Graph Neural Networks

Azad Afandizada

10.10.2025

Contents

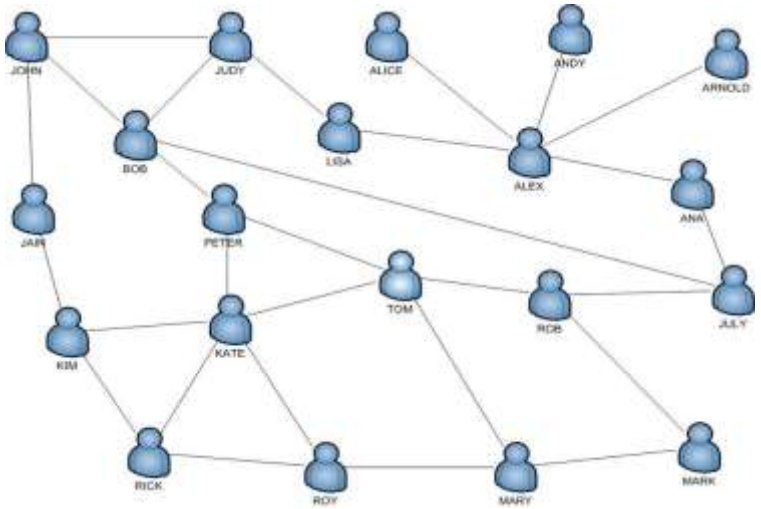
Introduction

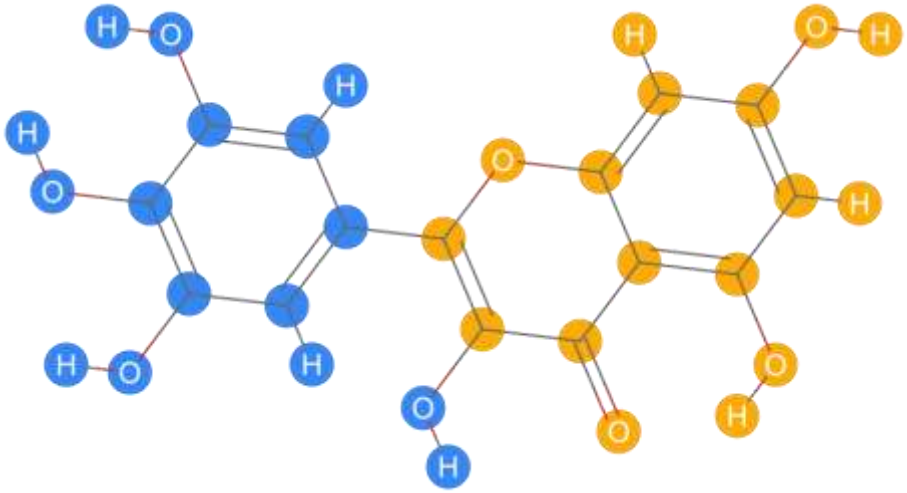
Introduction

Many real-world systems are naturally graphs:

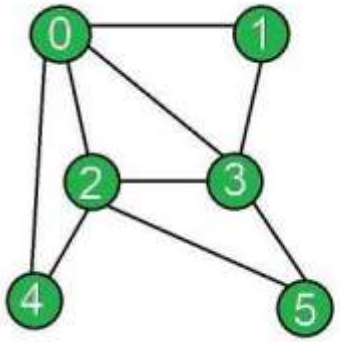
- Social networks → people as nodes, friendships as edges
- Molecules → atoms as nodes, bonds as edges

Graphs capture **relationships** and **structures**, not just individual data points.





What is Graph?



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

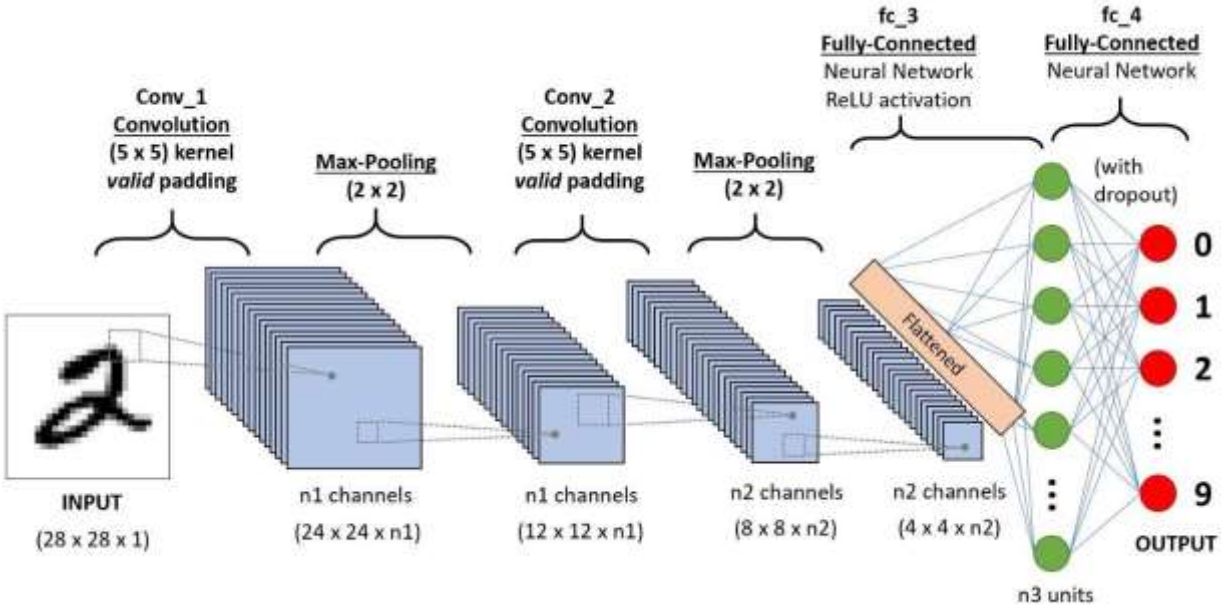
A graph G = (V, E)

- V = set of nodes (vertices)
- E = set of edges (connections)

Often represented by adjacency matrix or edge list.

Why Not Just Use Standard ML

- Most ML tasks assume data in Euclidean space (images, text, audio).
- CNNs work on images: fixed grid structure.
- RNNs work on sequences: ordered data.
- Increasingly, data come from non-Euclidean domains. Such data are often represented as graphs.
- Graphs are irregular:
    - No fixed ordering of nodes.
    - Variable size (different number of nodes).
    - Complex connectivity patterns.
    - Need special models that respect permutation invariance and local neighborhoods.



Basics of GNNs

Inputs to GNNs

- G = (V,E), where:
  - V – set of nodes
  - E –set of edges

1. Node Features:

$x_v \in \mathbb{R}^d$ - Feature vector for each node

Example: Particle kinematics

2. Edge Features:

$e_{uv} \in \mathbb{R}^k$ - Relationship between connected nodes

Example: Distance between particles

Node and Hidden Embeddings

- Node Embedding – each node is represented by a feature vector $h_v^{(k)}$ at layer k. Initially:

$$h_v^{(0)} = x_v$$

- Hidden Embeddings - Represent intermediate states of node information after k message-passing steps

General GNN propagation rule

- Nodes exchange information with their neighbors.
- Each node updates its representation using:
  - Its own features.
  - Features from its neighbors.
- Iterative process: after multiple layers, nodes “know” about distant parts of the graph.

General GNN message-passing update equation:

$$h_v^{(k)} = f_\theta^{(k)} \left(h_v^{(k-1)}, \left\{ h_u^{(k-1)} \mid u \in N_v \right\} \right)$$

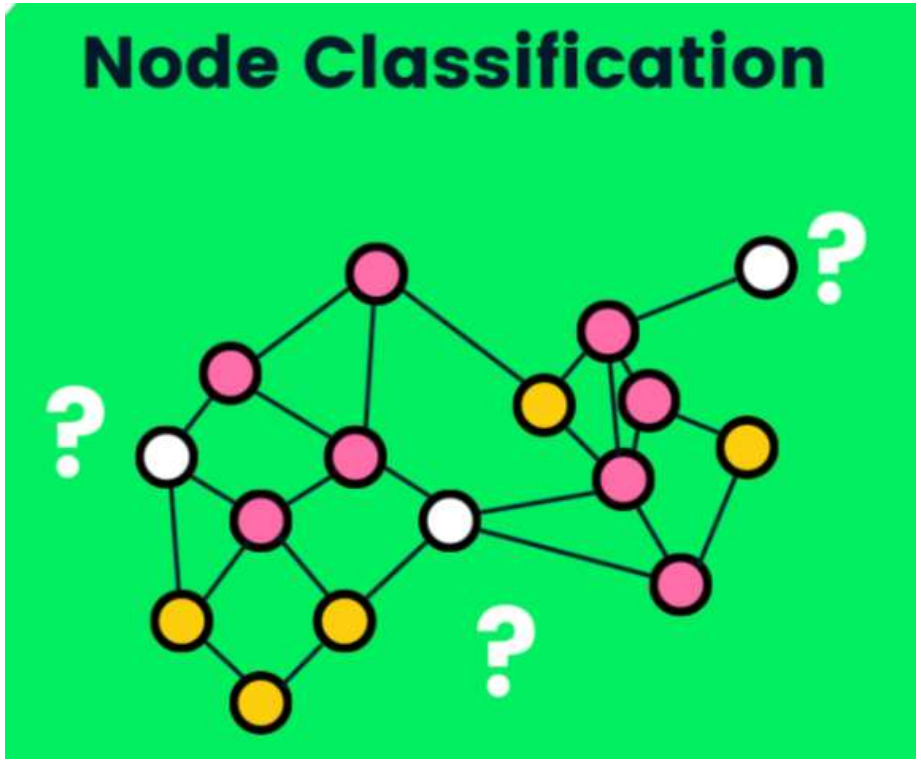
k = 1, … K – layers

$N_v = \{u \in V \mid (u, v) \in E\}$ - local neighborhood of node v

Classification Tasks

Node Classification

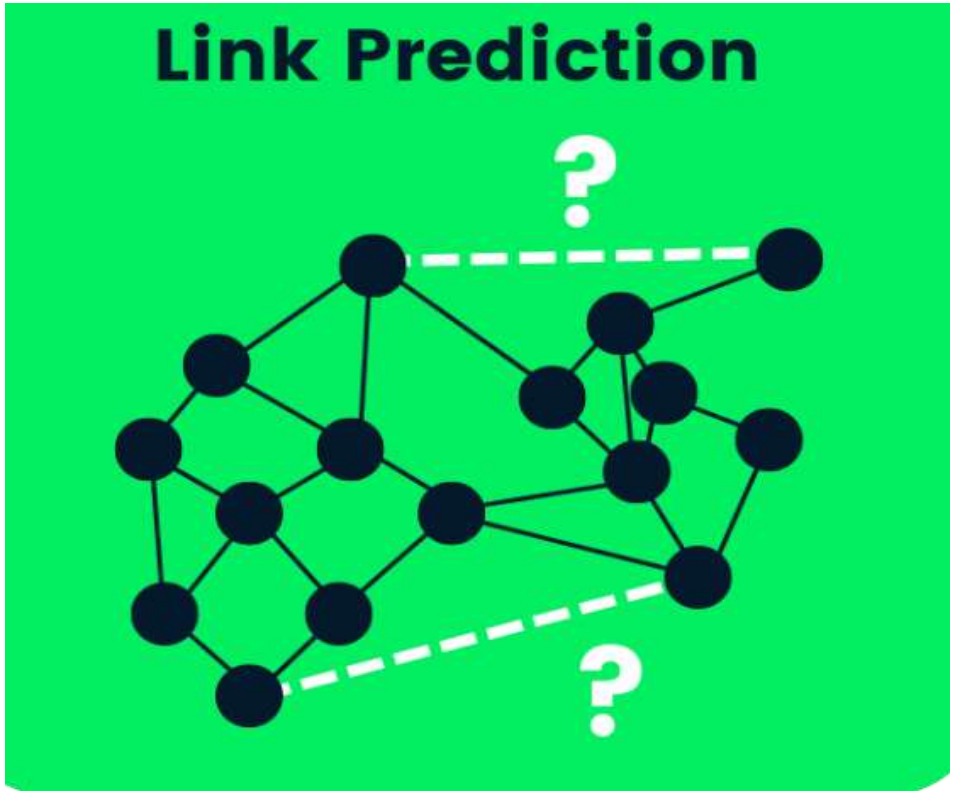
1. Node Classification:
    - Goal: Predict the label of individual nodes
    - Input: Single graph
    - Output: A label of each node
    - Example: Predicting the topic of a paper in a citation graph



Edge classification (Link Prediction)

2. Edge Classification:

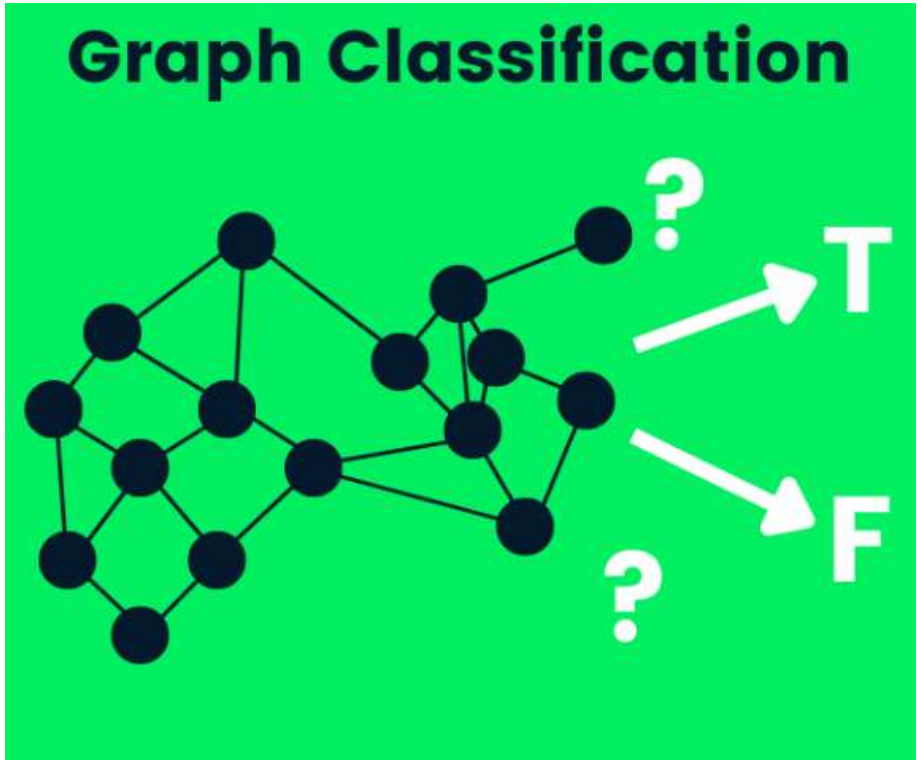
- Goal: Predict the type or existence of an edge between two nodes
- Input: Node embeddings or pair features
- Output: Label or probability for each possible edge
- Example: Predicting whether two users will be friends



Graph Classification

3. Graph Classification:

- Goal: Predict a label for the entire graph
- Input: Many small graphs
- Output: One label per graph
- Example: Classifying molecules as toxic or non-toxic



Types of GNN Architectures

Message Passing Neural Network

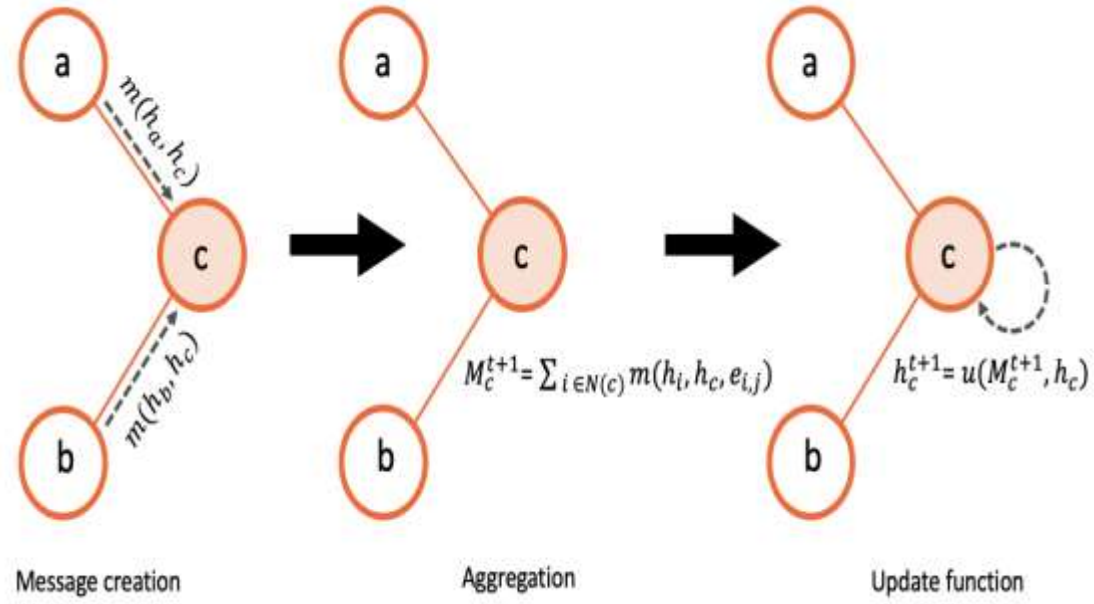
GNN layer is consist of three functions:

1. Message passing function $m_{vw}^{t+1} = M\left(h_v^t, h_w^t, e_{v,w}\right)$

2. Aggregation function $m_v^{t+1} = \sum_{w \epsilon N(v)} m_{vw}^{t+1}$

3. Update activation function $h_v^{t+1} = U(h_v^t, m_v^{t+1})$

4. Readout (per node output/global graph-level output) $y = R(h_v^t \forall v \in V)$



Pros & Cons

- Very general framework — most GNNs (GCN, GAT, etc.) can be expressed as MPNNs.
- Naturally captures local dependencies between nodes through iterative message aggregation.
- Flexible — allows custom message, aggregation, and update functions.

- Expensive for large graphs (each layer needs neighbor communication).
- Over-smoothing: after many layers, node embeddings become indistinguishable.
- Hard to scale to dynamic or time-varying graphs.
- Limited long-range dependency capture without deeper networks (which increases cost).

Graph Convolutional Networks

Introduced by Kipf & Welling (2017)

$$H^{(l+1)} = f(H^{(l)}, A) = \sigma\left(\check{\mathrm{D}}^{-\frac{1}{2}}\,\hat{\mathrm{A}}\check{\mathrm{D}}^{-\frac{1}{2}}\,H^{(l)}\,W^{(l)}\right)$$

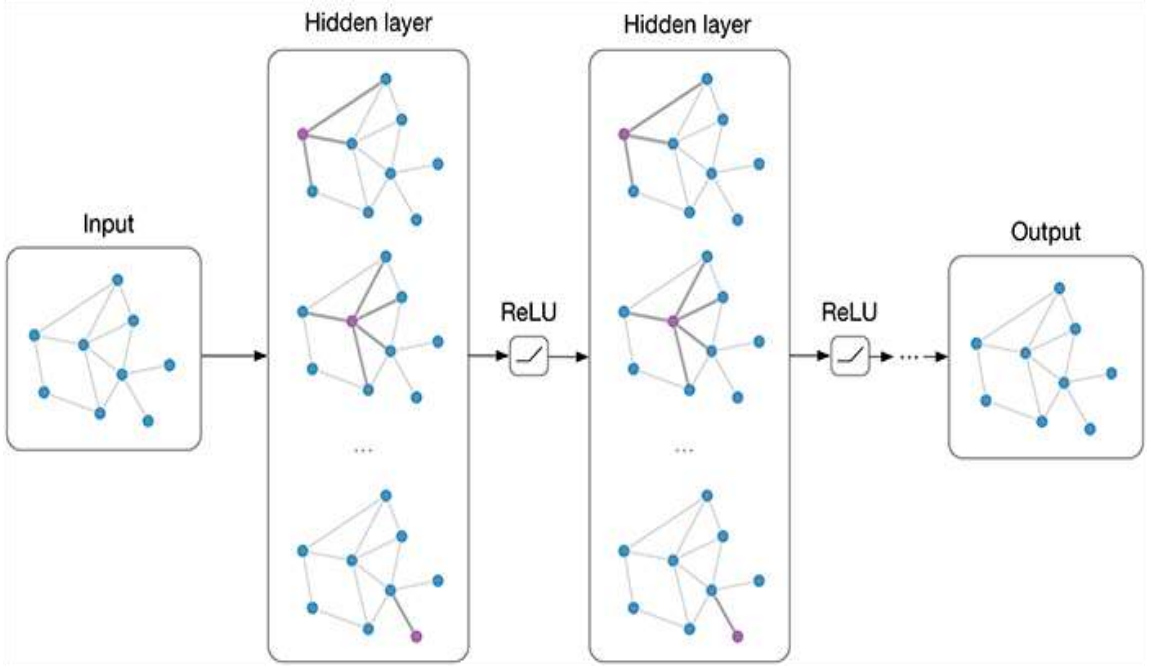
$\hat{\mathrm{A}} = \mathrm{A} + \mathrm{I}$ – Adjacency matrix with self-loops (Aggregate information from neighbors)

$\check{\mathrm{D}}$ – Degree of matrix $\hat{\mathrm{A}}$ (Used for normalization)

$H^{(l)}$ - Node embeddings at layer l

$W^{(l)}$ - Weight matrix

σ – Activation function (ReLu)



Pros & Cons

- Simpler and more efficient special case of MPNNs (uses linear aggregation).
- Performs well for node classification and semi-supervised learning.
- Easy to implement and widely supported in libraries (PyTorch Geometric, DGL).

- Assumes homophily — nodes connected by edges should have similar features.
- Over-smoothing with deeper layers.
- Struggles with directed, weighted, or heterophilic graphs.
- Information loss when averaging neighbors with equal weights.

Graph Attention Networks

- GATs extend GCNs using attention mechanisms.
- They allow nodes to weigh the importance of their neighbors dynamically.
- Motivation: In GCNs, all neighbors contribute equally, but not all are equally important.
- GATs learn attention coefficients to focus on the most relevant connections

Each node v computes attention coefficients with its neighbors u:

$$e_{vu} = a(Wh_v, Wh_u)$$

h_v - feature vector of v

W – learnable weight matrix

a – attention function

Then coefficients are normalized:

$$\alpha_{vu} = softmax_u(e_{vu})$$

α_{vu} - shows how important u to v

Graph Attention Networks

- The new embedding for node v is computed as a weighted sum of its neighbors' features:

$$h'_v = \sigma\left(\sum_{u \epsilon N(v)} \alpha_{vu} W h_u\right)$$

- To stabilize training and capture multiple types of relationships, GATs often use multiple attention heads:

$$h'_v = ||_{k=1}^{K} \sigma\left(\sum_{u \epsilon N(v)} \alpha_{vu}^{(k)} W^{(k)} h_u\right)$$

Pros & Cons

- Learns importance of each neighbor using attention — adaptive weighting improves performance.
- Handles graphs with varying node degrees naturally.
- Reduces noise from irrelevant neighbors.
- Good for heterogeneous or complex relationship graphs.

- Computationally heavier due to attention coefficient calculation.
- Attention may overfit on small or sparse datasets.
- Scaling to very large graphs requires sampling or approximation.
- Interpretability of attention weights can sometimes be misleading.

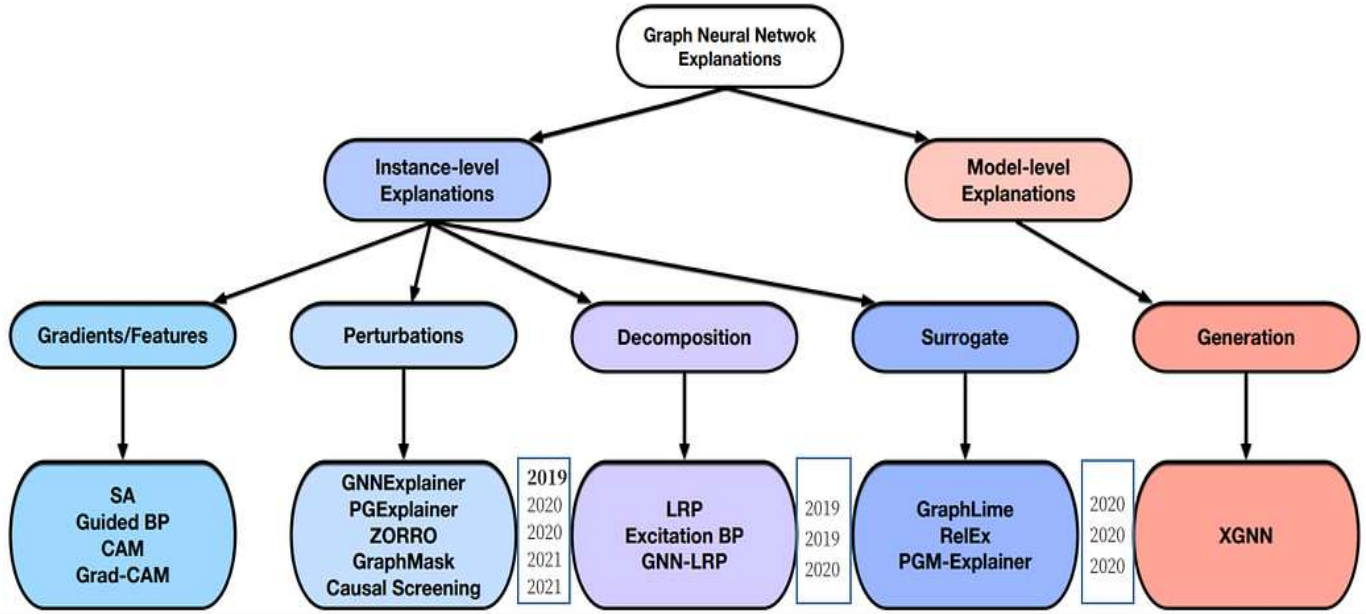
Explainability

Why Explainability for GNNs

- GNNs are powerful but often black boxes (tough to know why a prediction was made).
- Explanations help to debug models.
- Key nodes and edges: Shows the most important components within a prediction.
- Feature importance: Specific node and edge features that make decisions.

Explainablity Techniques

- Instance-level Explanations: Focus on understanding individual predictions making the complex more manageable.
- Model-level Explanations: Uncover the overall decision-making process providing a broader understanding.



이미지 출처: Explainability in Graph Neural Networks: A Taxonomic Survey (Yuan et al. 2020)

Instance-level Explanations

Gradients/Features-Based Methods

- Simplest approach used in picture and text jobs.
- Focuses on identifying important input features by backpropagating gradients.
- Feature-based approaches transfer hidden features to the input space using interpolation.
- Larger gradients or feature values typically denote increased importance in such methods.
- Given that hidden features and gradients have strong relationships with model parameters such explanations can reveal the model’s information.

Saliency Analysis (SA)

- This method highlights the input features that contribute the most to the model's prediction.
- Uses squared gradient values as value scores for input features.
- Simple and efficient.
- Saturation problems.
- Can only reflect the sensitivity between input and output which cannot accurately show the importance.

Guided Backpropagation (GuidedBP)

- This method uses a guided version of backpropagation to identify the most important input features.
- Guided BP only back propagates positive gradients while clipping negative gradients to zeros since negative gradients are difficult to understand.
- Shares the same problems as SA.

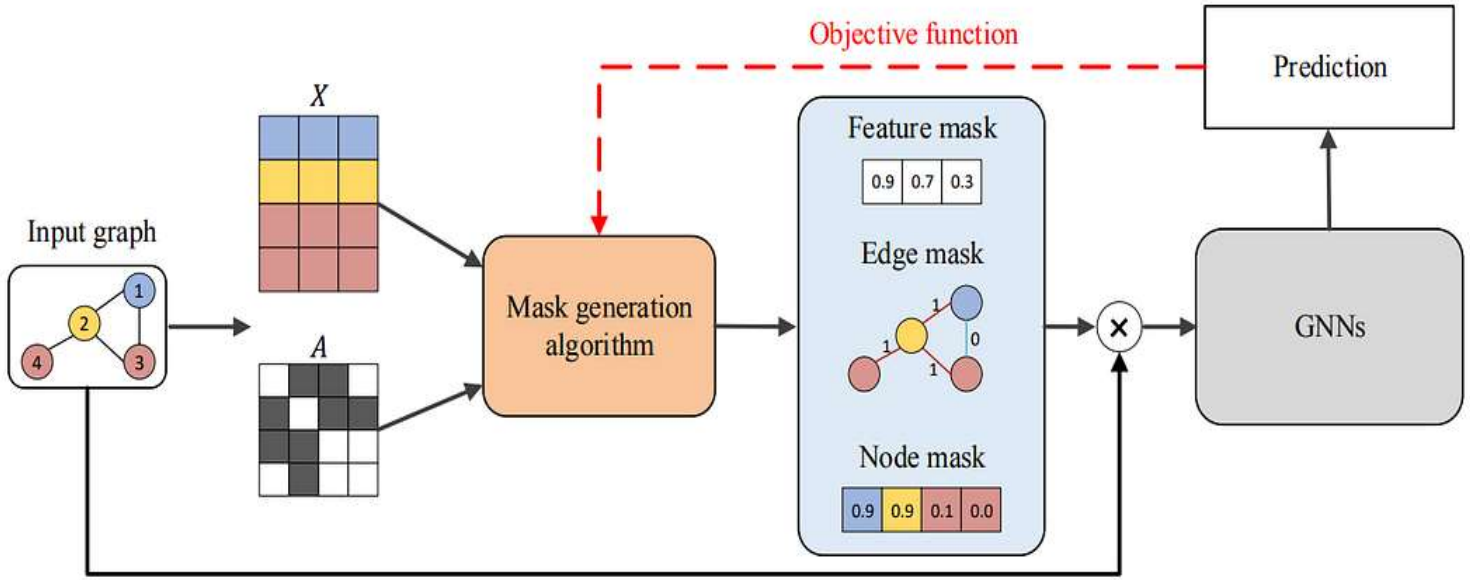
Class Activation Mapping (CAM)

- This method generates heatmaps to visualize the regions of the input that contribute the most to the model's prediction.
- CAM requires the use of a fully-connected layer as the final classifier and a global average pooling layer in the GNN model.
- Weights are obtained from the final FC layer connected with the target prediction.
- It takes the final node embeddings and combines different feature maps by weighted summations to obtain importance scores for input nodes.

- Restricts its application and generalization. (GNN must meet specific criteria).
- Only explain graph classification models, because the final FC layer is required to map predictions to various nodes.

Perturbation-based Methods

- Involve modifying the input graphs to observe the model's reactions, revealing crucial nodes and edges.
- But not all graphs can be resized to have the same number of edges and nodes.
- Structural information between nodes often plays a crucial role in characteristics of a graph.



GNNExplainer

- This method modifies the input graph and measures the change in the model's output to identify important nodes and edges.
- Initializes randomly soft masks for edges and node features. Improved by maximizing the mutual information between predictions obtained by the original graph and predictions of the new graph using backpropagation.
- Final masks explain the relative importance of various parts of the graph.

- Suffers from the "introduced evidence" (new nodes or edges).
- The created masks are graph-specific and not easily generalizable.

PGExplainer

•This method uses a probabilistic graphical model to explain the model's predictions.

• Each edge in a graph first gets an embedding formed by concatenating node embeddings of the nodes that the edge connects.

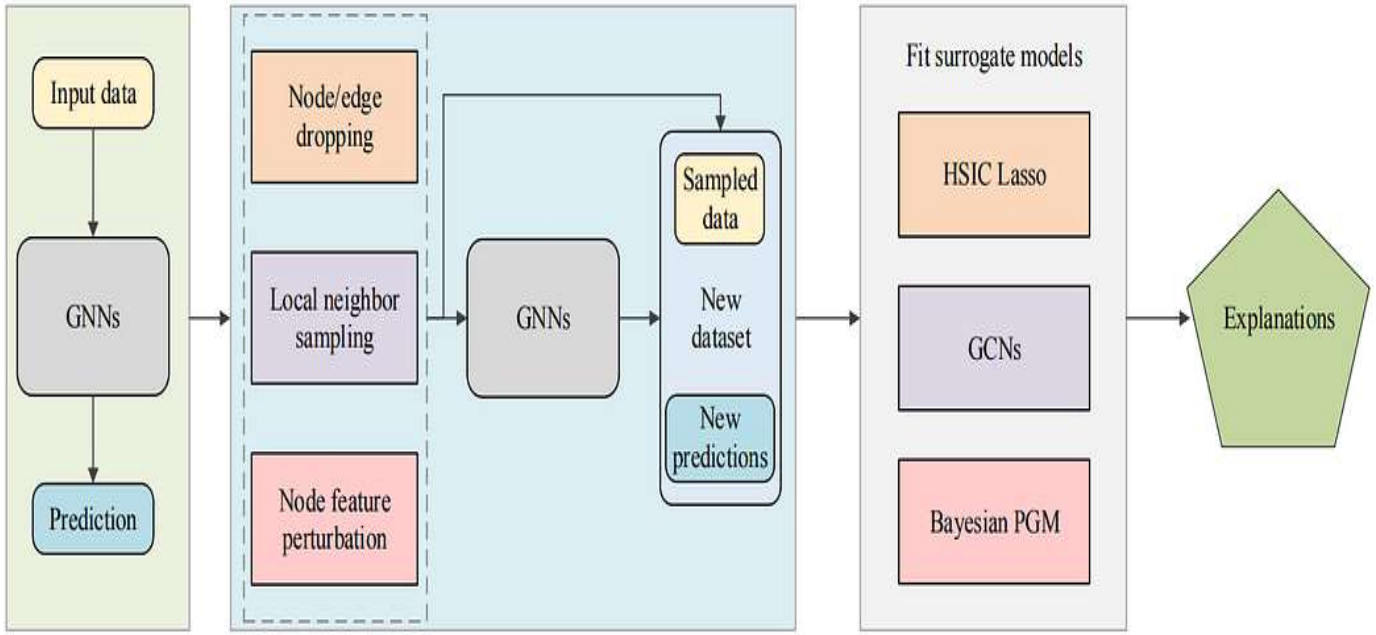
• These edge embeddings are used for predicting the importance of that edge.

•The mask predictor is trained similarly as in GNNExplainer

- The "introduced evidence" problem is largely avoided due to the use of the reparameterization trick.

Surrogate Methods

- The idea is to employ a surrogate model to approximate the predictions of the model.
- The explanations from surrogate model are used to explain the original prediction.



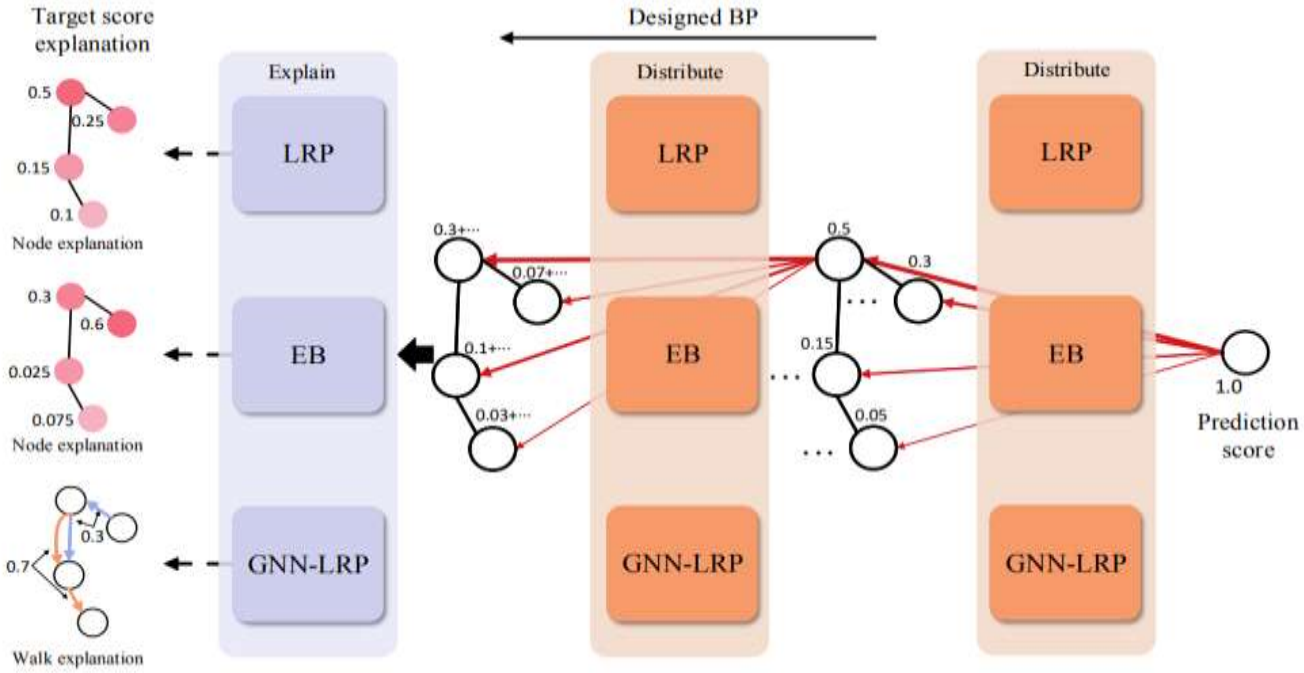
GraphLIME

- It builds a local neighborhood around a target node using its N-hop neighboring nodes (where N = number of GNN Layers).
- The local dataset is learnt using a nonlinear surrogate model.
- Based on the weights attributed to different features in HSIC Lasso it can select important features which are treated as explanations for both the surrogate and the GNN predictions.

- Can only provide explanations for node features and it ignores graph structures such as nodes and edges.
- Cannot be applied to graph classification problems.

Decomposition Methods

- Reveals the relationship between the features and the output predictions.
- Follows that the sum of the decomposed terms should equal the original prediction score.
- Challenging to apply to the graph domain. Structural properties cannot be ignored while distributing scores.



LRP

- Score decomposition rule is developed based on the hidden features and weights.
- The scores of a target neuron are represented as a linear approximation of neuron scores from the previous layer.
- Explanation results are more trustable as LRP is directly developed based on the model parameters.

Model-level Explanations

Model Level Explanations

- Aims for high-level understandings of deep graph models.
- Create graphs that maximally activate the model for specific classes revealing its internal representations.
- Investigates which input graph patterns can result in a particular GNN behavior like optimising a target prediction
- More difficult to explain GNNs at the model level. Underexplored subject.

XGNN

- Explains the GNN model from a global perspective.
- Creates the most important graph patterns for a given class.
- Makes predictions about how to add an edge to the existing graph at each step.
- Created graphs are then input into the trained GNNs to get feedback for policy gradient training of the generator.
- By employing policy gradient to maximize the reward the entire framework is optimized.

- Offers more knowledge of the GNN model through the use of class relevant graph patterns.
- Independent of any specific GNN architecture.
- Less accurate than the instance level techniques.

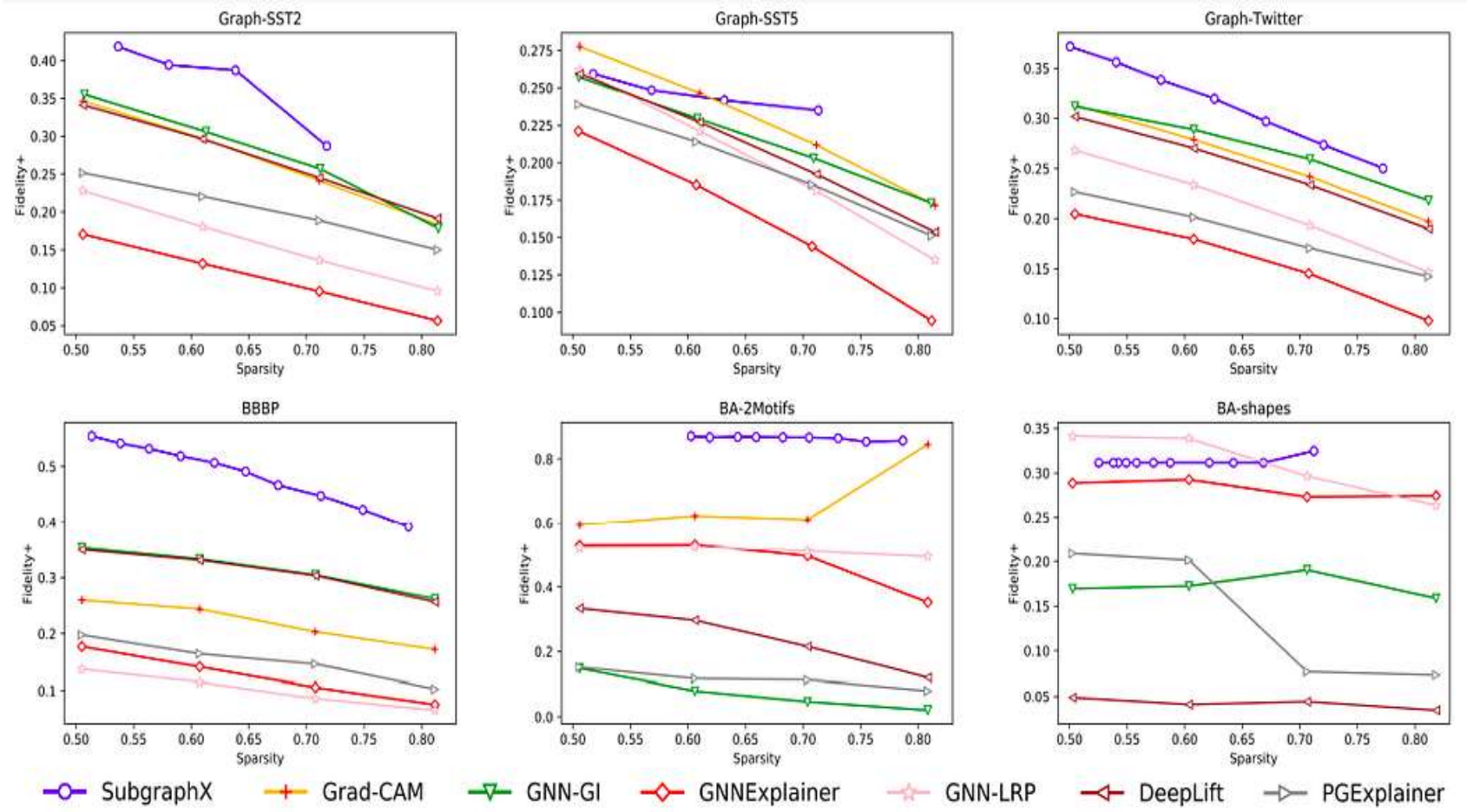
How to Tell If an Explainability Technique Works Well?

- **Fidelity:** checks if the features chosen by the algorithm really matter for the model's predictions. Remove important/unimportant feature and see if accuracy drops/stays the same.
- **Sparsity:** good explanations should capture only the most important input features and ignore the rest (lower is better).
- **Stability:** when small changes are applied to the input the prediction should not be affected
- **Accuracy:** tells how close the algorithm's explanation is to the real reason behind the model's prediction. Did the algorithm find the same important features as true ones?

Fidelity

1.
SubgraphX (**Gradients/Features-Based Methods**) outperforms all other methods.

2. GNN-GI and **GNN-LRP,** generally obtain higher scores than GNNExplainer and PGExplainer.



Accuracy & Stability

Synthetic datasets BA-Shapes and BA-Community. In these datasets the known true patterns (motifs) are turned into true edge masks. These true masks are compared with the edge importance scores predicted by different explanation methods. The comparison is measured using the AUC-ROC.

1.The highest accuracy is obtained by GNN-LRP for the BA-shapes dataset and by GNNExplainer on the BA-Community dataset.

2.DeepLift has the best Stability, but that comes at the cost of accuracy.

Methods	BA-shapes		BA-Community	
Metric	Accuracy	Stability	Accuracy	Stability
GNN-GI	0.8369	0.1361	0.8291	0.1723
GNNExplainer	0.8786	0.1721	0.9194	0.1820
PGExplainer	0.7147	0.0522	0.6843	0.1177
GNN-LRP	0.9243	0.1872	0.8357	0.1239
DeepLift	0.5698	0.0432	0.4190	0.0842

Limitations and Challenges

Graph Structure Limitation

- GNNs can get confused when all nodes have the same features (every node looks identical). In that case, even if two graphs have different connections GNN might still think they are the same.
- This happens because of the aggregation step. GNN combines information from neighbors. If function gives the same result for two different neighborhoods then the model can’t tell those structures apart.

Noise Vulnerability

- GNNs are not robust to noise in graph data
- Adding a slight noise in graph through node perturbation or edge addition/deletion is influencing the output of the GNNs.

Label Scarcity and Overfitting in Scientific Tasks

- In domains like chemistry or biology, labels are expensive to acquire (experiment results)
- Datasets are often small, which leads models to overfit

Future Challenges

- Making GNNs robust and stable under graph perturbations.
- Handling optimization over discrete graph structures effectively.
- Finding good pre-training strategies for GNNs.
- Finding trade offs between accuracy and robustness.

Applications

Charged particle tracking (HEP)

Paper: *Charged particle tracking via edge-classifying interaction networks* (DeZoort et al., 2021)

Application: Identify true track segments from silicon-tracker hits (HL-LHC-like pileup).

- GNN achieves high edge classification accuracy and strong tracking efficiency.
- Models trained on simpler graphs performed well on complex ones.
- The model runs quickly on GPUs (using TorchScript) and is much smaller than earlier GNNs.

Combinatorial optimization with physics-inspired GNNs

Paper: *Combinatorial Optimization with Physics-Inspired Graph Neural Networks* (Schuetz, Brubaker, Katzgraber, 2021)

Application: Solve NP-hard QUBO/PUBO problems (**MaxCut**, **MIS**) via an unsupervised PI-GNN trained on a relaxed Hamiltonian then project to integers.

- On MaxCut: PI-GNN is competitive with or better than classic solvers.
- On MIS: matches traditional algorithms on small graphs and scales; achieves high quality solutions up to very large graphs.

Conclusion

- GNNs have become a powerful framework.
- They show success across fields (social networks, recommendation systems, physics, chemistry).
- Despite these advances challenges remain

Thank you

Papers Used

https://arxiv.org/pdf/1901.00596

https://arxiv.org/pdf/1812.08434

https://arxiv.org/pdf/2107.01188

https://arxiv.org/pdf/2507.13703

https://medium.com/@debjoysaha/a-beginners-guide-to-explainability-in-graph-neural-networks-99927d527718

https://medium.com/@ykarray29/explainable-ai-for-graph-neural-networks-a4b89c89983a