ij} = 1 - \frac{\mathbf{H}_{i,:} \cdot \mathbf{H}_{j,:}}{|\mathbf{H}_{i,:}| \cdot |\mathbf{H}_{j,:}|} \quad i, j \in [1, 2, \dots, n],$$



Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View

Deli Chen,¹ Yankai Lin,² Wei Li,¹ Peng Li,² Jie Zhou,² Xu Sun¹

¹MOE Key Lab of Computational Linguistics, School of EECS, Peking University

²Pattern Recognition Center, WeChat AI, Tencent Inc., China

{chendeli, liweitj47, xusun}@pku.edu.cn, {yankailin, patrickpli, withtomzhou}@tencent.com,

How to detect over-smoothing?

Metric 2 : MADGap

$$\text{MADGap} = \text{MAD}^{\text{rmt}} - \text{MAD}^{\text{neb}}, \quad (5)$$

where MAD^{rmt} is the MAD value of the remote nodes in the graph topology and MAD^{neb} is the MAD value of the neighbouring nodes.

It is based on the main hypothesis that when nodes interact, they have access to either important information from nodes of the same class or noise from nodes of other classes.

Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View

Deli Chen,¹ Yankai Lin,² Wei Li,¹ Peng Li,² Jie Zhou,² Xu Sun¹

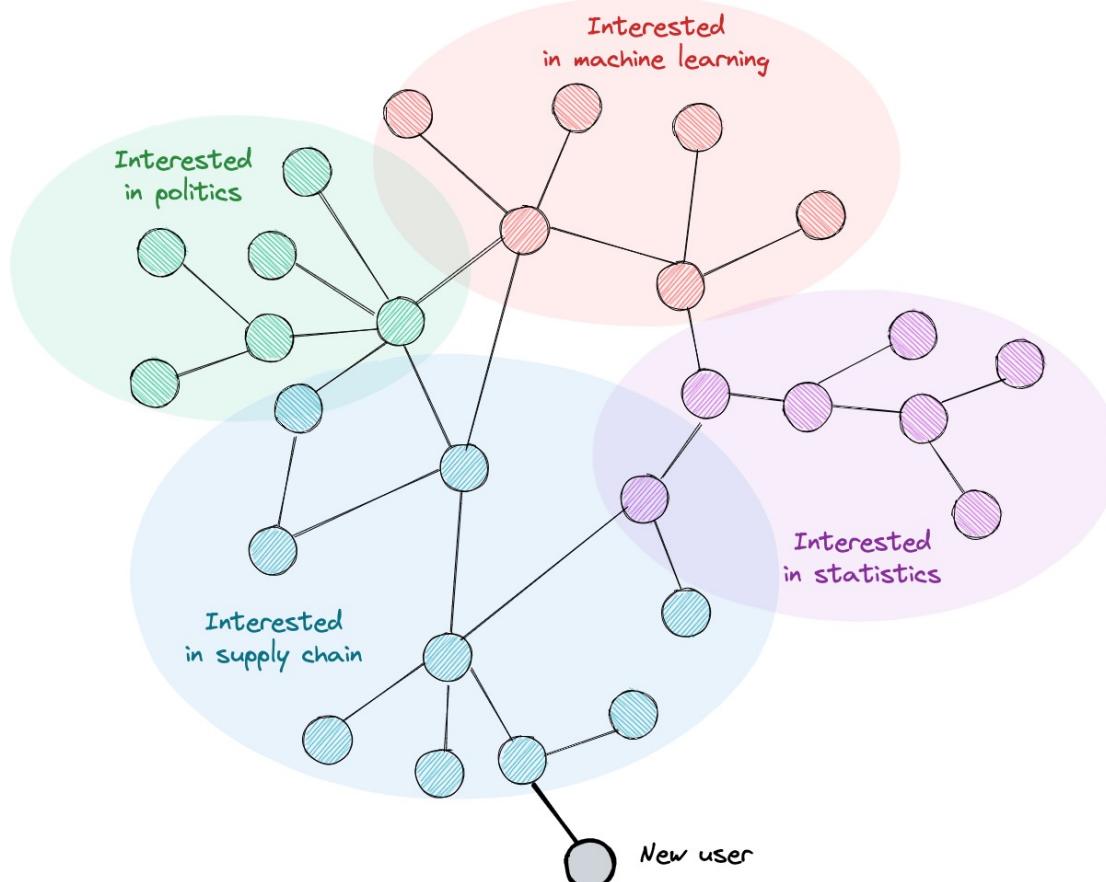
¹MOE Key Lab of Computational Linguistics, School of EECS, Peking University

²Pattern Recognition Center, WeChat AI, Tencent Inc., China

{chendeli,liweitj47,xusun}@pku.edu.cn, {yankailin,patrickpli,withtomzhou}@tencent.com,



Over-smoothing in GNN



How to reduce the effect of over-smoothing.

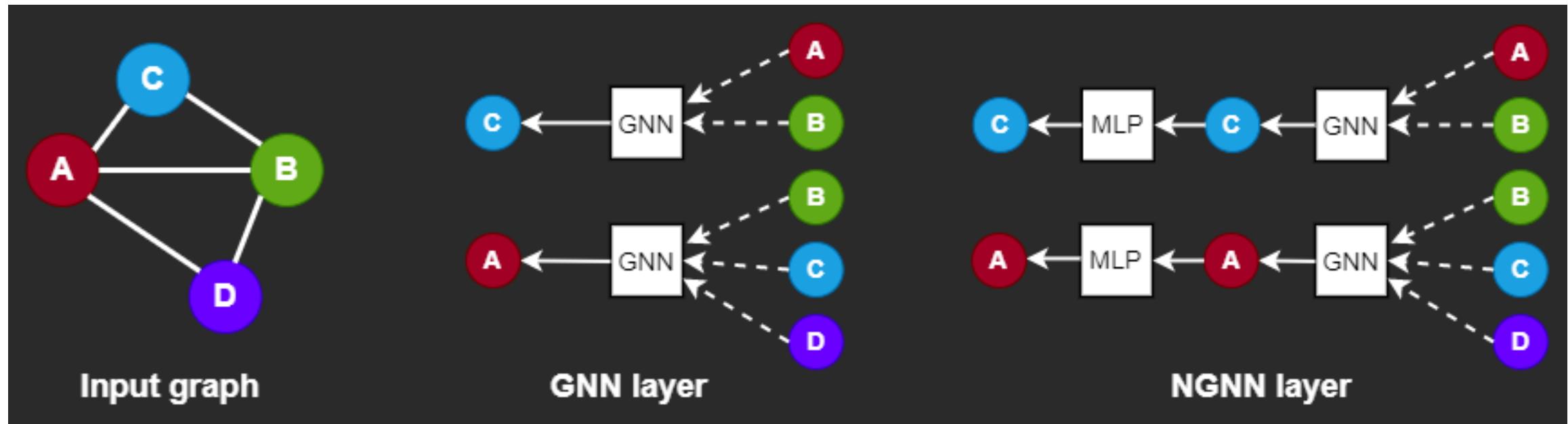
We encounter a trade-off between a low-efficiency model and a model with more depth but less expressivity in terms of node representations



Imagine that we're dealing with a social network graph with thousands of nodes. Some new users just signed in to the platform and subscribed to their friend's profiles. Our goal is to find topic suggestions to fill their feed.

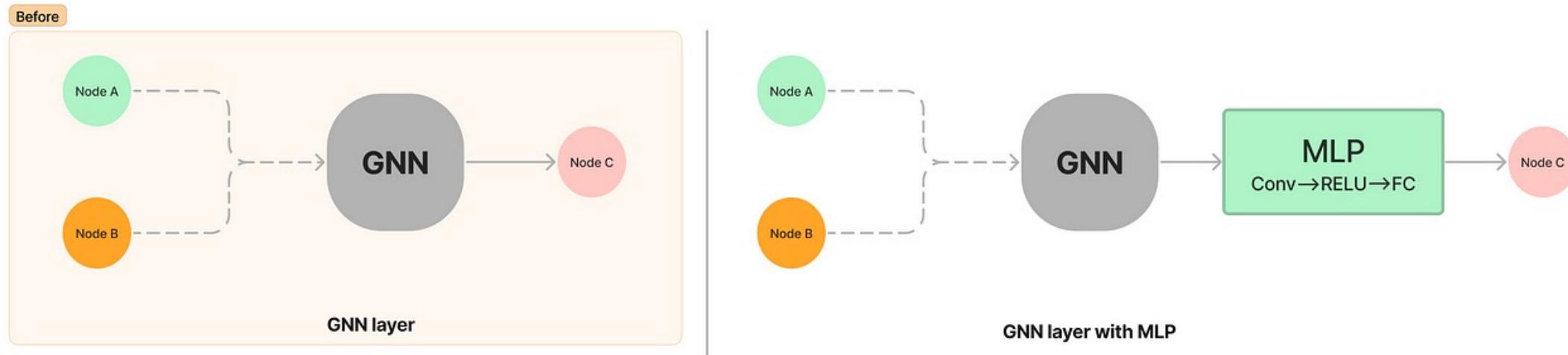
Over-smoothing in GNN

Solution → Inserting nonlinear feedforward neural network layer(s) within each GNN layer.



Over-smoothing in GNN

Solution → Inserting nonlinear feedforward neural network layer(s) within each GNN layer.

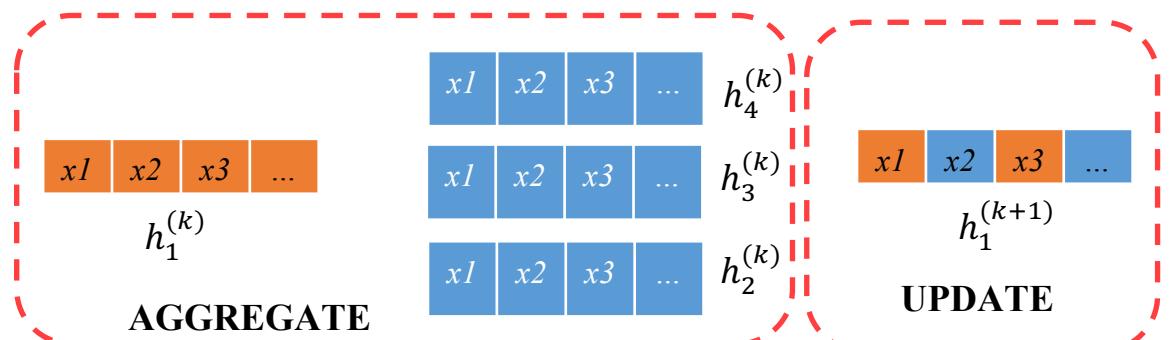
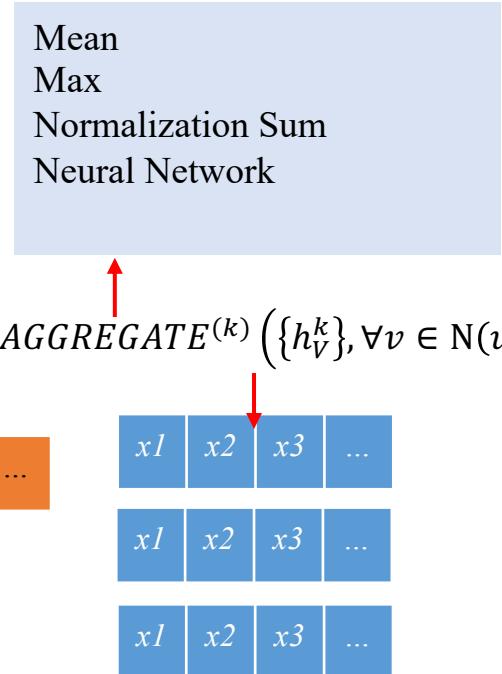
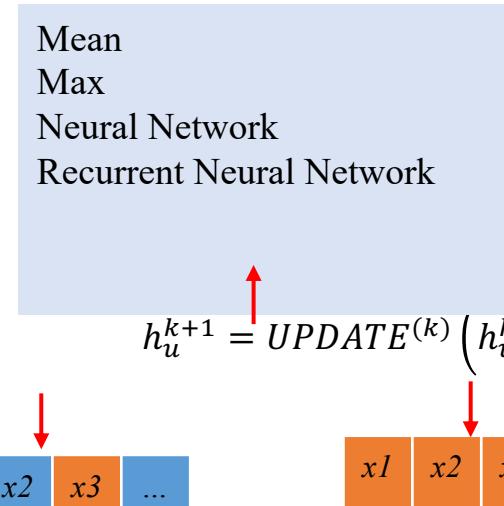
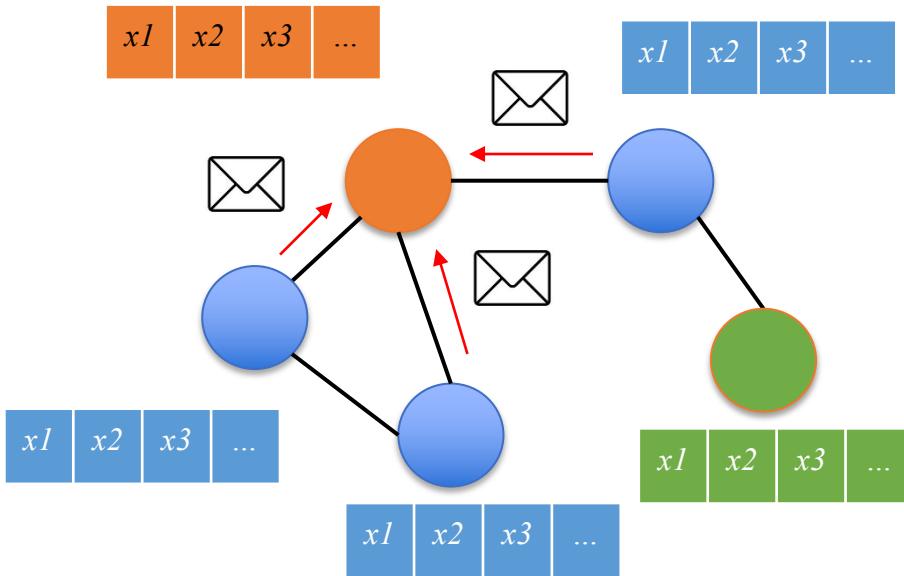


Over-smoothing in GNN

Solution → Inserting nonlinear feedforward neural network layer(s) within each GNN layer.

Dataset	Metric	Model	Performance	
ogbn-proteins	ROC-AUC(%)	GraphSage+Cluster Sampling	Vanilla	67.45 ± 1.21
			+NGNN	68.12 ± 0.96
ogbn-products	Accuracy(%)	GraphSage	Vanilla	78.27 ± 0.45
			+NGNN	79.88 ± 0.34
		GAT+Neighbor Sampling	Vanilla	79.23 ± 0.16
			+NGNN	79.67 ± 0.09
ogbl-collab	hit@50(%)	GCN	Vanilla	49.52 ± 0.70
			+NGNN	53.48 ± 0.40
		GraphSage	Vanilla	51.66 ± 0.35
			+NGNN	53.59 ± 0.56
ogbl-ppa	hit@100(%)	SEAL-DGCNN	Vanilla	48.80 ± 3.16
			+NGNN	59.71 ± 2.45
		GCN	Vanilla	18.67 ± 1.32
			+NGNN	36.83 ± 0.99

Message Passing: Math is Fun



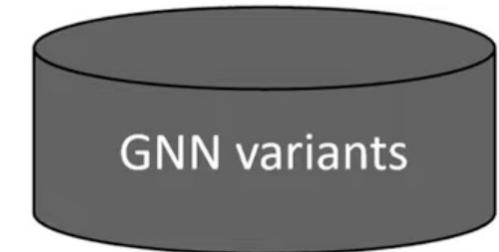
Message Passing: Math is Fun

Differentiable functions

$$h_u^{(k+1)} = \text{UPDATE}^{(k)} \left(h_u^{(k)}, \text{AGGREGATE}^{(k)}(\{h_v^{(k)}, \forall v \in \mathcal{N}(u)\}) \right)$$

- Mean
- Max
- Neural Network
- Recurrent NN

- Mean
- Max
- Normalized Sum
- Neural Network



Message Passing: Variants



Graph Convolutional Networks,
Kipf and Welling [2016]

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

Sum of normalized neighbor embeddings

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

Aggregated message

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

Send states through a MLP

Graph Attention Networks,
Veličković et al. [2017]

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} \mathbf{h}_v$$

Attention weights

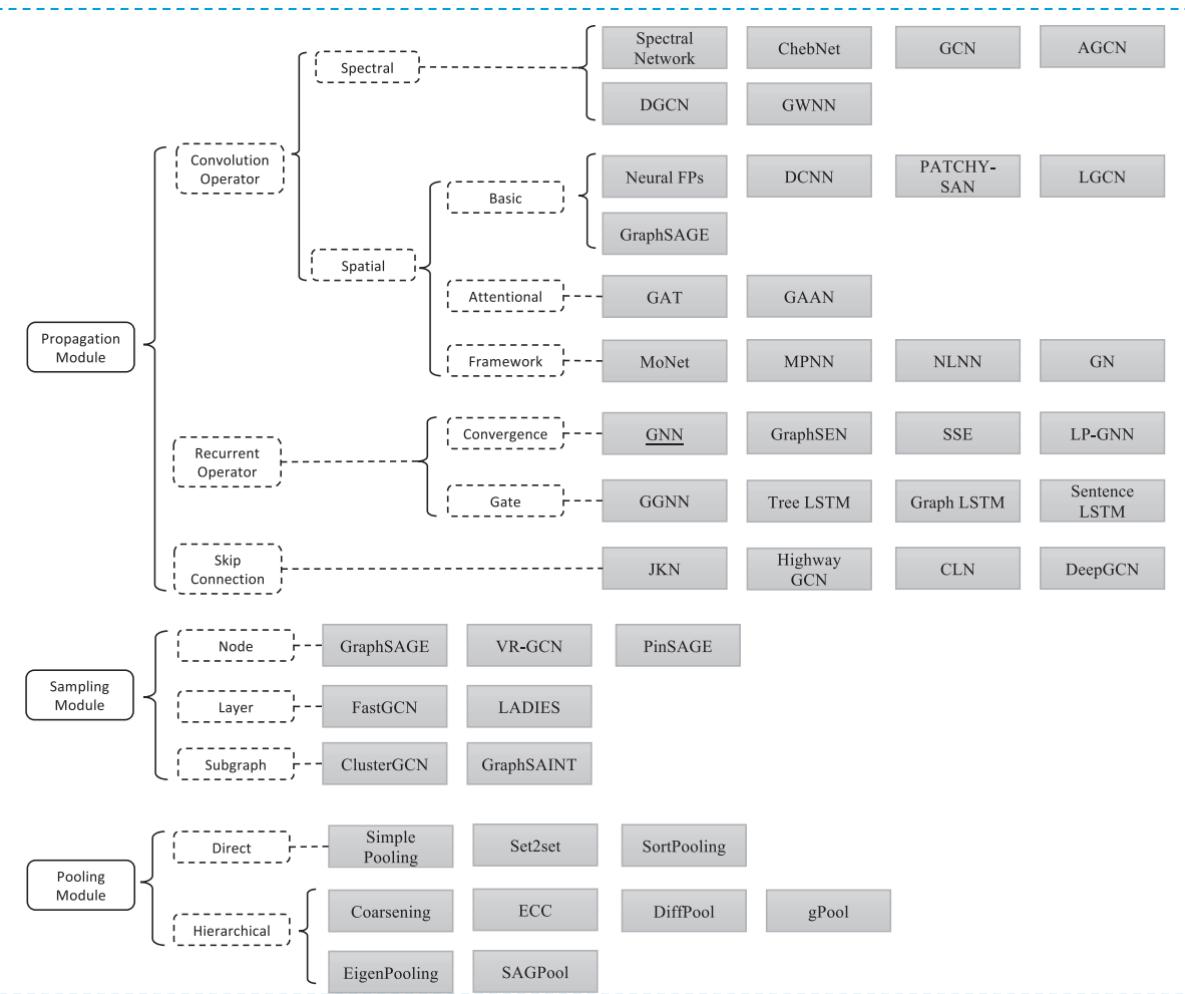
$$\alpha_{u,v} = \frac{\exp(\mathbf{a}^\top [\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_v])}{\sum_{v' \in \mathcal{N}(u)} \exp(\mathbf{a}^\top [\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_{v'}])}$$

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

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

Recurrent update of the state

Message Passing: Variants



Different variants of recurrent operators.

Variant	Aggregator	Updater
GGNN	$\mathbf{h}_{\mathcal{N}_v}^t = \sum_{k \in \mathcal{N}_v} \mathbf{h}_k^{t-1} + \mathbf{b}$	$\mathbf{z}_v^t = \sigma(\mathbf{W}^z \mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}^z \mathbf{h}_v^{t-1})$ $\mathbf{r}_v^t = \sigma(\mathbf{W}^r \mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}^r \mathbf{h}_v^{t-1})$ $\tilde{\mathbf{h}}_v^t = \tanh(\mathbf{W}^h \mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}(\mathbf{r}_v^t \odot \mathbf{h}_v^{t-1}))$ $\mathbf{h}_v^t = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{t-1} + \mathbf{z}_v^t \odot \tilde{\mathbf{h}}_v^t$
Tree LSTM (Child sum)	$\mathbf{h}_{\mathcal{N}_v}^t = \sum_{k \in \mathcal{N}_v} \mathbf{U}^i \mathbf{h}_k^{t-1}$ $\mathbf{h}_{\mathcal{N}_v,k}^f = \mathbf{U}^f \mathbf{h}_k^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{to} = \sum_{k \in \mathcal{N}_v} \mathbf{U}^o \mathbf{h}_k^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^u = \sum_{k \in \mathcal{N}_v} \mathbf{U}^u \mathbf{h}_k^{t-1}$	$\mathbf{i}_v^t = \sigma(\mathbf{W}^i \mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^t + \mathbf{b}^i)$ $\mathbf{f}_v^t = \sigma(\mathbf{W}^f \mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v,k}^f + \mathbf{b}^f)$ $\mathbf{o}_v^t = \sigma(\mathbf{W}^o \mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{to} + \mathbf{b}^o)$ $\mathbf{u}_v^t = \tanh(\mathbf{W}^u \mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^u + \mathbf{b}^u)$ $\mathbf{c}_v^t = \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{k \in \mathcal{N}_v} \mathbf{f}_v^t \odot \mathbf{c}_k^{t-1}$ $\mathbf{h}_v^t = \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)$
Tree LSTM (N-ary)	$\mathbf{h}_{\mathcal{N}_v}^t = \sum_{l=1}^K \mathbf{U}_l^i \mathbf{h}_{vl}^{t-1}$ $\mathbf{h}_{\mathcal{N}_v,k}^f = \sum_{l=1}^K \mathbf{U}_{kl}^f \mathbf{h}_{vl}^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{to} = \sum_{l=1}^K \mathbf{U}_l^o \mathbf{h}_{vl}^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^u = \sum_{l=1}^K \mathbf{U}_l^u \mathbf{h}_{vl}^{t-1}$	
Graph LSTM in (Peng et al., 2017)	$\mathbf{h}_{\mathcal{N}_v}^t = \sum_{k \in \mathcal{N}_v} \mathbf{U}_{m(v,k)}^i \mathbf{h}_k^{t-1}$ $\mathbf{h}_{\mathcal{N}_v,k}^f = \mathbf{U}_{m(v,k)}^f \mathbf{h}_k^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{to} = \sum_{k \in \mathcal{N}_v} \mathbf{U}_{m(v,k)}^o \mathbf{h}_k^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^u = \sum_{k \in \mathcal{N}_v} \mathbf{U}_{m(v,k)}^u \mathbf{h}_k^{t-1}$	

Graph neural networks: A review of methods and applications

Jie Zhou ^{a,1}, Ganqu Cui ^{a,1}, Shengding Hu ^a, Zhengyan Zhang ^a, Cheng Yang ^b, Zhiyuan Liu ^{a,*}, Lifeng Wang ^c, Changcheng Li ^c, Maosong Sun ^a

^a Department of Computer Science and Technology, Tsinghua University, Beijing, China

^b School of Computer Science, Beijing University of Posts and Telecommunications, China

^c Tencent Incorporation, Shenzhen, China

Outline

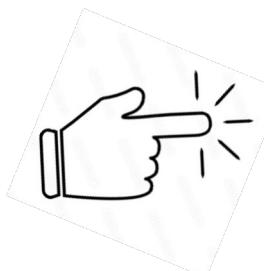
Objective

Introduction to Graph Data

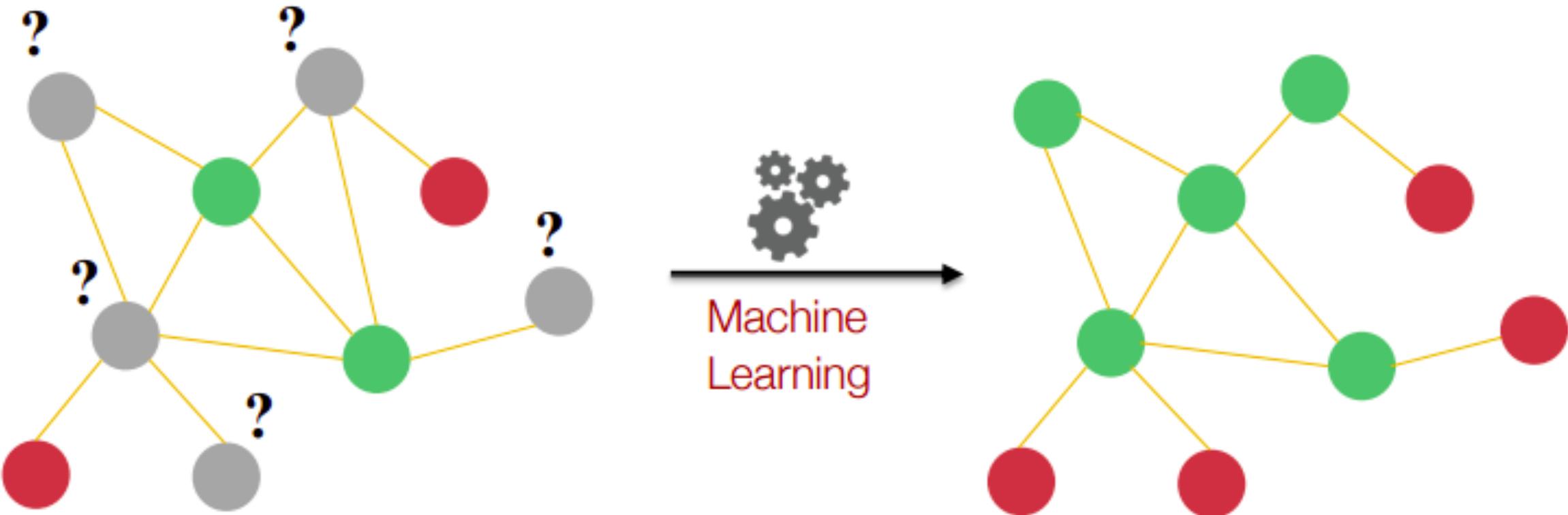
Graph Data with Neural Network

Node Classification Problem: Cora Citation Dataset

Summary



Node Classification Problem



GNN: Node Classification



Knowledge Graphs and Node Classification

We have one large graph and not many individual graphs (like molecules)

We infer on unlabeled nodes in this large graph and hence perform node-level predictions

--> We have to use different nodes of the graph depending on what we want to do



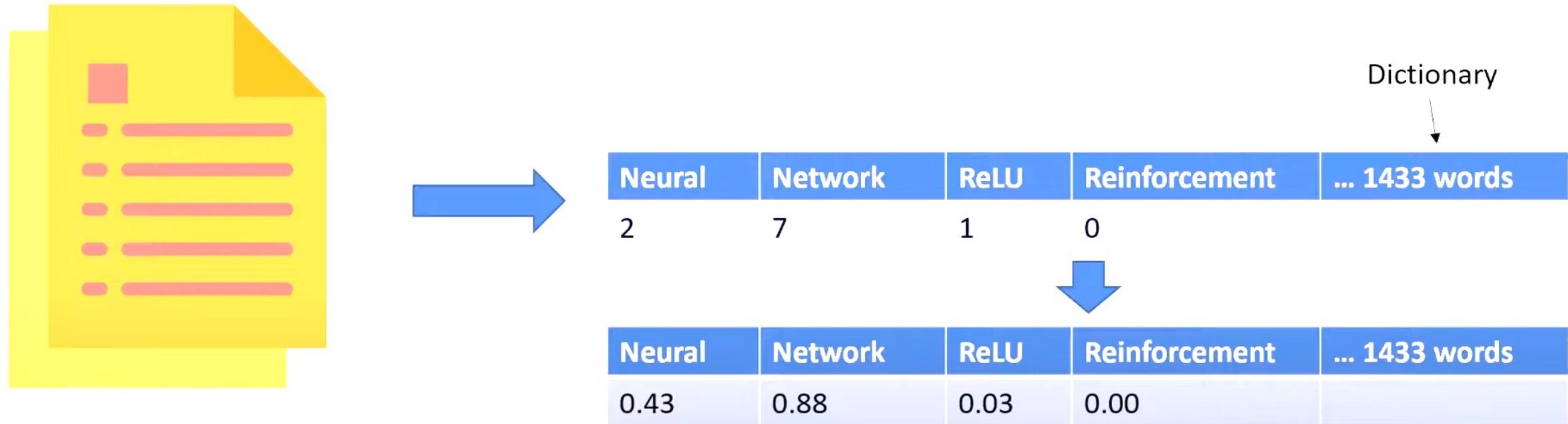
Dataset Introduction: Cora Citation Dataset in PyTorch Geometric

The Cora dataset consists of 2708 scientific publications classified into one of seven classes.

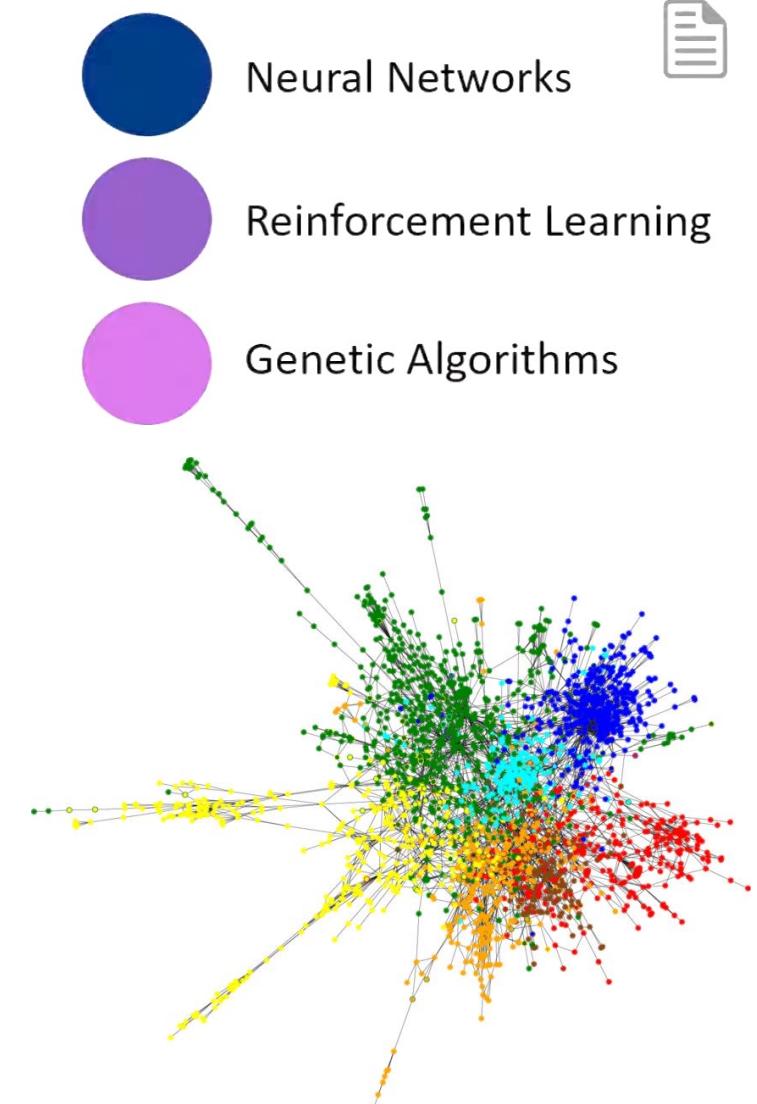
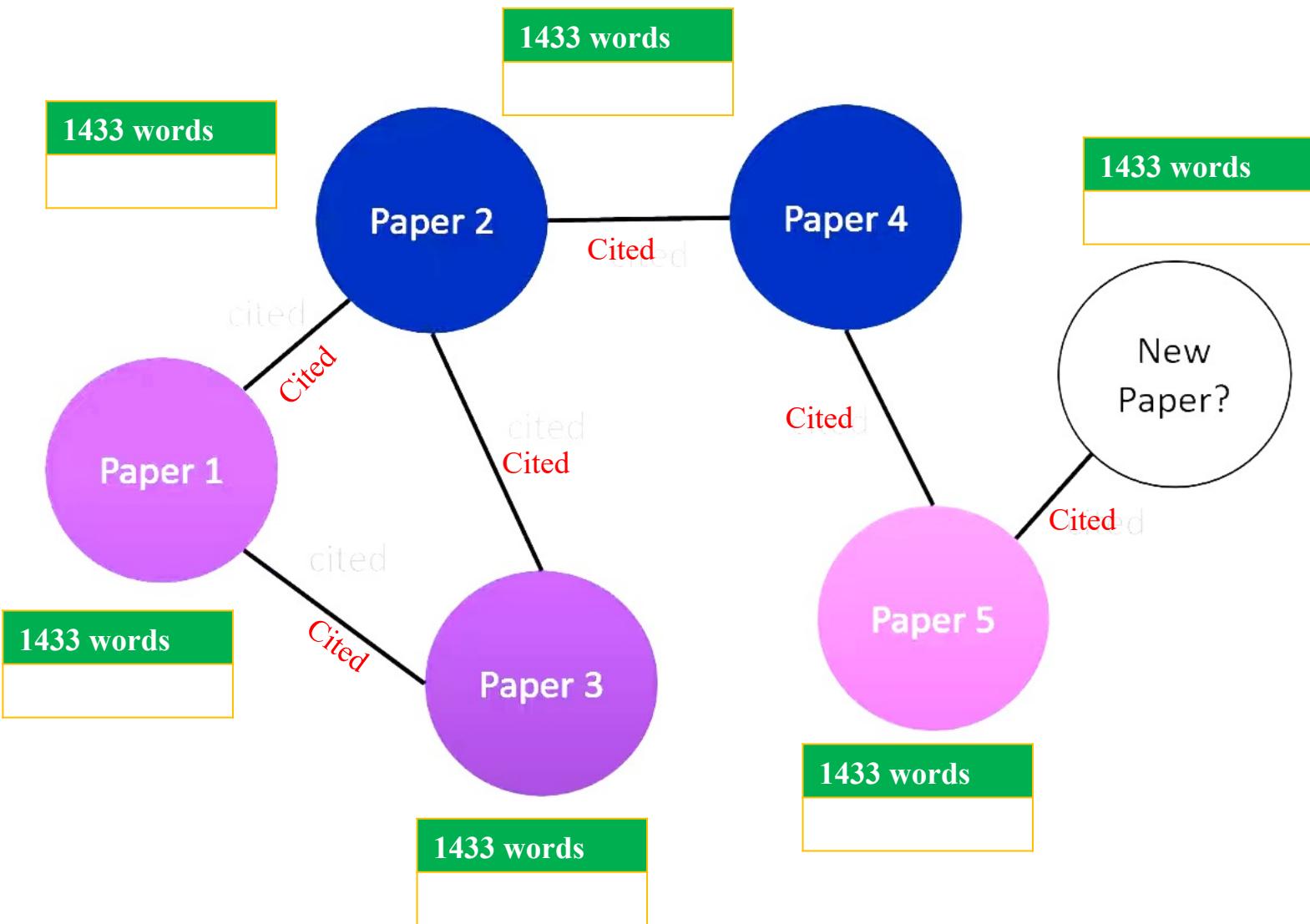
Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary.

- The dictionary consists of 1433 unique words.
- Nodes = Publications (Papers, Books ...)
- Edges = Citations
- Node Features = word vectors
- 7 Labels = Publication
- type e.g. Neural_Networks, Rule_Learning, Reinforcement_Learning, Probabilistic_Methods...

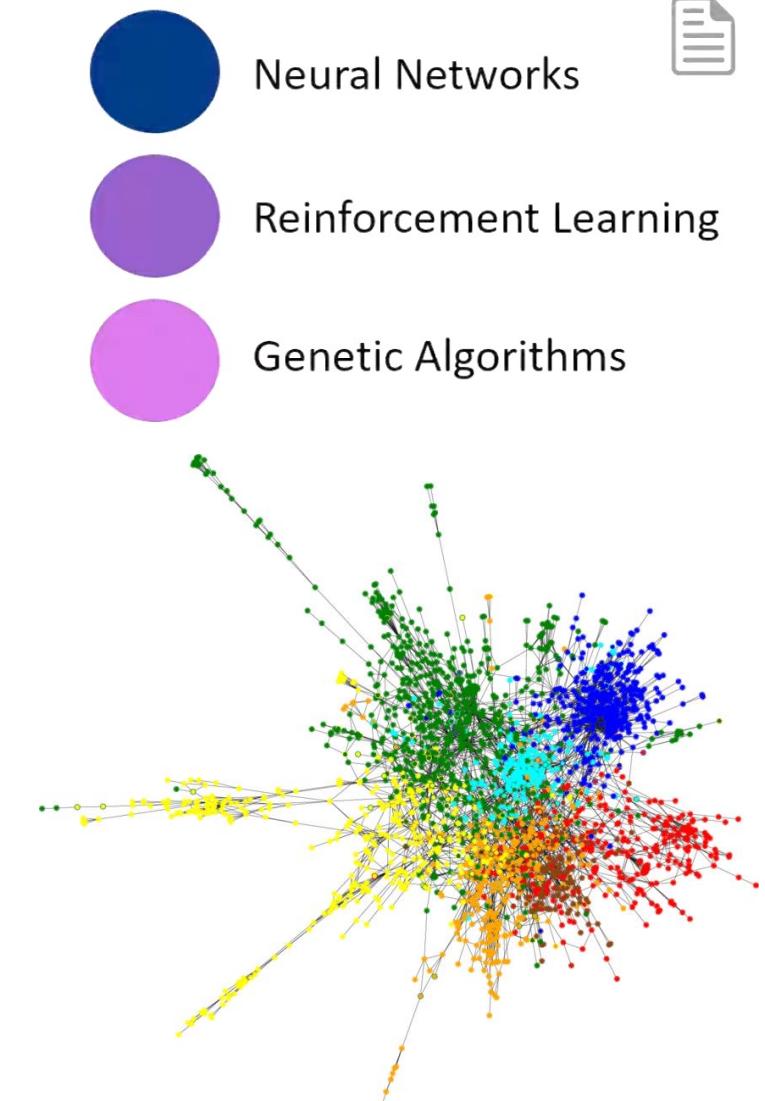
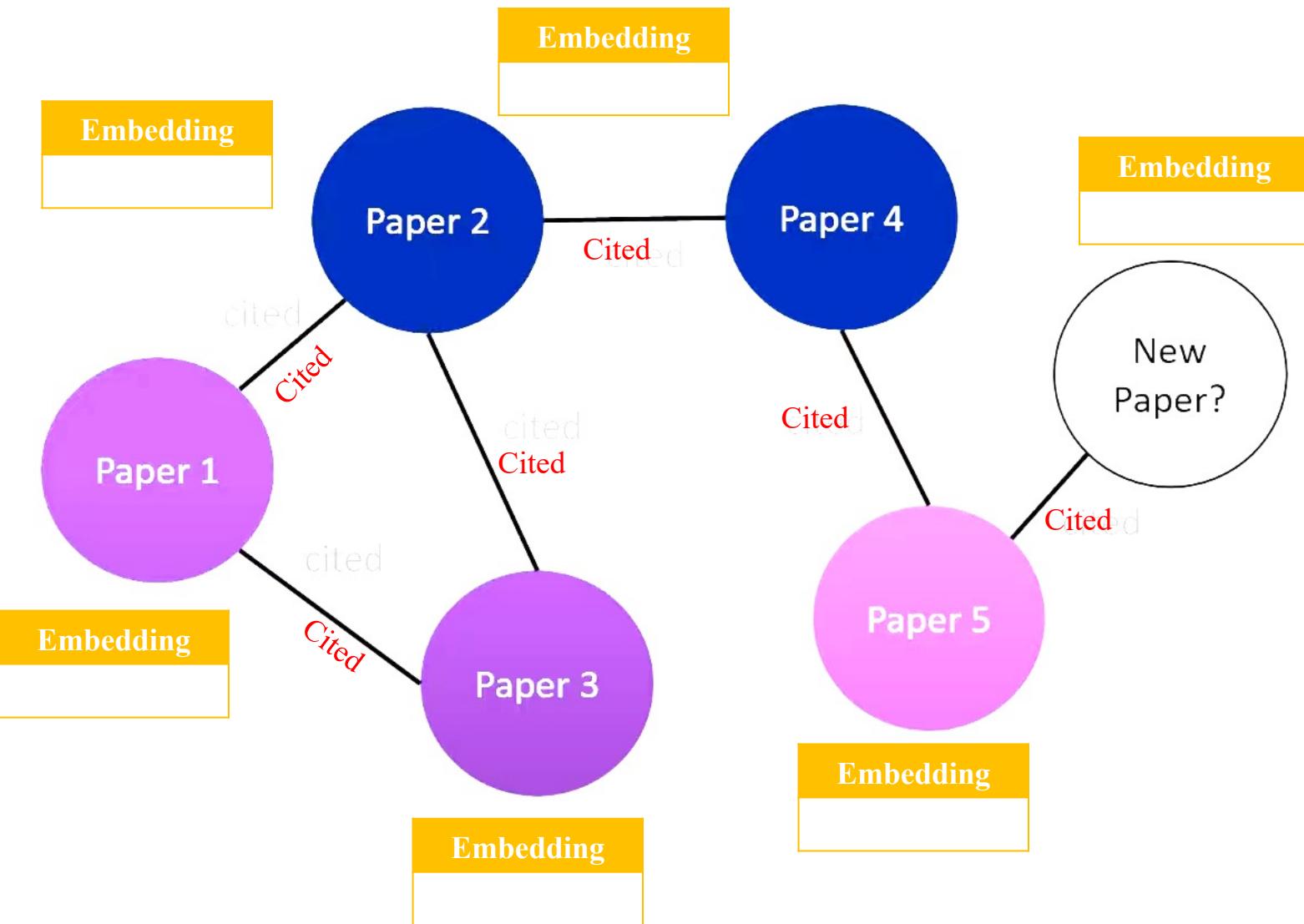
BoW Representation

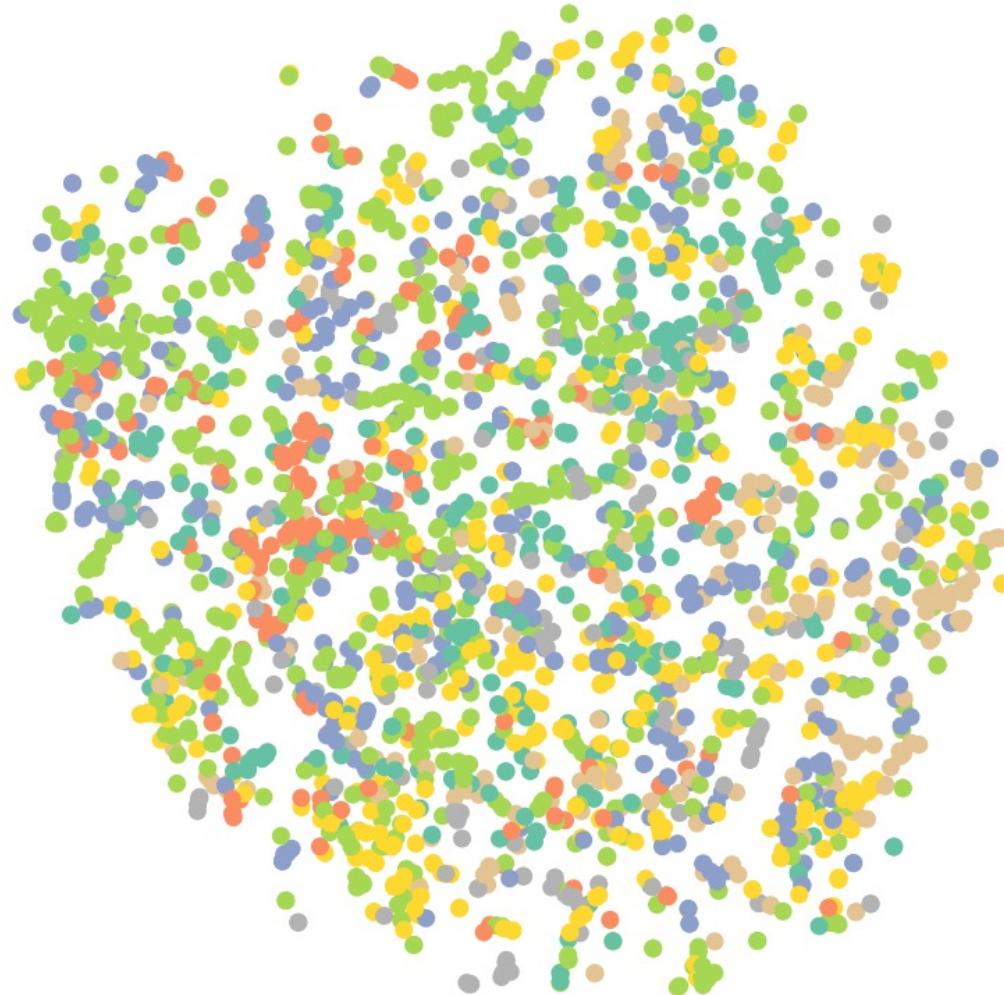


Cora Citation Dataset



Cora Citation Dataset





Visualize the node embeddings of the untrained GCN network

```
model = GCN(hidden_channels=16)
model.eval()

out = model(data.x, data.edge_index)
visualize(out, color=data.y)
```

GNN: Node Classification



Install Pytorch Geometric

```
# Check CUDA Version
!python -c "import torch; print(torch.version.cuda)"

# Add this in a Google Colab cell to install the correct version of Pytorch Geometric.
import torch

def format_pytorch_version(version):
    return version.split('+')[0]

TORCH_version = torch.__version__
TORCH = format_pytorch_version(TORCH_version)

def format_cuda_version(version):
    return 'cu' + version.replace('.', '')

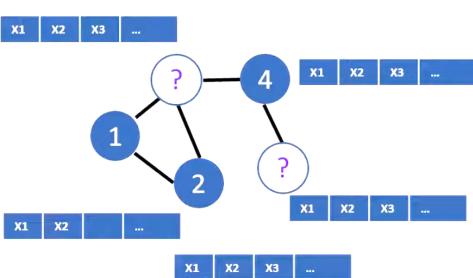
CUDA_version = torch.version.cuda
CUDA = format_cuda_version(CUDA_version)

!pip install torch-scatter      -f https://pytorch-geometric.com/whl/torch-{TORCH}+{CUDA}.html
!pip install torch-sparse       -f https://pytorch-geometric.com/whl/torch-{TORCH}+{CUDA}.html
!pip install torch-cluster      -f https://pytorch-geometric.com/whl/torch-{TORCH}+{CUDA}.html
!pip install torch-spline-conv  -f https://pytorch-geometric.com/whl/torch-{TORCH}+{CUDA}.html
!pip install torch-geometric
```

GNN: Node Classification



Load Cora Dataset



```
from torch_geometric.datasets import Planetoid
from torch_geometric.transforms import NormalizeFeatures

dataset = Planetoid(root='data/Planetoid', name='Cora', transform=NormalizeFeatures())

# Get some basic info about the dataset
print(f'Number of graphs: {len(dataset)}')
print(f'Number of features: {dataset.num_features}')
print(f'Number of classes: {dataset.num_classes}')
print(50*'=')

# There is only one graph in the dataset, use it as new data object
data = dataset[0]

# Gather some statistics about the graph.
print(data)
print(f'Number of nodes: {data.num_nodes}')
print(f'Number of edges: {data.num_edges}')
print(f'Number of training nodes: {data.train_mask.sum()}')
print(f'Training node label rate: {int(data.train_mask.sum()) / data.num_nodes:.2f}')
print(f'Is undirected: {data.is_undirected()}'
```

Number of graphs: 1
 Number of features: 1433
 Number of classes: 7
 ======
 Data(x=[2708, 1433], edge_index=[2, 10556], y=[2708], train_mask=[2708], val_mask=[2708], test_mask=[2708])
 Number of nodes: 2708
 Number of edges: 10556
 Number of training nodes: 140
 Training node label rate: 0.05
 Is undirected: True

GNN: Node Classification MLP



Define MLP

Here, we first reduce the 1433-dimensional feature vector to a low-dimensional embedding (hidden_channels=16), while the second linear layer acts as a classifier that should map each low-dimensional node embedding to one of the 7 classes.

```
import torch
from torch.nn import Linear
import torch.nn.functional as F

class MLP(torch.nn.Module):
    def __init__(self, hidden_channels):
        super().__init__()
        torch.manual_seed(12345)
        self.lin1 = Linear(dataset.num_features, hidden_channels)
        self.lin2 = Linear(hidden_channels, dataset.num_classes)

    def forward(self, x):
        x = self.lin1(x)
        x = x.relu()
        x = F.dropout(x, p=0.5, training=self.training)
        x = self.lin2(x)
        return x

model = MLP(hidden_channels=16)
print(model)
```

```
test_acc = test()
print(f'Test Accuracy: {test_acc:.4f}')
```

Test Accuracy: 0.5900

GNN: Node Classification MLP



How to Train

Here, we first reduce the 1433-dimensional feature vector to a low-dimensional embedding (hidden_channels=16), while the second linear layer acts as a classifier that should map each low-dimensional node embedding to one of the 7 classes.

```
test_acc = test()
print(f'Test Accuracy: {test_acc:.4f}')
```

Test Accuracy: 0.5900

```
from IPython.display import Javascript # Restrict height of output cell.
display(Javascript("google.colab.output.setIframeHeight(0, true, {maxHeight: 300})"))

model = MLP(hidden_channels=16)
criterion = torch.nn.CrossEntropyLoss() # Define loss criterion.
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4) # Define optimizer.

def train():
    model.train()
    optimizer.zero_grad() # Clear gradients.
    out = model(data.x) # Perform a single forward pass.
    loss = criterion(out[data.train_mask], data.y[data.train_mask]) # Compute the loss solely based on the
    loss.backward() # Derive gradients.
    optimizer.step() # Update parameters based on gradients.
    return loss

def test():
    model.eval()
    out = model(data.x)
    pred = out.argmax(dim=1) # Use the class with highest probability.
    test_correct = pred[data.test_mask] == data.y[data.test_mask] # Check against ground-truth labels.
    test_acc = int(test_correct.sum()) / int(data.test_mask.sum()) # Derive ratio of correct predictions.
    return test_acc
```

GNN: Node Classification GCN



Define GCN

```
class GCN(torch.nn.Module):
    def __init__(self, hidden_channels):
        super().__init__()
        torch.manual_seed(1234567)
        self.conv1 = GCNConv(dataset.num_features, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, dataset.num_classes)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = x.relu()
        x = F.dropout(x, p=0.5, training=self.training)
        x = self.conv2(x, edge_index)
        return x

model = GCN(hidden_channels=16)
print(model)
```

GCN(
 (conv1): GCNConv(1433, 16)
 (conv2): GCNConv(16, 7)
)

we will use one of the most simple GNN operators, the **GCN layer** ([Kipf et al. \(2017\)](#)), which is defined as

$$\mathbf{x}_v^{(\ell+1)} = \mathbf{W}^{(\ell+1)} \sum_{w \in \mathcal{N}(v) \cup \{v\}} \frac{1}{c_{w,v}} \cdot \mathbf{x}_w^{(\ell)}$$

GNN: Node Classification GCN



Define GCN

Dropout is only applied in the training step, but not for predictions

We have 2 Message Passing Layers and one Linear output layer

We use the softmax function for the classification problem

The output of the model are 7 probabilities, one for each class

```
class GCN(torch.nn.Module):
    def __init__(self, hidden_channels):
        super(GCN, self).__init__()
        torch.manual_seed(42)

        # Initialize the layers
        self.conv1 = GCNConv(dataset.num_features, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, hidden_channels)
        self.out = Linear(hidden_channels, dataset.num_classes)

    def forward(self, x, edge_index):
        # First Message Passing Layer (Transformation)
        x = self.conv1(x, edge_index)
        x = x.relu()
        x = F.dropout(x, p=0.5, training=self.training)

        # Second Message Passing Layer
        x = self.conv2(x, edge_index)
        x = x.relu()
        x = F.dropout(x, p=0.5, training=self.training)

        # Output layer
        x = F.softmax(self.out(x), dim=1)
        return x
```

we will use one of the most simple GNN operators, the **GCN layer** ([Kipf et al. \(2017\)](#)), which is defined as

$$\mathbf{x}_v^{(\ell+1)} = \mathbf{W}^{(\ell+1)} \sum_{w \in \mathcal{N}(v) \cup \{v\}} \frac{1}{c_{w,v}} \cdot \mathbf{x}_w^{(\ell)}$$

GNN: Node Classification GCN

```

from IPython.display import Javascript # Restrict height of output cell.
display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHeight: 300})'''))

model = GCN(hidden_channels=16)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)
criterion = torch.nn.CrossEntropyLoss()

def train():
    model.train()
    optimizer.zero_grad() # Clear gradients.
    out = model(data.x, data.edge_index) # Perform a single forward pass.
    loss = criterion(out[data.train_mask], data.y[data.train_mask]) # Compute the loss solely based on the training set.
    loss.backward() # Derive gradients.
    optimizer.step() # Update parameters based on gradients.
    return loss

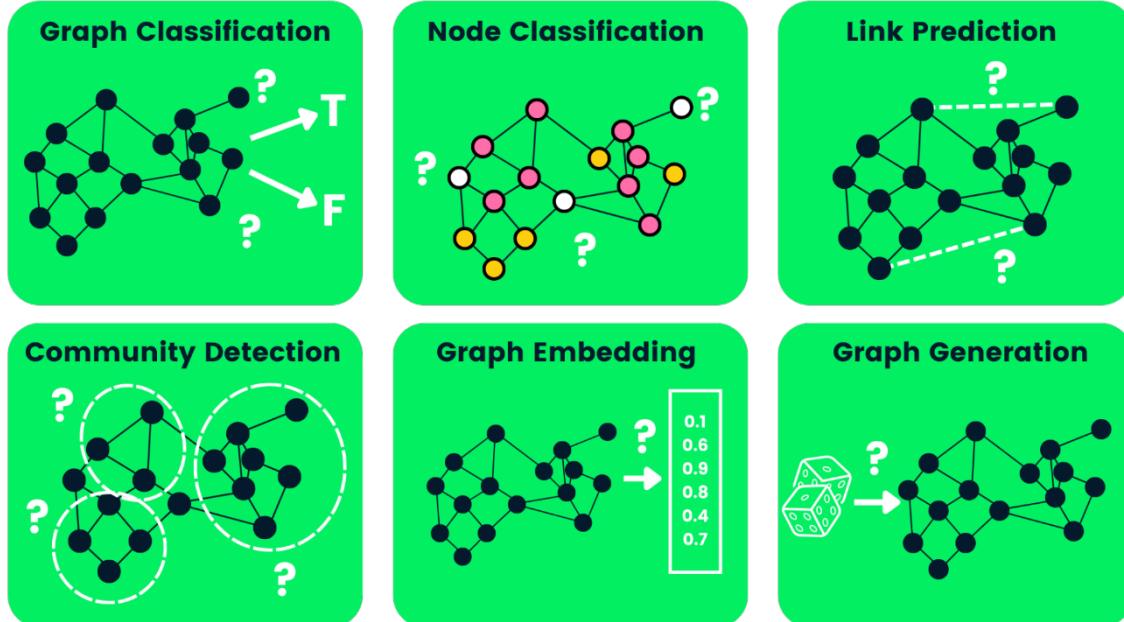
def test():
    model.eval()
    out = model(data.x, data.edge_index)
    pred = out.argmax(dim=1) # Use the class with highest probability.
    test_correct = pred[data.test_mask] == data.y[data.test_mask] # Check against ground-truth labels.
    test_acc = int(test_correct.sum()) / int(data.test_mask.sum()) # Derive ratio of correct predictions.
    return test_acc

for epoch in range(1, 101):
    loss = train()
    print(f'Epoch: {epoch:03d}, Loss: {loss:.4f}')

```

▶ test_acc = test()
 print(f'Test Accuracy: {test_acc:.4f}')
 Test Accuracy: 0.8150

Summary



- 1 • What is Graph Data Around Us
- 2 • Understand Graph Neural Network
- 3 • Understand Graph Convolutional Neural Network
- 4 • Node Classification with Cora Citation Dataset

Next Topics

Edge Feature in GNN: Last But Not Least



Why are edge features important



Love to watch movie



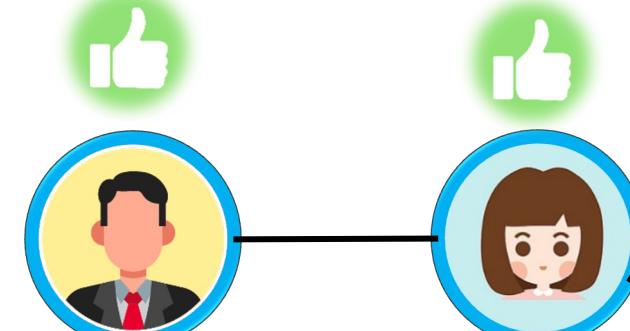
Not Love to watch movie

Node feature

Age	Weight	Drink

Age	Weight	Drink

Node feature



Node feature

Age	Weight	Drink

Edge information

Binary information

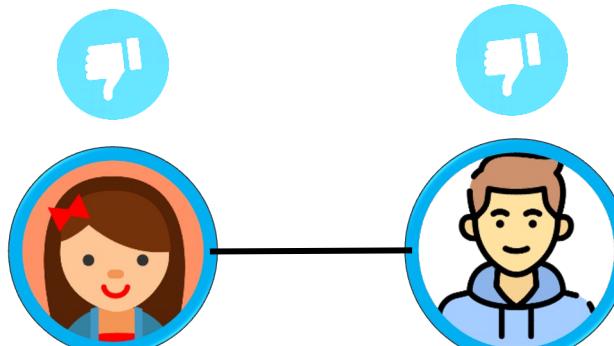
0-1



Does she love watching movie?

Node feature

Age	Weight	Drink



Node feature

Edge Feature in GNN: Last But Not Least



Why are edge features important



Love to watch movie



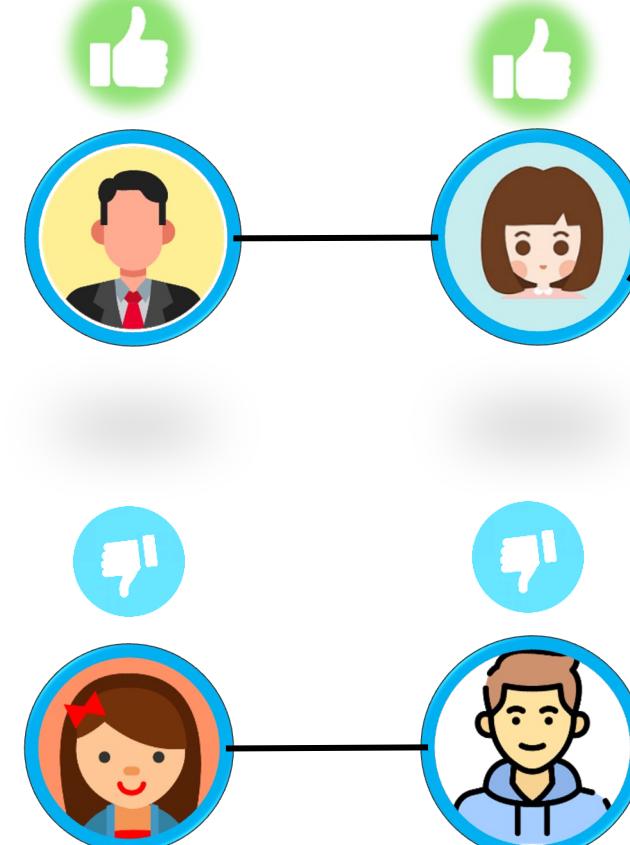
Not Love to watch movie

Node feature

Age	Weight	Drink

Age	Weight	Drink

Node feature



Node feature

Edge information

Binary information

0-1

How do edge features utilize in GNN?



Edge Feature in GNN

