上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Relational/Graph Learning

Teacher: Zheng Wang

Social Network Analysis (NIS8023)

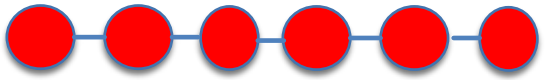
Shanghai Jiao Tong University

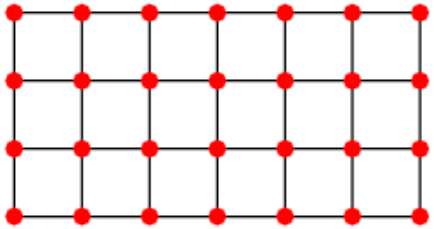
Outline

- Shallow Methods
  - Collective Classification
  - Label Propagation
- **Deep Learning Methods**
  - Graph Neural Networks

Deep Learning Meets Graphs: Challenges

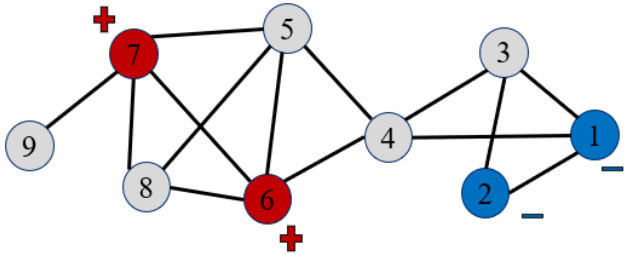
- Traditional DL is designed for simple grids or sequences
  - CNNs for fixed-size images/grids
  - RNNs for sequential data like text.



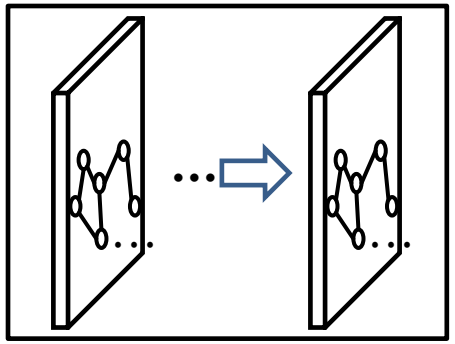


Deep Learning Meets Graphs: Challenges

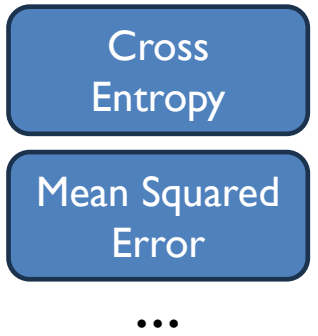
- Challenge with graphs
  - Structure: nodes have arbitrary connections and neighbor sizes.
  - Order: nodes lack a fixed ordering.



Graph



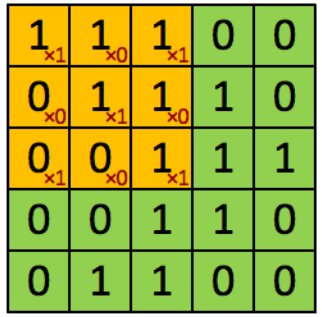
Our aim of GNNs



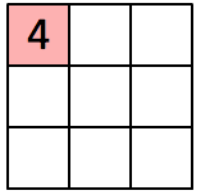
Various loss functions

Compare with CNN

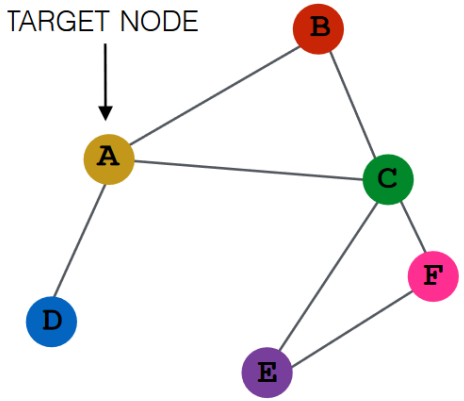
- Recall CNN
  - Regular "graph"
- Graph Neural Network (GNN)
  - Extend to irregular graph structure



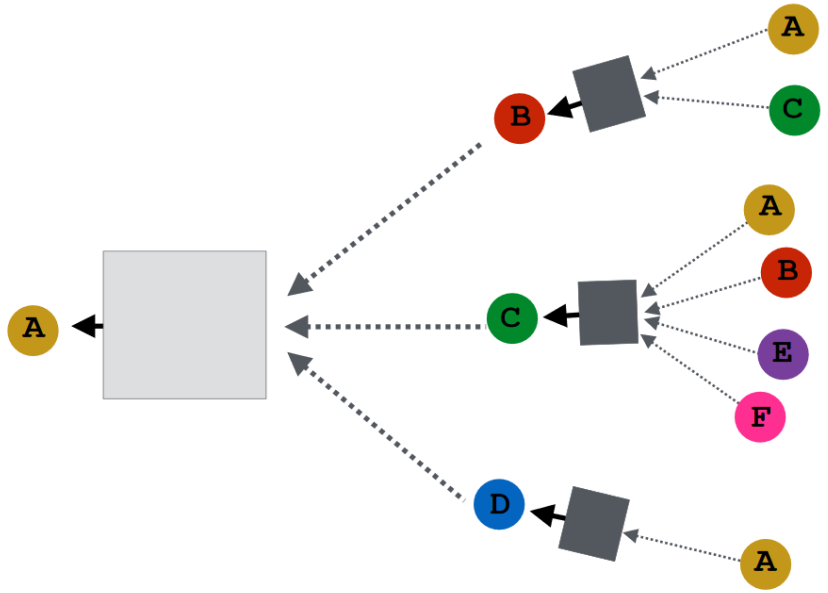
Image



Convolved Feature

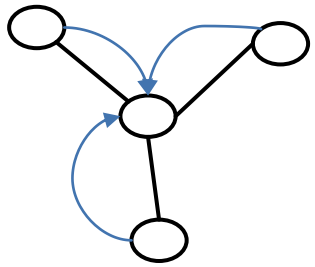


INPUT GRAPH



The General Architecture of GNNs

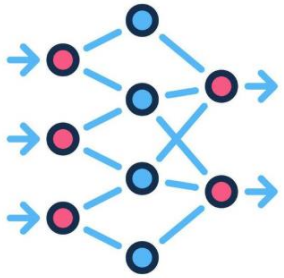
- GNNs iteratively compute node representations through two main components:
  - Message Passing: Aggregates information from a node and its neighbors.



message passing

  - Feature Mapping: Transforms features into a new representation space.

node features at layer $t-1$



feature mapping (via neural network)

node features at layer t

Formulation of the General Architecture of GNNs

- For a node v at layer t, the two key components GNNs are:
  - Message Passing: $$m_v^{(t)} = f_{msp}\left(h_v^{(t)}, \left\{h_u^{(t)} \mid u \in \mathcal{N}(v)\right\}\right)$$

    representation vector from previous layer for node v

    representation vectors from previous layer for node v's neighbors

    where $h_v^{(t)}$ represents the vector of node v at layer t, $N(v)$ is the neighbors of node v, $f_{msp}()$ is the message passing function, and $m_v^{(t)}$ is the aggregation result,

  - Feature Mapping: $$h_v^{(t+1)} = f_{map}(m_v^{(t)})$$

    where $f_{map}()$ is the feature mapping function.

Various GNNs: Graph Convolutional Network (GCN)

- Message passing: $$m_v^{(t)} = \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{h_u^{(t)}}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}}$$

- Feature mapping: $$h_v^{(t+1)} = \sigma\left(\mathbf{W}^{(t)} m_v^{(t)}\right)$$

$\mathbf{W}^{(t)}$: *weight matrix at layer t*
$\sigma()$: *nonlinear activation function*

One line formulation of GCN by writing the above two operations together:

$$\sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(t-1)}\mathbf{W}^{(t-1)})$$

A: the adjacent matrix
I: the identity matrix
$\hat{A} = A + I$: the adjacent matrix with self-loops
$\hat{D}$: the degree matrix of $\hat{A}$
H: the matrix form of node embedding $(\text{i.e.}, h_v)$

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

A toy example of 2-layer GCN on a 4-node graph

- Computation graph

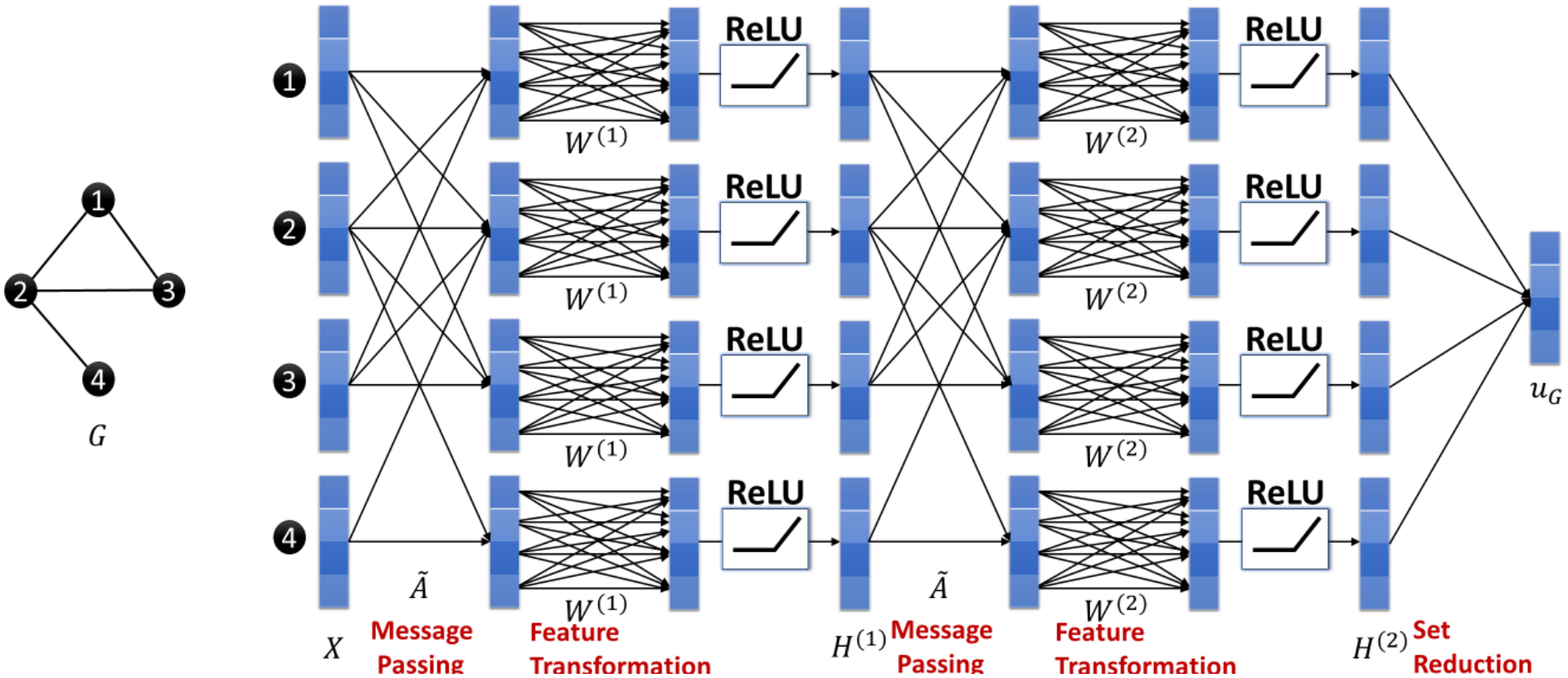


Image from CS247 UCLA

Various GNNs: Simple Graph Convolution (SGC)

- Message passing (same as GCN):

$$m_v^{(t)} = \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{h_u^{(t)}}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}}$$

- Feature mapping:
  - Layer [1, K-1] $h_v^{(t+1)} = m_v^{(t)}$

    ...
  - Layer K (same as GCN) $h_v^{(K)} = \mathbf{W} m_v^{(K-1)}$

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., & Weinberger, K. (2019, May). Simplifying graph convolutional networks. In *International conference on machine learning* (pp. 6861-6871). PMLR.

Various GNNs: GraphSAGE

- Message passing:

$$m_v^{(t)} = \text{CONCAT}\left(h_v^{(t)}, \left\{h_u^{(t)} \mid u \in \mathcal{N}(v)\right\}\right)$$

CONCAT(): the concatenation operator
Note: the authors argues that CONCAT() can be replaced by various ops, like ADD, LSTM, MEAN, and Pool.

- Feature mapping (same as GCN):

$$h_v^{(t+1)} = \sigma\left(\mathbf{W}^{(t)} m_v^{(t)}\right)$$

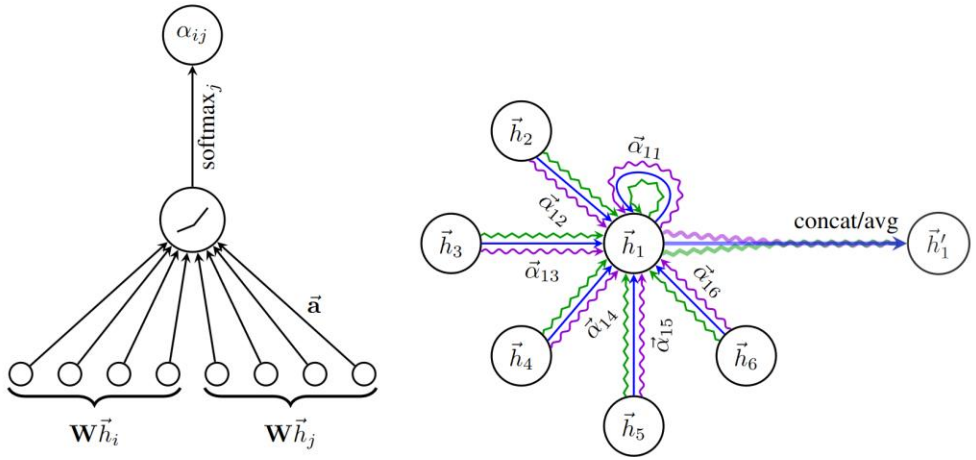
Hamilton, W., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. Advances in neural information processing systems, 30.

Various GNNs: Graph Attention Network (GAT)

- Message passing:

$$m_v^{(t)} = \sum_{u \in \mathcal{N}(v) \cup \{v\}} \alpha_{vu} h_u^{(t)}$$

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T [\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T [\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$



- Feature mapping: the same as GCN.

$$h_v^{(t+1)} = \sigma\left(\mathbf{W}^{(t)} m_v^{(t)}\right)$$

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. arXiv preprint arXiv:1710.10903.

A SOTA GNN: **O**ptimized Simple **G**raph **C**onvolution (**OGC**)

- The accuracy of OGC is SOTA

	Method	#Layer	Cora	Citeseer	Pubmed
supervised	LP	–	71.5	48.9	65.8
	ManiReg	–	59.5	60.1	70.7
	GCN	2	81.5	70.3	79.0
	APPNP	2	83.3	71.8	80.1
	Eigen-GCN	2	78.9± 0.7	66.5±0.3	78.6±0.1
	GNN-LF/HF	2	84.0±0.2	72.3±0.3	80.5±0.3
	C&S	3	84.6±0.5	75.0±0.3	81.2±0.4
	NDLS	2	84.6±0.5	73.7±0.6	81.4±0.4
	ChebNetII	2	83.7±0.3	72.8±0.2	80.5±0.2
	OAGS	2	83.9±0.5	73.7±0.7	81.9±0.9
	JKNet	{4, 16, 32}	82.7 ± 0.4	73.0 ± 0.5	77.9 ± 0.4
	Incep	{64, 4, 4}	82.8	72.3	79.5
	GCNII	{64, 32, 16}	85.5±0.5	73.4 ± 0.6	80.2 ± 0.4
	GRAND	{8, 2, 5}	85.4±0.4	75.4±0.4	82.7±0.6
	ACMP	{8, 4, 32}	84.9±0.6	75.5 ± 1.0	79.4±0.4
	OGC (ours)	> 2	**86.9 ± 0.0**	**77.5 ± 0.0**	**83.4 ± 0.0**

State of the Art | Node Classification on CiteSeer with Public Split: fixed 20 nodes per class

State of the Art | Node Classification on Cora with Public Split: fixed 20 nodes per class

State of the Art | Node Classification on PubMed with Public Split: fixed 20 nodes per class

Recorded by Paperswithcode.com

Wang, Z., Ding, H., Pan, L., Li, J., Gong, Z., & Philip, S. Y. (2024). From cluster assumption to graph convolution: Graph-based semi-supervised learning revisited. *TNNLS*.

A SOTA GNN: **O**ptimized Simple **G**raph **C**onvolution (**OGC**)

- The speed of OGC is very fast

Type	Method	Running on CPU							Running on GPU						
		Layers/Iterations						Total*	Layers/Iterations						Total*
		2	4	8	16	32	64		2	4	8	16	32	64	
supervised	GCNII	98.02	234.60	545.44	923.61	1555.37	4729.32	8086.36	4.60	10.76	29.64	54.08	200.20	780.24	1079.52
	GRAND	857.23	952.29	2809.84	4361.91	6225.39	15832.91	31039.57	35.45	70.99	274.95	2090.65	2961.71	6829.78	12263.54
	OGC	1.88	3.69	7.29	14.63	30.36	61.88	61.88				-			-
unsupervised	SGC	0.22	0.28	0.44	0.77	1.46	2.80	5.78	0.52	0.52	0.55	0.61	0.70	0.82	3.59
	S^2GC	0.25	0.36	0.56	0.94	2.16	3.66	6.88	0.57	0.60	0.62	0.66	0.75	0.92	3.99
	GGC	0.27	0.50	0.85	1.60	3.10	6.02	10.26	0.58	0.59	0.62	0.67	0.80	1.01	4.08
	GGCM	0.37	0.57	1.06	1.93	3.77	7.27	12.45	0.60	0.63	0.64	0.69	0.85	1.09	4.35

Running time (seconds) on Pubmed.

OGC is 130 to 500 times more efficient than the best runner-up deep GNNs.

How OGC works?

- Each layer of OGC includes two parts:
  - Message passing (lazy graph convolution):

$$(1-\beta)U_i^{(k)} + \beta \underbrace{\sum_{i \in N(j) \cup \{j\}} \frac{U_i^{(k)}}{\sqrt{|\mathcal{N}(i)||\mathcal{N}(j)|}}}_{\text{Same as GCN}}$$

  - Supervised EmBedding (SEB):

$$-\eta_{sup} S(-Y + Z^{(k)})W^{\top}$$

W is a label prediction function
Y is the original label information
$Z^{(k)}$ is the predicted soft labels at the k-th iteration
S is a diagonal matrix (for supervised label indicator)

$\beta \in (0,1)$: the moving probability that a node moves to its neighbors in every period.
Note: U is used to stand for node embedding matrix to be consist with the paper.

One Line Formulation of OGC

At the each (i.e., k-th) layer, OGC can be formulated in one line:

$$U^{(k+1)} = \underbrace{[\beta \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} + (1-\beta) I_n] U^{(k)}}_{\text{Lazy Graph Convolution}} - \underbrace{\eta_{sup} S(-Y + Z^{(k)}) W^\top}_{\text{Supervised EmBedding (SEB)}}$$

For node classification, at the last (i.e., k-th) layer, OGC gets the label predictions:

$$\hat{Y}^{(k)} \text{ from: } Z^{(k)} = U^{(k)} W$$

How to Train OGC: Less Is More (LIM) Trick

- OGC is a shallow method, which means it does not need huge validation set. In the learning process, it uses both train and validation sets as the supervised knowledge.
  - Parameter learning objective: Supervised using additional labels from both the training and validation sets.
  - Parameter update objective: Performed solely on the training set, excluding the validation set to avoid overfitting.

The core idea behind the LIM Trick is to enable AI models to learn from extensive supervised datasets—including both training and validation data—while minimizing parameter updates to prevent overfitting.

Pseudo-code of OGC

Algorithm 1 OGC

Input: Graph information (A and X), the max iteration number K, and the label information of a small node set $\mathcal{V}_L$

Output: The label predictions

1: Initialize $U^{(0)} = X$ and $k = 0$
2: **repeat**
3: $k = k + 1$
4: Update W manually or by some automatic-differentiation toolboxes (Sect. 4.1)
5: Update $U^{(k)}$ via (lazy) supervised graph convolution (Eq. 6)
6: Get the label predictions $\hat{Y}^{(k)}$ from: $Z^{(k)} = U^{(k)}W$
7: **until** $\hat{Y}^{(k)}$ converges or $k \geq K$
8: **return** $\hat{Y}^{(k)}$

OGC is very simple with only three steps (at each iteration):

1. Update the label prediction function W
2. Update the node embedding results U
3. Get the label prediction results

More Types of GNNs

- Heterogeneous GNNs (HGNNs)
- Graph-Level GNNs
- Other Advanced GNNs (e.g., Transformers for graphs, GNN-based Autoencoders)

Thanks for your time.
QA.