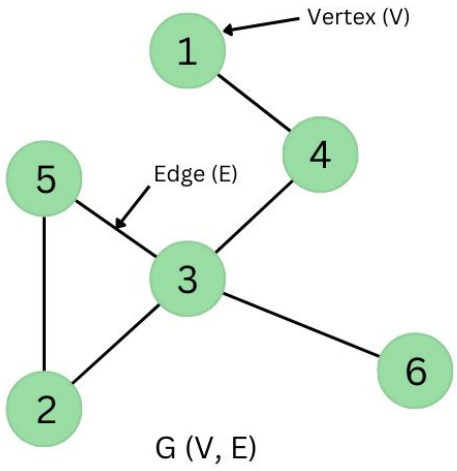
Graph Neural Networks

Shelly Levy, Adam Bublil, Tom Saacks and Roy Efroni

What is a Graph?

- A graph is a data structure comprising 2 essential elements: nodes (or vertices) and edges.
  - Nodes = things (people, molecules, articles)
  - Edges = relationships (friendship, chemical bond, citation)



Why Graphs ?

- Many datasets are not grids (like images or text) - they are networks!

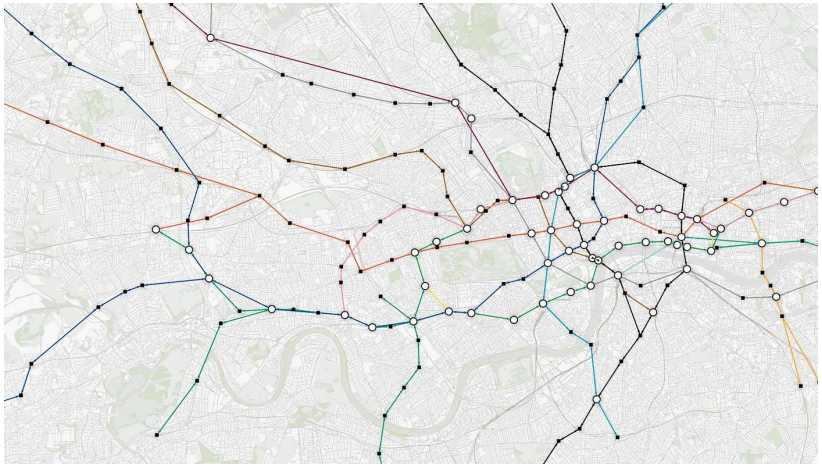
Why Graphs ?

- Many datasets are not grids (like images or text) - they are networks!

Nodes = people
Edges = relationships (friendship, communication, follows)

Why Graphs ?

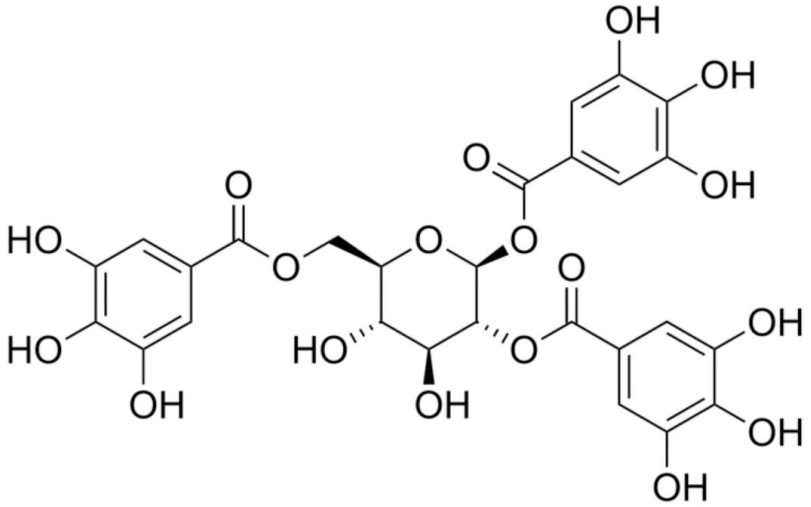
- Many datasets are not grids (like images or text) - they are networks!



Nodes = intersections or stations
Edges = roads or connections

Why Graphs ?

- Many datasets are not grids (like images or text) - they are networks!



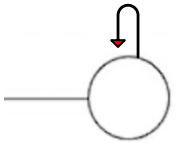
Nodes = atoms
Edges = chemical bonds

What is a Graph?

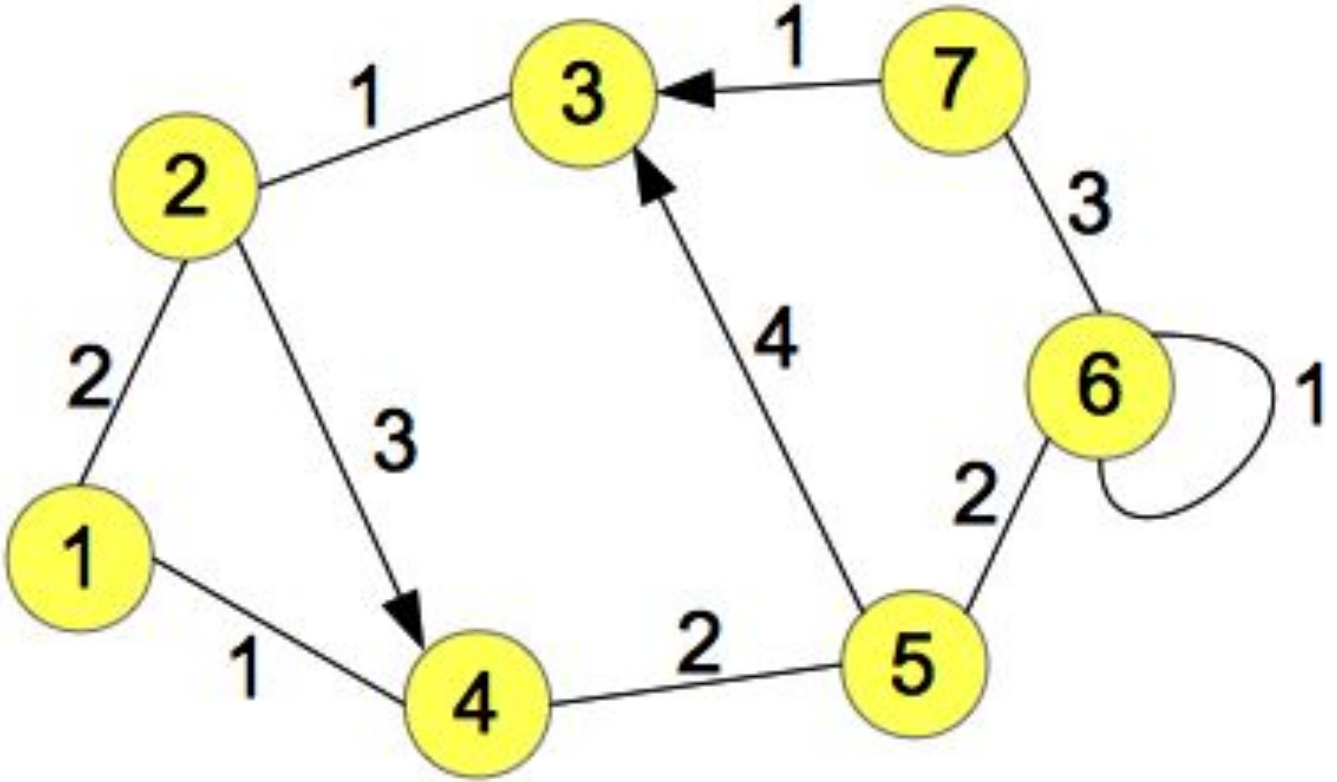
- **Types of Edges:**
  - Undirected: relationships are symmetric (e.g., friendship)
  - Directed: one-way relationships (e.g., follower → followed)
  - Self-loops: edges from a node to itself (used in GNNs to include a node's own information during updates)
  - Homogeneous: All edges represent the same kind of relationship (e.g., all "friend" links)
  - Heterogeneous: different types of relationships (e.g., "follows," "likes," "comments")
  - Weighted: edges have numeric values (e.g., strength of connection)
  - Attributed: edges contain feature vectors (e.g., timestamp, type)

Undirected

Directed



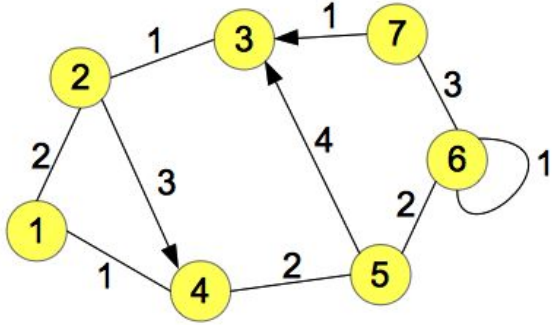
Self Loop


1
2
3
4
5
6
7
1
1
2
3
4
2
3
1
2
1

Social Network Scenario

- **Nodes** = Users, **Edges** = Interactions (friendship, follows, tags, etc.),

  Weights = Interaction strength (messages, tags, etc.)



Self-loop

User 6 often engages with their own content (e.g., likes or reposts their posts)

Directed edge

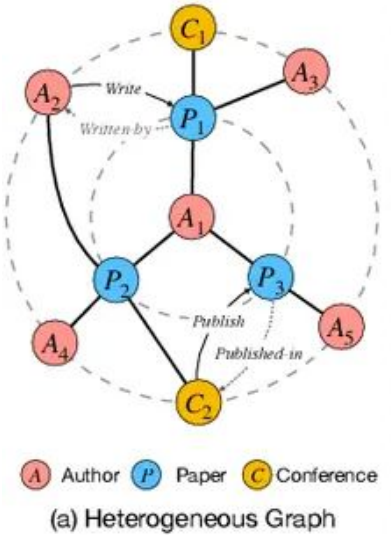
User 2 often tags/messages User 4, but not vice versa (fan → influencer)

Undirected edge

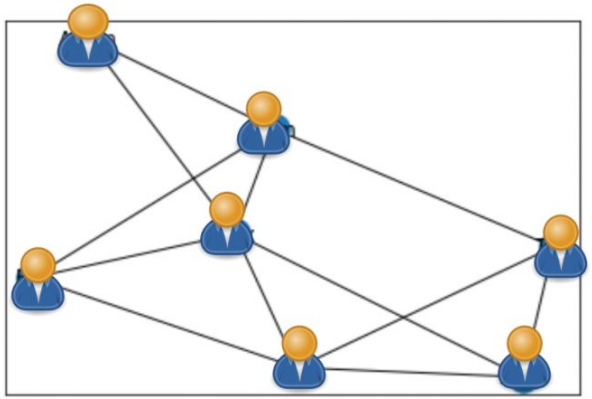
Users 1 and 2 are mutual friends who regularly interact

What is a Graph?

- **Types of Nodes:**
  - Homogeneous: all nodes are of the same type (e.g., all users)
  - Heterogeneous: multiple node types (e.g., users, items, tags)
  - Attributed Nodes: each node has features (e.g., age, text, category)
  - Dynamic Nodes: nodes can appear/disappear over time (e.g., users joining/leaving a network)

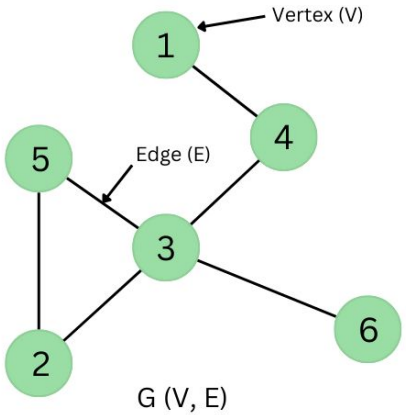


(a) Heterogeneous Graph


Homogeneous Graph

What is a Graph?

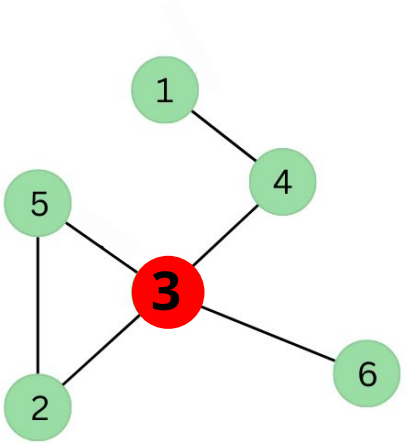
- **Neighbors $N(v)$** - the set of nodes directly connected to a given node
  - For node v, the neighborhood is denoted $N(v)$
  - $N(v)=\{u \mid (u,v) \in E\}$



What is a Graph?

- **Neighbors $N(v)$** - the set of nodes directly connected to a given node
  - For node v, the neighborhood is denoted $N(v)$
  - $N(v)=\{u \mid (u,v) \in E\}$

Who are the neighbors of node 3 ($N(v=3)$)?

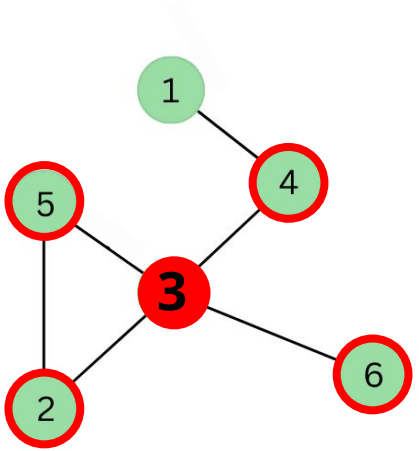


What is a Graph?

- **Neighbors $N(v)$** - the set of nodes directly connected to a given node
  - For node v, the neighborhood is denoted $N(v)$
  - $N(v)=\{u \mid (u,v) \in E\}$

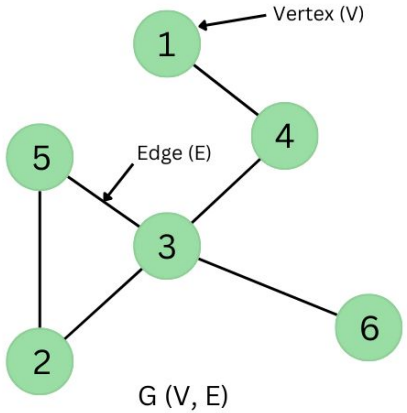
Who are the neighbors of node 3 ($N(v=3)$)?

$$N(v=3)=\{2,4,5,6\}$$



What is a Graph?

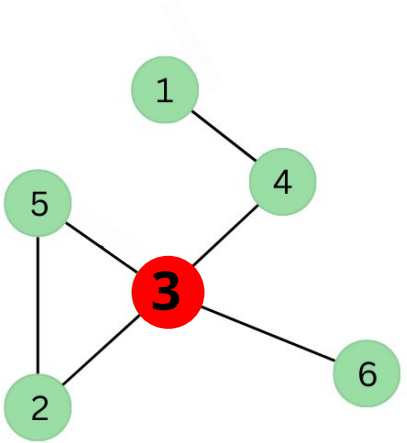
- **Degree $|N(v)|$** - the number of edges connected to a node
  - In undirected graphs, it's the total number of neighbors
  - In directed graphs, we define in-degree and out-degree



What is a Graph?

- **Degree $|N(v)|$** - the number of edges connected to a node
  - In undirected graphs, it's the total number of neighbors
  - In directed graphs, we define in-degree and out-degree

What is the degree of node 3 ($|N(v=3)|$)?

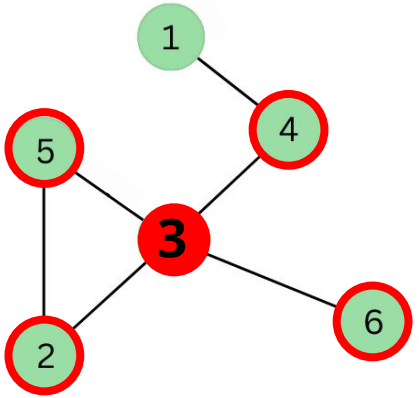


What is a Graph?

- **Degree $|N(v)|$** - the number of edges connected to a node
  - In undirected graphs, it's the total number of neighbors
  - In directed graphs, we define in-degree and out-degree

What is the degree of node 3 ($|N(v=3)|$)?

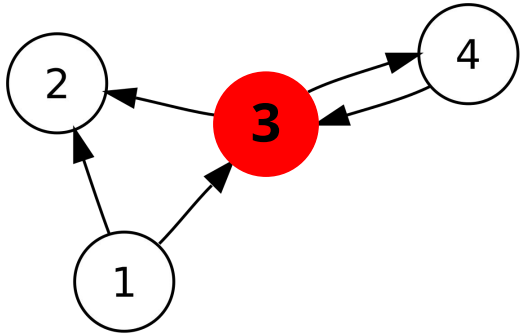
$$N(v=3)=|\{2,4,5,6\}| = 4$$



What is a Graph?

- **Degree $|N(v)|$** - the number of edges connected to a node
  - In undirected graphs, it's the total number of neighbors
  - In directed graphs, we define in-degree and out-degree

What are the in-degree and out-degree of node 3?



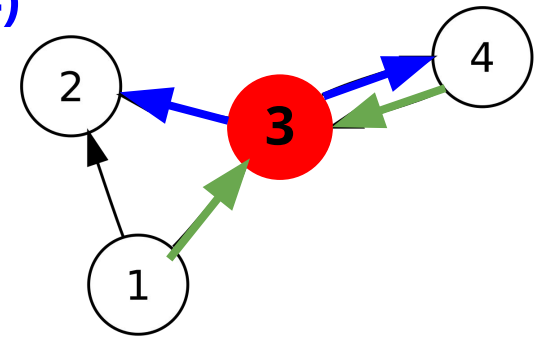
What is a Graph?

- **Degree $|N(v)|$** - the number of edges connected to a node
  - In undirected graphs, it's the total number of neighbors
  - In directed graphs, we define in-degree and out-degree

What are the in-degree and out-degree of node 3?

In-degree = 2 (arrow from 4 → 3 and 1→ 3)

Out-degree = 2 (arrows from 3 → 2 and 3 → 4)

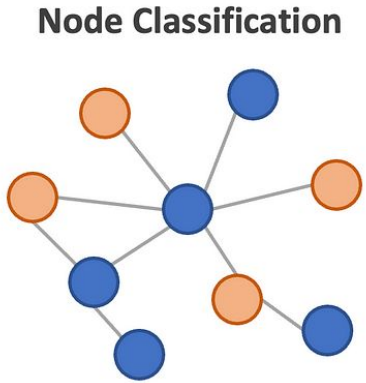


Graph Learning Tasks in GNNs

- GNNs are powerful because they support a variety of prediction tasks:
  - On nodes
  - On edges
  - On whole graphs
- These tasks help solve problems in social networks, chemistry, transportation, and more

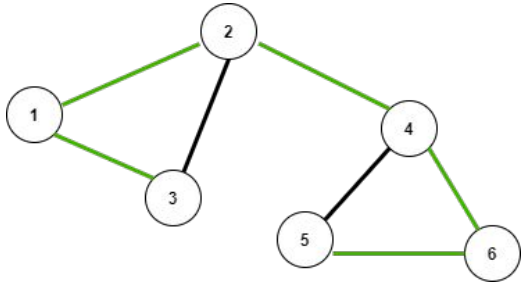
Common Node-Level Tasks

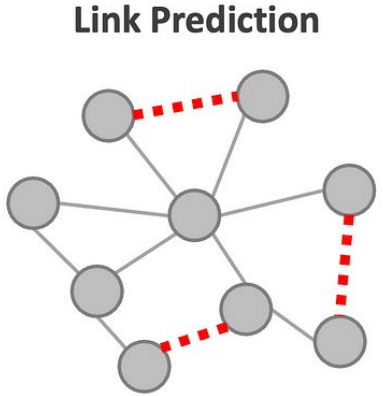
- Node Classification
  - Predict category/label for each node
  - Example: Is this user a bot?
- Node Regression
  - Predict a continuous value per node
  - Example: Estimate air quality at each sensor node
- Node Clustering
  - Group nodes based on structure/features
  - Example: Detect social communities



Common Edge-Level Tasks

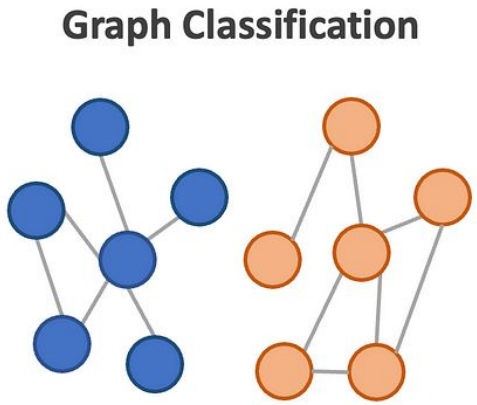
- Edge Classification
  - Classify relationships
  - Example: Is this relationship strong or weak?
- Link Prediction
  - Predict if an edge should exist
  - Example: Recommend a new friend





Common Graph-Level Tasks

- Graph Classification
  - Predict a label for the entire graph
  - Example: Is this molecule toxic?
- Graph Regression
  - Predict a value for the whole graph
  - Example: Predict a property (e.g., boiling point)
- Graph Matching
  - Compare graph similarity
  - Example: Match similar documents or proteins



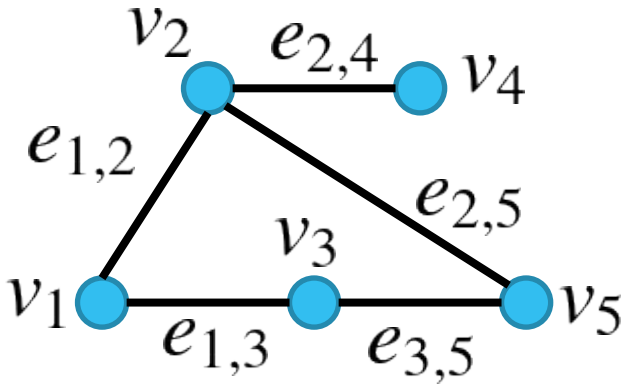
Learning Settings in GNNs

- **Supervised:** All labels are known
- **Semi-supervised:** Some labels are known
- **Unsupervised:** No labels, rely on patterns

- Different applications require different GNN tasks
- GNNs adapt to the graph structure and task type
- Next: Explore architectures that support these tasks

Graph

- **Graph:** $G = (V, E)$
- **Set of nodes (vertices):** $V = \{v_i\}$
- **Set of edges:** $E = \{e_{i,j}\}$
  - Directional/non-directional



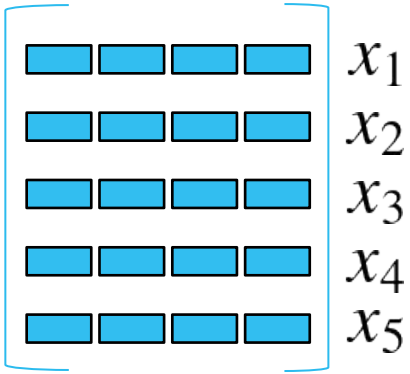
- Nodes and/or edges have some features (label, number, vectors..)

Matrix Representation of Graphs in GNNs

- Matrix representation of Attributed graph $G = (X, A)$

- Feature Matrix $X = (x_{i,d}) = (x_1\ x_2\ \dots)^T$

$X \in \mathbb{R}^{N \times D}$ N nodes, D dim. features

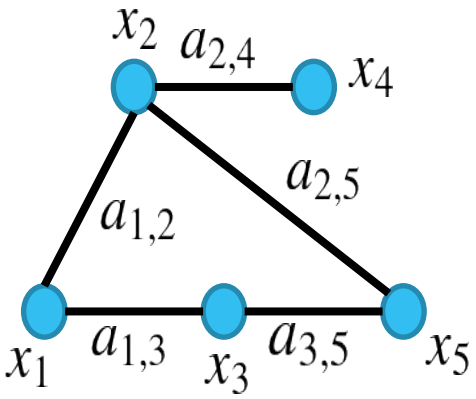


Matrix Representation of Graphs in GNNs

- Adjacency Matrix $A = (a_{i,j})_{i,j=1,2,\ldots}$

$A \in \{0,1\}^{N \times N}$ Edge existence between N X N node pairs

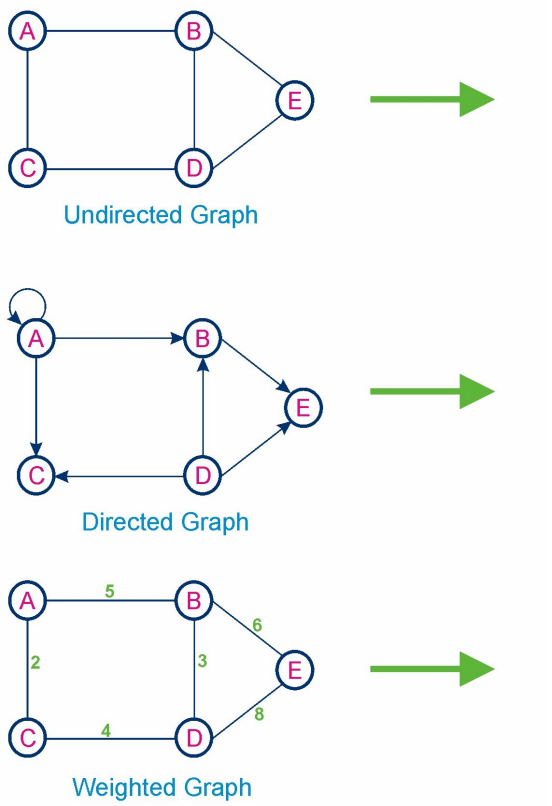
Symmetric A <-- non-directional edges



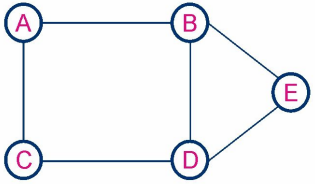
$a_{i,j}$

	x_1	x_2	x_3	x_4	x_5
x_1	0	1	1 $\alpha_{1,3}$	0	0
x_2	1 $\alpha_{1,3}$	0	0	1	1
x_3	1	0	0	0	1
x_4	0	1	0	0	0
x_5	0	1 $\alpha_{2,5}$	1	0	0

Matrix Representation of Graphs in GNNs

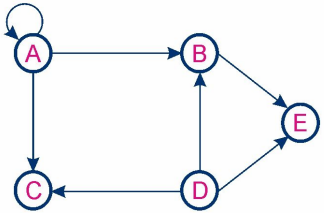


Matrix Representation of Graphs in GNNs



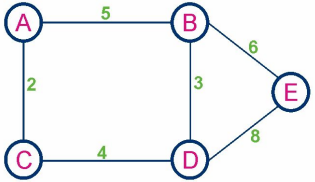
Undirected Graph

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



Directed Graph

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



Weighted Graph

	A	B	C	D	E
A	0	5	2	0	0
B	5	0	0	3	6
C	2	0	0	4	0
D	0	3	4	0	8
E	0	6	0	8	0

General GNN Layer Formula

- A typical GNN layer can be written as:

$$h_v^{(k)} = \text{UPDATE}^{(k)}\left(h_v^{(k-1)}, \text{AGGREGATE}^{(k)}\left(\left\{\text{MESSAGE}^{(k)}\left(h_v^{(k-1)}, h_u^{(k-1)}, e_{uv}\right) \mid u \in \mathcal{N}(v)\right\}\right)\right)$$

- Where:
  - $h_v^{(k)}$: node v's embedding at layer k.
  - $\mathcal{N}(v)$: neighbors of node v.
  - e_{uv} : edge features between nodes u and v (optional).
  - AGGREGATE collects neighbor messages
  - UPDATE combines the aggregated message and the node's prior representation.

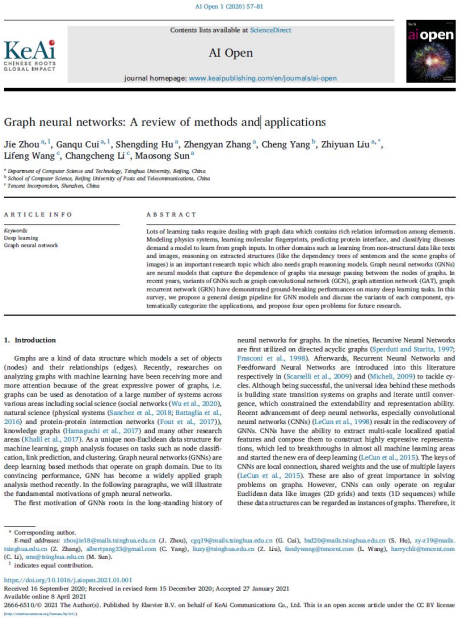
GNN Models From Literature

Title: *Graph Neural Networks: A Review of Methods and Applications*

Authors: Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Maosong Sun.

Published in: *AI Open* (Elsevier)

Year: 2020



AI Open 1 (2020) 57–81

KeAi CHINESE ROOTS GLOBAL IMPACT

Contents lists available at ScienceDirect

AI Open

journal homepage: www.keaipublishing.com/en/journals/ai-open

Graph neural networks: A review of methods and applications

Jie Zhou [a,1], Ganqu Cui [b,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

ARTICLE INFO

Keywords:
Deep learning
Graph neural network

ABSTRACT

Lots of learning tasks require dealing with graph data which contains rich relation information among elements. Modeling physics systems, learning molecular fingerprints, predicting protein interface, and classifying diseases demand a model to learn from graph inputs. In other domains such as learning from non-structural data like texts and images, reasoning on extracted structures (like the dependency trees of sentences and the scene graphs of images) is an important research topic which also needs graph reasoning models. Graph neural networks (GNNs) are neural models that capture the dependence of graphs via message passing between the nodes of graphs. In recent years, variants of GNNs such as graph convolutional network (GCN), graph attention network (GAT), graph recurrent network (GRN) have demonstrated ground-breaking performances on many deep learning tasks. In this survey, we propose a general design pipeline for GNN models and discuss the variants of each component, systematically categorize the applications, and propose four open problems for future research.

1. Introduction

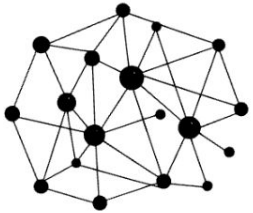
Graphs are a kind of data structure which models a set of objects (nodes) and their relationships (edges). Recently, researches on analyzing graphs with machine learning have been receiving more and more attention because of the great expressive power of graphs, i.e. graphs can be used as denotation of a large number of systems across various areas including social science (social networks (Wu et al., 2020)), natural science (physical systems (Sanchez et al., 2018; Battaglia et al., 2016) and protein-protein interaction networks (Fout et al., 2017)), knowledge graphs (Hamaguchi et al., 2017) and many other research areas (Khalil et al., 2017). As a unique non-Euclidean data structure for machine learning, graph analysis focuses on tasks such as node classification, link prediction, and clustering. Graph neural networks (GNNs) are deep learning based methods that operate on graph domain. Due to its convincing performance, GNN has become a widely applied graph analysis method recently. In the following paragraphs, we will illustrate the fundamental motivations of graph neural networks.

The first motivation of GNNs roots in the long-standing history of neural networks for graphs. In the nineties, Recursive Neural Networks are first utilized on directed acyclic graphs (Sperduti and Starita, 1997; Frasconi et al., 1998). Afterwards, Recurrent Neural Networks and Feedforward Neural Networks are introduced into this literature respectively in (Scarselli et al., 2009) and (Micheli, 2009) to tackle cycles. Although being successful, the universal idea behind these methods is building state transition systems on graphs and iterate until convergence, which constrained the extendability and representation ability. Recent advancement of deep neural networks, especially convolutional neural networks (CNNs) (LeCun et al., 1998) result in the rediscovery of GNNs. CNNs have the ability to extract multi-scale localized spatial features and compose them to construct highly expressive representations, which led to breakthroughs in almost all machine learning areas and started the new era of deep learning (LeCun et al., 2015). The keys of CNNs are local connection, shared weights and the use of multiple layers (LeCun et al., 2015). These are also of great importance in solving problems on graphs. However, CNNs can only operate on regular Euclidean data like images (2D grids) and texts (1D sequences) while these data structures can be regarded as instances of graphs. Therefore, it

* Corresponding author.
E-mail addresses: zhoujie18@mails.tsinghua.edu.cn (J. Zhou), cgq19@mails.tsinghua.edu.cn (G. Cui), hsd20@mails.tsinghua.edu.cn (S. Hu), zy-z19@mails.tsinghua.edu.cn (Z. Zhang), albertyang33@gmail.com (C. Yang), liuy@tsinghua.edu.cn (Z. Liu), fandywang@tencent.com (L. Wang), harrychli@tencent.com (C. Li), sms@tsinghua.edu.cn (M. Sun).
[1] indicates equal contribution.

https://doi.org/10.1016/j.aiopen.2021.01.001
Received 16 September 2020; Received in revised form 15 December 2020; Accepted 27 January 2021
Available online 8 April 2021

Graph Convolutional Networks (GCNs)

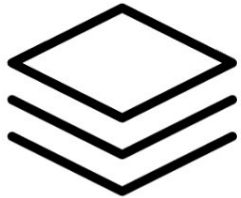


GRAPH DATA

Extend the concept of convolution from grid-like data (as in CNNs) to graph data

FEATURE AGGREGATION

Aggregates features from a node's neighbors to update the node's representation



LAYER-WISE PROPAGATION

Aggregated features are passed through a neural network layer (linear transformation + non-linear activation)

UPDATING NODE REPRESENTATIONS

Captures the local structure of the graph

GCN - Short Example

Each node updates its features by **averaging** its own features with its neighbors' features, **weighted by the graph structure**.

Formula:

$$H = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}XW \qquad H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right)$$

Where:

- $\tilde{A} = A + I$ (adjacency matrix + self-loop),
- $\tilde{D}$ is the degree matrix of $\tilde{A}$,
- $H^{(l)}$ = node features at layer l,
- $W^{(l)}$ = learnable weights,
- σ = non-linear activation (e.g., ReLU)

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right)$$

The Graph:

The **adjacency matrix A** is:

$$A =$$

Add self-loops:

$$\tilde{A} = A + I$$

$$\tilde{A} =$$

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right)$$

The Graph:

The **adjacency matrix A** is:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Add self-loops:

$$\tilde{A} = A + I$$

$$\tilde{A} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right)$$

The Graph:

$$\tilde{A} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Compute the degree of each node:

$$\tilde{D} =$$

Normalize:

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

This scales the influence of neighbors based on degree.
So high-degree nodes don’t dominate.

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right)$$

The Graph:

$$\tilde{A} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Compute the degree of each node:

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

Normalize:

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

This scales the influence of neighbors based on degree.
So high-degree nodes don’t dominate.

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right)$$

The Graph:

Each node has 2 features

$$H^{(0)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$

Learnable weight matrix

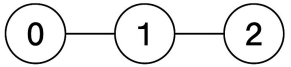
$$W^{(0)} = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

$$H^{(0)}W^{(0)} =$$

We want to transform the 2 input features into 3 output features

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right)$$

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



$$\hat{A} \approx \begin{bmatrix} 0.5 & 0.4082 & 0 \\ 0.4082 & 0.3333 & 0.4082 \\ 0 & 0.4082 & 0.5 \end{bmatrix}$$

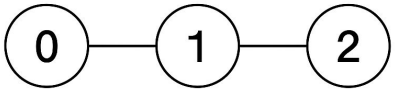
$$H^{(0)}W^{(0)} = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 1 \\ 1 & 3 & 1 \end{bmatrix}$$

Multiply by the normalized adjacency matrix to mix features from neighbors.

This step gives each node a new representation, combining:

- Its own transformed features
- Neighboring nodes' transformed features

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right)$$



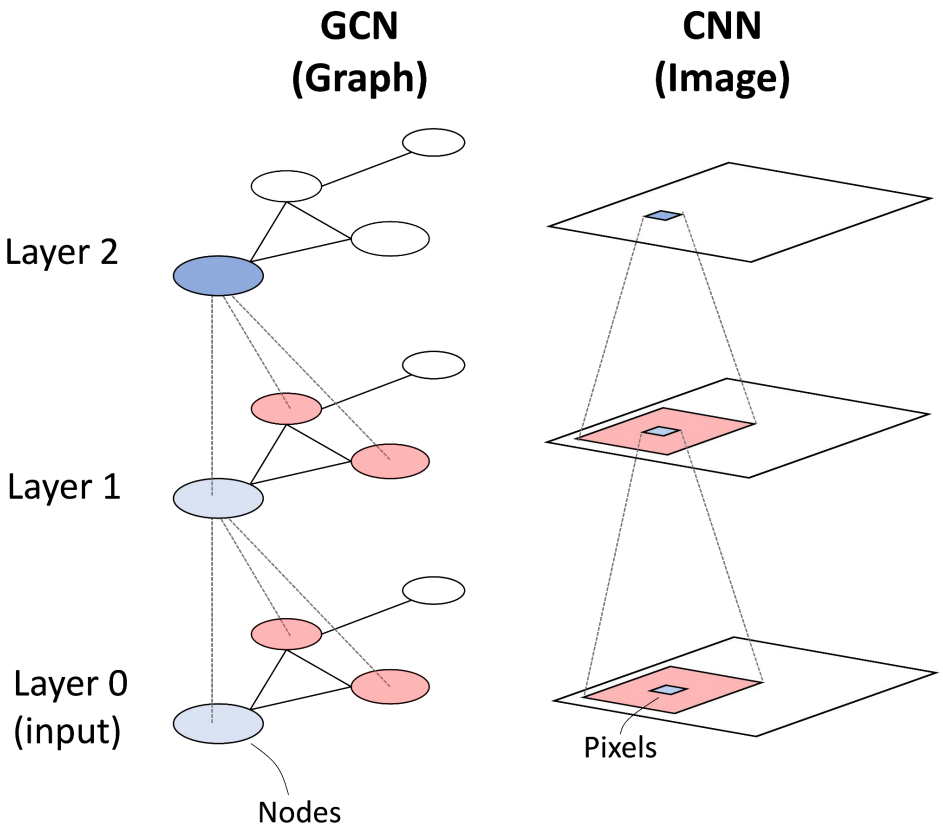
$$H^{(1)} = \begin{bmatrix} 0.5 & 1.4082 & 0.4082 \\ 0.8164 & 2.3743 & 0.7415 \\ 0.5 & 1.9082 & 0.9082 \end{bmatrix} \longrightarrow H^{(1)}_{\text{ReLU}} = \begin{bmatrix} 0.5 & 1.4082 & 0.4082 \\ 0.8164 & 2.3743 & 0.7415 \\ 0.5 & 1.9082 & 0.9082 \end{bmatrix}$$

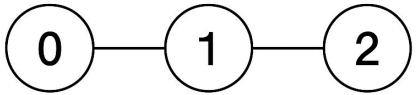
Apply a non-linear function (ReLU) to introduce non-linearity:

- Now each node has a **learned representation** that combines its **own info** and its **neighbors' info**, passed through a learnable transformation.

- This is the final output after **one GCN layer**

Next Layers





$$H^{(1)}_{\text{ReLU}} = \begin{bmatrix} 0.5 & 1.4082 & 0.4082 \\ 0.8164 & 2.3743 & 0.7415 \\ 0.5 & 1.9082 & 0.9082 \end{bmatrix}$$

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right)$$

GNN Model Timeline

2005 - GNN Framework:
Scarselli et al.

The first general framework for neural networks on graphs.

- Uses **recurrent updates** until convergence, similar to RNN.
- Nodes update based on neighbors iteratively until the entire graph reaches a stable state.
- Very **flexible**, but **slow and computationally heavy**.
- Paved the way for modern GNNs.

GNN Model Timeline

2017 – Graph Convolutional Network (GCN):
Kipf & Welling

Simplifies graph convolutions using normalized adjacency and matrix operations.

- Based on spectral graph theory.
- Aggregates neighbors using fixed weights (degree-normalized).
- Fast, scalable — sparked modern GNN research.

GNN Model Timeline

2017 – GraphSAGE (Sample and AGGregate):
Hamilton et al.

Introduces inductive learning via neighborhood sampling and aggregation.

- Makes predictions on nodes or graphs it has never seen during training.
- Samples fixed-size neighborhoods.
- Aggregation methods: mean, LSTM, max pooling.
- Learns a function to compute node embeddings.

GNN Model Timeline

2018 – Graph Attention Network (GAT):
Veličković et al.

Learns which neighbors to attend to using attention scores.

- Computes attention scores
- Learns importance weights for each neighbor.
- Uses multi-head attention for stability.
- More expressive than GCN.

GNN Model Timeline

2021–2025 – Graph Transformers:
SOTA models like Graphormer, SAN, TokenGT

Extend Transformer-style global attention to graph data.

- Inspired by NLP Transformers
- Add positional encodings
- Scale well to large and dense graphs.
- Use global self-attention, not just local neighbors.

Advantages of GNNs and GCNs

- **Flexibility:** Can handle a wide variety of data types and structures beyond grid-like data.
- **Expressiveness:** Capture complex relationships and dependencies within the data.
- **Performance:** Often outperform traditional models on tasks involving graph-structured data due to their ability to leverage the graph's inherent structure.

GCNs VS. CNNs

CNNs

GCNs

Data Structure

- Designed for gridlike structures (e.g. 2D images).
- Designed for graph-structured data, which can be irregular and non-Euclidean.

Convolution Operation

- Use fixed-size kernels for convolution on grid-like data.
- Use neighborhood aggregation for convolution on graphs.

Applications

- Primarily used for image and video processing.
- Used for tasks involving graph-structured data (node classification, link prediction, graph classification).

Practical Example - Zachary Karate Club

The Zachary Karate Club dataset is a classic real-world example used to demonstrate how Graph Convolutional Networks (GCNs) work. It captures the social structure of 34 members of a university karate club, with edges representing social interactions between them outside the club. During the observation period, a dispute between the club’s instructor ("Mr. Hi") and the administrator ("John A.") led to the club splitting into factions. This provides a natural setting for a node classification task, where the goal of the GCN is to predict each member’s group affiliation based solely on the graph’s structure.

Zachary Karate Club GCN Steps

01
Data Preparation

- Prepare graph data, including node features and adjacency matrix

02
Model Definition

- Define the GCN architecture

03
Training

- Train the GCN on the dataset

04
Evaluation

- Evaluate the model's performance using accuracy method

Overview

•PyTorch Geometric: is a specialized extension of PyTorch that has been created specifically for the development and implementation of GNNs.

```
data = dataset[0]

print(f'x = {data.x.shape}')
print(data.x)
```

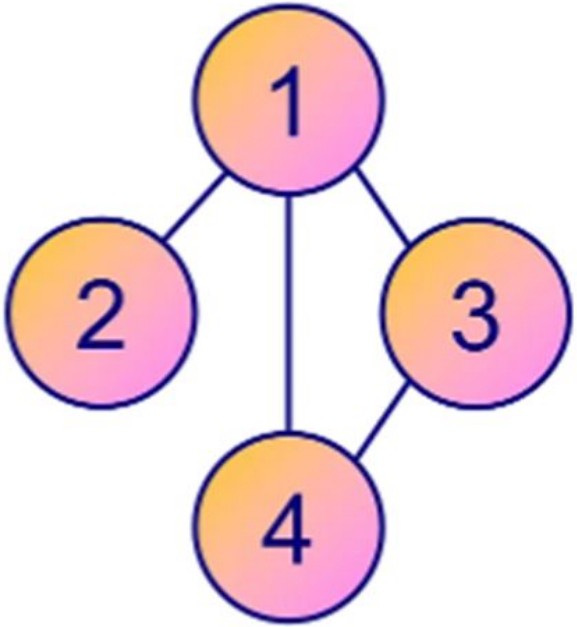
```
x = torch.Size([34, 34])
tensor([[1., 0., 0.,  ..., 0., 0., 0.],
        [0., 1., 0.,  ..., 0., 0., 0.],
        [0., 0., 1.,  ..., 0., 0., 0.],
        ...,
        [0., 0., 0.,  ..., 1., 0., 0.],
        [0., 0., 0.,  ..., 0., 1., 0.],
        [0., 0., 0.,  ..., 0., 0., 1.]])
```

```
print(f'edge_index = {data.edge_index.shape}')
print(data.edge_index)
```

```
edge_index = torch.Size([2, 156])
tensor([[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,
          1,  1,  1,  1,  1,  1,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  3,
          3,  3,  3,  3,  3,  4,  4,  4,  5,  5,  5,  5,  6,  6,  6,  6,  7,  7,
          7,  7,  8,  8,  8,  8,  8,  9,  9, 10, 10, 10, 11, 12, 12, 13, 13, 13,
         13, 13, 14, 14, 15, 15, 16, 16, 17, 17, 18, 18, 19, 19, 19, 20, 20, 21,
         21, 22, 22, 23, 23, 23, 23, 23, 24, 24, 24, 25, 25, 25, 26, 26, 27, 27,
         27, 27, 28, 28, 28, 29, 29, 29, 29, 30, 30, 30, 30, 31, 31, 31, 31, 31,
         31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33,
         33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33],
        [ 1,  2,  3,  4,  5,  6,  7,  8, 10, 11, 12, 13, 17, 19, 21, 31,  0,  2,
          3,  7, 13, 17, 19, 21, 30,  0,  1,  3,  7,  8,  9, 13, 27, 28, 32,  0,
          1,  2,  7, 12, 13,  0,  6, 10,  0,  6, 10, 16,  0,  4,  5, 16,  0,  1,
          2,  3,  0,  2, 30, 32, 33,  2, 33,  0,  4,  5,  0,  0,  3,  0,  1,  2,
          3, 33, 32, 33, 32, 33,  5,  6,  0,  1, 32, 33,  0,  1, 33, 32, 33,  0,
          1, 32, 33, 25, 27, 29, 32, 33, 25, 27, 31, 23, 24, 31, 29, 33,  2, 23,
         24, 33,  2, 31, 33, 23, 26, 32, 33,  1,  8, 32, 33,  0, 24, 25, 28, 32,
         33,  2,  8, 14, 15, 18, 20, 22, 23, 29, 30, 31, 33,  8,  9, 13, 14, 15,
         18, 19, 20, 22, 23, 26, 27, 28, 29, 30, 31, 32]])
```

Overview

- **Adjacency matrix**



$$A = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix}$$

Overview

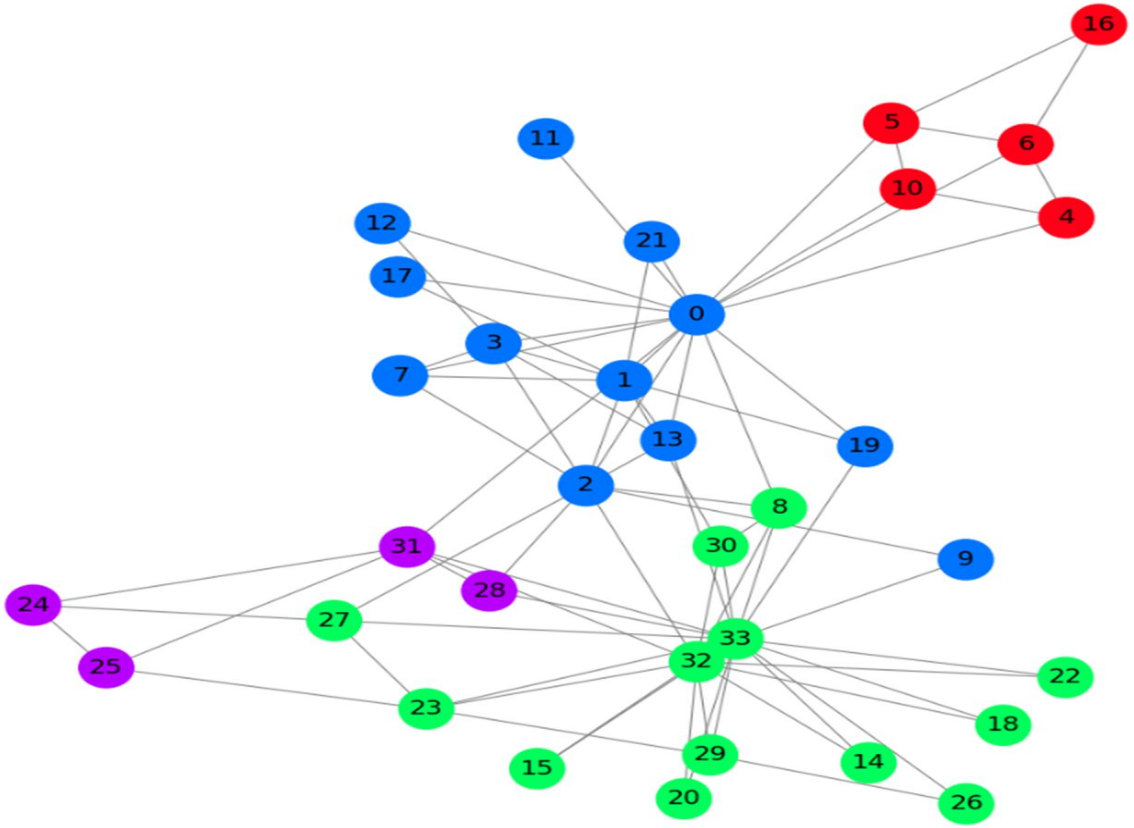
- **Adjacency matrix**

```
from torch_geometric.utils import to_dense_adj

A = to_dense_adj(data.edge_index)[0].numpy().astype(int)
print(f'A = {A.shape}')
print(A)
```

```
A = (34, 34)
[[0 1 1 ... 1 0 0]
 [1 0 1 ... 0 0 0]
 [1 1 0 ... 0 1 0]
 ...
 [1 0 0 ... 0 1 1]
 [0 0 1 ... 1 0 1]
 [0 0 0 ... 1 1 0]]
```

Overview


16
5
6
11
10
4
12
21
17
0
3
7
1
13
19
2
8
31
30
9
24
28
27
33
25
32
22
23
18
29
14
15
20
26

Graph Convolutional Network equation

$$h = \mathbf{W}x$$

CNN

$$h_i = \sum_{j \in \tilde{\mathcal{N}}_i} \mathbf{W}x_j$$

Connections between nodes

$$h_i = \frac{1}{\deg(i)} \sum_{j \in \tilde{\mathcal{N}}_i} \mathbf{W}x_j$$

normalized

Improved Graph Convolutional Network equation

$$h_i = \sum_{j \in \tilde{\mathcal{N}}_i} \frac{1}{\sqrt{\deg(i)}\sqrt{\deg(j)}} \mathbf{W} x_j$$

Bigger weights to features from nodes with fewer neighbors

- h_i: The new feature vector for node i
- $\tilde{\mathcal{N}}_i$: The set of neighbors of node i including node i itself
- $\deg(i)$: The degree (number of neighbors) of node i
- $\mathbf{W}$: A weight matrix applied to the feature vectors
- $\mathbf{x}_j$: The feature vector of neighbor node j

Graph Convolutional Network Implementation

- **‘GCNConv’ Function**

```python
class GCN(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.gcn = GCNConv(dataset.num_features, 3)  # GCN layer transforming 34D input to 3D hidden features
        self.out = Linear(3, dataset.num_classes)    # Output layer transforming 3D hidden features to num_classes

    def forward(self, x, edge_index):
        h = self.gcn(x, edge_index).relu()  # Apply GCN layer and ReLU activation
        z = self.out(h)                     # Apply the output layer
        return h, z                         # Return hidden and output features
```

```
GCN(
  (gcn): GCNConv(34, 3)
  (out): Linear(in_features=3, out_features=4, bias=True)
)
```

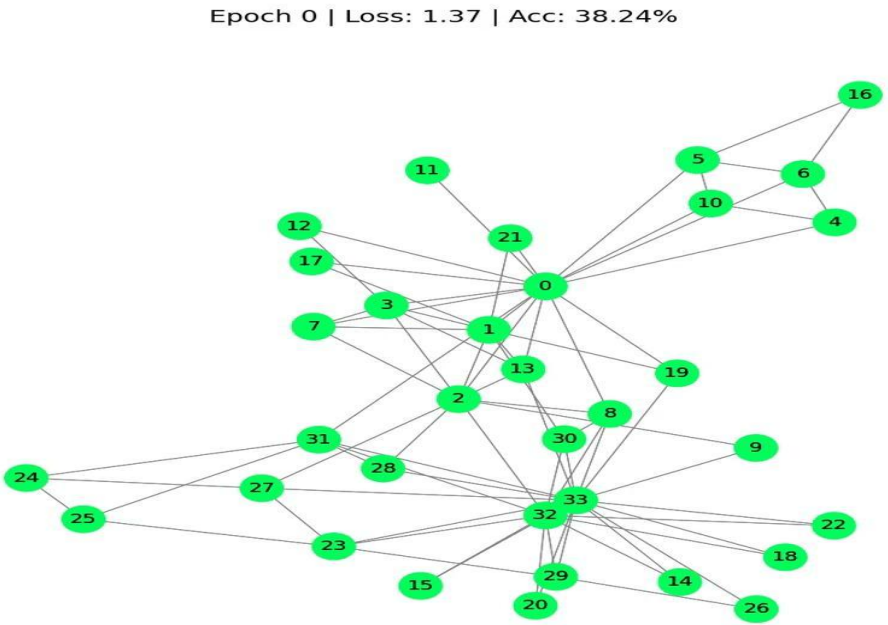
```python
model = GCN()
print(model)
```

Graph Convolutional Network Implementation

- ‘Cross Entropy’ loss function for multi-class classification.
- ‘Adam’ optimizer with a learning rate of 0.02.

```
Epoch   0 | Loss: 1.37 | Acc: 38.24%
Epoch  10 | Loss: 1.28 | Acc: 38.24%
Epoch  20 | Loss: 1.14 | Acc: 70.59%
Epoch  30 | Loss: 0.93 | Acc: 73.53%
Epoch  40 | Loss: 0.69 | Acc: 94.12%
Epoch  50 | Loss: 0.47 | Acc: 100.00%
```

Graph Convolutional Network Implementation



Graph Convolutional Network Implementation

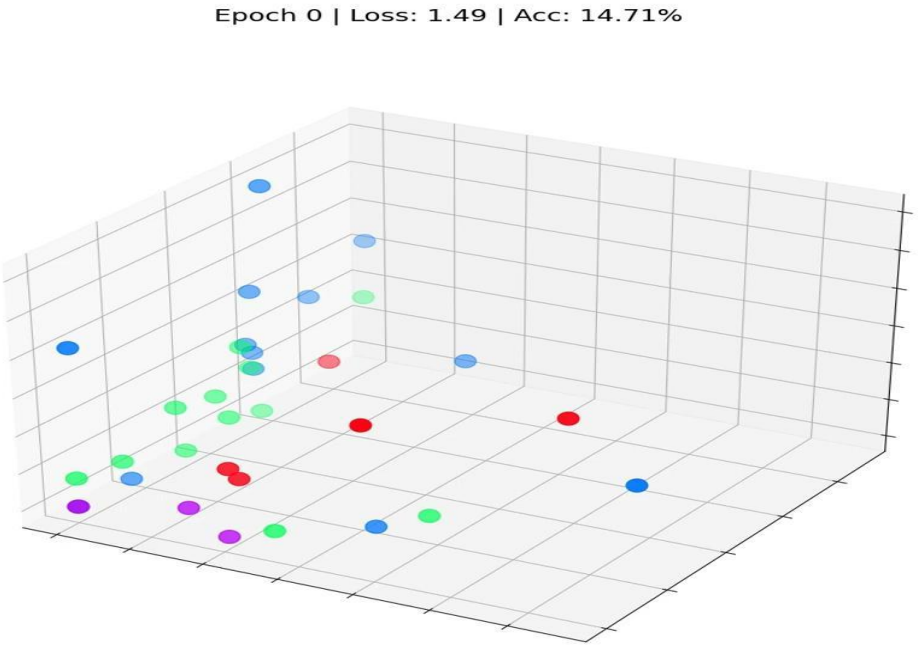
```
Final embeddings = torch.Size([34, 3])
tensor([[0.0000e+00, 3.3839e-02, 3.6172e+00],
        [0.0000e+00, 0.0000e+00, 2.8734e+00],
        [0.0000e+00, 0.0000e+00, 1.8273e+00],
        [0.0000e+00, 0.0000e+00, 2.8750e+00],
        [0.0000e+00, 2.5313e+00, 2.6427e+00],
        [0.0000e+00, 2.5743e+00, 2.8960e+00],
        [0.0000e+00, 2.6633e+00, 2.9342e+00],
        [0.0000e+00, 0.0000e+00, 2.4379e+00],
        [0.0000e+00, 0.0000e+00, 4.8642e-02],
        [0.0000e+00, 0.0000e+00, 1.8188e+00],
```

Graph Convolutional Network Implementation

```
Final embeddings = torch.Size([34, 3])
tensor([[0.0000e+00, 3.3839e-02, 3.6172e+00],
        [0.0000e+00, 0.0000e+00, 2.8734e+00],
        [0.0000e+00, 0.0000e+00, 1.8273e+00],
        [0.0000e+00, 0.0000e+00, 2.8750e+00],
        [0.0000e+00, 2.5313e+00, 2.6427e+00],
        [0.0000e+00, 2.5743e+00, 2.8960e+00],
        [0.0000e+00, 2.6633e+00, 2.9342e+00],
        [0.0000e+00, 0.0000e+00, 2.4379e+00],
        [0.0000e+00, 0.0000e+00, 4.8642e-02],
        [0.0000e+00, 0.0000e+00, 1.8188e+00],
```

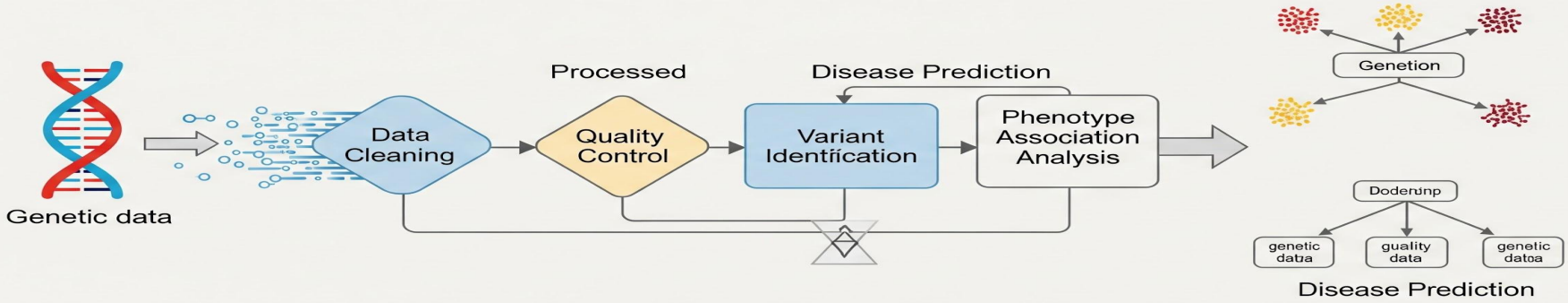
- BERT produce embeddings with 768 or even 1024 dimensions.

Graph Convolutional Network Implementation



Our Research Question

- Can Graph Neural Networks (e.g., GCN, GAT) predict novel gene–disease associations from known biological interactions?



The Complexity Of Biological Interactions

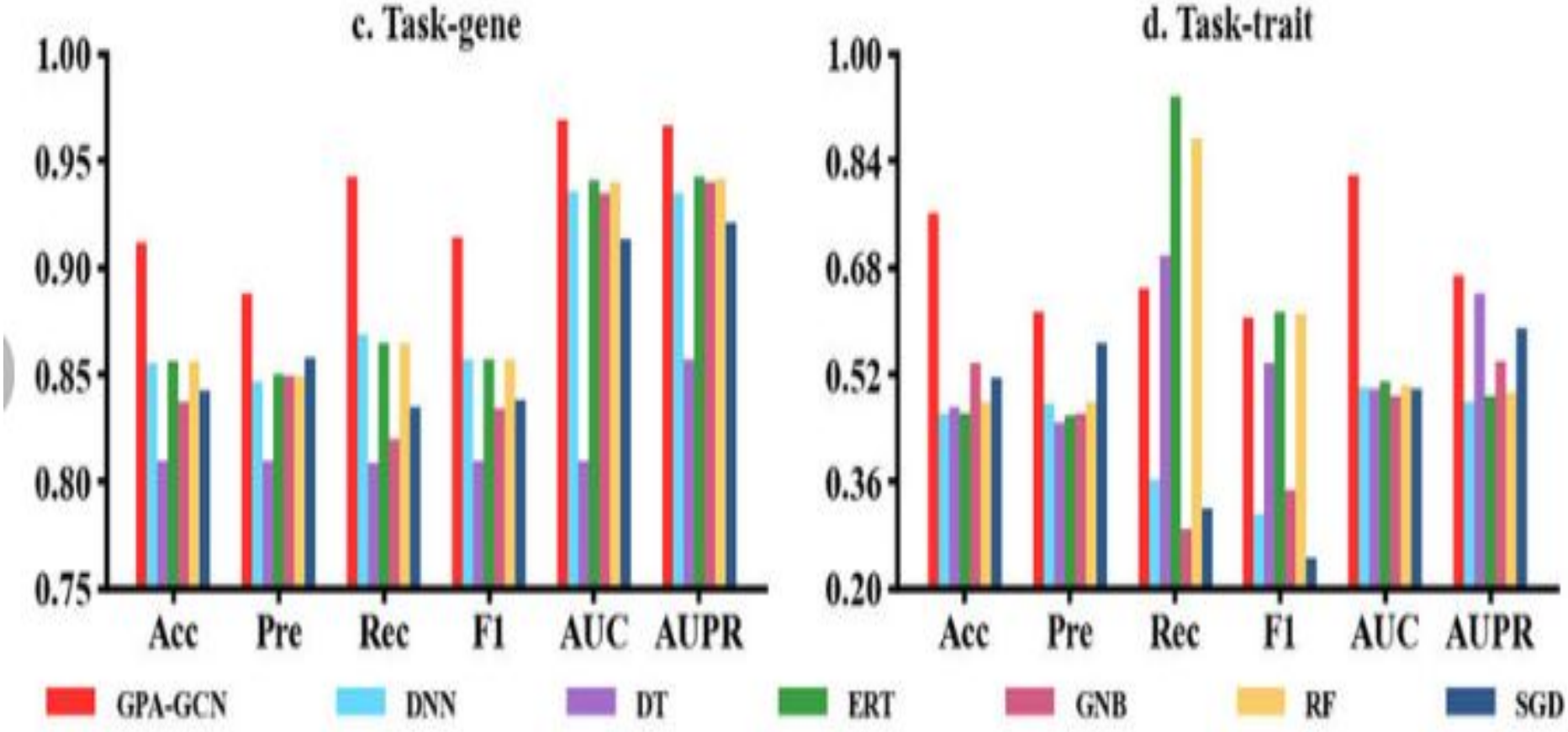
- Interconnectedness
- Data Characteristics
- Limitation of Traditional ML

Why Use GNN

- Biological systems can be represented as graphs: nodes = genes/diseases, edges = interactions.
- GNNs preserve the graph structure and enable propagation of information across nodes.
- Unlike flat models like MLP, GNNs are well-suited for structured and relational biological data.
- Effective in scenarios with limited labeled data.

How Was The Literature Survey Conducted?

- Reviewed scientific articles using GCN or GAT for biological prediction tasks.
- Focused on two distinct cases with clear evaluation:
  - Crop-GPA (plant genomics)
  - HOGCN (human gene–disease prediction)


c. Task-gene
1.00
0.95
0.90
0.85
0.80
0.75
Acc
Pre
Rec
F1
AUC
AUPR
d. Task-trait
1.00
0.84
0.68
0.52
0.36
0.20
Acc
Pre
Rec
F1
AUC
AUPR
GPA-GCN
DNN
DT
ERT
GNB
RF
SGD

Using GCN to Predict Gene - Phenotype Associations in Crops

Source: Gao et al., 2024 – Crop-GPA Platform

- Platform includes over 374,000 validated gene–trait associations across ten crops.
- Two key tools:
  - GPA-BERT: Uses BERT to mine scientific texts and automatically extract gene–trait associations.
  - GPA-GCN: A graph convolutional network that predicts new associations based on semantic proximity in the graph.
- Performance:
  - GPA-GCN outperformed classical models (Random Forest, Naive Bayes, DNN) with AUC = 0.969 and F1 = 0.818.
  - Successfully identified associations not found in existing databases but validated by research.
- Main contribution: Demonstrated that GCN can discover hidden associations in complex biological graphs.

From Agricultural genomics to Biomedical prediction

- Having seen GCN's capabilities in crop genetics, we move to explore its evolution in medical contexts.
- Enter HOGCN – a high-order graph model designed for biomedical networks.

What Is HOGCN

- HOGCN stands for High-Order Graph Convolutional Network.
- Unlike traditional GCNs, HOGCN extends message passing beyond direct neighbors.
- It allows learning from multi-hop connections to capture deeper and more abstract relationships in the graph.
- Why it matters: In biomedical networks, relevant gene–disease interactions may not be directly connected. HOGCN leverages indirect paths to uncover those hidden patterns.
- Architecture highlights: Combines first-order and high-order proximity information while preserving training stability.

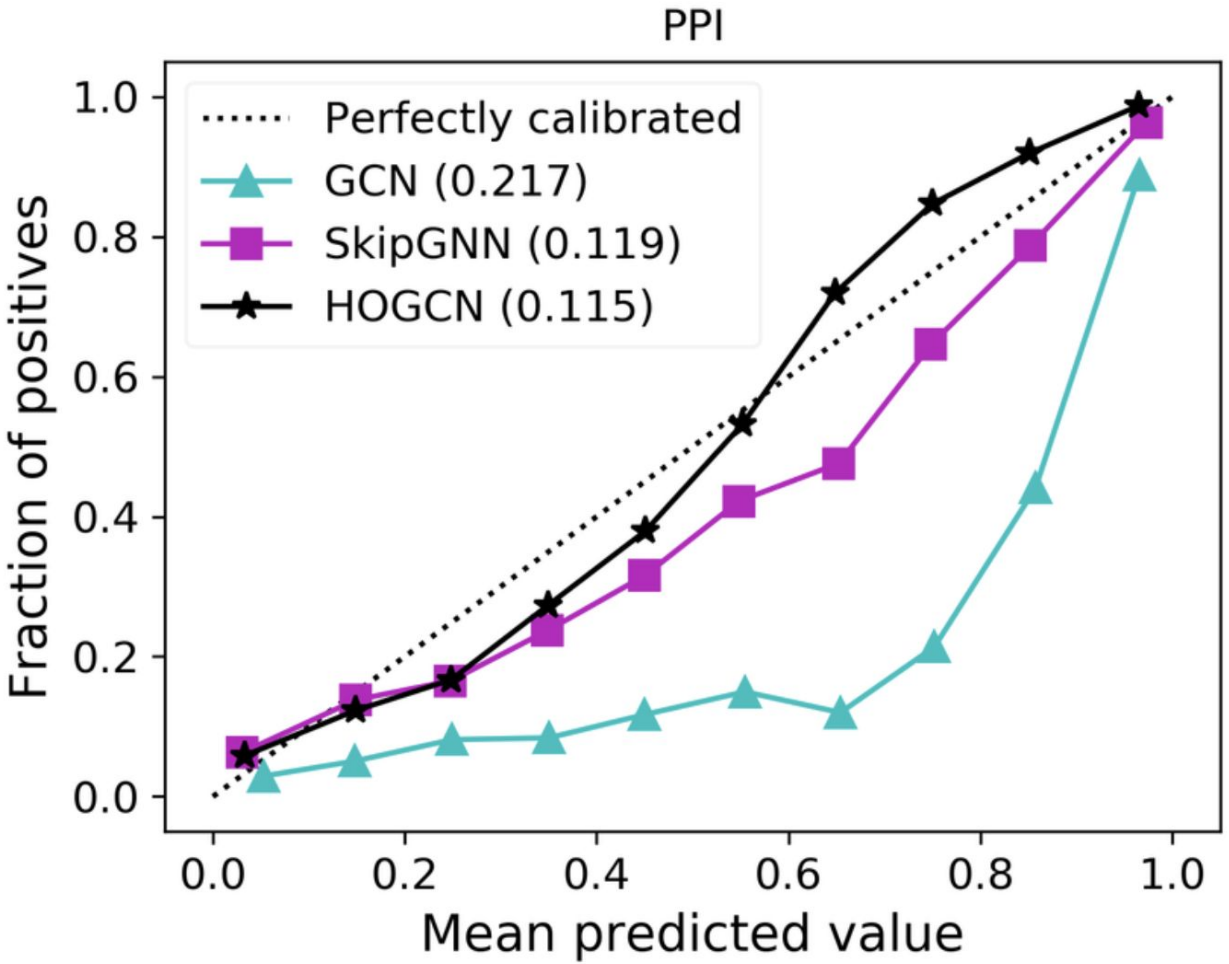
Differences Between GCN and HOGCN

Feature	GCN	HOGCN
Neighborhood Depth	Immediate neighbors only	Higher-order neighbors included
Message Passing	One-hop information	Multi-hop with semantic control
Prediction Capacity	Shallow patterns	Captures deep/indirect associations
Graph Type Suitability	Simple biological networks	Complex biomedical/heterogeneous graphs
Visual Representation	Less semantic clustering (t-SNE)	Clearer clusters in high-dimensional space

Predicting Gene-Disease Associations With HOGCN

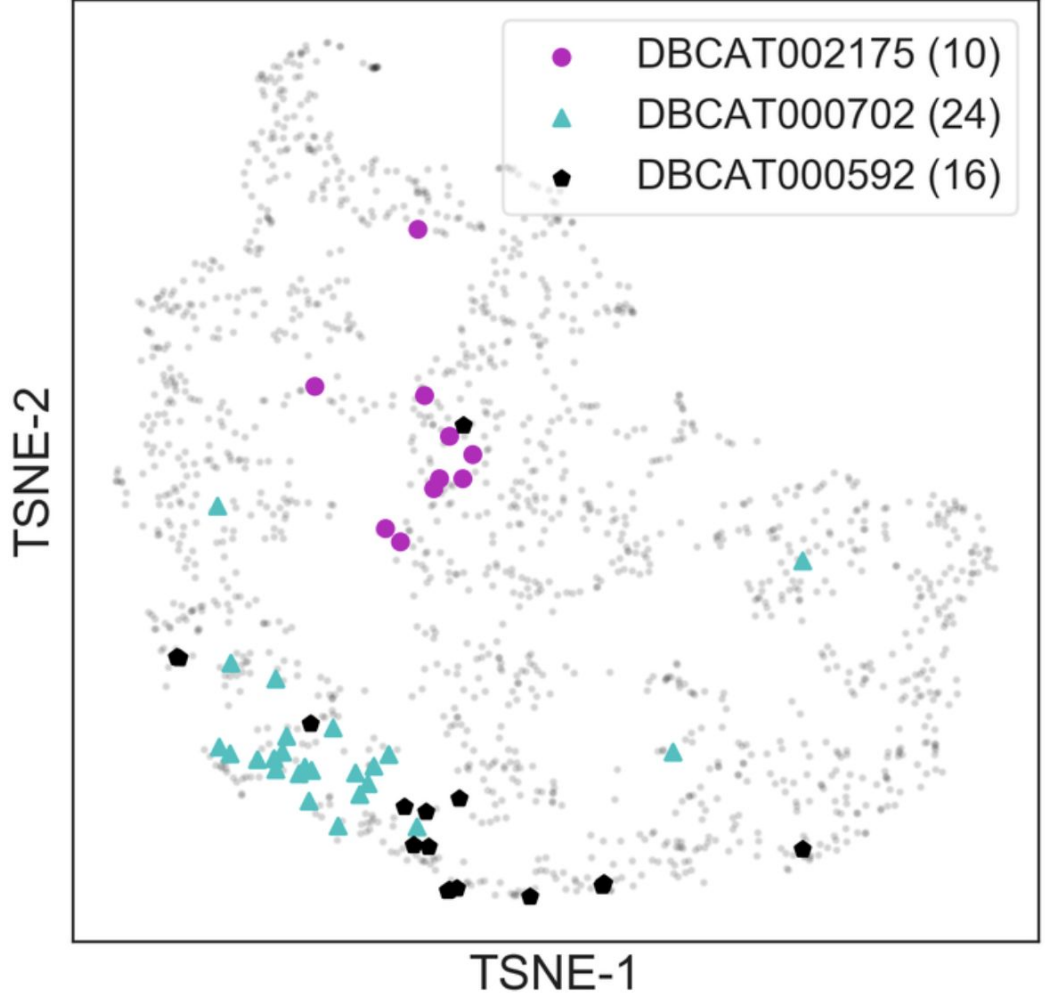
- Source: MDA-HOGCN – applying a high-order GCN to biomedical association prediction.
- Network includes nodes for genes, diseases, and regulatory molecules (e.g., miRNA).
- Key innovation: Unlike basic GCNs, HOGCN transmits messages from higher-order neighbors while maintaining prediction stability.
- Results:
  - Improved AUPRC and AUC compared to traditional GCNs.
  - Produced clearer semantic representations of biological associations via deep graph learning.
- Importance: Shows that adding graph depth significantly enhances predictions, especially for indirect or hidden gene–disease links.

Dataset	Method	AUPRC	AUROC
DTI	DeepWalk	0.753 ± 0.008	0.735 ± 0.009
	node2vec	0.771 ± 0.005	0.720 ± 0.010
	L3	0.891 ± 0.004	0.793 ± 0.006
	VGAE	0.853 ± 0.010	0.800 ± 0.010
	GCN	0.904 ± 0.011	0.899 ± 0.010
	SkipGNN	0.928 ± 0.006	0.922 ± 0.004
	HOGCN	**0.937 ± 0.001**	**0.934 ± 0.001**
DDI	DeepWalk	0.698 ± 0.012	0.712 ± 0.009
	node2vec	0.801 ± 0.004	0.809 ± 0.002
	L3	0.860 ± 0.004	0.869 ± 0.003
	VGAE	0.844 ± 0.076	0.878 ± 0.008
	GCN	0.856 ± 0.005	0.875 ± 0.004
	SkipGNN	0.866 ± 0.006	0.886 ± 0.003
	HOGCN	**0.897 ± 0.003**	**0.911 ± 0.002**
PPI	DeepWalk	0.715 ± 0.008	0.706 ± 0.005
	node2vec	0.773 ± 0.010	0.766 ± 0.005
	L3	0.899 ± 0.003	0.861 ± 0.003
	VGAE	0.875 ± 0.004	0.844 ± 0.006
	GCN	0.909 ± 0.002	0.907 ± 0.006
	SkipGNN	0.921 ± 0.003	0.917 ± 0.004
	HOGCN	**0.930 ± 0.002**	**0.922 ± 0.001**
GDI	DeepWalk	0.827 ± 0.007	0.832 ± 0.003
	node2vec	0.828 ± 0.006	0.834 ± 0.003
	L3	0.899 ± 0.001	0.832 ± 0.001
	VGAE	0.902 ± 0.006	0.873 ± 0.009
	GCN	0.909 ± 0.002	0.906 ± 0.006
	SkipGNN	0.915 ± 0.003	0.912 ± 0.004
	HOGCN	**0.941 ± 0.001**	**0.936 ± 0.001**


PPI
Perfectly calibrated
GCN (0.217)
SkipGNN (0.119)
HOGCN (0.115)
Fraction of positives
Mean predicted value
0.0
0.2
0.4
0.6
0.8
1.0

Performance Comparison - Who Is Better?

- GCN: Only aggregates from immediate neighbors – suited for simpler graphs.
- HOGCN: Learns from higher-order neighbors for deeper graph insight.
- HOGCN demonstrated improved metrics (precision, recall, AUC), especially for indirect connections.
- t-SNE visualizations show better semantic clustering of data using HOGCN


DBCAT002175 (10)
DBCAT000702 (24)
DBCAT000592 (16)
TSNE-2
TSNE-1

Summary Of Literature Insights

- Crop-GPA illustrates how GCN can predict associations in complex plant genomics.
- HOGCN offers significant performance improvement in human biomedical predictions.
- The combination of graph-based learning, semantic modeling, and deep learning points toward a promising future in gene–disease prediction research.

Summary Of Literature Insights

- Why it can be done – supporting points:
  - GCN successfully predicted missing links across 10 plant species (Crop-GPA).
  - GCN captures semantic structure via graph proximity and textual embedding (GPA-GCN).
  - HOGCN effectively handles indirect and higher-order biological interactions (MDA-HOGCN).
  - Improved metrics show practical feasibility (AUC, F1-score, cluster)

Dataset Overview & Collection Process

- Can Graph Neural Networks (e.g., GCN, GAT) predict novel gene–disease associations from known biological interactions?
- Dataset Name - TBGA Gene–Disease Association Dataset
- Collection Process - Downloaded directly from Kaggle
- Data Source - Curated from multiple biomedical databases
- Structure of the Data - Each row represents a known gene–disease association (with score)
- Graph Construction - Nodes: Genes + Diseases , Edges: Known associations

Evaluation Strategy

Evaluate how well the model can predict novel gene–disease associations, beyond the known training pairs.

Metrics:

- **Accuracy –** overall correctness
- **Precision –** how many predicted positives are correct
- **Recall –** how many actual positives were found
- **F1-Score –** balance between precision and recall
- **AUC-ROC –** ability to separate positives from negatives

Evaluation Strategy

Baseline Comparison:

- Baseline Model: GCN (Graph Convolutional Network)
- Literature Model: [e.g., HOGCN or other from the paper]
- We compare:
  - Performance on validation metrics (Accuracy, F1, ROC AUC)
  - Ability to propose meaningful false positives and Validate them using Known biomedical databases.

Project Plan

Step 1: Data Preparation

- Load and preprocess TBGA_train.txt and TBGA_val.txt
- Build bipartite graph (genes only connected to diseases)
- Add node features (e.g., one-hot encoding or node degree)
- EDA- Statistics, Graph Structure Overview, check for errors, etc.

Project Plan

Step 2: Model Implementation

- Baseline: GCN using PyTorch Geometric
- Optionally implement GAT or Graph Transformer for comparison

Project Plan

Step 3: Training & Evaluation

- Train model to predict edges (gene–disease links)

- Fine tune the models- Optimize Layer Depth, Adjust Attention Heads in GAT, Choose Best Aggregator in GraphSAGE, etc...

- Evaluate on validation set using:
    - Accuracy, Precision, Recall, F1-Score, AUC

Project Plan

Step 4: Post-analysis

- Identify false positives from the model

- Validate potential novel associations:
  - Cross-reference with different datasets
  - Literature review

Project Plan

Step 5: Comparison & Reporting

- Compare our GNN models with paper models (e.g., HOGCN)
- Present performance metrics and biological insight
- Write final report and prepare presentation
- Maybe help the world with finding new gene-disease connections

Thank you

Questions?