Graph 14. Reasoning with Heterogeneous Graphs

- Graph Transformer Networks (GTN) [NeurIPS 2019]
- Graphormer [NeurIPS 2021]

1

Common limitation with conventional GNNs

- Assuming graph structures are homogeneous
  - Such as GCN, GAT, GIN, etc.
- However, real-world graphs are often heterogeneous
  - Knowledge graphs are typical examples (previous lectures)
  - Citation networks also have heterogeneous types of entities (authors, documents, conferences) and the relations (author-paper, paper-conference, etc.)

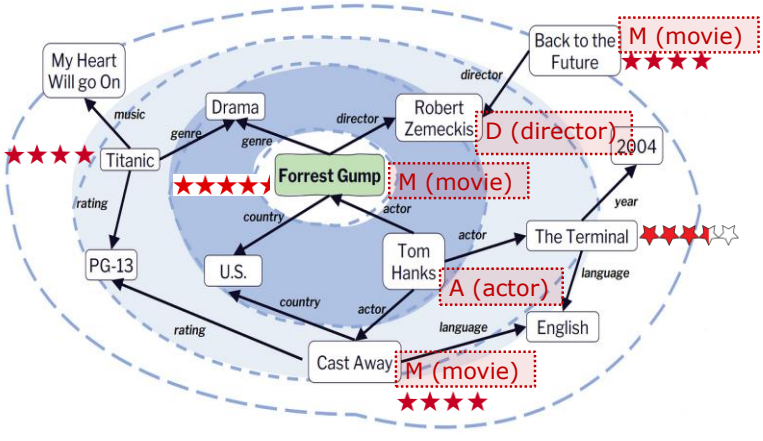
2

Approaches

- Extract useful meta-paths from heterogeneous graph by humans [WWW 2018, WWW 2019]
- Learn useful meta-paths from heterogeneous graph automatically [NeurIPS 2019]

3

What is a meta path?



Meta-paths

- MDM
- MAM
- MDMDM
- etc.

4

Graph Transformer Networks (GTN)
[S Yun et al., NeurIPS 2019]

- Key Idea & Contributions
  - Automatically extract useful meta-paths from the input graph instead of relying on hand-crafted ones;
  - Constructing a new graph where the edges are replaced by the learned meta-paths;
  - Obtaining node embedding with a regular GCN on the newly constructed graph;
  - SOTA results on evaluation benchmarks for node (multi-class) classification;
  - Offering interpretable explanation with meta-paths for the system's predictions,

    e.g., MAD = (Forest Gump, Tom Hanks, R Zemeckis) → 5-star movie rating

5

Notation [S Yun et al., NeurIPS 2019]

- Denote by $G = (V, E)$ the input graph with $|V| = N$ nodes.
- Denote by T^e as the set of edge types and $|T^e| = K$.
- For each edge type, $A_k \in \mathbb{R}^{N \times N}$ denote the adjacency matrix with $A_k[\mathrm{i}, \mathrm{j}\} \neq 0$ if there is an edge of type k from i to j and $A_k[i, j] = 0$ otherwise.
- Denote by $\mathbb{A} \in \mathbb{R}^{N \times N \times K}$ the tensor of the adjacency matrices for edge type $k = 1, \ldots, K$.
- Denote by p a meta-path of length l as $v_1 \xrightarrow{t_1} v_2 \xrightarrow{t_2} \cdots \xrightarrow{t_l} v_{l+1}$ such that $t_j \in T^e$.
- Denote by $R_p = t_1 \circ t_2 \circ \cdots \circ t_l$ the composite relation of meta-path p .
- Denote by $A_p = A_{t_1} A_{t_2} \ldots A_{t_l}$ the adjacency matrix of meta-path p (in a previous work).

6

Meth-path Generation

- Previous work [WWW 2019]

$$A_p = A_{t_1} A_{t_2} \dots A_{t_l}$$

- Proposed new way [S Yun et al., NeurIPS 2019]

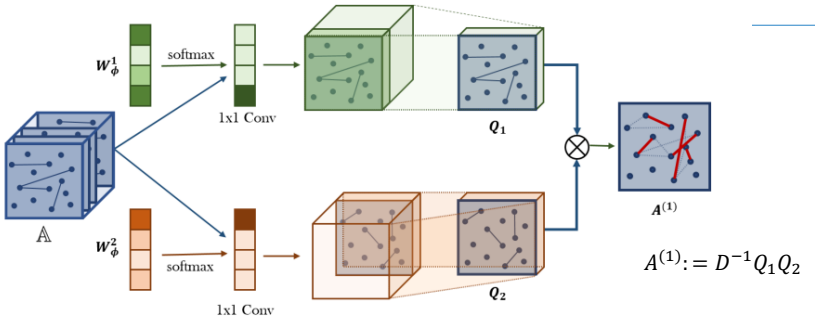
$$A_p^{(l)} = \left(\sum_{t_1 \in T^e} \alpha_{t_1}^{(1)} A_{t_1}\right)\left(\sum_{t_2 \in T^e} \alpha_{t_2}^{(2)} A_{t_2}\right) \dots \left(\sum_{t_l \in T^e} \alpha_{t_l}^{(l)} A_{t_l}\right)$$

$$= \sum_{t_1 t_2, \dots, t_L \in T^e} \alpha_{t_1}^{(1)} \alpha_{t_2}^{(2)} \dots \alpha_{t_1}^{(l)} A_{t_1} A_{t_2} \dots A_{t_l}$$

  - Resulted $A_p^{(l)} \in R^{N \times N}$ is the matrix of meta-path weights, where each edge in the meta-path has a learnable weight (in red) .

7

Graph Transformer with soft selection of edge types

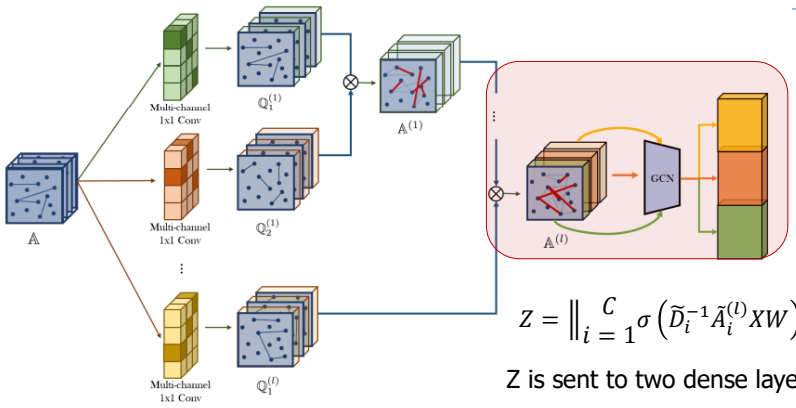


$$A^{(1)} := D^{-1} Q_1 Q_2$$

Figure 1: **Graph Transformer Layer** softly selects adjacency matrices (edge types) from the set of adjacency matrices $\mathbb{A}$ of a heterogeneous graph G and learns a new meta-path graph represented by $A^{(1)}$ via the matrix multiplication of two selected adjacency matrices Q_1 and Q_2. The soft adjacency matrix selection is a weighted sum of candidate adjacency matrices obtained by 1×1 convolution with non-negative weights from softmax(W_ϕ^1).

8

Multi-channel Convolution Layers + GCN for Learning $A^{(l)}$



$$Z = \Big\|_{i=1}^{C} \sigma\left(\tilde{D}_i^{-1}\tilde{A}_i^{(l)}XW\right) \qquad \tilde{A}_i^{(l)} = A_i^{(l)} + \mathrm{I}$$

Z is sent to two dense layers followed by a SoftMax layer.

Figure 2: Graph Transformer Networks (GTNs) learn to generate a set of new meta-path adjacency matrices $\mathbb{A}^{(l)}$ using GT layers and perform graph convolution as in GCNs on the new graph structures. Multiple node representations from the same GCNs on multiple meta-path graphs are integrated by concatenation and improve the performance of node classification. $\mathbb{Q}_1^{(l)}$ and $\mathbb{Q}_2^{(l)} \in \mathbf{R}^{N\times N\times C}$ are intermediate adjacency tensors to compute meta-paths at the lth layer.

9

9

Adding Identity matrix in $\mathbb{A}$ to learn variable length meta-paths

- Originally

$$\mathbb{A} = \{\mathrm{A}_1, \mathrm{A}_2, \ldots, A_k\}$$

- Adding identity matrix in $\mathbb{A}$

$$\mathbb{A} = \{\mathrm{A}_0, \mathrm{A}_1, \mathrm{A}_2, \ldots, A_k\} \quad \text{where } \mathrm{A}_0 = \mathrm{I}$$

- Why the latter is better?
  - Consider $f(x_1, x_2) = (ax_1 + bx_2)^2 = \mathrm{a}^2\mathrm{x}_1^2 + 2\mathrm{ab}\mathrm{x}_1\mathrm{x}_2 + \mathrm{b}^2\mathrm{x}_2^2$
  - Verses $\phi(x_1, x_2) = (1 + ax_1 + bx_2)^2 = 1 + 2\mathrm{a}\mathrm{x}_1 + 2\mathrm{b}\mathrm{x}_2 + \mathrm{a}^2\mathrm{x}_1^2 + 2\mathrm{ab}\mathrm{x}_1\mathrm{x}_2 + \mathrm{b}^2\mathrm{x}_2^2$
  - The former only have quadratic terms and the latter have scaler, linear and quadratic ones.

10

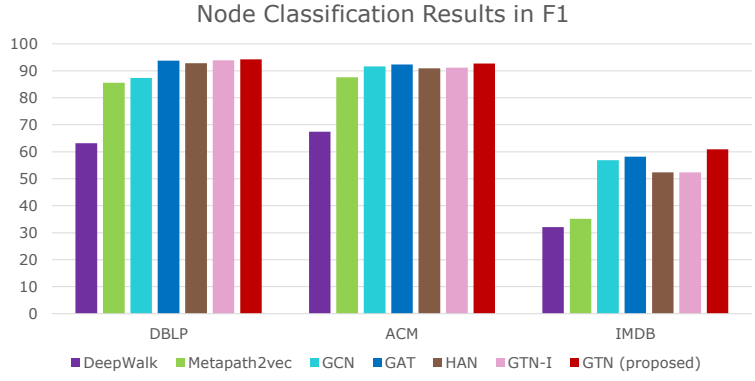
Evaluation Results on Multi-class Classification of Nodes
[S Yun et al., NeurIPS 2019]

Table 2: Evaluation results on the node classification task (F1 score).

	DeepWalk	metapath2vec	GCN	GAT	HAN	GTN$_{-I}$	**GTN (proposed)**
DBLP	63.18	85.53	87.30	93.71	92.83	93.91	**94.18**
ACM	67.42	87.61	91.60	92.33	90.96	91.13	**92.68**
IMDB	32.08	35.21	56.89	58.14	56.77	52.33	**60.92**

- $GTN > all\ other\ methods$ (Significantly?)

Evaluation Results on Multi-class Classification of Nodes
(my graph)



- $DeepWalk \ll Others$
- $GTN > Others$ (Significantly?)
- $GTN \approx GAT$ (2nd best)
- $GAT > GCN$ (Why?)
- $GTN > GTN_{-I}$ (Why?)
- $GAT > GTN_{-I}$!

Concluding Remarks on GTN

- Supporting automated extraction of meta-paths with learned (task-adapted) weights.
- Providing understandable interpretation of predictions
- Allowing traditional GNNs to be plugged in the generic framework (instead of designing new GNNs for reasoning with meta-paths).
- Results (on node classification tasks) are slightly better than GAT.
- Can we improve GAT (originally designed for sequences) in graph-based reasoning?

Do Transformers Relly Perform Bad for Graph Representation?
[C Ying et al., NeurIPS 2021]

- **Transformer → Graphormer** (proposed)
  - Building upon a standard Transformer architecture
  - Extending Transformer by leveraging graph-based features (e.g., node degrees, edge types, shortest path between any pair of nodes)
  - Excellent results on evaluation benchmarks, especially the **Open Graph Benchmark Large-Scale Challenge (OGP-LSC) with 3.8M graphs** and several popular leaderboards

Graphormer Layers

$$h'^{(l)} = MHA\left(LN(h^{(l-1)}\right) + h^{(l-1)}$$

$$h^{(l)} = FFN\left(LN(h'^{(l)}\right) + h'^{(l)}$$

$$h^{(G)} = \text{READOUT}\left(\left\{h_i^{(L)} | v_i \in G\right\}\right)$$

where LN stands for Layer Normalization,;

MHA stands for multi-head self-attention;

FFN stands for point-wise feed-forward neural network.

- Exactly the same as in standard Transformer? **The devil is in the details.**

Graphormer Encoding Trick 1

- **Centrality encoding of each node** (new parts in red)

$$h_i^{(0)} = x_i + z^-_{\deg^-(v_i)} + z^+_{\deg^+(v_i)} \quad (1)$$

where x_i is the feature vector of node v_i ; $z^-_{\deg^-(v_i)} \in \mathbb{R}^d$ and $z^+_{\deg^+(v_i)} \in \mathbb{R}^d$ are the embeddings of the in/out degrees of the node, respectively.

Graphormer Encoding Trick 2

- **Spatial Encoding in Attention**

$$A_{ij} = \frac{(h_i W_Q)(h_j W_K)^T}{\sqrt{d}} + b_{\phi(v_i,v_j)} \quad (2)$$

where $\phi(v_i, v_j)$ is the length of the shortest path between nodes v_i and v_j;

$b_{\phi(v_i,v_j)} \in \mathbb{R}$ is a learnable scalar indexed by $\phi(v_i, v_j)$ and shared across all layers.

Graphormer Encoding Trick 3

- **Edge Encoding**
  - We calculate the attention between a node pair (v_i, v_j) as

$$A_{ij} = \frac{(h_i W_Q)(h_j W_K)^T}{\sqrt{d}} + b_{\phi(v_i,v_j)} + c_{ij} \quad (4)$$

where e_n denote the n'th edge in the shortest path $SP_{ij} = (e_1, e_2, \dots, e_N)$ between the two nodes;

$x_{e_n} \in \mathbb{R}^{d_E}$ is the edge feature (e.g., the bond type between to atoms is a molecular graph);

$w_n \in \mathbb{R}^{d_E}$ is the positional embedding of the $n'th$ weight in the shortest path;

$c_{ij} = \frac{1}{N}\sum_{n=1}^{N} x_{e_n}^T w_n$ is the average edge weight in the shortest path.

Graph Benchmarks: Large-scale Challenges

[C Ying et al., NeurIPS 2021]

Table 6: Statistics of the datasets.

Dataset	Scale	# Graphs	# Nodes	# Edges	Task Type
PCQM4M-LSC	Large	3,803,453	53,814,542	55,399,880	Regression
OGBG-MolPCBA	Medium	437,929	11,386,154	12,305,805	Binary classification
OGBG-MolHIV	Small	41,127	1,048,738	1,130,993	Binary classification
ZINC (sub-set)	Small	12,000	277,920	597,960	Regression

The task of PCQM4M-LSC is to predict DFT(density functional theory)-calculated HOMO-LUMO energy gap of molecules given their 2D molecular graphs, which is one of the most practically-relevant quantum chemical properties of molecule science. PCQM4M-LSC is unprecedentedly large in scale comparing to other labeled graph-level prediction datasets, which contains more than 3.8M graphs. Besides, we conduct experiments on two molecular graph datasets in popular OGB leaderboards, i.e., OGBG-MolPCBA and OGBG-MolHIV. They are two molecular property prediction datasets with different sizes. The pre-trained knowledge of molecular graph on PCQM4M-LSC could be easily leveraged on these two datasets. We adopt official scaffold split on three datasets following [21, 22]. In addition, we employ another popular leaderboard, i.e., benchmarking-gnn [14]. We use the ZINC datasets, which is the most popular real-world molecular dataset to predict graph property regression for contrained solubility, an important chemical property for designing generative GNNs for molecules. Different from the scaffold spliting in OGB, uniform sampling is adopted in ZINC for data splitting.

19

Benchmark Evaluation Results

[C Ying et al., NeurIPS 2021]

Table 1: Results on PCQM4M-LSC. * indicates the results are cited from the official leaderboard [21].

method	#param.	train MAE	validate MAE
GCN [26]	2.0M	0.1318	0.1691 (0.1684*)
GIN [54]	3.8M	0.1203	0.1537 (0.1536*)
GCN-VN [26, 15]	4.9M	0.1225	0.1485 (0.1510*)
GIN-VN [54, 15]	6.7M	0.1150	0.1395 (0.1396*)
GINE-VN [5, 15]	13.2M	0.1248	0.1430
DeeperGCN-VN [30, 15]	25.5M	0.1059	0.1398
GT [13]	0.6M	0.0944	0.1400
GT-Wide [13]	83.2M	0.0955	0.1408
Graphormer$_{\text{SMALL}}$	12.5M	0.0778	0.1264
Graphormer	47.1M	**0.0582**	**0.1234**

VN: virtual node

12-layer → DeeperGCN-VN

20

Benchmark Evaluation Results

[C Ying et al., NeurIPS 2021]

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 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

Concluding Remarks on Graphormer

- Graphomer extends Transformer's capability in graph-based learning with relatively simple yet effective encoding tricks (such as degree-based node embedding, SP-enriched attention, and incorporating edge-type information and SP-positional embedding into attention).
- Strong results on a broad range of regression/classification tasks on the largest evaluation benchmarks, significantly outperforming GAT, GCN and GIN (although direct comparison with GTN is not possible due to the lack of available results on the same benchmarks.)

References

- [NeurIPS 2019] Graph Transformer Networks Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, Hyunwoo J. Kim
- [NeurIPS 2021] Do Transformers Relly Perform Bad for Graph Representation? Chengxuan Ying , Tianle Cai , Shengjie Luo , Shuxin Zheng , Guolin Ke , Di He , Yanming Shen , Tie-Yan Liu.