Learning from Graphs

- Part 1: Methodology

Quanming Yao

Assistant Prof. EE. Tsinghua

qyaoaa@tsinghua.edu.cn

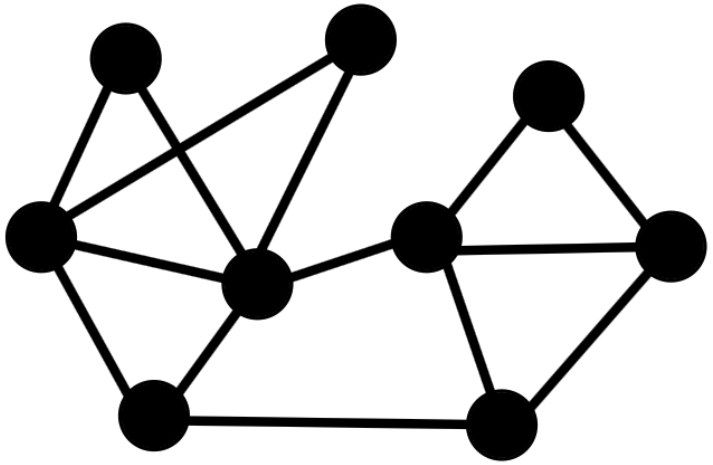
Acknowledgement to CS224W: Machine Learning with Graphs Stanford / Winter 2023 for parts of content.

Outline

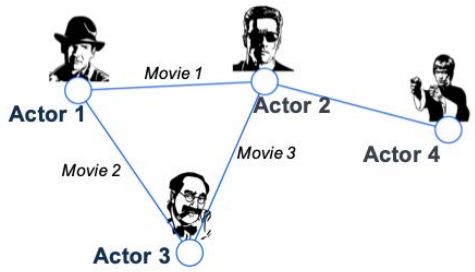
- Applications in Graphs
  - Graph-structured Data
  - Graph Learning Application
- Exemplar Learning Methods on Graphs
  - Overview on Graph Representation Learning
  - Graph Embedding
  - Graph Neural Network (GNN)
  - Graph Transformer
  - Versatile Graph Learning Methods
  - Summary and Discussion
- Theory Insights
  - Theory: expressivity
  - Theory: training
  - Theory: generalization

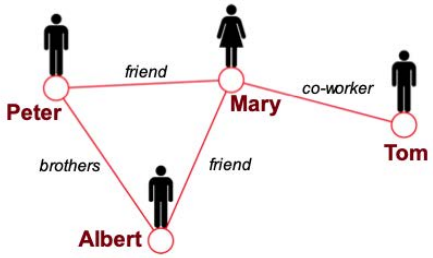
What is Graph?

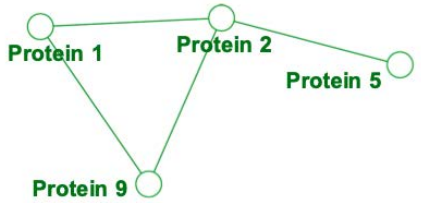
- What is Graph?
- Graphs are a general language for describing and analyzing entities with relations/interactions
  - Node: same / different types, with/without attributes
  - Edge (interaction): directed/undirected, same/different types

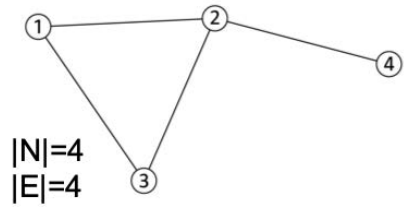


$G = (N, E)$



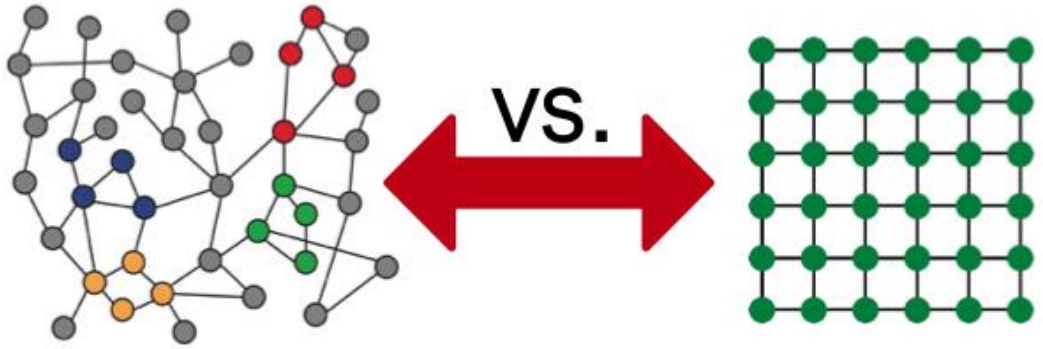






Why Graph?

- Graph connect things, a more general type of structured data
- Not everything can be represented as a sequence or a grid



Networks

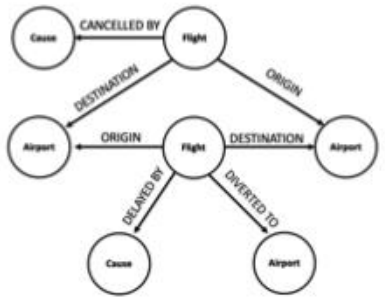
Images

Nodes: pixel
Edges: adjacent pixels

Text

Nodes: word
Edges: adjacent word pairs

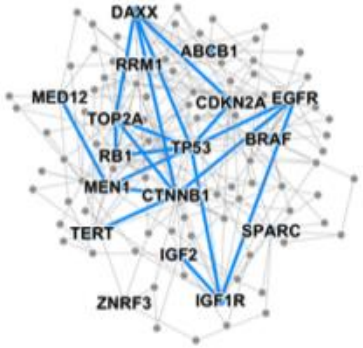
Graph Data in Real-world



Event Graphs

Image credit: SalientNetworks

Computer Networks



Disease Pathways

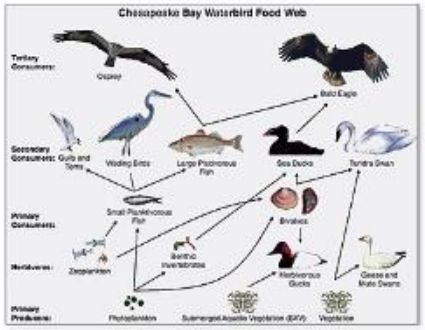


Image credit: Wikipedia

Food Webs

Image credit: Pinterest

Particle Networks

Image credit: visitlondon.com

Underground Networks

Graph Data in Real-world

Image credit: Medium

Social Networks

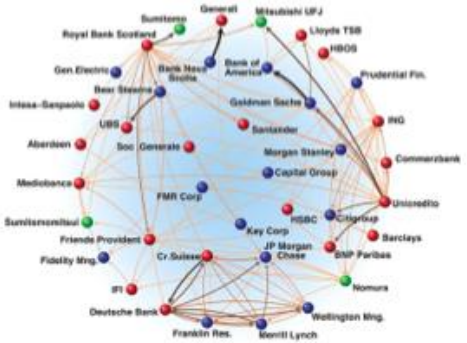


Image credit: Science

Economic Networks

Image credit: Lumen Learning

Communication Networks

Citation Networks

Image credit: Missoula Current News

Internet

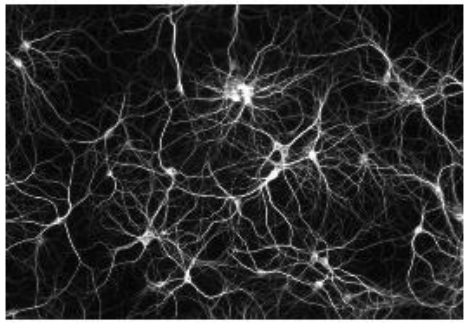


Image credit: The Conversation

Networks of Neurons

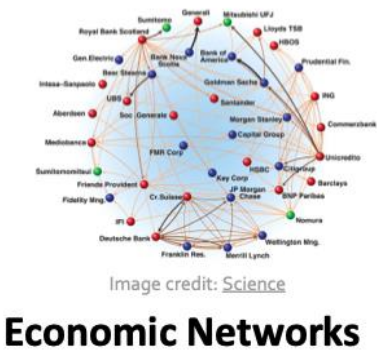
Outline

- Applications in Graphs
  - Graph-structured Data
  - Graph Learning Application
- Exemplar Learning Methods on Graphs
  - Overview on Graph Representation Learning
  - Graph Embedding
  - Graph Neural Network (GNN)
  - Graph Transformer
  - Versatile Graph Learning Methods
  - Summary and Discussion
- Theory Insights
  - Theory: expressivity
  - Theory: training
  - Theory: generalization

What can We Learn from Graph?

Financial anti-fraud task

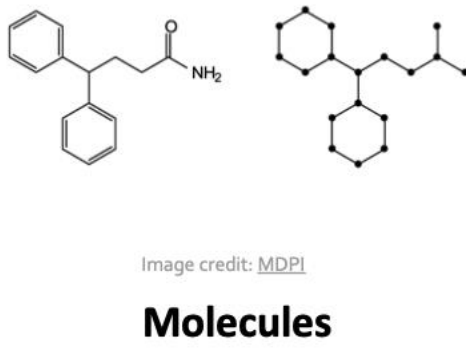
- Different commercial fraud had grown significantly → Anti-fraud model
- Construct economic graph and solved as node-level task
  - Nodes: person / account
  - Edges: transaction record between nodes
  - Target: measuring the risk of each node

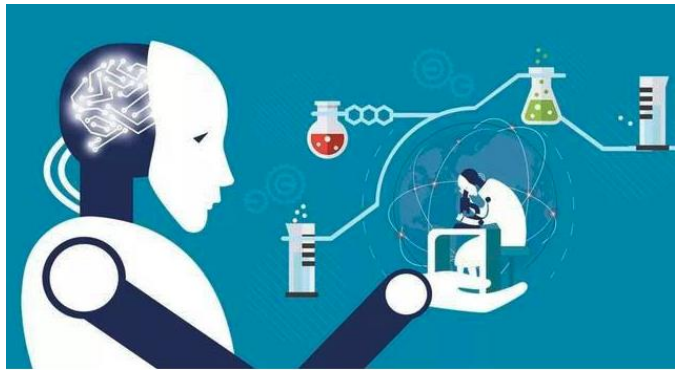


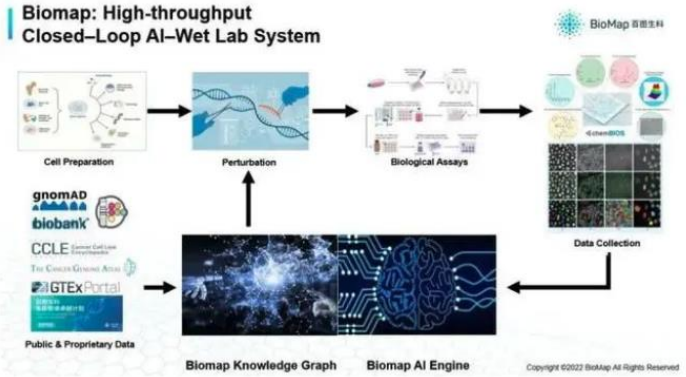
What can We Learn from Graph?

Molecule property prediction

- Traditional drug research is time-consuming and low-efficiency → Solving with ML
- Construct molecule/protein graph and solved as graph-level task
  - Nodes: atom/amino acid
  - Edges: chemical bond between atoms / distances between amino acid
  - Target: predicting the molecule/protein graph properties w.r.t. diseases



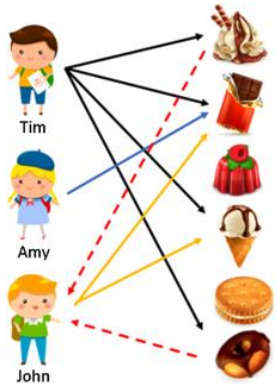




What can We Learn from Graph?

Recommendation system

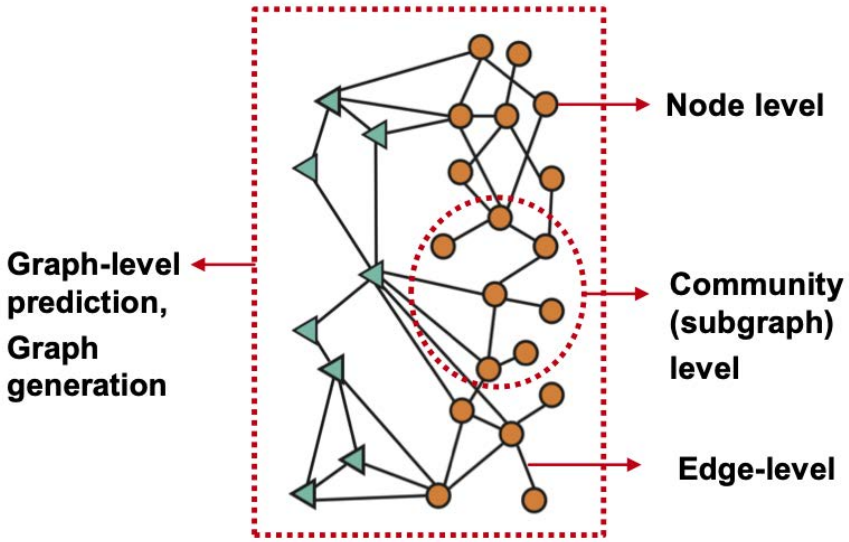
- Recommend goods/videos/news
- Construct User-item graphs and predict the links among them
  - Nodes: goods and users in the database
  - Edges: purchase record between user-item pairs
  - Target: recommend potential goods to users



User-item graph

Summary

Different tasks on the graph



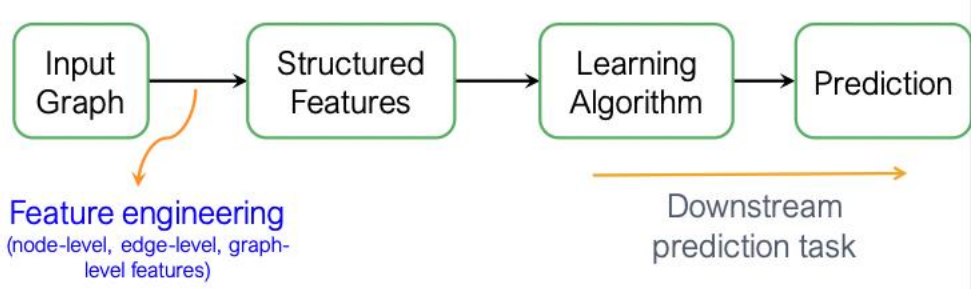
- Node classification: Predict a property of a node
  - Example: Financial anti-fraud Task/categorize online users and items
- Link prediction: Predict whether there are missing links between two nodes
  - Example: Path Planning / Recommendation system
- Graph Classification: Categorize different graphs
  - Example: Molecule property prediction
- Clustering: Detect if nodes from a community
  - Example: Social circle detection
- Other tasks:
  - Graph generation: drug discovery
  - Graph evolution: Physical simulation

Outline

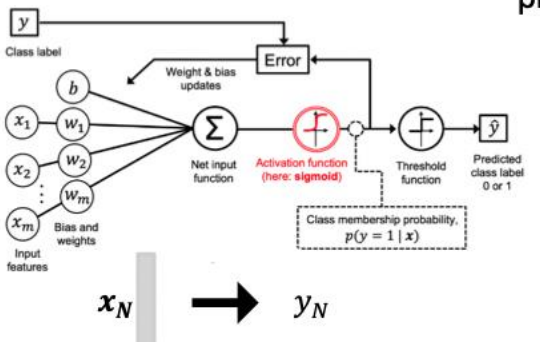
- Applications in Graphs
  - Graph-structured Data
  - Graph Learning Application
- Exemplar Learning Methods on Graphs
  - Overview on Graph Representation Learning
  - Graph Embedding
  - Graph Neural Network (GNN)
  - Graph Transformer
  - Versatile Graph Learning Methods
  - Summary and Discussion
- Theory Insights
  - Theory: expressivity
  - Theory: training
  - Theory: generalization

Graph Representation

- Mapping discrete graph into continuous representations
- Make prediction for the downstream task based on the representations

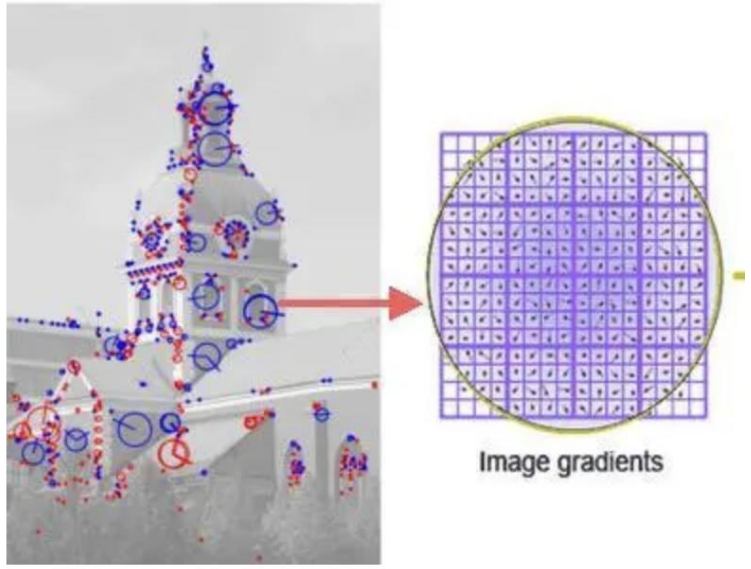


- **Train an ML model:**
  - Logistic Regression
  - Random forest
  - Neural network, etc.
- **Apply the model:**
  - Given a new node/link/graph, obtain its features and make a prediction



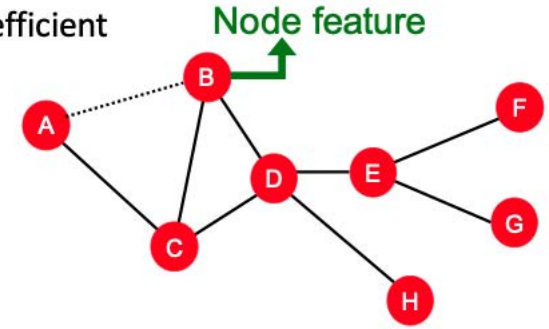
Hand-designed Representation/Feature

- Image: extract rotation /scaling/ brightness-invariant local features of images



- Graph: the permutation-invariant features in the graph structure (irregular structure of each node)

- Node degree
- Node centrality
- Clustering coefficient
- Graphlets

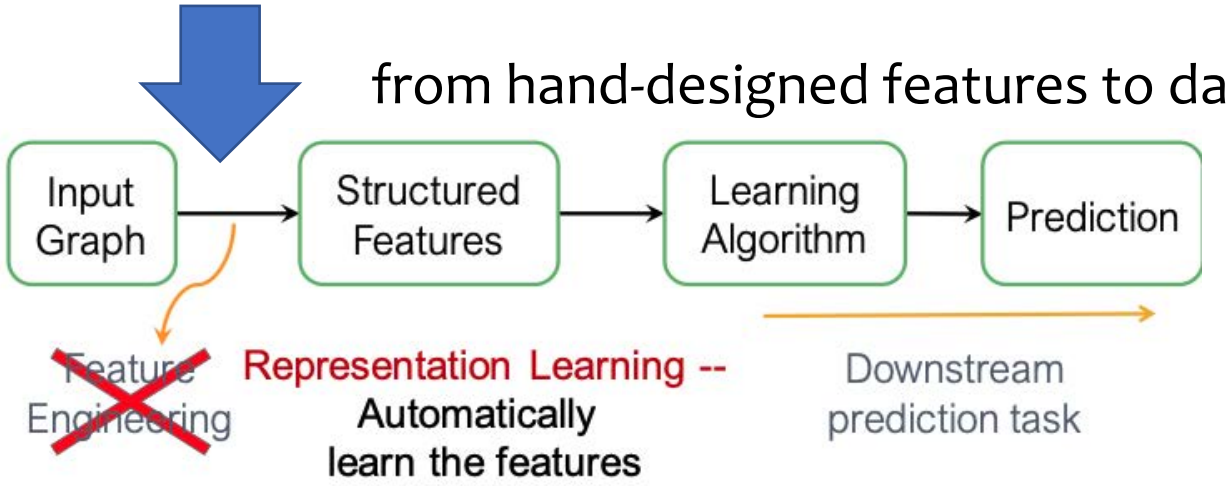


Representation Learning

Hand-designed features:

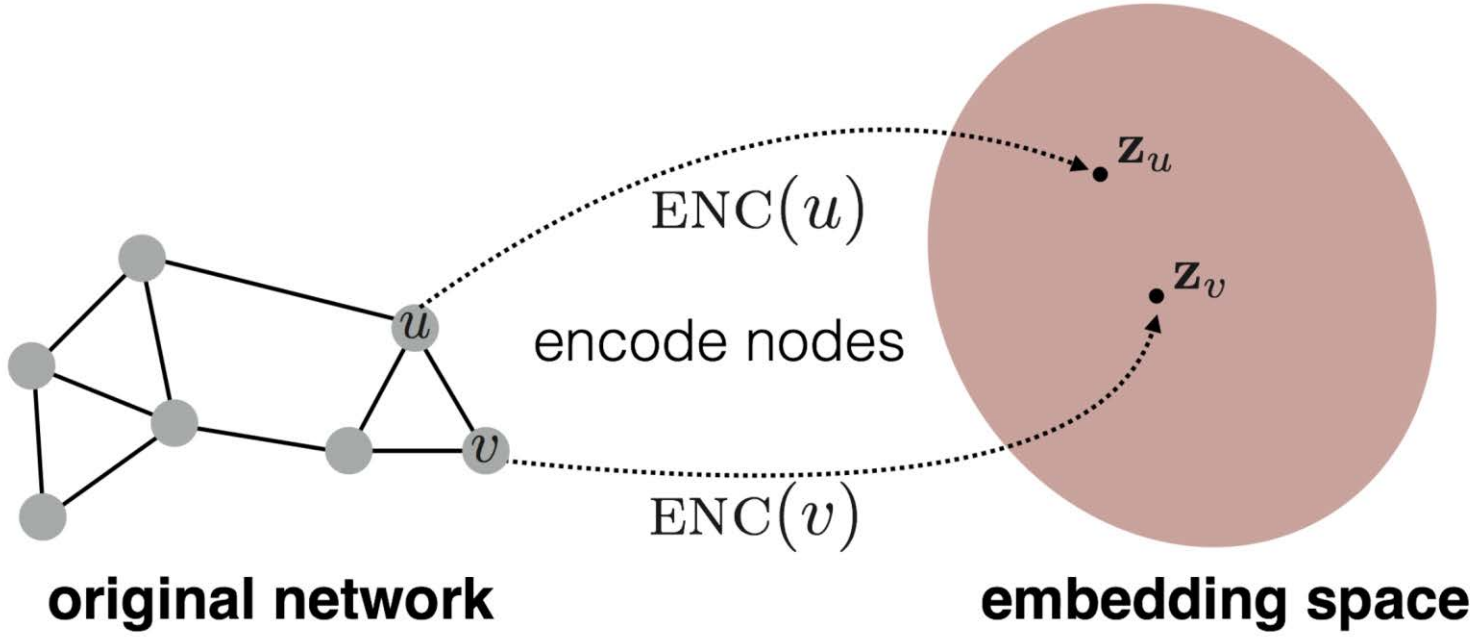
- Largely rely on human expertise
- Low precision on large-scale and diverse real-world data

How to character the properties into representations in a data-driven way?

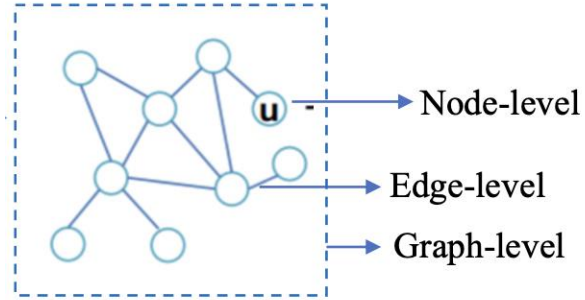


Graph Representation Learning

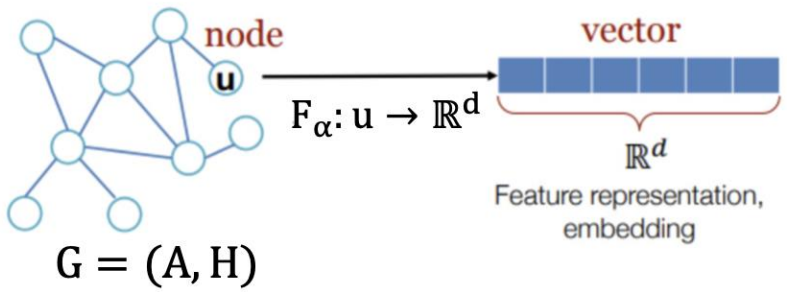
- Target: obtaining the representations which can efficiently capture the graph characterizations



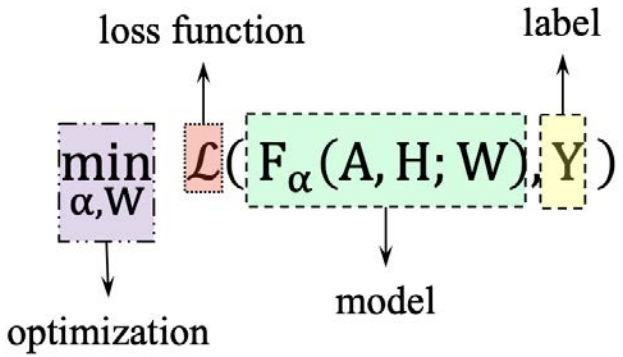
Framework



- Graph representation learning framework

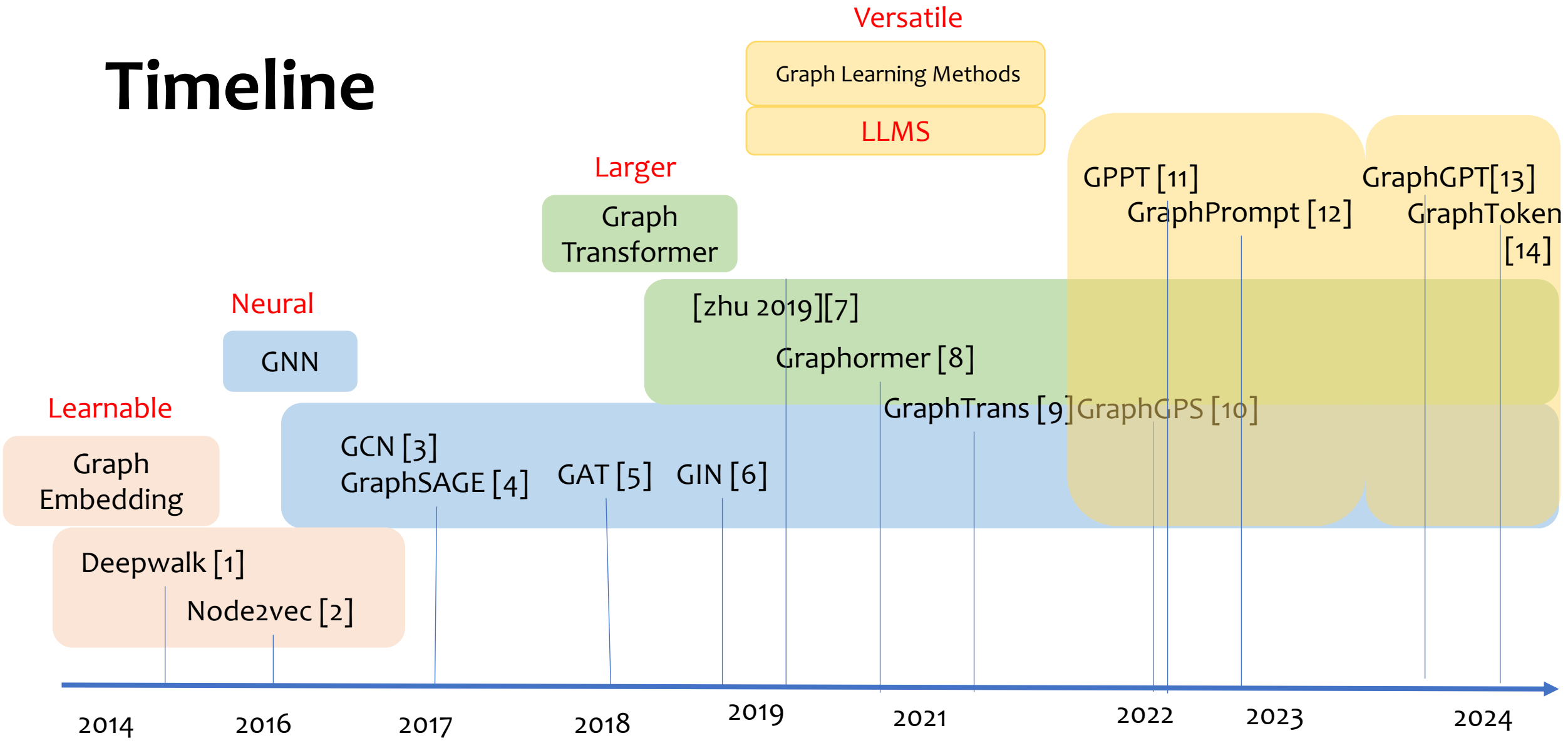


$h_u \in \mathbb{R}^d$



- Node classification:
  - node label y_u
  - Mapping representations into label space $f: \mathbb{R}^d \to \mathbb{R}^C$
  - f: MLP network for example
- Graph classification
  - Graph label y_G
  - Learning the graph representation $f_{rd}: (A, H) \to \mathbb{R}^d$
  - Mapping representations into label space $f: \mathbb{R}^d \to \mathbb{R}^C$
  - f_{rd}: global sum for example $z = \sum_{u \in G} h_u$
- Link Prediction
  - Edge label y_e
  - Learning the edge representation $f_{sf}: (h_u, h_v) \to \mathbb{R}^d$
  - Learning edge score $f: \mathbb{R}^d \to \mathbb{R}^1$
  - Predicting the existence of edge (u, v)
  - f_{sf}: concatenation for example $h = W(h_u || h_v)$

Timeline


Versatile
Graph Learning Methods
LLMS
Larger
Graph Transformer
Neural
GNN
Learnable
Graph Embedding
Deepwalk [1]
Node2vec [2]
GCN [3]
GraphSAGE [4]
GAT [5]
GIN [6]
[zhu 2019][7]
Graphormer [8]
GraphTrans [9]
GraphGPS [10]
GPPT [11]
GraphPrompt [12]
GraphGPT[13]
GraphToken [14]
2014
2016
2017
2018
2019
2021
2022
2023
2024

Reference

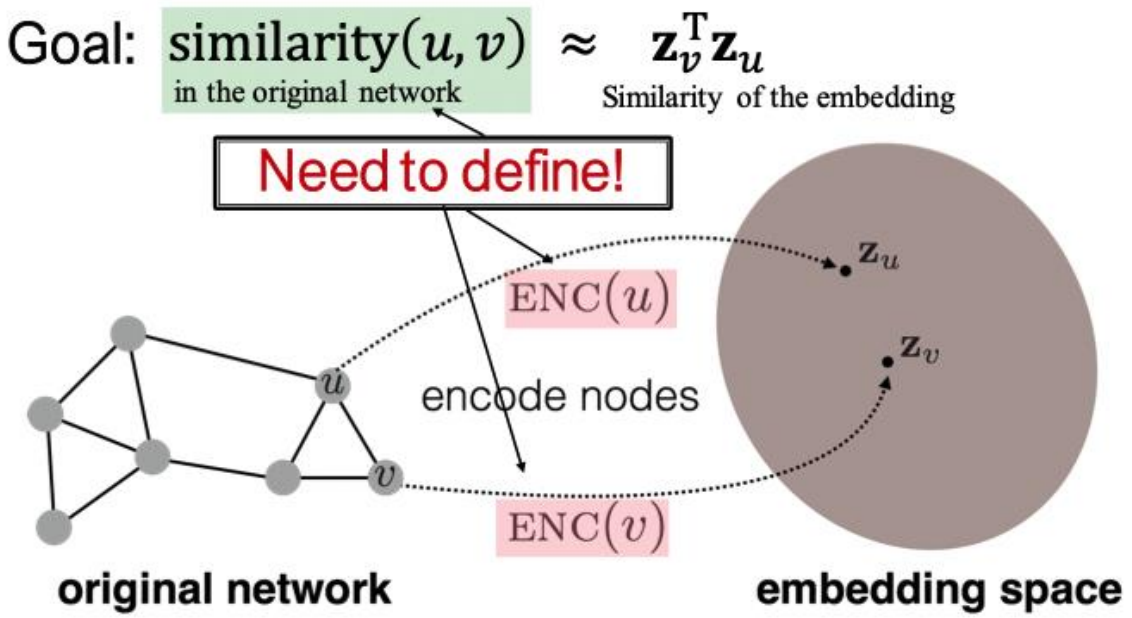
- [1] DeepWalk: Online Learning of Social Representations. KDD 14
- [2] node2vec: Scalable feature learning for networks. KDD 16
- [3] SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS. ICLR 2017
- [4] Inductive Representation Learning on Large Graphs. NeurIPS 2017
- [5] GRAPH ATTENTION NETWORKS ICLR 2018
- [6] HOW POWERFUL ARE GRAPH NEURAL NETWORKS? ICLR 2019
- [7] Modeling Graph Structure in Transformer for Better AMR-to-Text Generation. EMNLP 2019
- [8] Do Transformers Really Perform Bad for Graph Representation? NeurIPS 2021
- [9] Representing Long-Range Context for Graph Neural Networks with Global Attention. NeurIPS 2021
- [10] Recipe for a General, Powerful, Scalable Graph Transformer. NeurIPS 2022
- [11] GPPT: Graph pre-training and prompt tuning to generalize graph neural networks. KDD 2022
- [12] GraphPrompt: Unifying Pre-Training and Downstream Tasks for Graph Neural Networks. WWW 2023
- [13] GraphGPT: Graph Instruction Tuning for Large Language Models. SIGIR 2024
- [14] Let your graph do the talking: Encoding structured data for llms. Arxiv 2024

Outline

- Applications in Graphs
  - Graph-structured Data
  - Graph Learning Application
- Exemplar Learning Methods on Graphs
  - Overview on Graph Representation Learning
  - Graph Embedding
  - Graph Neural Network (GNN)
  - Graph Transformer
  - Versatile Graph Learning Methods
  - Summary and Discussion
- Theory Insights
  - Theory: expressivity
  - Theory: training
  - Theory: generalization

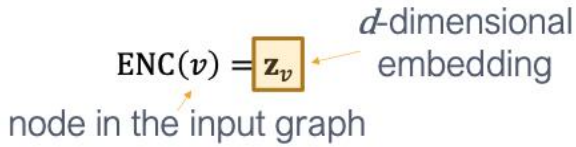
Graph Embedding

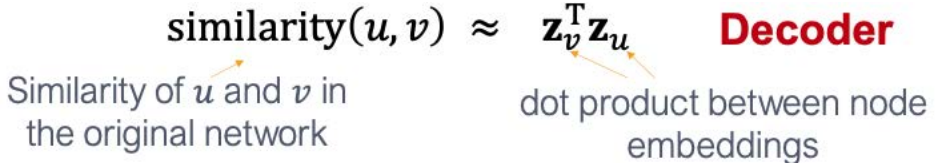
The goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the graph



Graph Embedding – Overview

1. **Encoder** maps from nodes to embeddings
2. **Define a node similarity function** (i.e., a measure of similarity in the original network)
3. **Decoder DEC** maps from embeddings to the similarity score
4. Optimize the parameters of the encoder so that the similar nodes in the origin graph have the similar representations in the embedding space



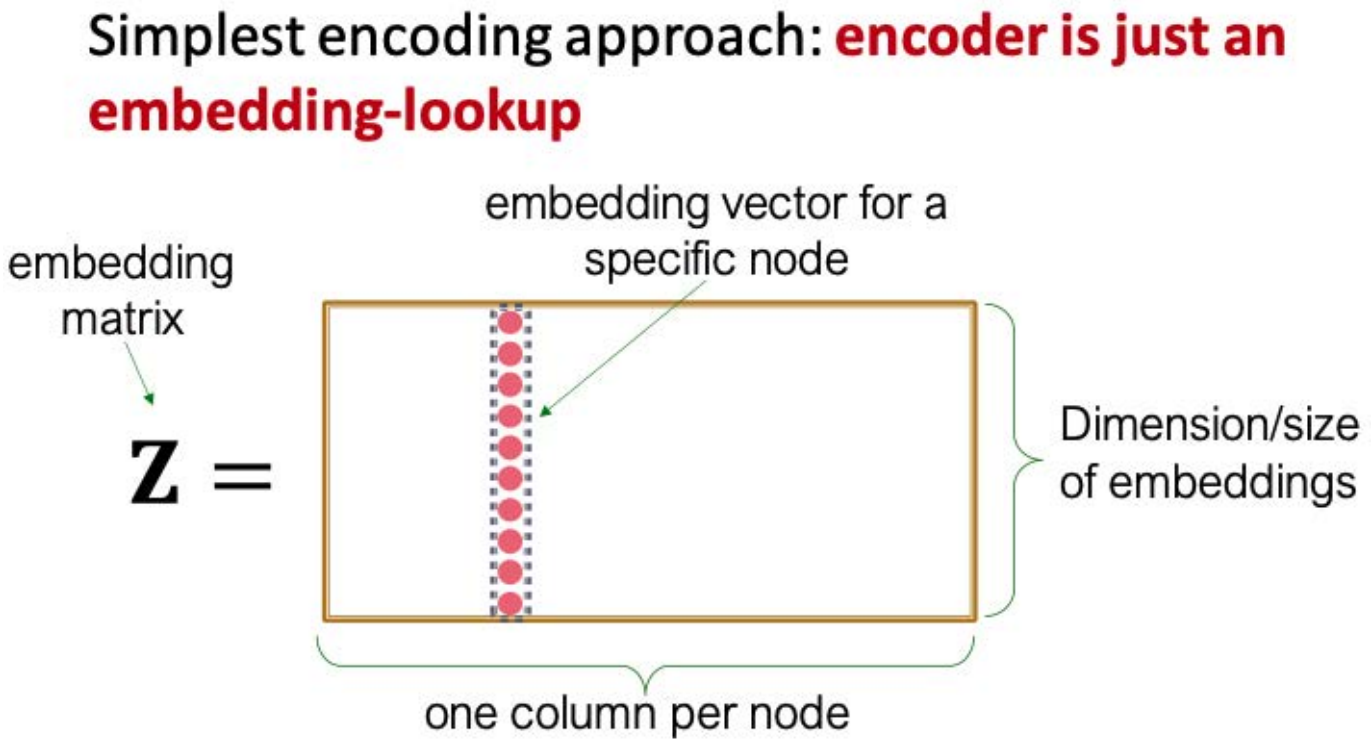


$$\max_f \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) | \mathbf{z}_u)$$

- $N_R(u)$ is the neighborhood of node u by strategy R

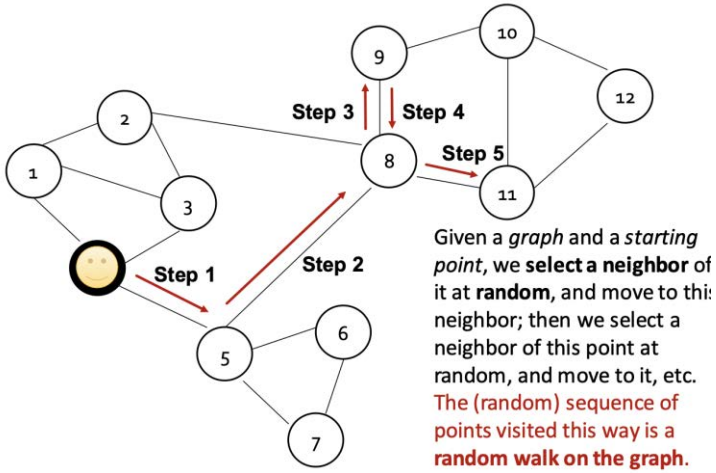
Graph Embedding – Encoder

- Shallow encoder: **Encoder is just an embedding-lookup**
  - $Enc(v) = z_v$



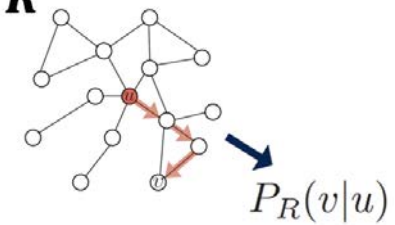
Graph Embedding – Similarity function

- Random-walk based similarity function: Nodes in the same walk have similar node representations



Estimate probability of visiting node v on a random walk starting from node u using some random walk strategy R

Walk: $u \to \cdots \to v$



Two nodes which can be accessed in few steps in the network

Two nodes have large similarities in the origin network (graph view)

Nodes appeared in the same walk (technical view)

Graph Embedding – Optimization

- Nodes in the same walk have similar node representations
  - Log-likelihood objective:

$$\max_f \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) | \mathbf{z}_u) \quad f(u) = \mathbf{z}_u$$

  - $N_R(u)$ is the neighborhood of node u by strategy R

expensive

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u)) = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_n)}\right)$$

sum over all nodes u

sum over nodes v seen on random walks starting from u

predicted probability of u and v co-occuring on random walk

Graph Embedding – Optimization

- Use negative sampling for quick likelihood calculation

$$\log\left(\frac{\exp\left(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v\right)}{\sum_{n\in V}\exp\left(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_n\right)}\right)$$

random distribution over nodes

$$\approx \log\left(\sigma\left(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v\right)\right) - \sum_{i=1}^{k}\log\left(\sigma\left(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_{n_i}\right)\right), n_i \sim P_V$$

- **Gradient Descent**: a simple way to minimize $\mathcal{L}$:
  - Initialize z_u at some randomized value for all nodes u.
  - Iterate until convergence:
    - For all u, compute the derivative $\frac{\partial \mathcal{L}}{\partial z_u}$.
    - For all u, make a step in reverse direction of derivative: $z_u \leftarrow z_u - \eta\frac{\partial \mathcal{L}}{\partial z_u}$.

η: learning rate

Graph Embedding – Procedures

- Run **short fixed-length** random walks starting from each node on the graph
- For each node u collect $N_R(u)$, the multiset of nodes visited on random walks starting from u
- Optimize embeddings using Stochastic Gradient Descent:

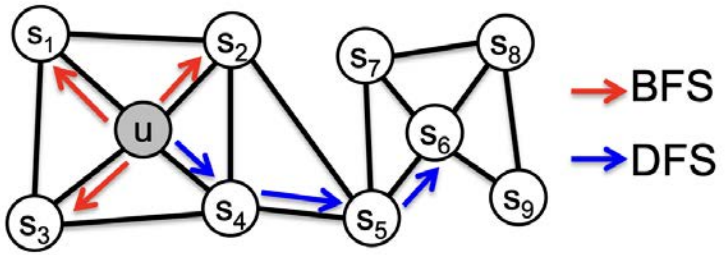
$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

We can efficiently approximate this using negative sampling!

Graph Embedding Extension – Node2vec

- **What strategies should we use to run these random walks?**
- **Goal:** Embed nodes with similar network neighborhoods close in the feature space.

Two classic strategies to define a neighborhood $N_R(u)$ of a given node u:

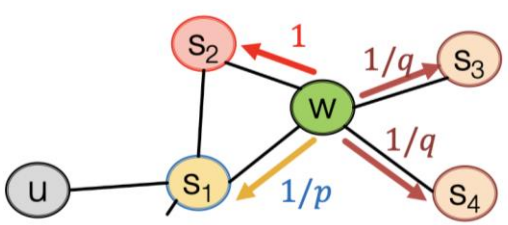


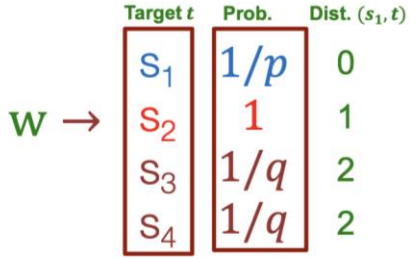
Walk of length 3 ($N_R(u)$ of size 3):

$N_{BFS}(u) = \{s_1, s_2, s_3\}$ Local microscopic view

$N_{DFS}(u) = \{s_4, s_5, s_6\}$ Global macroscopic view

Explore local (BFS) and global (DFS) neighbors





- **BFS-like** walk: Low value of p
- **DFS-like** walk: Low value of q

$N_R(u)$ are the nodes visited by the biased walk

Biases walk: trade-off between local and global

Graph Embedding Extension

- **Different kinds of biased random walks:**
  - Based on node attributes (Dong et al., 2017).
  - Based on learned weights (Abu-El-Haija et al., 2017)
- **Alternative optimization schemes:**
  - Directly optimize based on 1-hop and 2-hop random walk probabilities (as in LINE from Tang et al. 2015).
- **Network preprocessing techniques:**
  - Run random walks on modified versions of the original network (e.g., Ribeiro et al. 2017's struct2vec, Chen et al. 2016's HARP).

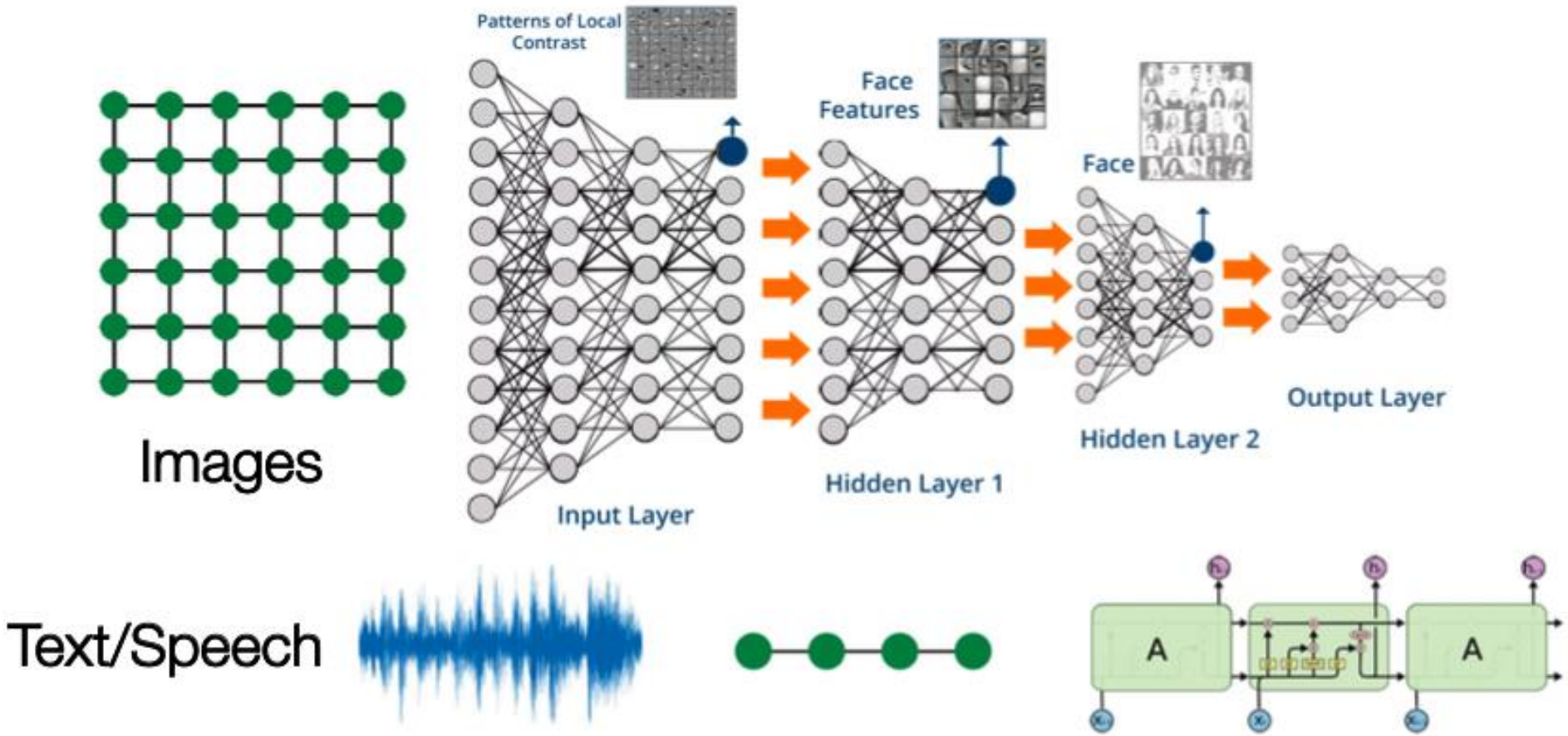
Graph Embedding – Summary

- **Core idea**: Embed nodes so that distances in embedding space reflect node similarities in the original network
- **Different notions of node similarity:**
  - Naive: Similar if two nodes are connected (next)
  - Neighborhood overlap (covered in Lecture 2)
  - Random walk approaches (covered today)
- Limitations:
  - Limited receptive field (node pairs in small distances are preferred)
  - Cannot handle node features.
    → limited performance on real-world dataset.

Outline

- Applications in Graphs
  - Graph-structured Data
  - Graph Learning Application
- Exemplar Learning Methods on Graphs
  - Overview on Graph Representation Learning
  - Graph Embedding
  - Graph Neural Network (GNN)
  - Graph Transformer
  - Versatile Graph Learning Methods
  - Summary and Discussion
- Theory Insights
  - Theory: expressivity
  - Theory: training
  - Theory: generalization

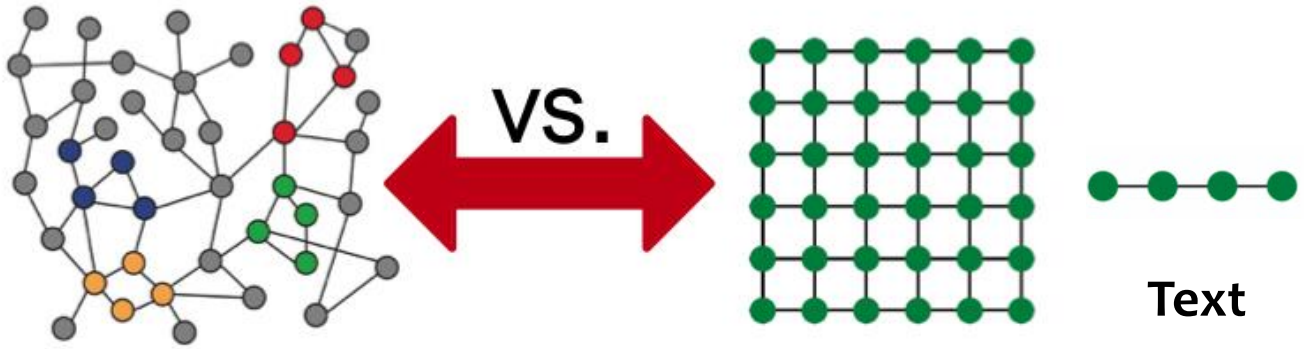
Deep Learning



Deep Learning on Graph

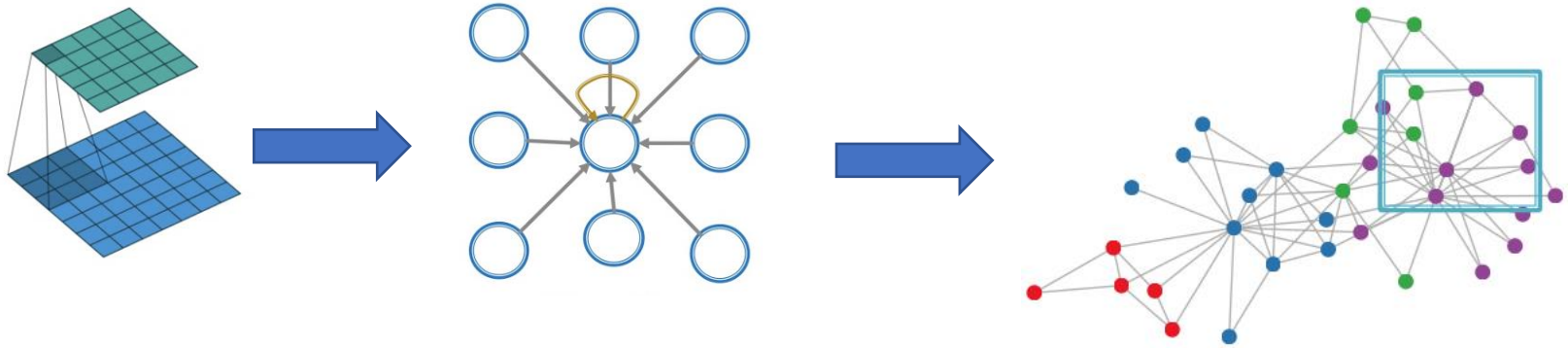
Graphs are more complex

- Arbitrary size
- Complex topological structures (not like grids as images)
- No fixed node ordering or reference point
- Have multimodal/dynamic features



Deep Learning on Graph

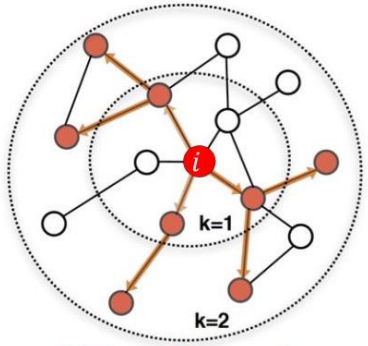
- Convolution operation on images
- To generalize convolutions beyond simple lattices
- Leverage features/attributes of nearby pixels



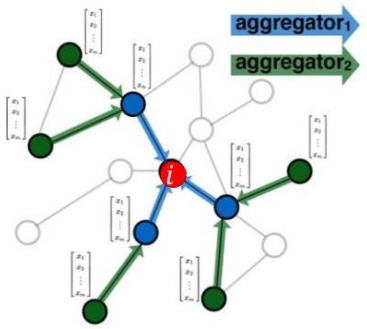
- Convolution on Graph
  - Transform "messages" h_i from neighbors: $W_i h_i$
  - Add them up: $\sum_i W_i h_i$

Graph Neural Networks (GNN)

- Key idea:
  - Node's neighborhood defines a computation graph
  - Learn how to **propagate information** across the graph to compute node features
- Generate node embeddings based on **local network neighborhoods, i.e.,** aggregate information from their neighbors using neural networks



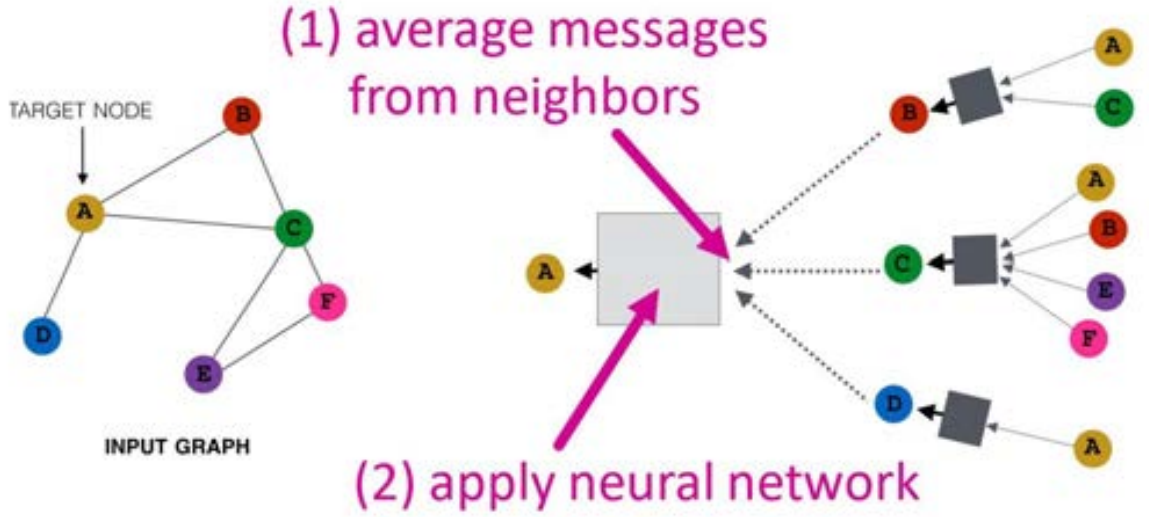
Determine node computation graph



Propagate and transform information

How does Graph Neural Networks Work?

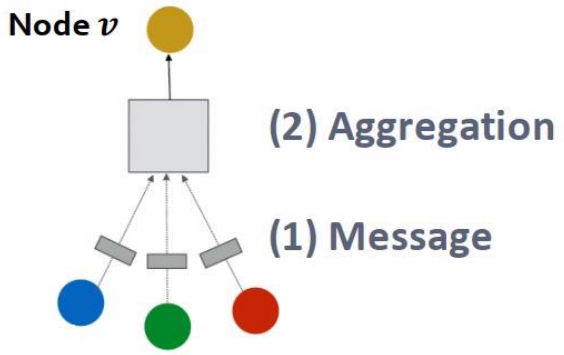
- Aggregate information from neighbors using neural networks
- Basic approach: use a message passing function
  - Average information (messages) from neighbors
  - Apply a neural network



Graph Neural Networks

Message passing function

- Input: node features of v and its neighbors
- Message function: $m_u^{(l)} = \mathrm{MSG}^{(l)}\left(h_u^{(l-1)}\right)$, e.g., A linear layer $m_u^{(l)} = \mathrm{W}^{(l)} h_u^{(l-1)}$
- Aggregation function: $h_v^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{m_u^{(l)}, u \in N(v)\right\}\right)$, e.g., Sum/Mean/Max



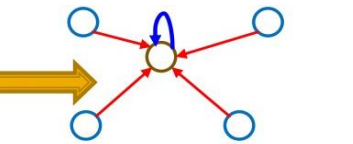
Graph Neural Networks

- In each layer, learn representations for all nodes

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

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

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

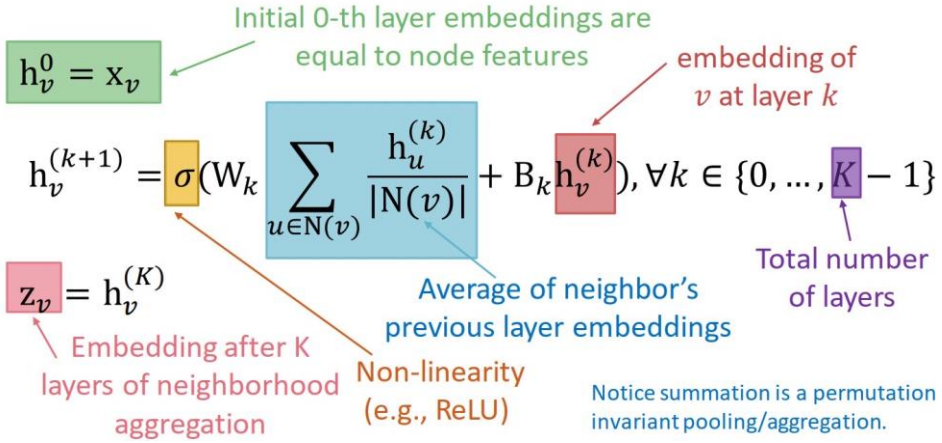


$$H^{(k)} = [h_1^{(k)} \dots h_{|V|}^{(k)}]^T$$

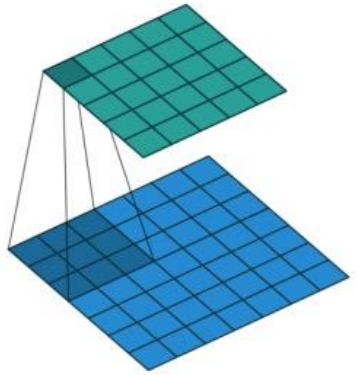
- Red: neighborhood aggregation
- Blue: self transformation

All nodes share the same transform parameter

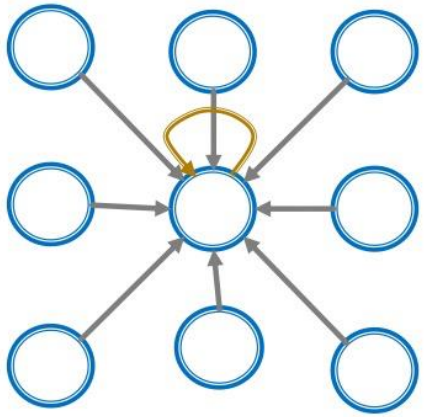
- Construct GNN by stacking message-passing-layer



Graph Neural Networks – CNN vs. GCN



Image



Graph

- GNN formulation: $\mathrm{h}_v^{(l+1)} = \sigma(\mathbf{W}_l \sum_{u \in \mathrm{N}(v)} \frac{\mathrm{h}_u^{(l)}}{|\mathrm{N}(v)|} + \mathrm{B}_l \mathrm{h}_v^{(l)}), \forall l \in \{0, \dots, L-1\}$
- CNN formulation: (previous slide) $\mathrm{h}_v^{(l+1)} = \sigma(\sum_{u \in \mathrm{N}(v) \cup \{v\}} \mathrm{W}_l^u \mathrm{h}_u^{(l)}), \forall l \in \{0, \dots, L-1\}$

  if we rewrite: $\mathrm{h}_v^{(l+1)} = \sigma(\sum_{u \in \mathrm{N}(v)} \mathbf{W}_l^u \mathrm{h}_u^{(l)} + \mathrm{B}_l \mathrm{h}_v^{(l)}), \forall l \in \{0, \dots, L-1\}$

Graph Neural Networks – Summary

- Existing GNNs can be summarized with the message-passing scheme, and more methods can be designed based on this scheme

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

Method	Message	Aggregate	total
GCN	$m_u^{(l+1)} = \frac{1}{\lvert N(v)\rvert} W^{(l)} h_u^{(l)}$	Sum	$h_v^{(l+1)} = \sigma\left(W^{(l)} \sum_{u \in N(v)} \frac{1}{\lvert N(v)\rvert} h_u^{(l)}\right)$
GraphSAGE	Sum/Mean/ Max/LSTM	Sum/Mean/ Max/LSTM	$h_v^{(l+1)} = \sigma\left(W^{(l)} \mathrm{CONCAT}\left(h_v^{(l)}, \mathrm{AGG}\left(\left\{h_u^{(l)}, u \in N(v)\right\}\right)\right)\right)$
GAT	Attention weighted	Sum	$h_v^{(l+1)} = \sigma\left(W^{(l)} \sum_{u \in N(v)} \alpha_{vu} h_u^{(l)}\right)$

Graph Neural Networks – Extension

- Different kinds of aggregation layers to capture expressive local information.
  - GCN / GAT / GraphSAGE / GIN
  - MixHop [1] / PNA [2] / IDGNN [3]
- Design skip-connections in the network to enable the deep networks.
  - JKNet[4] / DeepGCN[5] / IGNN[6]
- Applying GNNs on different tasks
  - Pooling operations in graph classification task: DGCNN [7] / SAGPool [8] / DiffPool [9]
  - Scoring functions in link-prediction task: SEAL[10]
  - …

Graph Neural Networks – Extension

Reference

[1] MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. ICML 2019

[2] Principal Neighbourhood Aggregation for Graph Nets. NeurIPS 2020

[3] Identity-aware Graph Neural Networks. AAAI 2021

[4] Representation Learning on Graphs with Jumping Knowledge Networks. ICML 2018

[5] DeepGCNs: Can GCNs Go as Deep as CNNs? ICCV 2019

[6] Implicit Graph Neural Networks. NeurIPS 2020

[7] An end-to-end deep learning architecture for graph classification. AAAI 2018

[8] Self-Attention Graph Pooling. ICML 2019

[9] Hierarchical graph representation learning with differentiable pooling. NeurIPS 2018

[10] Link Prediction Based on Graph Neural Networks. NeurIPS 2018

Graph Neural Networks – Comparison

- Strength (compared with graph embedding)
  - Larger receptive field (rely on model depth)
  - Take node features into consideration
  - Learnable parameters bring strong model capacity and better performance
- Weakness
  - Limited ability to capture long-range dependency
  - Rely on given structure, which may noise or inaccurate

Graph Neural Networks – Comparison

- GNN achieve higher performance compared with graph embedding methods

Graph embedding methods

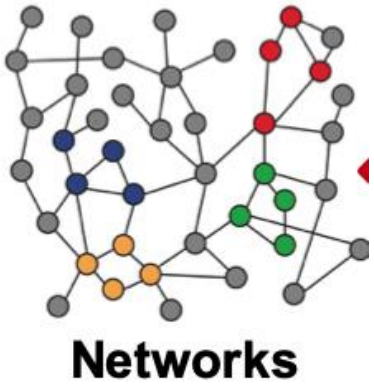
Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)
GCN (rand. splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7	58.4 ± 1.7

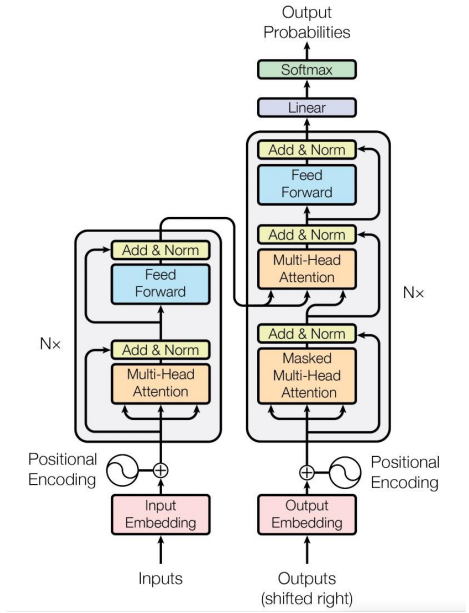
Outline

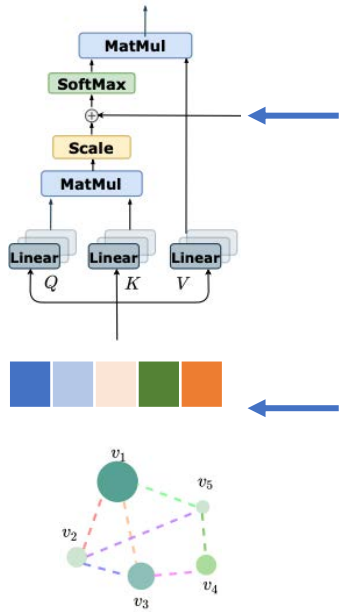
- Applications in Graphs
  - Graph-structured Data
  - Graph Learning Application
- Exemplar Learning Methods on Graphs
  - Overview on Graph Representation Learning
  - Graph Embedding
  - Graph Neural Network (GNN)
  - Graph Transformer
  - Versatile Graph Learning Methods
  - Summary and Discussion
- Theory Insights
  - Theory: expressivity
  - Theory: training
  - Theory: generalization

Transformer

- Transformer has achieved great success in many artificial intelligence fields due to its strong modeling ability
- How to extract the graph information with transformer?





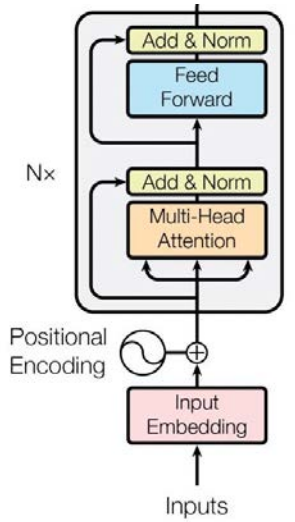


⭐⭐
Learning global attention according to graph structures.
- Local/global/relative

⭐⭐
- Select token for graph structured data
  - Node/edge/substructures/…
- Encoding the relations between token based on graph structures.
  - Local/global/relative

Transformer

- Transformer encoder is often used in NLP tasks
- Positional Encoding
  - Treat each word as one learnable token
  - Encoding the context information among the words
- Multi-head Attention
  - Updating representations with global attention
- Layer Normalization
  - Stable and fast training
- Feed-Forward Networks (FFN)
  - Feature transformation



$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

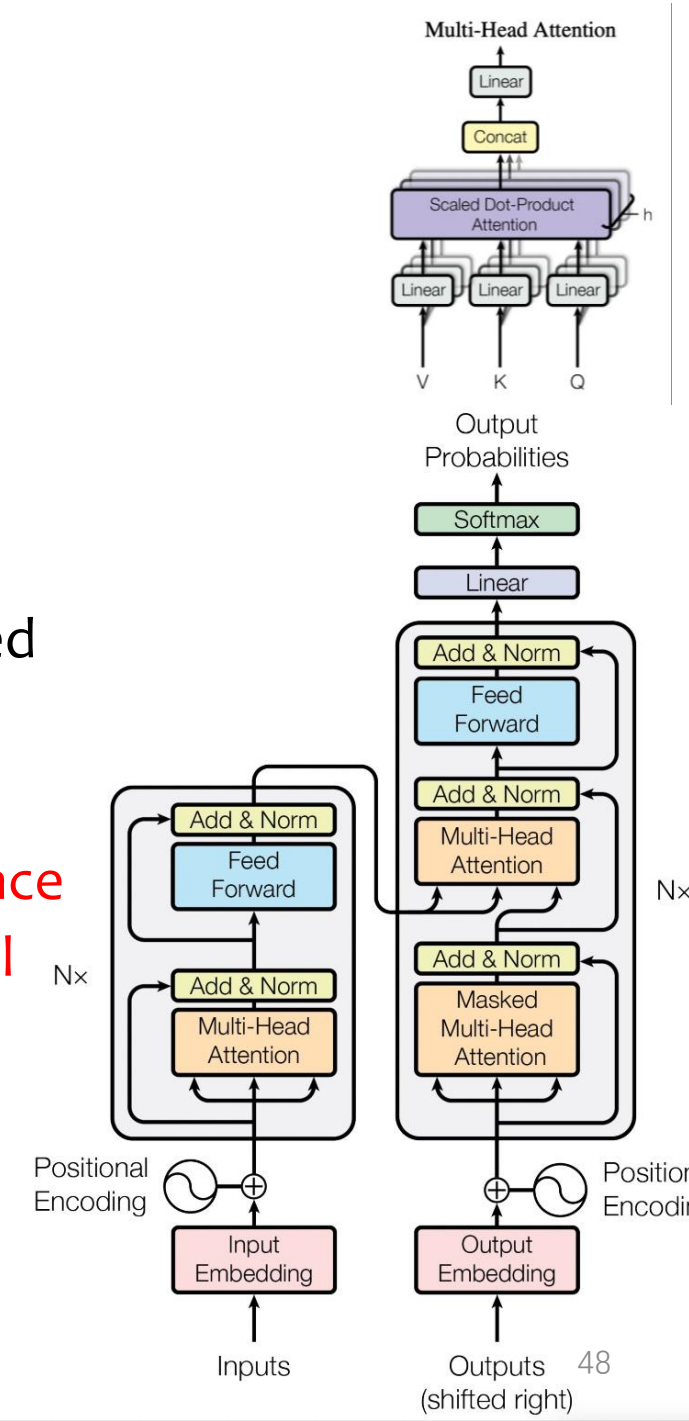
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$\text{LayerNorm}\big(x + \text{Sublayer}(x)\big)$$

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Transformer

- Key design in Transformer
  - Positional encoding: encoding the context information in the word sequence
  - Multi-head self-attention: learning token embedding based on the global attention on all token set
- Comparison with RNN
  - RNN: learning node representations following the sequence
  - Transformer: break the sequence and learning in a parallel manner



Attention Is All You Need, NIPS 2017

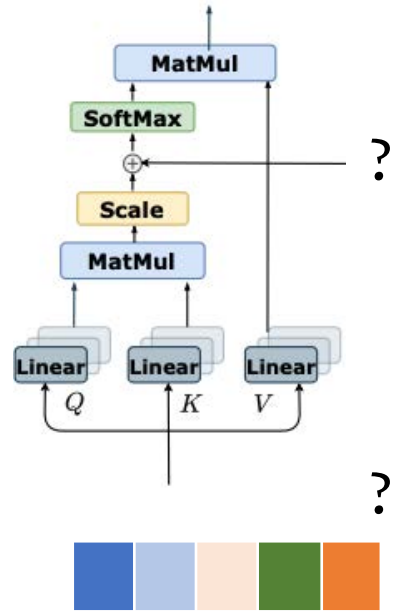
Graph Transformer

Key point to achieve graph + transformer

- Token: Nodes in general, edge/substructures are allowed
- Structure Encoder: encode the graph structures in transformer
  - what to encode: local / global / relative information in graph
  - where to use:
    - With node token: positional encoding
    - With attention map: structural encoding
- Transformer module:

$$Q = HW_Q, \quad K = HW_K, \quad V = HW_V,$$

$$A = \frac{QK^\top}{\sqrt{d_K}}, \quad \mathrm{Attn}\,(H) = \mathrm{softmax}\,(A)\,V,$$



Graph Transformer – Encoder

- Calculate token embedding
  - Using degree centrality to measure the node importance in local structure

$$h_i^{(0)} = x_i + z^-_{\deg^-(v_i)} + z^+_{\deg^+(v_i)},$$

- Attaching spatial encoding in the attention map
  - Measures the relative spatial relations in the graph with shortest path length
  - Preserve the edge features in the path

$$A_{ij} = \frac{(h_i W_Q)(h_j W_K)^T}{\sqrt{d}} + b_{\phi(v_i,v_j)} + c_{ij},$$

$$c_{ij} = \frac{1}{N}\sum_{n=1}^{N} x_{e_n}(w_n^E)^T,$$

Encoding type	Description	Examples
Local PE *node features*	Allow a node to know its position and role within a local cluster of nodes. *Within a **cluster**, the closer two nodes are to each other, the closer their local PE will be, such as the position of a word in a sentence (not in the text).*	• Sum each column of non-diagonal elements of the m-steps random walk matrix. • Distance between a node and the centroid of a cluster containing the node.
Global PE *node features*	Allow a node to know its global position within the graph. *Within a **graph**, the closer two nodes are, the closer their global PE will be, such as the position of a word in a text.*	• Eigenvectors of the Adjacency, Laplacian [15, 35] or distance matrices. • SignNet [38] (includes aspects of relative PE and local SE). • Distance from the graph's centroid. • Unique identifier for each connected component of the graph.
Relative PE *edge features*	Allow two nodes to understand their distances or directional relationships. *Edge embedding that is correlated to the distance given by any global or local PE, such as the distance between two words.*	• Pair-wise node distances [37, 3, 35, 62, 43] based on shortest-paths, heat kernels, random-walks, Green's function, graph geodesic, or any local/global PE. • Gradient of eigenvectors [3, 35] or any local/global PE. • PEG layer [56] (includes aspects of global PE). • Boolean indicating if two nodes are in the same cluster.
Local SE *node features*	Allow a node to understand what substructures it is a part of. *Given an SE of radius m, the more similar the m-hop subgraphs around two nodes are, the closer their local SE will be.*	• Degree of a node [62]. • Diagonal of the m-steps random-walk matrix [16]. • Time-derivative of the heat-kernel diagonal (gives the degree at $t = 0$). • Enumerate or count predefined structures such as triangles, rings, etc. [6, 67]. • Ricci curvature [53].
Global SE *graph features*	Provide the network with information about the global structure of the graph. *The more similar two graphs are, the closer their global SE will be.*	• Eigenvalues of the Adjacency or Laplacian matrices [35]. • Graph properties: diameter, girth, number of connected components, # of nodes, # of edges, nodes-to-edges ratio.
Relative SE *edge features*	Allow two nodes to understand how much their structures differ. *Edge embedding that is correlated to the difference between any local SE.*	• Pair-wise distance, encoding, or gradient of any local SE. • Boolean indicating if two nodes are in the same substructure [5] (similar to the gradient of sub-structure enumeration).

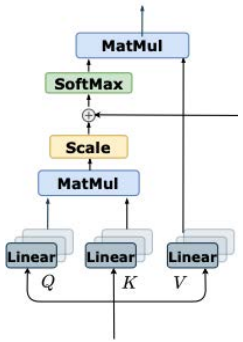
Graph Transformer – Overall Architecture

Architectures of graph transformer

- Stacking transformer modules.
- Construct tokens and calculate embeddings
- In each transformer module
  - Multi-head self-attention + structural encoder + Norm
  - FFN + Norm

$$\hat{A} = \frac{(H_l W^Q)(H_l W^K)^\top}{\sqrt{d_K}}, \quad A = \mathrm{softmax}(\hat{A}),$$
$$\hat{\mathrm{Attn}}(H_l) = A H_l W^V W^O = A H_l W^{VO},$$
$$\mathrm{Attn}(H) = \mathrm{LN}(AHW^{VO}))$$
$$H_{l+1} = \mathrm{LN}(ReLU(H' W^{F_1} + \mathbf{1} b^{F_1 T}) W^{F_2} + \mathbf{1} b^{F_2 T}).$$

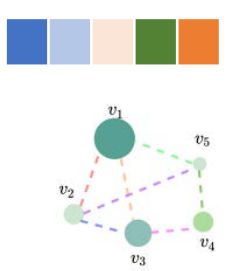
- The downstream task based on learned node representation



⭐⭐

Learning global attention according to graph structures.

- Local/global/relative



⭐⭐

- Select token for graph structured data
  - Node/edge/substructures/…
- Encoding the relations between token based on graph structures.
  - Local/global/relative

Graph Transformer – Performance

- Carefully designed encoder + transformers can perform well on the graph learning tasks

Table 2: Results on MolPCBA.

method	#param.	AP (%)
DeeperGCN-VN+FLAG [30]	5.6M	28.42±0.43
DGN [2]	6.7M	28.85±0.30
GINE-VN [5]	6.1M	29.17±0.15
PHC-GNN [29]	1.7M	29.47±0.26
GINE-APPNP [5]	6.1M	29.79±0.30
GIN-VN[54] (fine-tune)	3.4M	29.02±0.17
Graphormer-FLAG	119.5M	**31.39**±0.32

Table 3: Results on MolHIV.

method	#param.	AUC (%)
GCN-GraphNorm [5, 8]	526K	78.83±1.00
PNA [10]	326K	79.05±1.32
PHC-GNN [29]	111K	79.34±1.16
DeeperGCN-FLAG [30]	532K	79.42±1.20
DGN [2]	114K	79.70±0.97
GIN-VN[54] (fine-tune)	3.3M	77.80±1.82
Graphormer-FLAG	47.0M	**80.51**±0.53

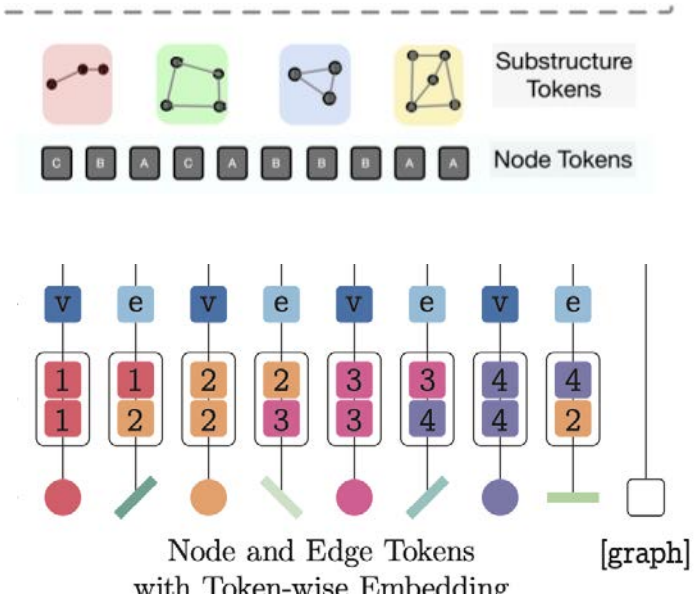
Table 4: Results on ZINC.

method	#param.	test MAE
GIN [54]	509,549	0.526±0.051
GraphSage [18]	505,341	0.398±0.002
GAT [50]	531,345	0.384±0.007
GCN [26]	505,079	0.367±0.011
GatedGCN-PE [4]	505,011	0.214±0.006
MPNN (sum) [15]	480,805	0.145±0.007
PNA [10]	387,155	0.142±0.010
GT [13]	588,929	0.226±0.014
SAN [28]	508, 577	0.139±0.006
$\text{Graphormer}_{\text{SLIM}}$	489,321	**0.122**±0.006

Graph transformer methods

Graph Transformer – Extension

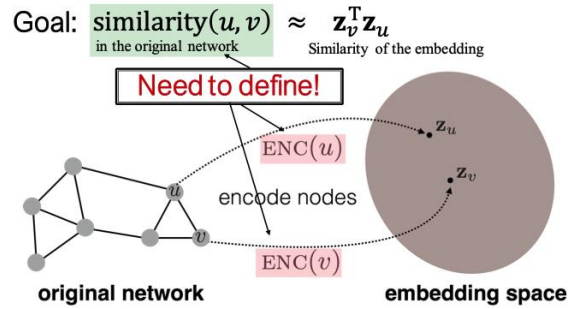
- New tokens to represent the graph
  - Use edge [1] / sub-structured [2] as graph token
- Better encoders to encode the rich information in graph
  - Simple structure in Image / Speech / text data
  - Rich information on graph structures [3]
- Combine GNN and graph transformers
  - Extract local information with GNN
  - Extract global information with transformer

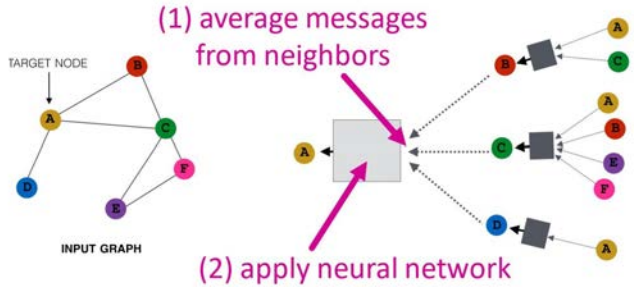


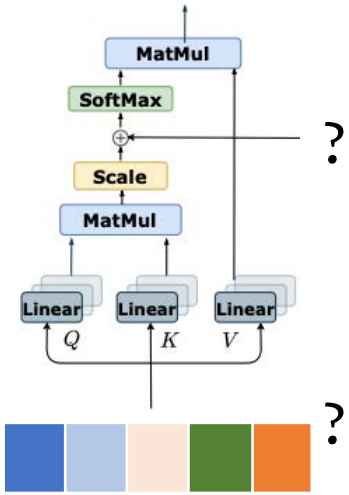
[1] Pure Transformers are Powerful Graph Learners. NeurIPS 2022
[2] ARE MORE LAYERS BENEFICIAL TO GRAPH TRANSFORMERS? ICLR 2023
[3] Recipe for a General, Powerful, Scalable Graph Transformer NeurIPS 2022

Summary

- Graph Embedding
  - Encode similar nodes in the graph with similar embeddings
  - Focus on local structures and cannot use the node features
- GNN
  - Learning node representations on local subgraphs
  - Largely rely on the graph structure and have limited ability in capturing the long-range dependency
- Graph Transformer
  - Encoding structure into representations and updating embeddings with global attention
  - The performance rely on the encoder which may have information loss
  - Hard to optimize

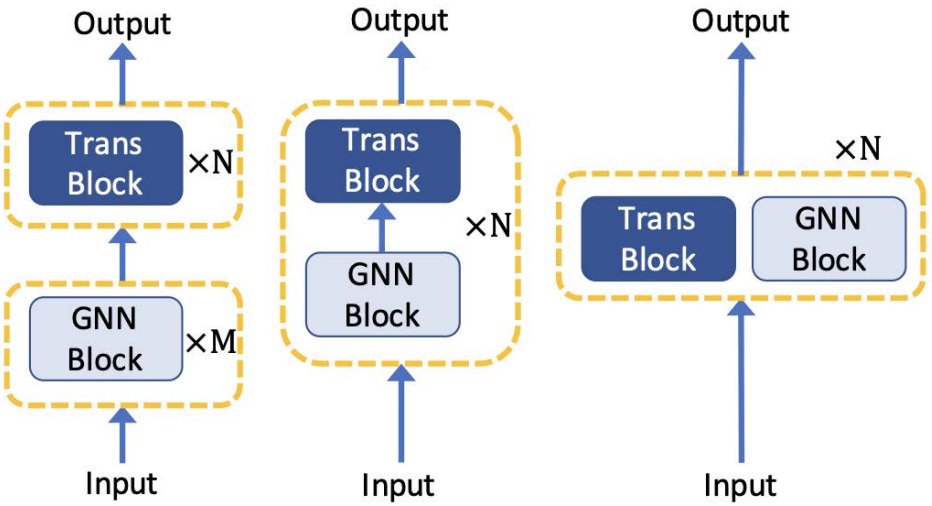






Combining GNNs and Transformers

- GNNs have advantages in local information extraction
- Transformers have strength in capturing the long-range dependency
- Combine GNNs and transformers to improve model performance



(a) Before Trans (b) Alternatively (c) Parallel

Three typical ways to combine GNNs and Transformer

Combining GNNs and Transformers

- Combining GNNs and Transformers needs careful design

Table 4: Performance comparison on graph-level tasks. For each column, the values with underline are the best results in each group. The values in bold are the best results across the groups.

		graph-level tasks									node-level tasks								
		ZINC(MAE↓)			molhiv(ROC-AUC↑)			molpcba(AP↑)			Flickr(Acc↑)			arixv(Acc↑)			product(Acc↑)		
		Small	Middle	Large	Small	Middle	Large	Small	Middle	Large	Small	Middle	Large	Small	Middle	Large	Small	Middle	Large
TF	vanilla	0.6689	0.6700	0.6699	0.7466	0.7230	0.7269	0.1624	0.1673	0.1676	0.5270	0.5279	0.5214	0.5539	0.5571	0.5598	0.7887	0.7887	0.7956
GA	before	0.4700	0.4809	0.5169	0.6758	0.7339	0.7182	0.2105	0.1989	0.2269	0.5352	0.5369	0.5272	0.5608	0.5590	0.5614	**0.7953**	0.7888	0.8012
	alter	0.3771	0.3412	0.2956	0.7200	0.7086	0.7433	0.2474	0.2417	0.2244	**0.5374**	0.5357	0.5162	0.5599	0.5555	0.5592	0.7905	0.7915	**0.8057**
	parallel	0.3803	**0.2749**	0.2984	0.7138	0.7750	0.7603	0.2345	0.2444	0.2205	0.5370	**0.5379**	0.5209	**0.5647**	0.5600	0.5529	0.7878	0.7896	0.7999

Outline

- Applications in Graphs
  - Graph-structured Data
  - Graph Learning Application
- Exemplar Learning Methods on Graphs
  - Overview on Graph Representation Learning
  - Graph Embedding
  - Graph Neural Network (GNN)
  - Graph Transformer
  - Versatile Graph Learning Methods
  - Summary and Discussion
- Theory Insights
  - Theory: expressivity
  - Theory: training
  - Theory: generalization

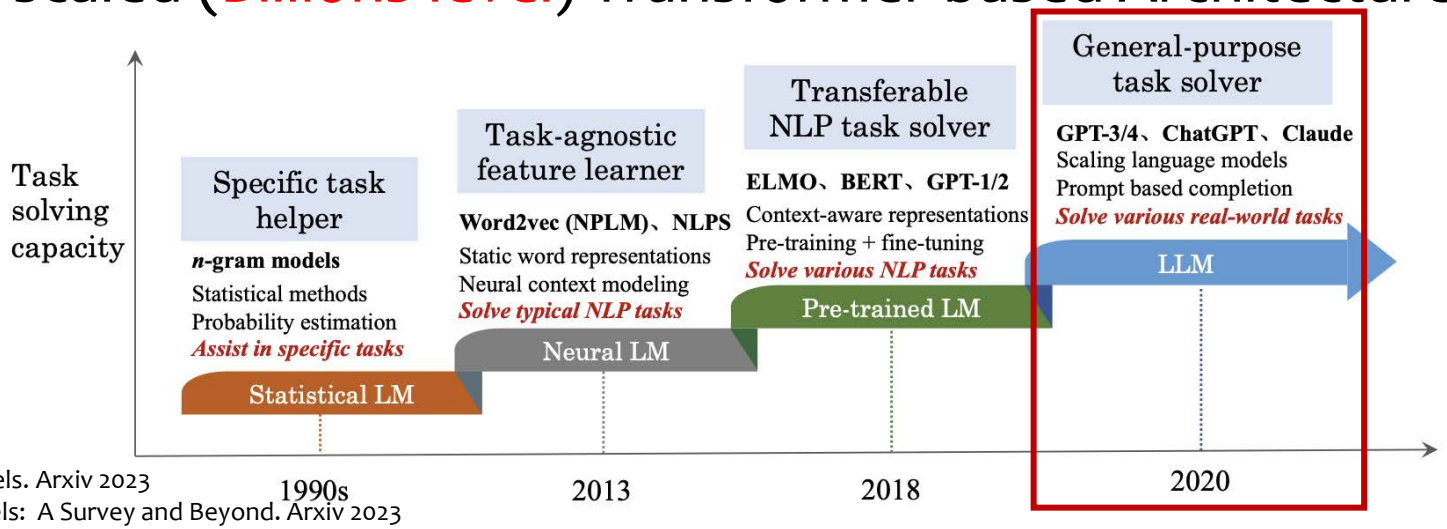
Versatile Graph Learning Methods + LLMs

When Large Language Models (LLMs) meet graph learning methods

- Towards graph foundation model
  - Adopting the pre-training and prompting framework on graphs
- LLMs as assistant in graph learning
  - LLMs as enhancer when facing the graphs
  - LLMs as predictor to generate the results
  - LLMs as alignment to learn graphs with GNNs/Trans

Preliminary: Large Language Model

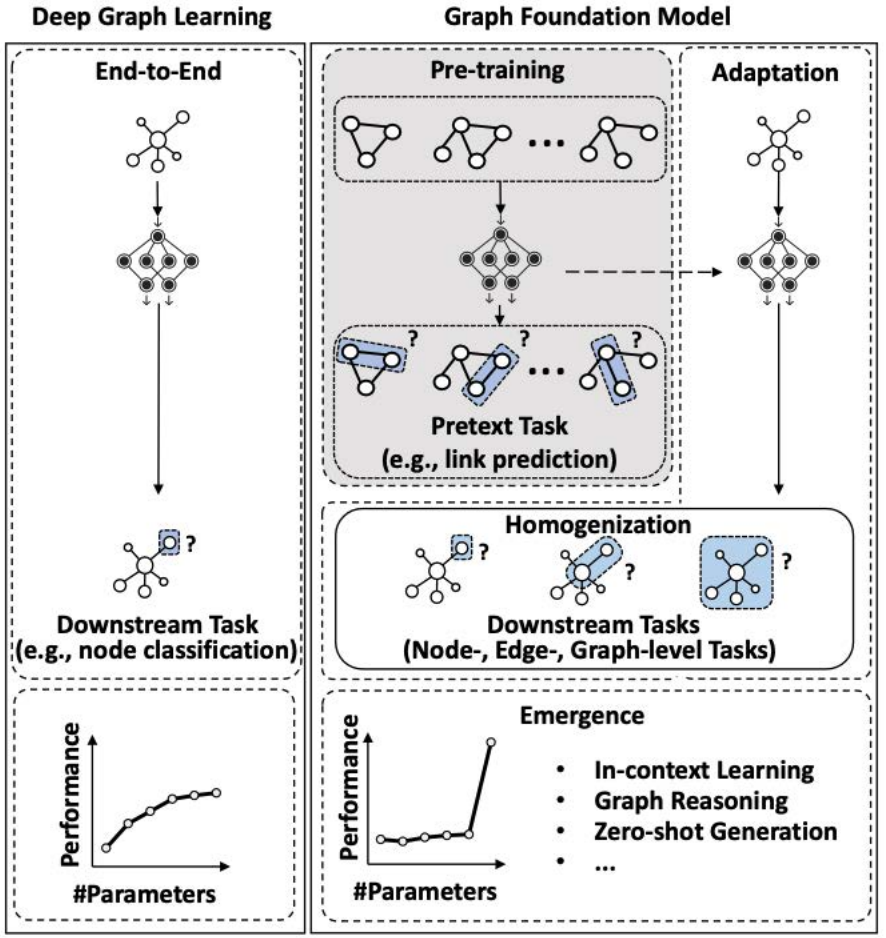
- Target: handling different downstream tasks with one method
- Training strategy:
    - Pre-training on diverse data to capture task-agnostic knowledge
    - Prompt tuning to adapt to downstream tasks
- Model: large-scaled (Billions-level) Transformer-based Architectures.



[1] A Survey of Large Language Models. Arxiv 2023
[2] Towards Graph Foundation Models: A Survey and Beyond. Arxiv 2023

Graph Foundation Model

- *Definition: A graph foundation model (GFM) is a model that is expected to benefit from the pre-training of broad graph data, and can be adapted to a wide range of downstream graph tasks.*
- Two key characteristics
  - Emergence: GFMs will exhibit some new abilities when having a large parameters or trained on more data.
  - Homogenization: applied to different formats of tasks in node/link/graph levels.



Towards Graph Foundation Models: A Survey and Beyond. Arxiv 2023

Graph prompt

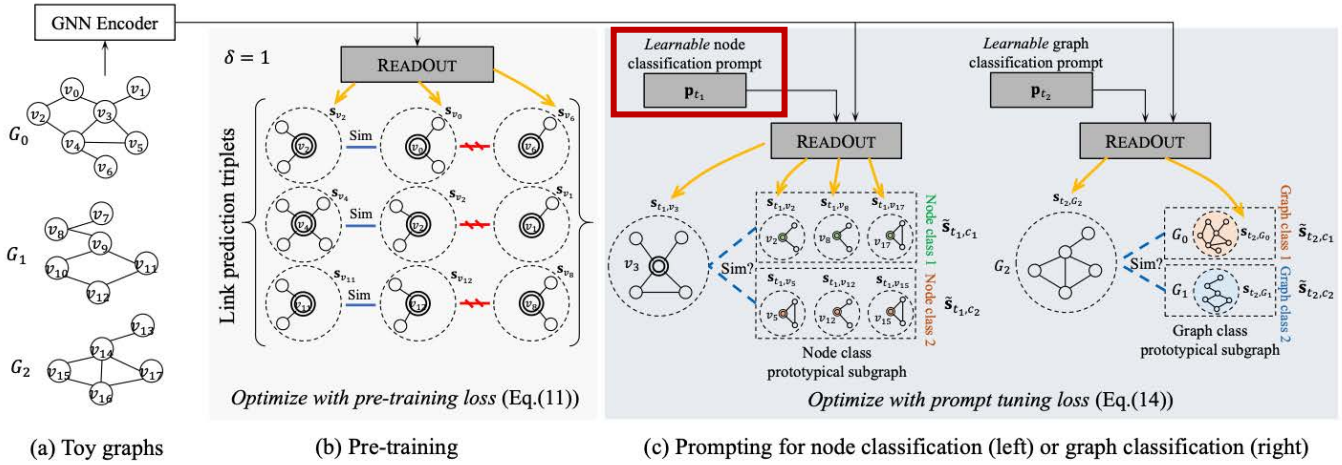


Figure 2: Overall framework of GRAPHPROMPT.

Preliminary: Prompt Tuning
- Prompt: Short text containing task-related and context info
- Target: Generating target-related response w/o tuning parameters.

- Examples:
  - text classification task: 'Very boring movie'→ LLMs →{ 'positive', 'negative'}
  - Prompted data: 'Very boring movie. It was MASK' → LLMs → {'terrible', 'bad', 'awesome', 'good', etc}

Pre-training strategy

- Data: Formatting node/graph-level tasks as unified link-level tasks.
- Model: Using GNNs as backbone
- Training: Contrastive loss:

$$\mathcal{L}_{\text{pre}}(\Theta) = -\sum_{(v,a,b)\in\mathcal{T}_{\text{pre}}} \ln \frac{\exp(\text{sim}(\mathbf{s}_v, \mathbf{s}_a)/\tau)}{\sum_{u\in\{a,b\}} \exp(\text{sim}(\mathbf{s}_v, \mathbf{s}_u)/\tau)},$$

Prompt Tuning for downstream task

- Design prompted graph: graph embedding ⊙ task vector
- Tuning with contrastive loss

GraphPrompt: Unifying Pre-Training and Downstream Tasks for Graph Neural Networks. WWW 2023

Towards Graph Foundation Models

Three key differences existed when designing GFMs, i.e., architecture, pre-training strategy, and adaption strategy [1].

Model	Backbone Architecture	Pretraining	Adaptation
All In One [61]	GCN, GAT, Graph Transformer	Same-Scale CL	Prompt Tuning
PRODIGY []	GCN, GAT	Graph Reconstruction, Supervised	Prompt Tuning
DGI [88]	GCN	Cross-Scale CL	Parameter-Efficient FT
GRACE [68]	GCN	Same-Scale CL	Vanilla FT
VGAE [89]	GCN	Graph Reconstruction, Property Prediction	Vanilla FT
MA-GCL [90]	GCN	Same-Scale CL	Vanilla FT
GraphMAE [70]	GAT	Graph Reconstruction	Parameter-Efficient FT
GraphMAE2 [71]	GAT	Graph Reconstruction	Parameter-Efficient FT
GPPT [74]	GraphSAGE	Graph Reconstruction, Cross-Scale CL	Prompt Tuning
GPT-GNN [91]	HGT	Graph Reconstruction	Vanilla FT
GraphPrompt [60]	GIN	Graph Reconstruction	Prompt Tuning
GCC [92]	GIN	Same-Scale CL	Vanilla FT
GraphCL [93]	GIN	Same-Scale CL	Parameter-Efficient FT
AdapterGNN [72]	GIN	Cross-Scale CL, Graph Reconstruction, Same-Scale CL	Parameter-Efficient FT
AAGOD [94]	GIN	Same-Scale CL, Supervised	Prompt Tuning
GPF [95]	GIN	Cross-Scale CL, Graph Reconstruction	Prompt Tuning
SGL-PT [96]	GIN	Same-Scale CL, Graph Reconstruction	Prompt Tuning
Graph-BERT [97]	Graph Transformer	Graph Reconstruction, Supervised	Vanilla FT
GROVER [98]	Graph Transformer	Property Prediction	Vanilla FT
G-Adapter [73]	Graph Transformer	Supervised, Graph Reconstruction, Property Prediction	Parameter-Efficient FT

Towards Graph Foundation Models: ~~A Survey and Beyond~~. Arxiv 2023

Challenges of GFMs

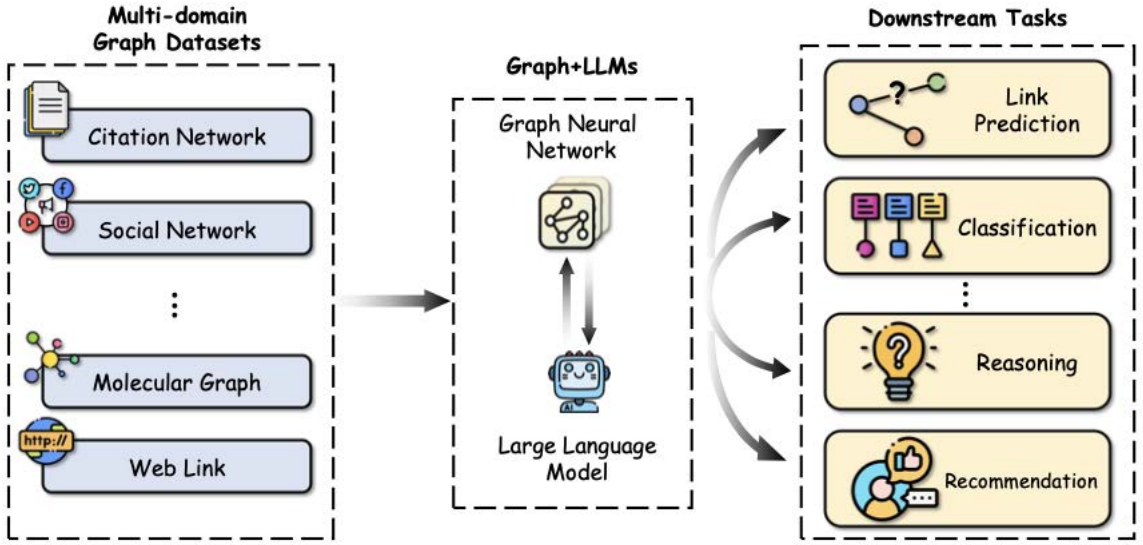
- In obtaining the GFM
  - Data: Graph data lack common foundations compared with natural language [1].
  - Pre-training: The diversity between domains [2]
  - Model: The constructions of billons-level backbone model for emergence ability.
- In using the GFM
  - The potential applications of GFMs

The exploration on GFMs is unknown and still ongoing

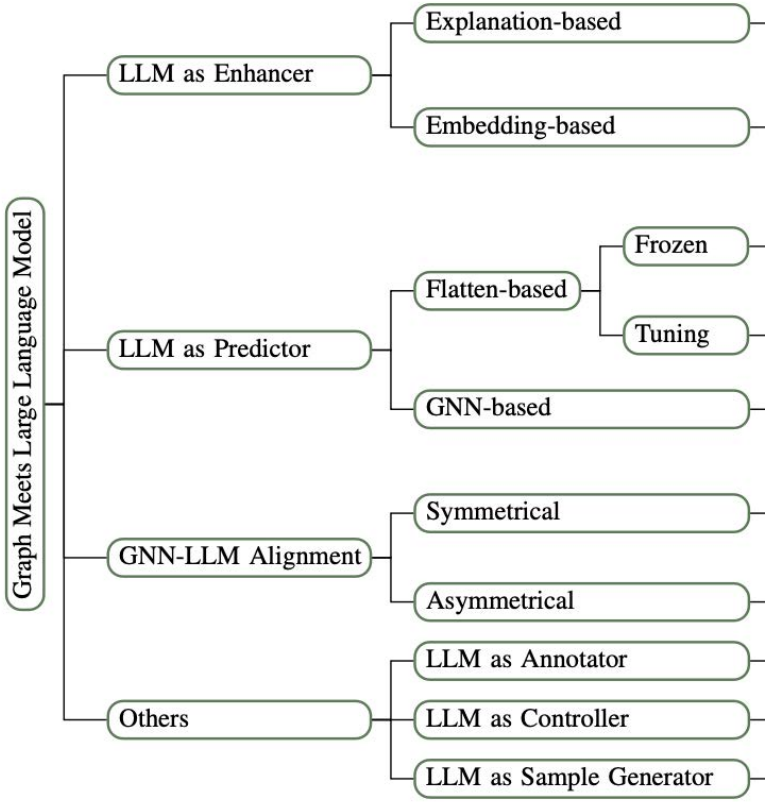
[1] Towards Graph Foundation Models: A Survey and Beyond. Arxiv 2023
[2] Graph Meets LLMs: Towards Large Graph Models. Arxiv 2023

Applying LLMs in graph learning

Applying LLMs in graph learning



Three categories based on the



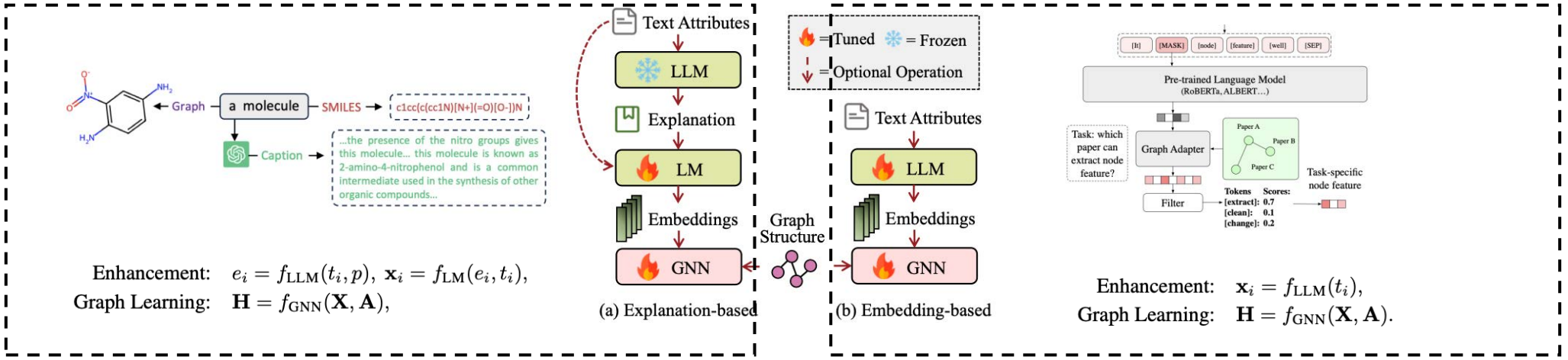
A Survey of Graph Meets Large Language Model: Progress and Future Directions. arxiv 2023

Applying LLMs in graph learning

LLM as enhancer: enhancing node embedding with LLMs.

- Explanation-based: capture high-level explanations with LLMs.
- Embedding-based: utilize LLMs to generate embeddings.

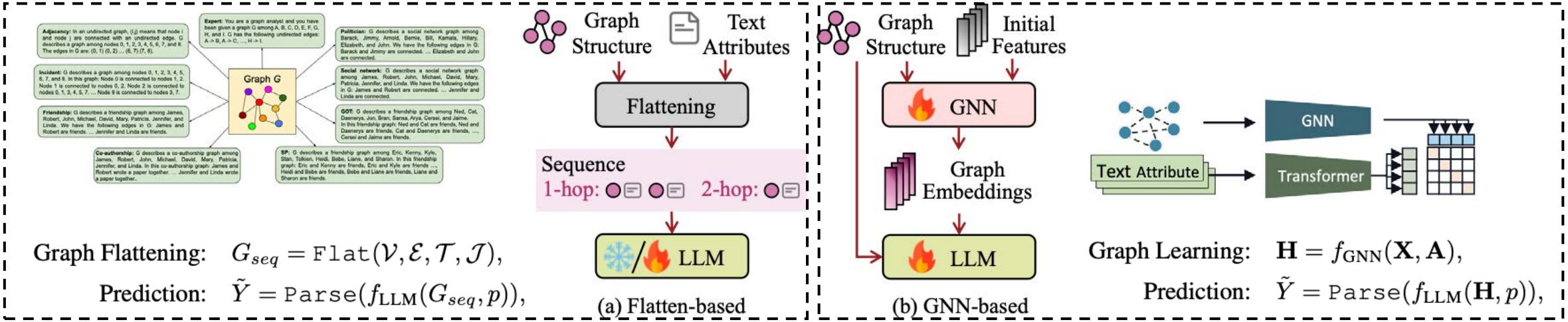
- Well performance on TAG & flexibility / Scalability



[1] Can Large Language Models Empower Molecular Property Prediction?
[2] Prompt-based Node Feature Extractor for Few-shot Learning on Text-Attributed Graphs

Applying LLMs in graph learning

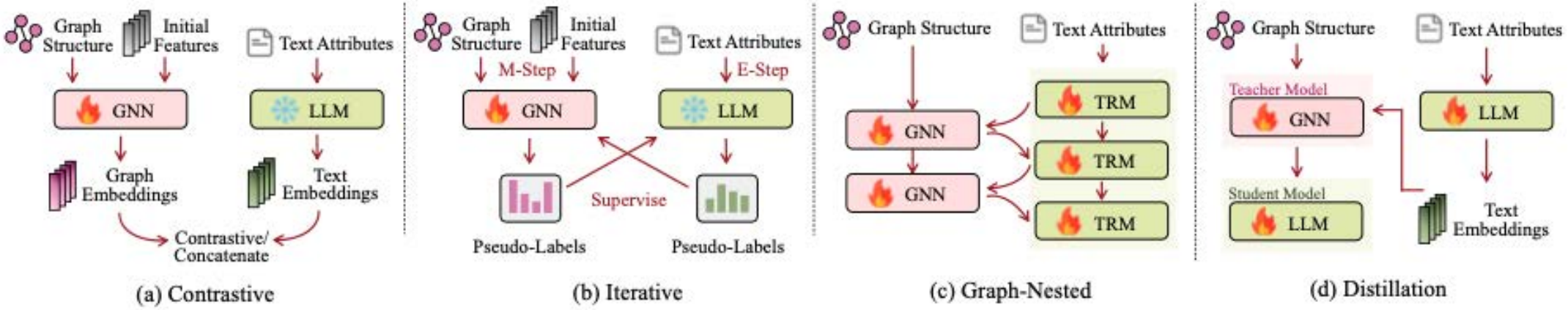
- LLM as predictor: utilize LLMs to make predictions like classification and reasoning.
  - Flatten-based: flatten graphs into sequence of nodes and then encoded by LLMs
  - GNN-based: Predict based on GNN embeddings and LLMs.
- Effectiveness / Efficiency



[1]TALK LIKE A GRAPH: ENCODING GRAPHS FOR LARGELANGUAGE MODELS
[2] GraphGPT: Graph Instruction Tuning for Large Language Models

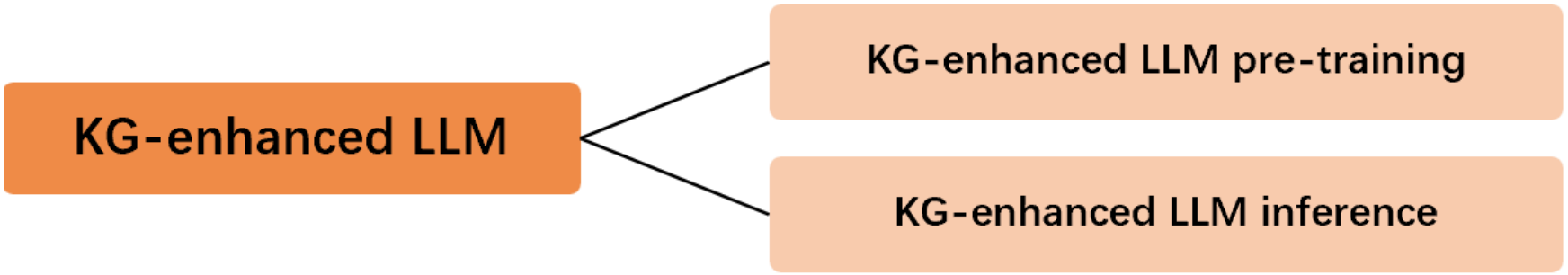
Applying LLMs in graph learning

- GNN-LLM Alignment: aligning the embedding spaces of GNNs and LLMs.
  - Symmetric: two encoders achieve comparable performance
  - Asymmetric: allowing one encoder to assist or enhance another
- Effective learning on both modalities / Data scarcity



Graph-enhanced LLM

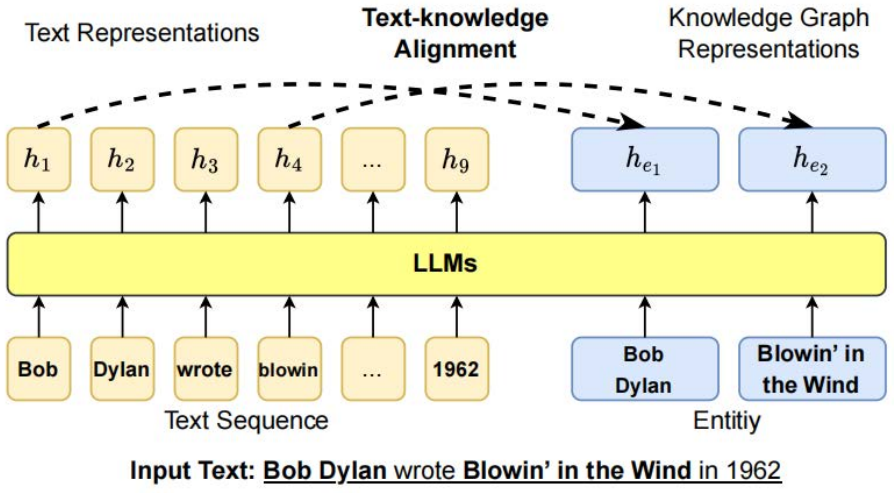
- LLMs have been criticized for their lack of practical knowledge and tendency to generate factual errors during inference (hallucinations).
- To alleviate these issues, one potential approach is to take advantage of the Knowledge Graphs (KGs), which store high-quality factual knowledge in a structured format.

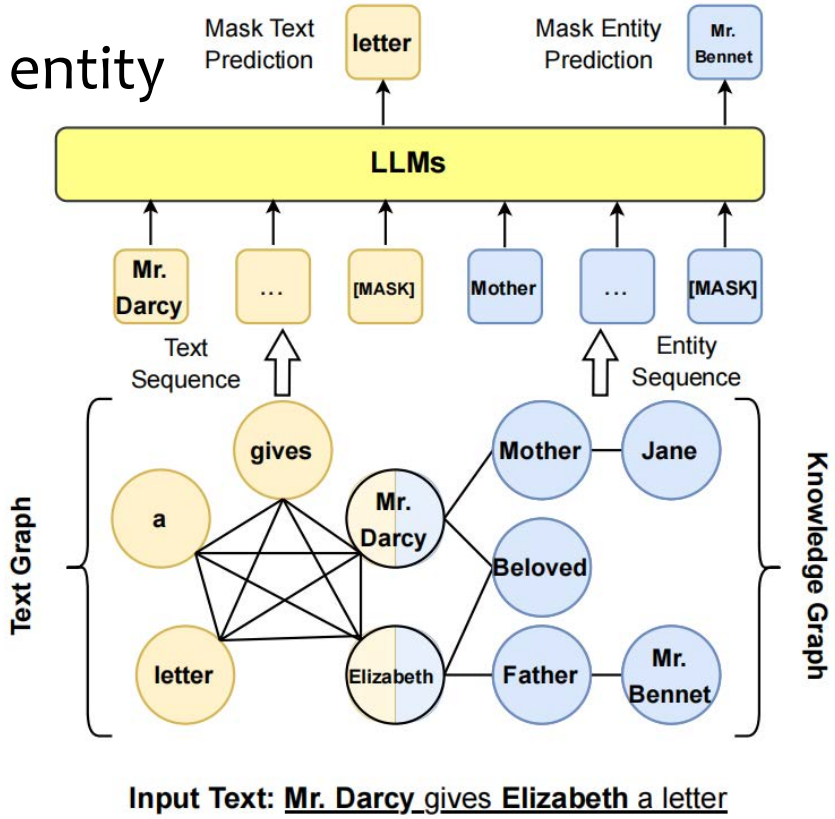


[1] Graph Machine Learning in the Era of Large Language Models (LLMs)
[2] Unifying Large Language Models and Knowledge Graphs: A Roadmap

Graph-enhanced LLM

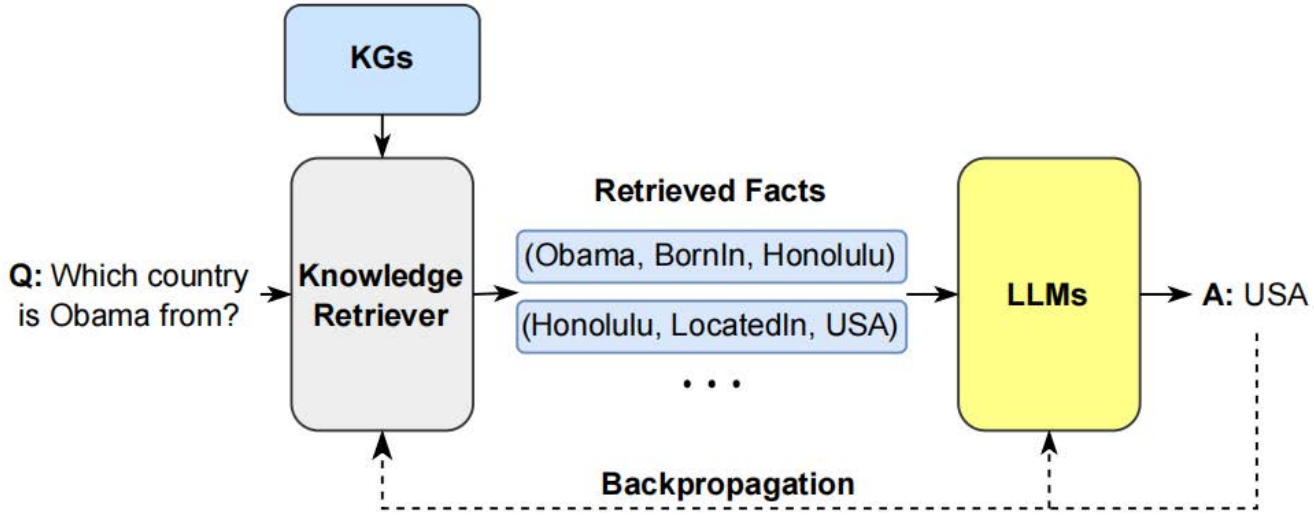
- Graph-enhanced LLM pre-training:
  - Integrating KGs into training objective: alignment links between textual tokens and entity
  - Integrating KGs into LLM Inputs: introducing subgraph into the inputs





Graph-enhanced LLM

- Graph-enhanced LLM inference:
  - Graph Retrieval-Augmented Generation (GraphRAG): retrieve relevant information from graph data to augment the generation process of large language models.



[1] Graph Retrieval-Augmented Generation: A Survey

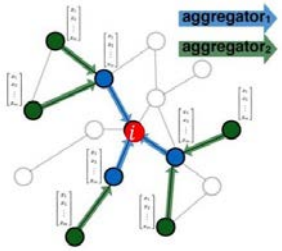
Summary and future directions

- Summary
  - GFMs pre-train and adapt to diverse graph-structured data.
  - LLMs can be used to assist the graph learning procedures.
  - Graphs can be used to enhance LLMs.
  - Generative(GFMs) is not all we need.
- Future directions
  - Backbone architecture constructions and training.
  - Combination strategy of textual and structure modalities.

Outline

- Applications in Graphs
  - Graph-structured Data
  - Graph Learning Application
- Exemplar Learning Methods on Graphs
  - Overview on Graph Representation Learning
  - Graph Embedding
  - Graph Neural Network (GNN)
  - Graph Transformer
  - Versatile Graph Learning Methods
  - Summary and Discussion
- Theory Insights
  - Theory: expressivity
  - Theory: training
  - Theory: generalization

Theory of GNN

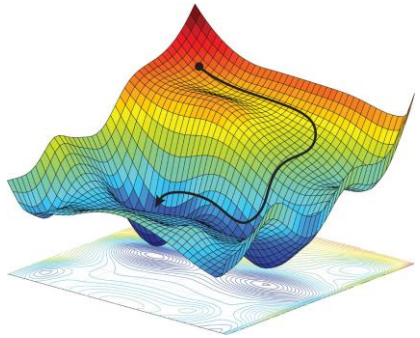


Model Design： Expressivity

- What kind of graph can be distinguished?
- What kind of graph structure can be learned?

Model Training: hard to train

- Oversmoothing
- Oversquahing



Upper bound of performance

Actual achievable performance

Model Performance: Generalization

Outline

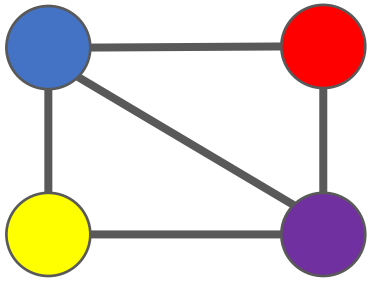
- Applications in Graphs
  - Graph-structured Data
  - Graph Learning Application
- Exemplar Learning Methods on Graphs
  - Overview on Graph Representation Learning
  - Graph Embedding
  - Graph Neural Network (GNN)
  - Graph Transformer
  - Versatile Graph Learning Methods
  - Summary and Discussion
- Theory Insights
  - Theory: expressivity
  - Theory: training
  - Theory: generalization

Theory: expressivity

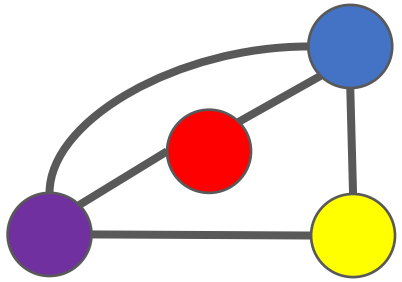
- What kind of graph can be distinguished?
  - Graph Isomorphism
  - WL-test
- What kind of graph structure can be learned?
  - Expressivity of GNN in rule learning

Graph Isomorphism

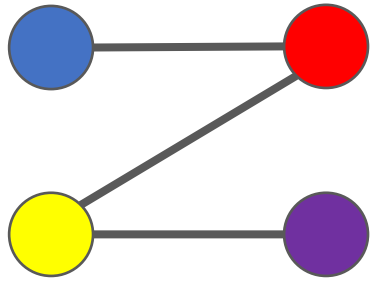
- The graph isomorphism problem asks whether two graphs are topologically identical
- If an isomorphism exists between two graphs G and H,
  - They have same number of node set and same edge connection
  - Graph isomorphism is a bijection f between nodes in G and H, two nodes u and v are adjacent in G, iff $f(u)$ and $f(v)$ are adjacent in H
- Take an example on $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$
  - $\mathcal{G}_1, \mathcal{G}_2$ are isomorphic, but $\mathcal{G}_1, \mathcal{G}_3$ are non-isomorphic



$\mathcal{G}_1$



$\mathcal{G}_2$

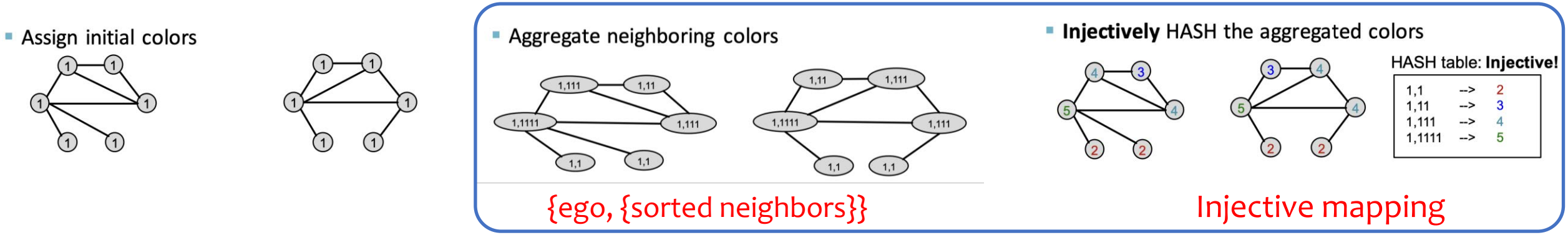


$\mathcal{G}_3$

WL Test

- Graph isomorphism: check whether two graphs have the same structure. NP hard problem
- Weisfeiler-Lehman test (WL-test): a color refinement strategy to check graph isomorphism (necessary but not sufficient)
  - Different results on two graphs → Two different graphs

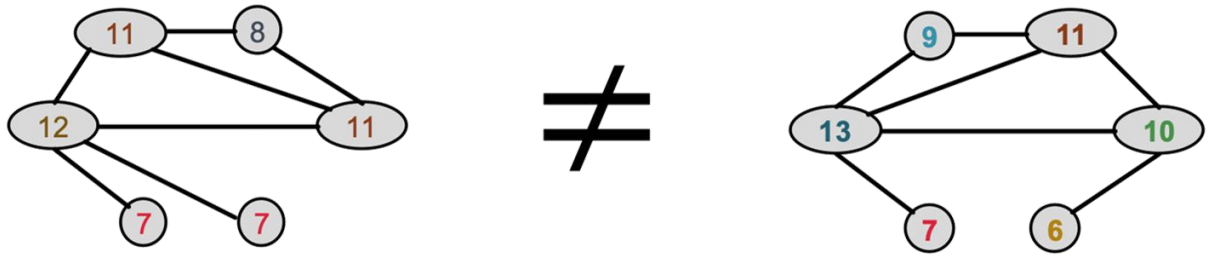
Iterate until convergence



WL Test

Example of color refinement given two graphs

- Two graphs are considered isomorphic if they have the same set of colors



{7,7,8,11,11,12}

{6,7,9,10,11,13}

Algorithm *1-WL (color refinement)*

Input: $G = (V, E, X_V)$

1. $c_v^0 \leftarrow \text{hash}(X_v)$ for all $v \in V$
2. **repeat**
3. $\quad c_v^\ell \leftarrow \text{hash}(c_v^{\ell-1}, \{\!\{c_w^{\ell-1} : w \in \mathcal{N}_G(v)\}\!\})\ \forall v \in V$
4. **until** $(c_v^\ell)_{v \in V} = (c_v^{\ell-1})_{v \in V}$
5. **return** $\{\!\{c_v^\ell : v \in V\}\!\}$

Message-passing framework

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

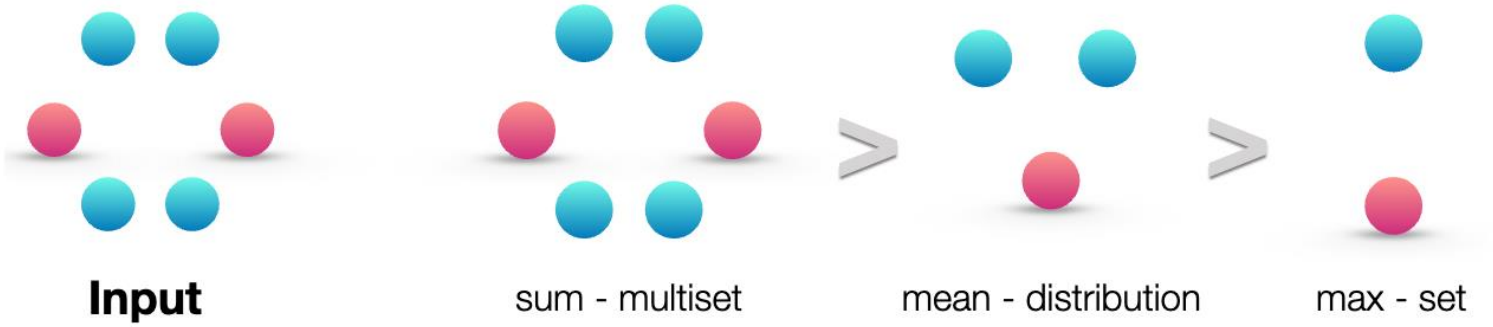
$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

Message-passing GNN is at most as expressive as 1-WL test

How Powerful are Graph Neural Networks? ICLR 2019

Expressivity of different GNN architectures

- Motivated by WL-test, the expressivity of GNN can be analyzed by their ability of distinguishing graphs
- We can analyze the expressivity of GNN with different aggregation functions



How Powerful are Graph Neural Networks? ICLR 2019

Graph Isomorphism Network (GIN)

- Designing the Graph neural networks as powerful as WL-Test
- Key point: hash injective mapping based on node and neighbors

$$c^{(k+1)}(v) = \text{HASH}\left(c^{(k)}(v), \{c^{(k)}(u)\}_{u \in N(v)}\right)$$

$$\text{MLP}_\Phi\left((1+\epsilon)\cdot \text{MLP}_f(c^{(k)}(v)) + \sum_{u \in N(v)} \text{MLP}_f(c^{(k)}(u))\right)$$

- Specifically, we will model the injective function over the tuple:

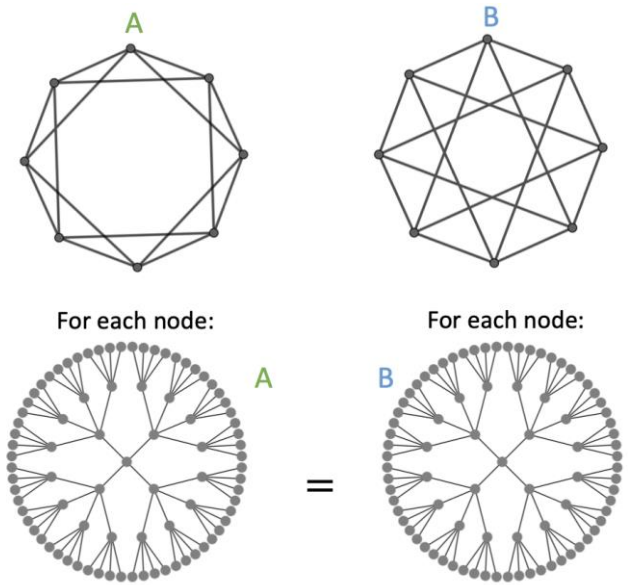
$$\left(c^{(k)}(v), \{c^{(k)}(u)\}_{u \in N(v)}\right)$$

Root node features

Neighboring node colors

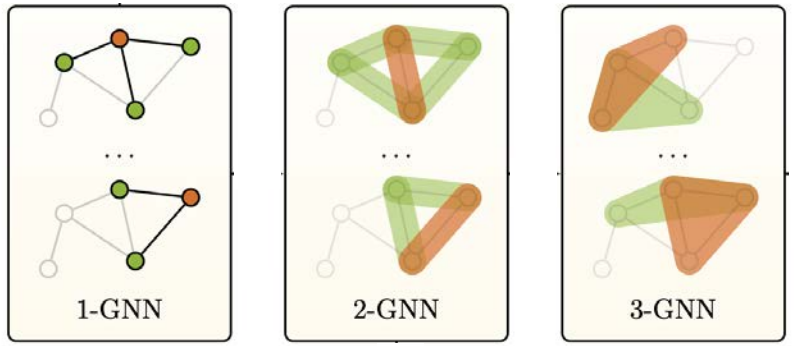
From 1-WL to k-WL

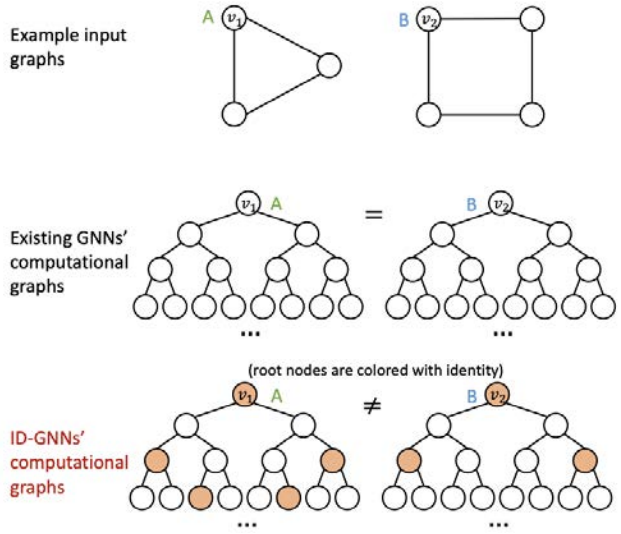
Two different graphs (G1,G2) + 1-WL test→ Same results in 1-WL test



Design GNNs beyond 1-WL test

- Design GNNs with higher-order WL-test [1].
- Identity-aware nodes [2]

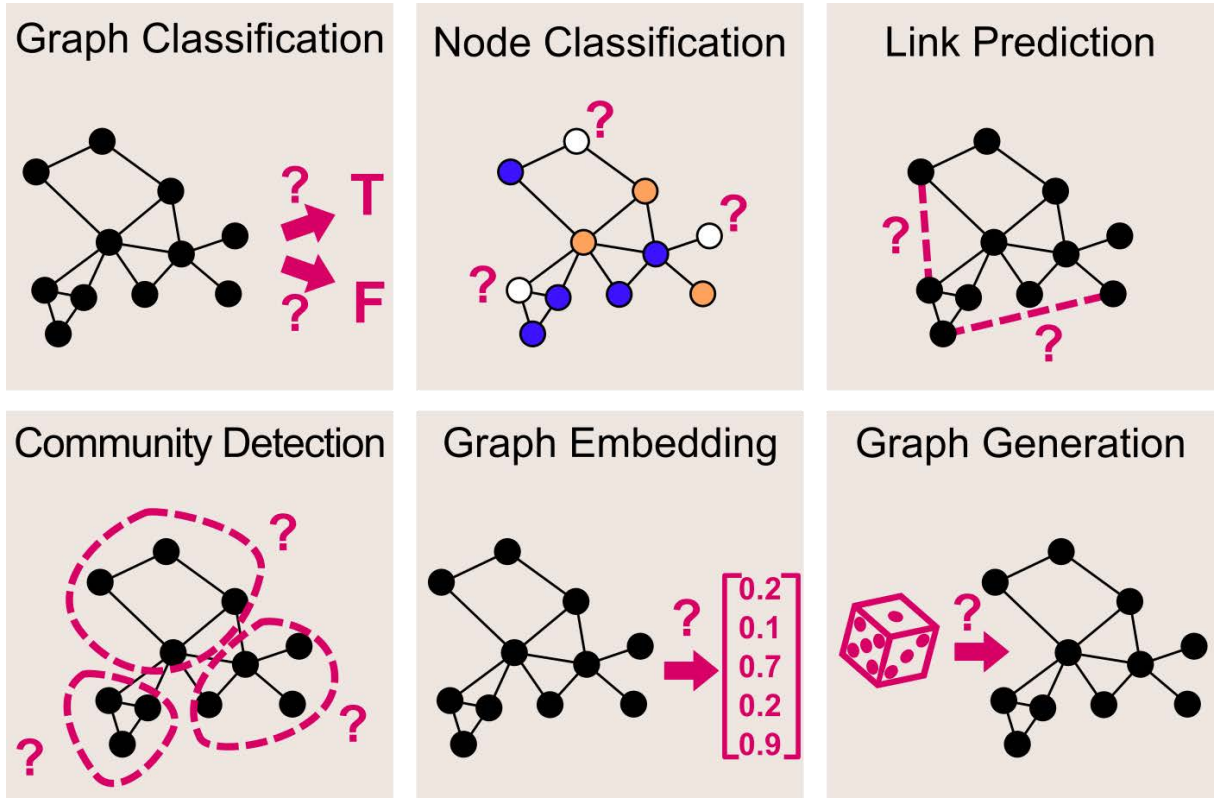




[1] Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks. AAAI 2019
[2] Identity-aware Graph Neural Networks. AAAI 2021

WL Hierarchy: standardized comparison

Isomorphism: core structural difference between graphs



Quantified expressivity

- k-WL strictly distinguishes more graphs than (k-1)-WL (k > 1)
- We have a Hierarchy:

  1-WL = 2-WL < 3-WL < ...

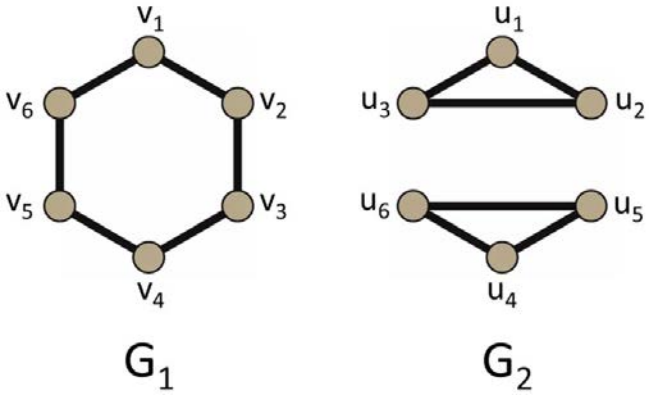
- This Hierarchy can be used to describe expressivity of GNN models
  - E.g. MPNN, GCN = 1-WL
  - k-GNN = k-WL (high cost)
  - Most latest GNN models > 2-WL

WL Hierarchy: examples

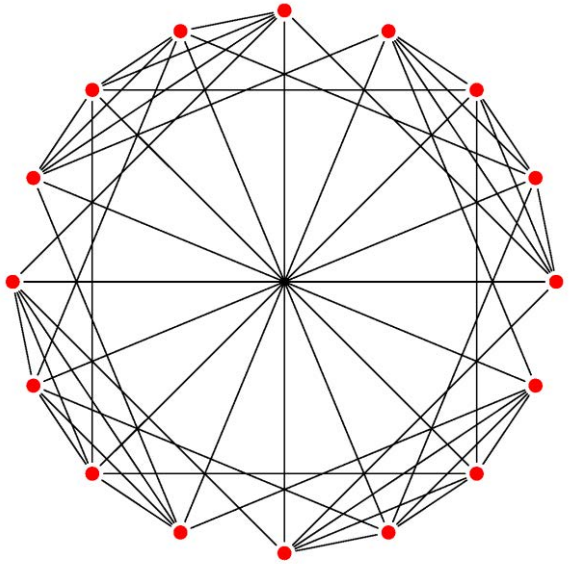
1-WL=2-WL

3-WL

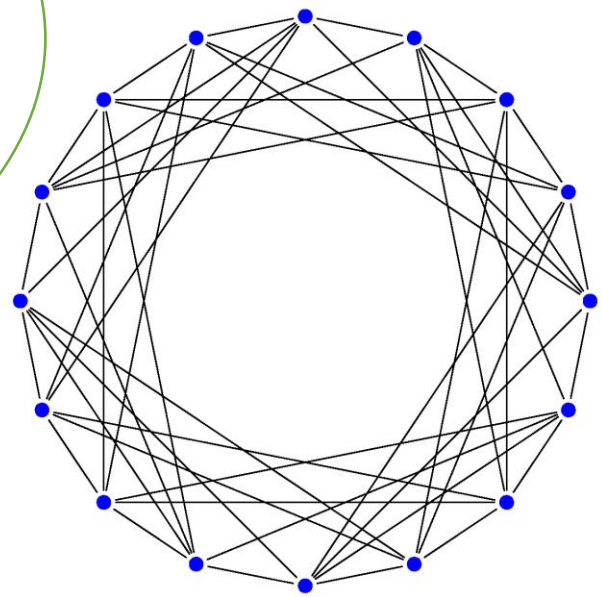
...



Regular graphs with same size and degree are not in 1-WL



(a) 4 × 4 Rook Graph.

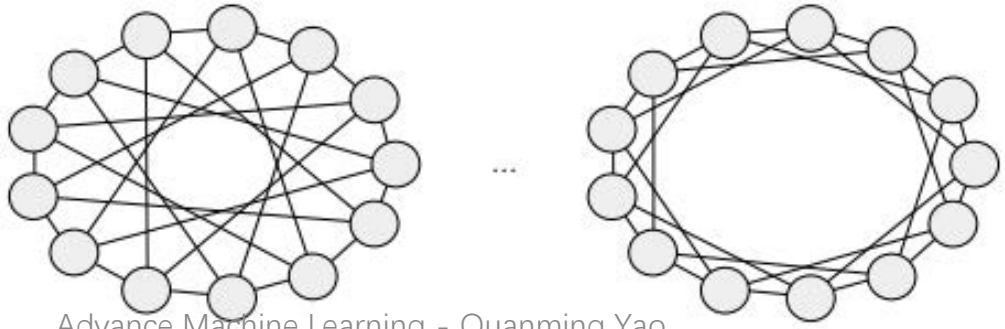


(b) Shrikhande Graph.

WL Hierarchy: drawbacks

- Granularity: most latest GNN models are located in (2-WL, 4-WL)
  - Not directly comparable within WL Hierarchy
- Hard to evaluate Graph Transformers
  - Expressivity of GTs are usually illustrated with examples
  - We can prove expressivity $\not<$k-WL, but not $\geq$ k-WL
- Practical applications
  - Hard to define "enough expressivity" for practical tasks
  - WL-based synthetic tests are usually toy experiments on regular graphs

Example: Circle Skip Link (CSL) graphs



Theory: expressivity

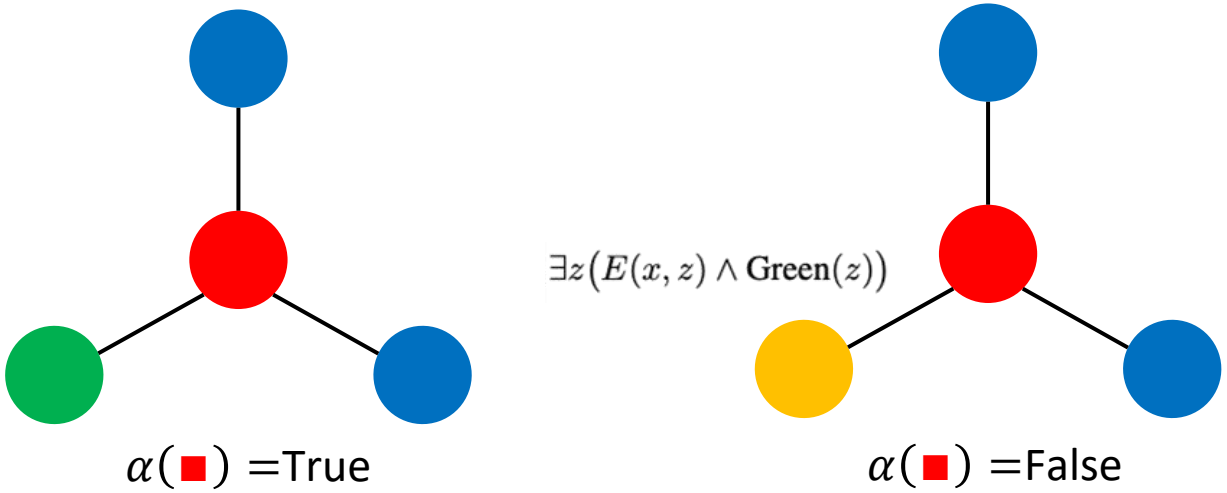
- What kind of graph can be distinguished?
  - Graph Isomorphism
  - WL-test
- What kind of graph structure can be learned?
  - Expressivity of GNN in rule learning

Logic on graph

- Logic can describe structures in graph
- Logical classifiers

edges nodes

$$\alpha(x) := \mathbf{Red}(x) \wedge \exists y\big(E(x,y) \wedge \mathbf{Blue}(y)\big) \wedge \exists z\big(E(x,z) \wedge \mathbf{Green}(z)\big).$$

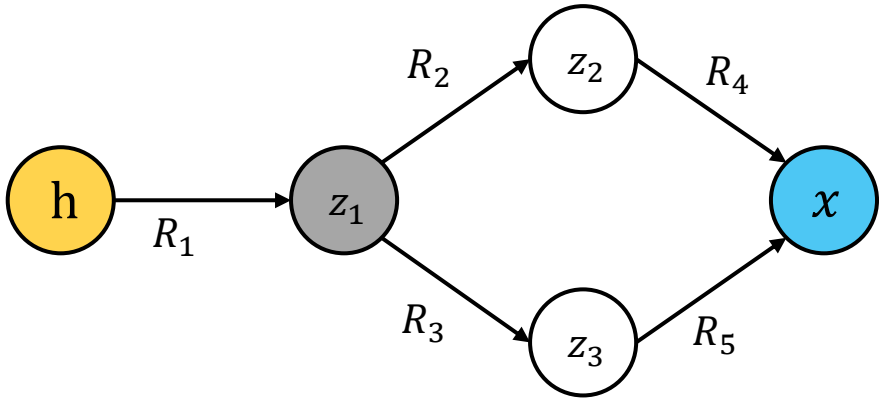


$\alpha(\blacksquare)$ =True $\alpha(\blacksquare)$ =False

Logic describes node classifier

If the logical formula is true,
relation U exists between h and x

$$U(\mathsf{h},x) := \exists^1 z_1, R_1(\mathsf{h},z_1) \wedge \big(\exists z_2, z_3, R_2(z_1,z_2) \wedge R_4(z_2,x) \wedge R_3(z_1,z_3) \wedge R_5(z_3,x)\big)$$



Logic describes rule structure

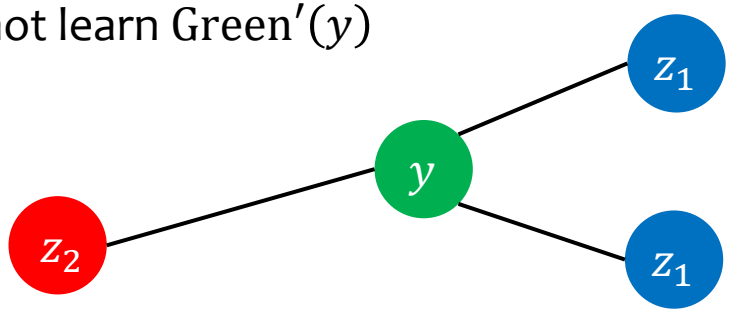
Expressivity of GNN

- GNN

$$\boldsymbol{x}_v^{(i)} = \mathrm{COM}^{(i)}\left(\boldsymbol{x}_v^{(i-1)}, \mathrm{AGG}^{(i)}\left(\{\!\{\boldsymbol{x}_u^{(i-1)} \mid u \in \mathcal{N}_G(v)\}\!\}\right)\right).$$

Theorem A logical classifier is captured by GNN if and only if it can be expressed in graded modal logic.

GNN cannot learn $\mathrm{Green}'(y)$



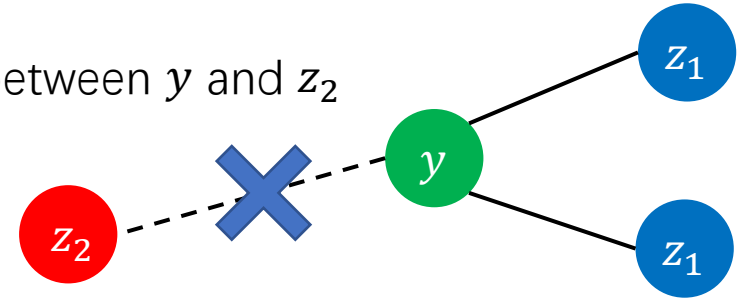
$$\mathrm{Green}(y) = \exists^{\geq 2} z_1, R(y, z_1) \wedge \mathrm{Blue}(z_1) \wedge \exists z_2, \mathrm{R}(\mathrm{y}, \mathrm{z}_2) \wedge \mathrm{Red}(z_2)$$

- GNN + layer-wise readout

$$\boldsymbol{x}_v^{(i)} = \mathrm{COM}^{(i)}\left(\boldsymbol{x}_v^{(i-1)}, \mathrm{AGG}^{(i)}\left(\{\!\{\boldsymbol{x}_u^{(i-1)} \mid u \in \mathcal{N}_G(v)\}\!\}\right), \mathrm{READ}^{(i)}\left(\{\!\{\boldsymbol{x}_u^{(i-1)} \mid u \in G\}\!\}\right)\right).$$

Theorem GNN with layer-wise readout can capture FOC2 classifier

No edge between y and z_2



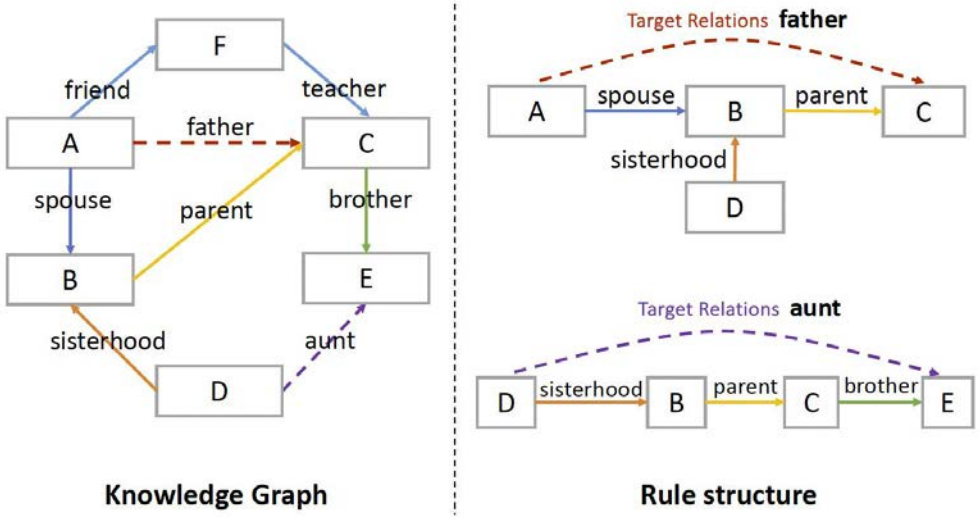
GNN cannot learn $\mathrm{Green}'(y)$, but GNN + layer-wise readout can.

$$\mathrm{Green}'(y) = \exists^{\geq 2} z_1, R(y, z_1) \wedge \mathrm{Blue}(z_1) \wedge \exists z_2\ \mathrm{Red}(z_2)$$

The Logical Expressiveness of Graph Neural Networks, ICLR 2020

Expressivity of GNN in rule learning

- What kind of rule structures can GNN learn?



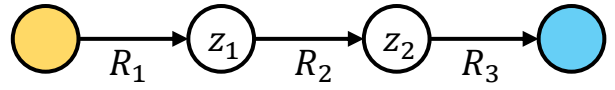
Knowledge Graph | Rule structure

Rule structures is important deducing new facts in graph

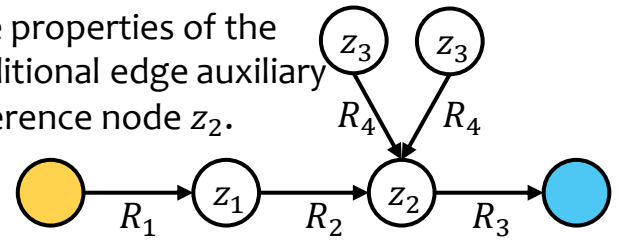
Corollary 3.3. *The rule structures learned by QL-GNN can be constructed with the recursion:*

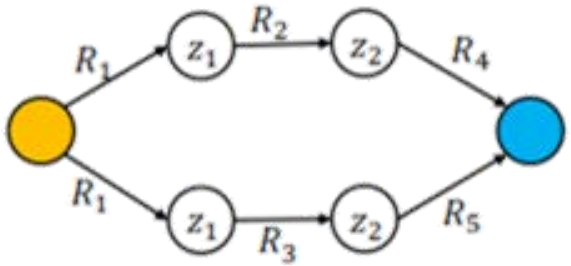
- ***Base case:*** *all unary predicates $P_i(x)$ can be learned by QL-GNN; the constant predicate $P_h(x)$ can be learned by QL-GNN;*
- ***Recursion rule:*** *if the rule structures $R_1(\mathsf{h}, x), R_2(\mathsf{h}, x), R(\mathsf{h}, y)$ are learned by QL-GNN, $R_1(\mathsf{h}, x) \wedge R_2(\mathsf{h}, y)$, $\exists^{\geq N} y\,(R_i(y, x) \wedge R(\mathsf{h}, y))$ are learned by QL-GNN.*

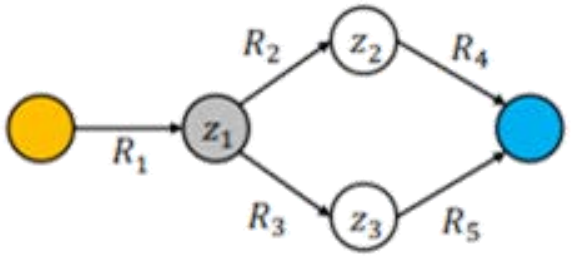
Understanding Expressivity of GNN in Rule Learning, ICLR 2024



The properties of the additional edge auxiliary inference node z_2.



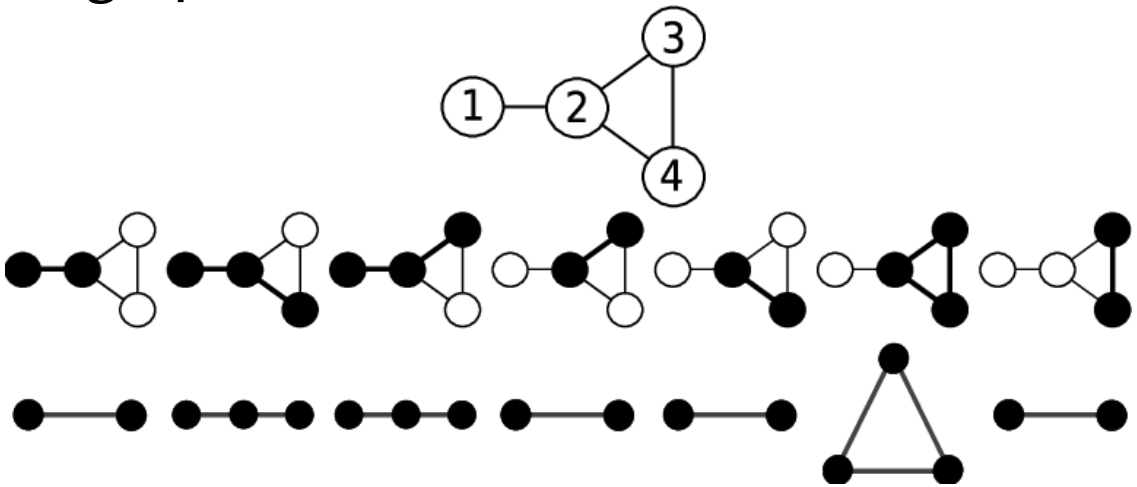




Other metrics for expressivity

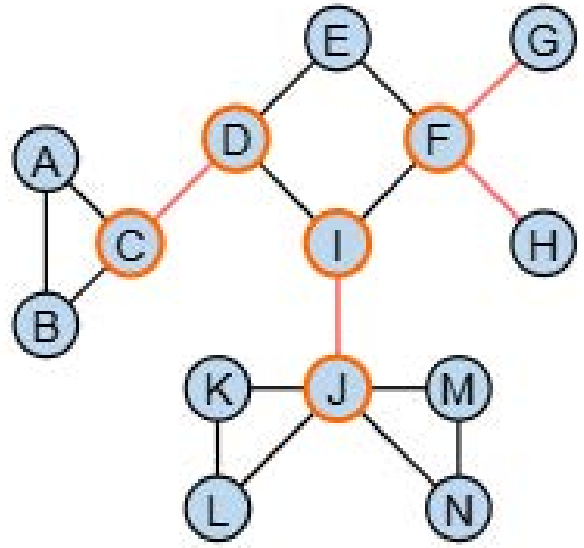
- Besides isomorphism, we can also define GNN expressivity based on other problems:

subgraph counting: counting the occurence of a given pattern in graphs



Some common “patterns” (graphlets)

graph biconnectivity: Identifying cut-edges and cut-vertices in graphs



[1] Can graph neural networks count substructures? NeurIPs 2020.
[2] Rethinking the expressive power of gnns via graph biconnectivity. ICLR 2023

Outline

- Applications in Graphs
  - Graph-structured Data
  - Graph Learning Application
- Exemplar Learning Methods on Graphs
  - Overview on Graph Representation Learning
  - Graph Embedding
  - Graph Neural Network (GNN)
  - Graph Transformer
  - Versatile Graph Learning Methods
  - Summary and Discussion
- Theory Insights
  - Theory: expressivity
  - Theory: training
  - Theory: generalization

Theory: training

- Oversmoothing
  - What is oversmoothing?
  - Methods for oversmoothing
- Oversquashing
  - What is oversquashing?
  - Methods for oversquashing
  - Weakness of existing method

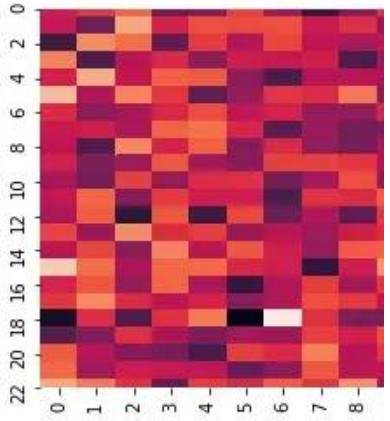
Oversmoothing

- Graph convolution

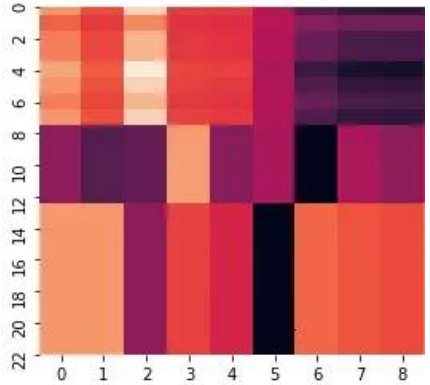
$$X^{(k+1)} = \sigma\left(PX^{(k)}W_{k+1}\right)$$

$$P = D^{-\frac{1}{2}}\hat{A}D^{-\frac{1}{2}}, \qquad \hat{A} = A + I$$

$$\lambda_i(P) \in (-1,1]$$ Contraction mapping



Node features $X^{(0)}$



Smoothed GNN features $X^{(L)}$

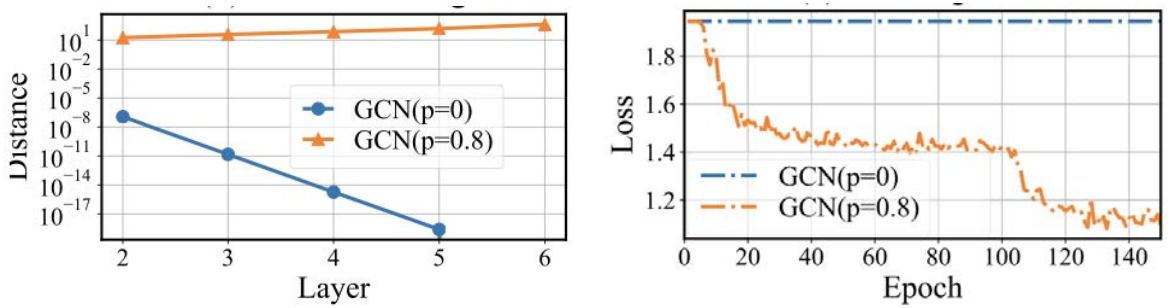
- Oversmoothing
  - Assume σ is identity mapping
  - $X^{(L)} = P^L X^{(0)} W_1 W_2 \cdots W_L$
  - Since P has many eigenvalues smaller than 1, $X^{(L)}$ will converge to be low rank as L increases (details in [1])
- How to deepen GNN to obtain better performance while prevent over-smoothing problem?

[1] Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. AAAI 2018

Alleviate Over-smoothing Problem

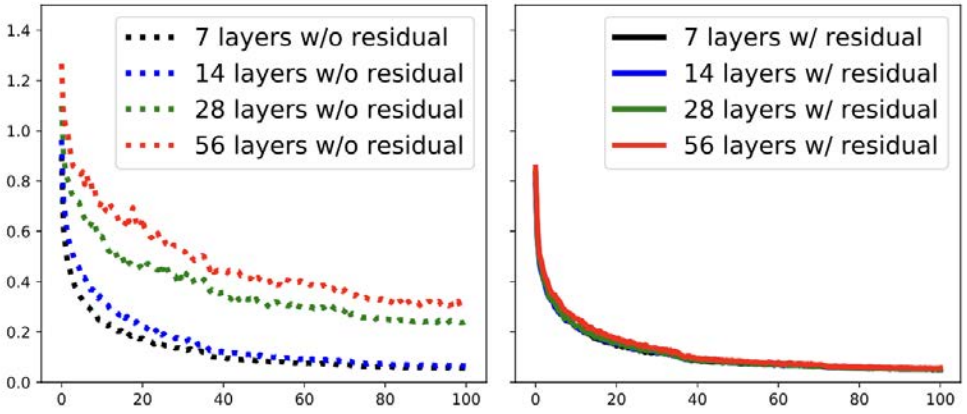
Using different adjacent matrix (drop edges in the graph)

$$A_{drop} = A \odot M$$



Using features in different layers to deepen GNNs (skip-connection)

$$H_{l+1} = WAH_l + H_l$$

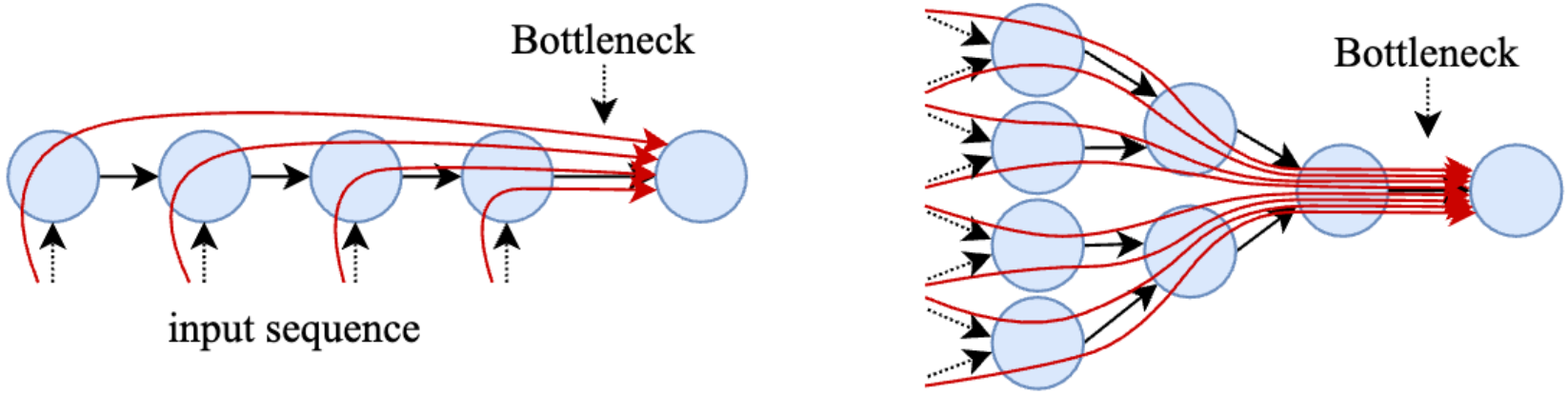


Theory: training

- Oversmoothing
  - Graph Isomorphism
  - WL-test
- Oversquashing
  - What is oversquashing?
  - Methods for oversquashing
  - Weakness of existing method

Oversquashing

- GNN compresses information from potentially exponentially many nodes (with respect to the number of layers) into fixed-sized node vectors



(a) The bottleneck of RNN seq2seq models

(b) The bottleneck of graph neural networks

Oversquashing

- Mathematical formulation

The distance between i and s is $r+1$

Lemma 1. *Assume an MPNN as in equation 1. Let $i, s \in V$ with $s \in S_{r+1}(i)$. If $|\nabla \phi_\ell| \leq \alpha$ and $|\nabla \psi_\ell| \leq \beta$ for $0 \leq \ell \leq r$, then*

$$\left| \frac{\partial h_i^{(r+1)}}{\partial x_s} \right| \leq (\alpha\beta)^{r+1} (\hat{A}^{r+1})_{is}. \tag{2}$$

Gradient vanishing **Graph structure leads to oversquashing**

Jacobian measure

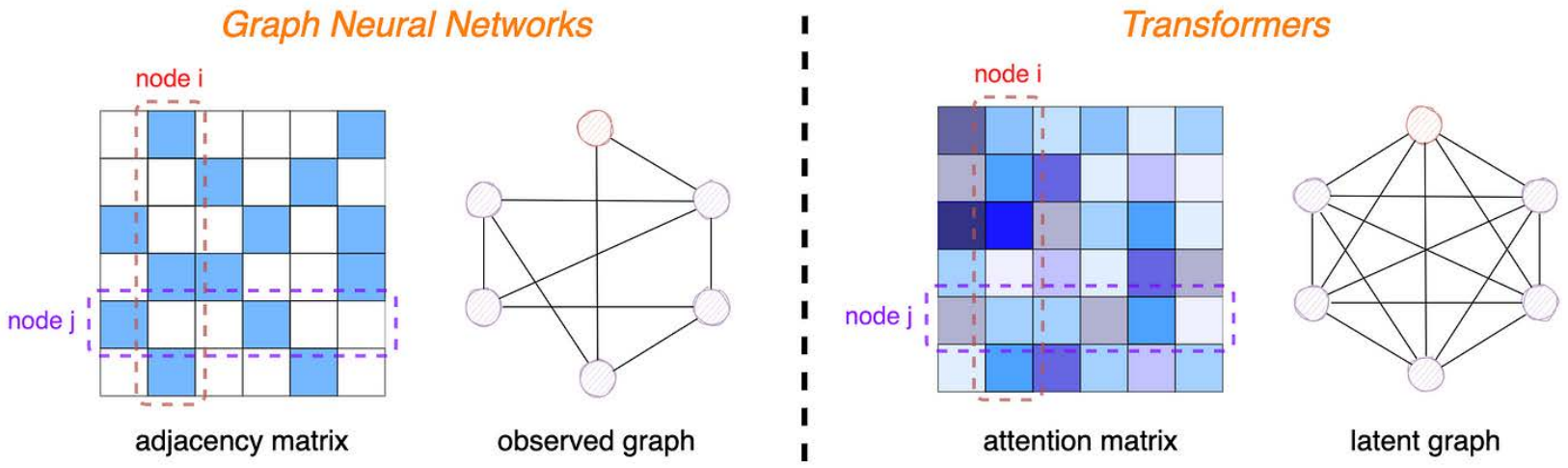
Measures the dependency between node i and s

$$h_i^{(\ell+1)} = \phi_\ell \left(h_i^{(\ell)}, \sum_{j=1}^{n} \hat{A}_{ij} \psi_\ell(h_i^{(\ell)}, h_j^{(\ell)}) \right). \tag{1}$$

Understanding over-squashing and bottlenecks on graphs via curvature. ICLR 2022

Methods for alleviating oversquashing

- Graph Transformer
  - Fully-connected graph
  - distant nodes are directly connected



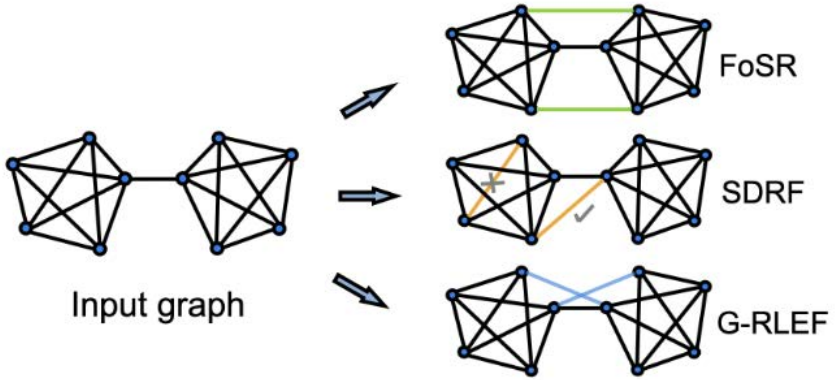
Destroy graph structures & High computational complexity

Methods for alleviating oversquashing

- Rewiring methods
  - Add/remove edges to improve Jacobian measure
    - SDRF [1]: add and remove edges to improve curvature (upper bound for spectral gap)
    - FoSR [2]: estimate spectral gap, and add edges to maximum it
    - GTR [3]: greedily add edges to minimize total resistance (optimize entire spectrum)

Smallest eigenvalue for normalized Laplacian

$$R_{tot} = n \cdot \mathrm{tr}\, L^{+} = n \sum_{i=2}^{n} \frac{1}{\sigma_i}$$



$$\left| \frac{\partial h_i^{(r+1)}}{\partial x_s} \right| \leq (\alpha\beta)^{r+1} (\hat{A}^{r+1})_{is}.$$

Optimize oversquashing measure (in)directly/greedily

[1] Understanding over-squashing and bottlenecks on graphs via curvature, ICLR 2022
[2] FoSR: First-order spectral rewiring for addressing oversquashing in GNNs, ICLR 2023
[3] Understanding Oversquashing in GNNs through the Lens of Effective Resistance, ICML 2023

$$A = P\Lambda P^{-1}$$
$$A^{r+1} = P\Lambda^{r+1}P^{-1}$$

Exponential term

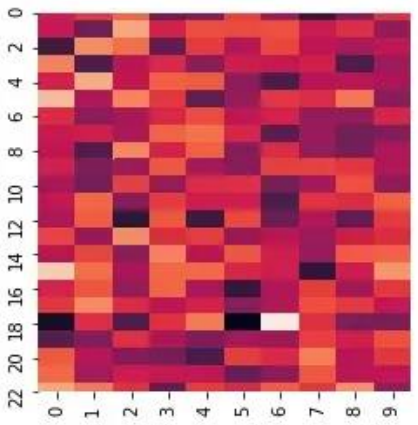
Destroy graph structures & Suboptimal oversquashing measure

Comparison

Numeric precision & Graph structure

Oversmoothing ← | ↓ | → **Oversquashing**

Hard to train deep GNN



Node features

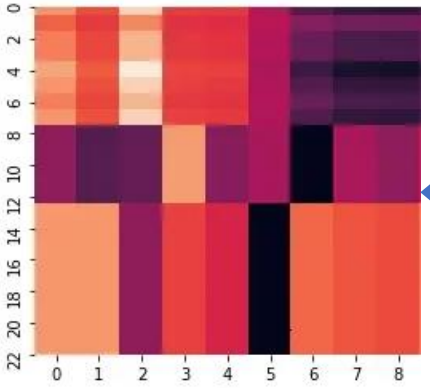
Input

$h_v^{(0)}$

GNN

$h_v^{(L)}$

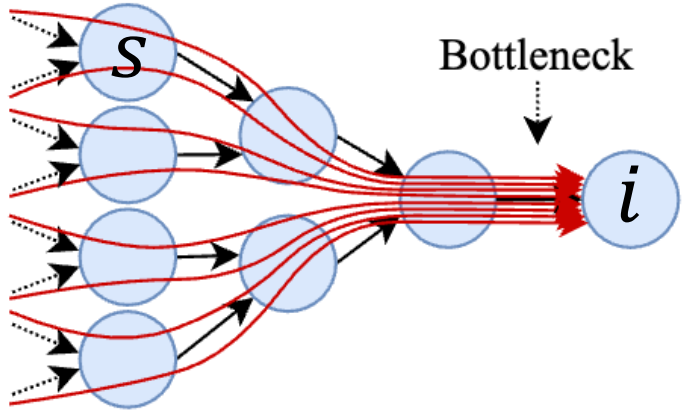
Output

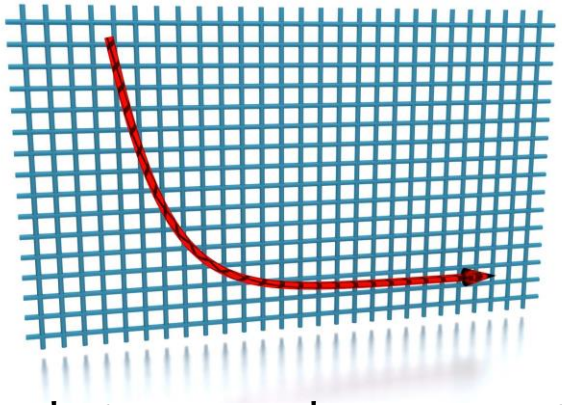


Indistinguishable

Node features outputted by GNN

$$\frac{\partial h_i^{(L)}}{\partial h_s^{(0)}}$$





Dependency between nodes exponentially decay

Outline

- Applications in Graphs
  - Graph-structured Data
  - Graph Learning Application
- Exemplar Learning Methods on Graphs
  - Overview on Graph Representation Learning
  - Graph Embedding
  - Graph Neural Network (GNN)
  - Graph Transformer
  - Versatile Graph Learning Methods
  - Summary and Discussion
- Theory Insights
  - Theory: expressivity
  - Theory: training
  - Theory: generalization

Theory: generalization

- Sample complexity
  - the number of training-samples that it needs in order to successfully learn a target function
- Sample complexity of GNN classifier

GNN with more layers (L) and larger dimension size (d), and graphs with more diverse structures (u) require more samples to train

$$N = O\left(\frac{C\ LP\log(2uP) + \ln\frac{1}{\delta}}{\epsilon}\right)$$

Less learning error and failure probability require more sample to train

Parameters related to GNN

- $P = d(2dL + L + 1)$
- L the number of layers of GNN
- d the size of feature dimension of GNN
- u the number of colors in WL test
- C is a constant

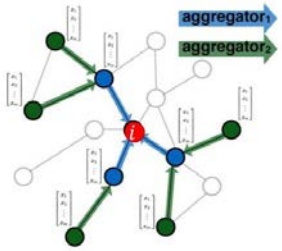
Parameters related to general machine learning

$$\Pr[\mathcal{E}(h_n) - \mathcal{E}^* \geq \epsilon] < \delta$$

- $\mathcal{E}(h_n)$ is the empirical risk of model h_n trained from n samples
- $\mathcal{E}^*$ is the optimal risk
- ϵ is the the learning error
- δ is the failure probability

WL meet VC. ICML 2023.

Challenge: Unify different GNN theories



Propagate and transform information

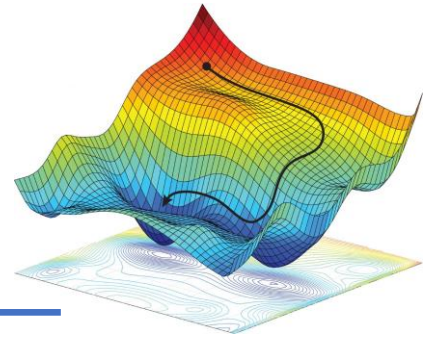
Model Design: Expressivity

- Architecture ✓
- Training dynamics ✗

Upper bound of performance

Model Training: Hard to train

- Architecture ✗
- Training dynamics ✓



Actual achievable performance

Model Performance: Generalization

- Architecture ✗
- Training dynamics ✗

How to analyze the performance for specific architecture and training algorithm?

Summary

- Graph structured data has wide applications in the real-world
- Encoding graph property into low dimensional embedding is the target of graph representation learning
- Three mainstream methods to learning graph representations: Graph embedding / Graph Neural Networks / Graph Transformer
- Versatile graph learning methods with LLMs: Graph foundation models / applying LLMs in graph learning directly
- GNN is at most as powerful as 1-WL test, and the expressivity of GNN for learning structures can be analyzed with logic
- There are some challenges to train GNN, e.g., oversmoothing and oversquashing

Thanks!

Q&A