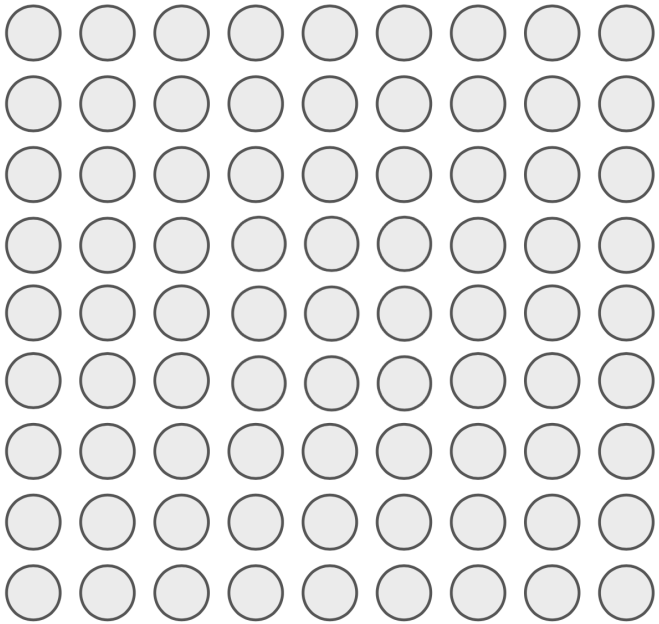
Part II: Graph Convolutional Networks

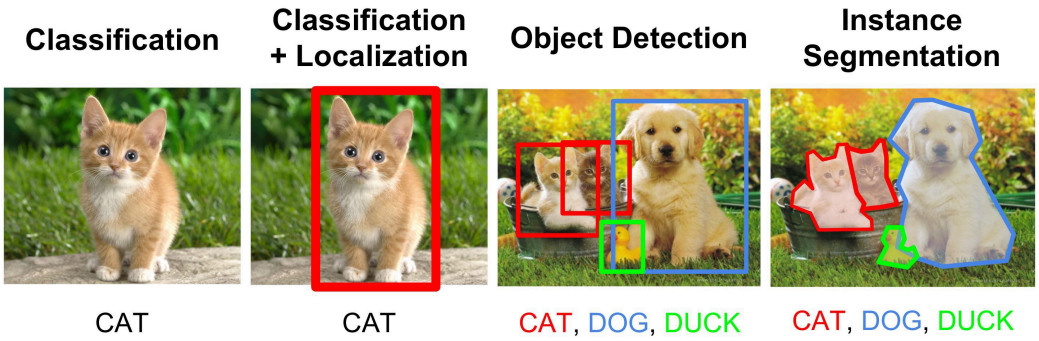
Jinhong Jung
Seoul National University
jinhongjung@snu.ac.kr

Motivation (1)

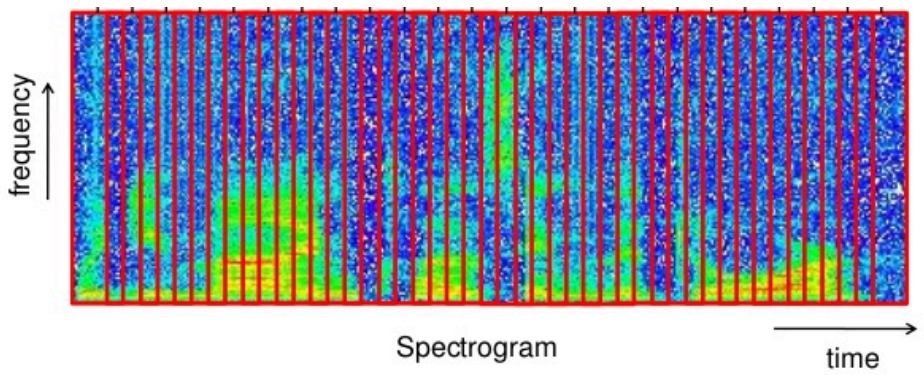
- **Traditional deep learning models have focused on grid data (e.g., image, video, text, and audio)**



Grid structure
(Multi-dimensional array)



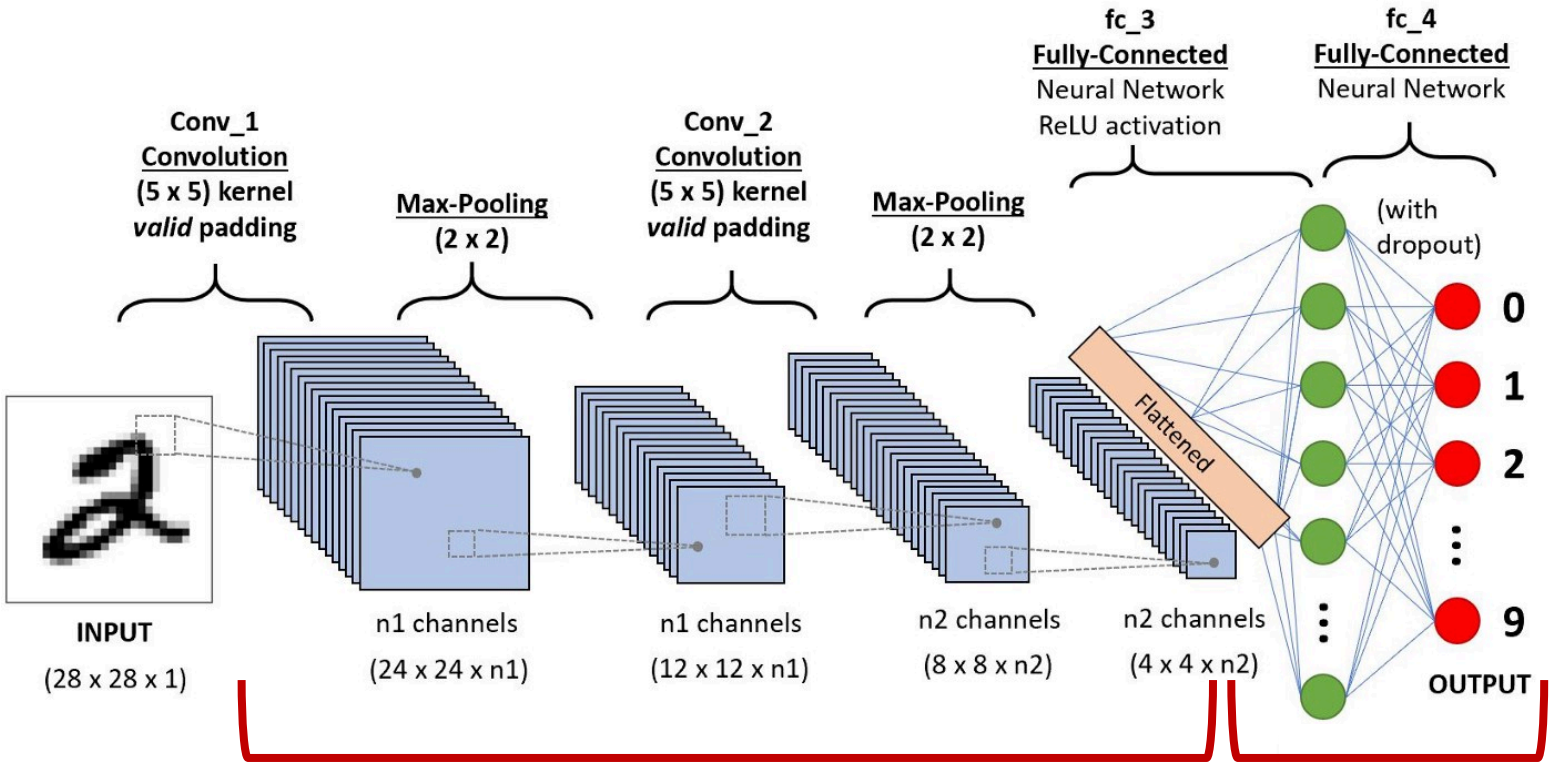
ML tasks on image data



ML tasks on audio data

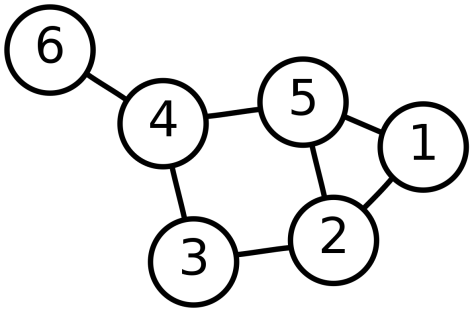
Motivation (2)

- **CNN works well on such grid-typed data**
  - ***End-to-end learning***: jointly learn hidden features and a supervised task (e.g., classification)



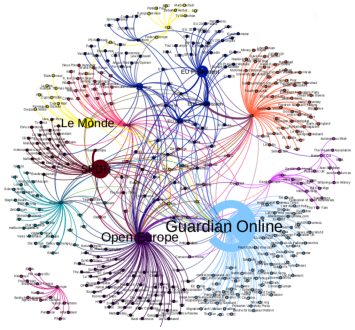
Motivation (3)

- **Traditional models such as CNN do not work on non-grid graph structured data**
  - Traditional convolution assumes grid structure
  - There are tremendous graph structured data

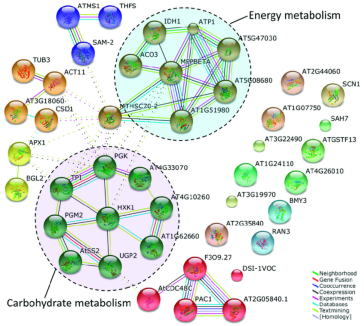


Non-grid graphs

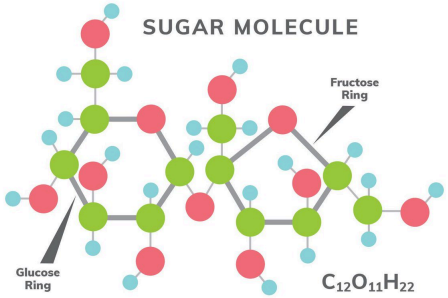
Friendship



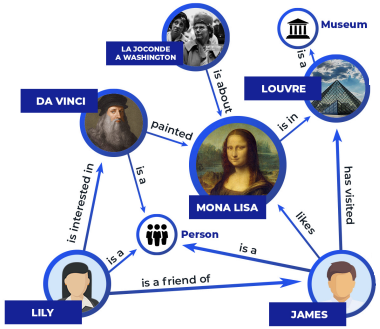
Hyperlink



Gene

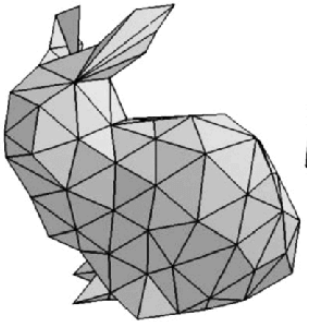


Molecule



Knowledge

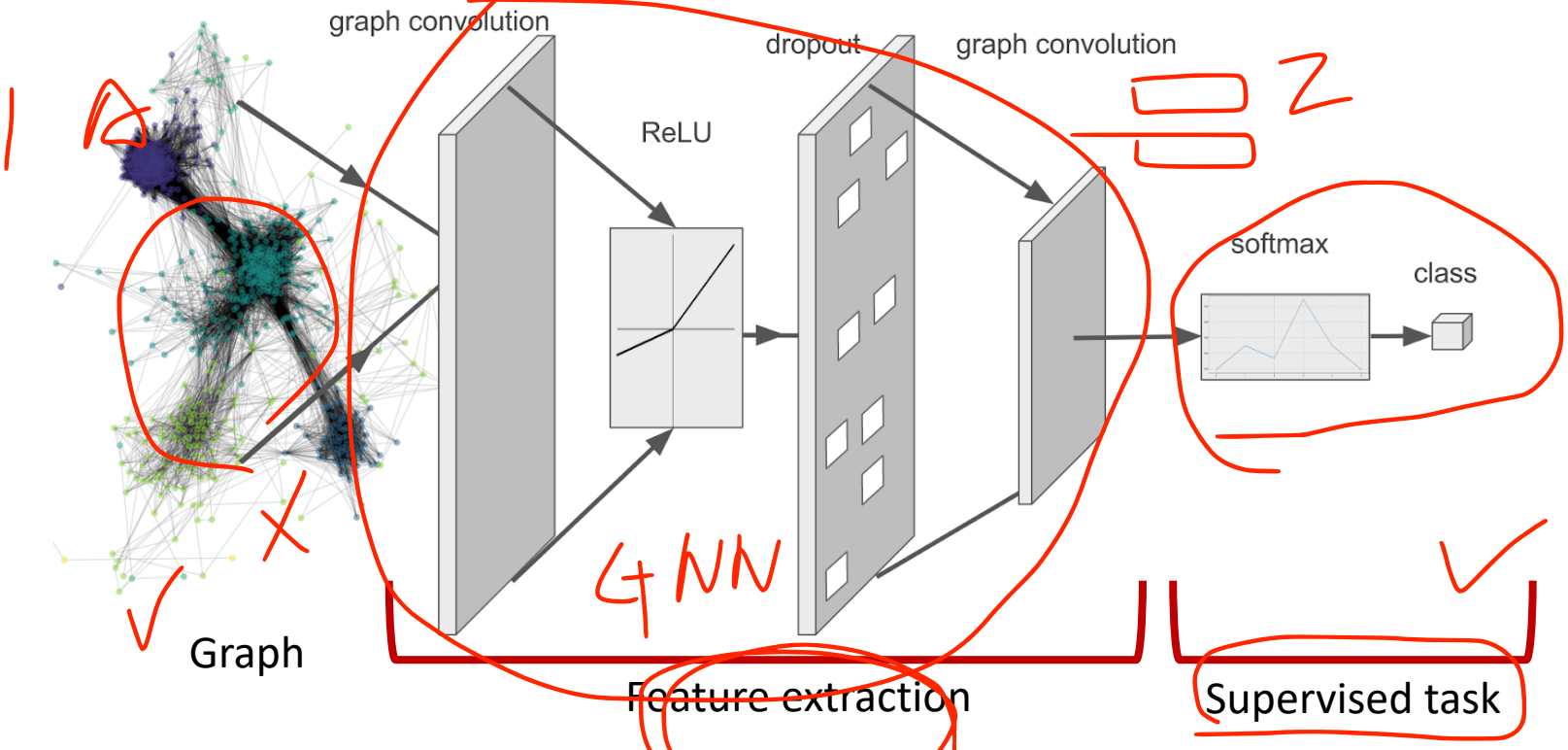
Point cloud



3D mesh

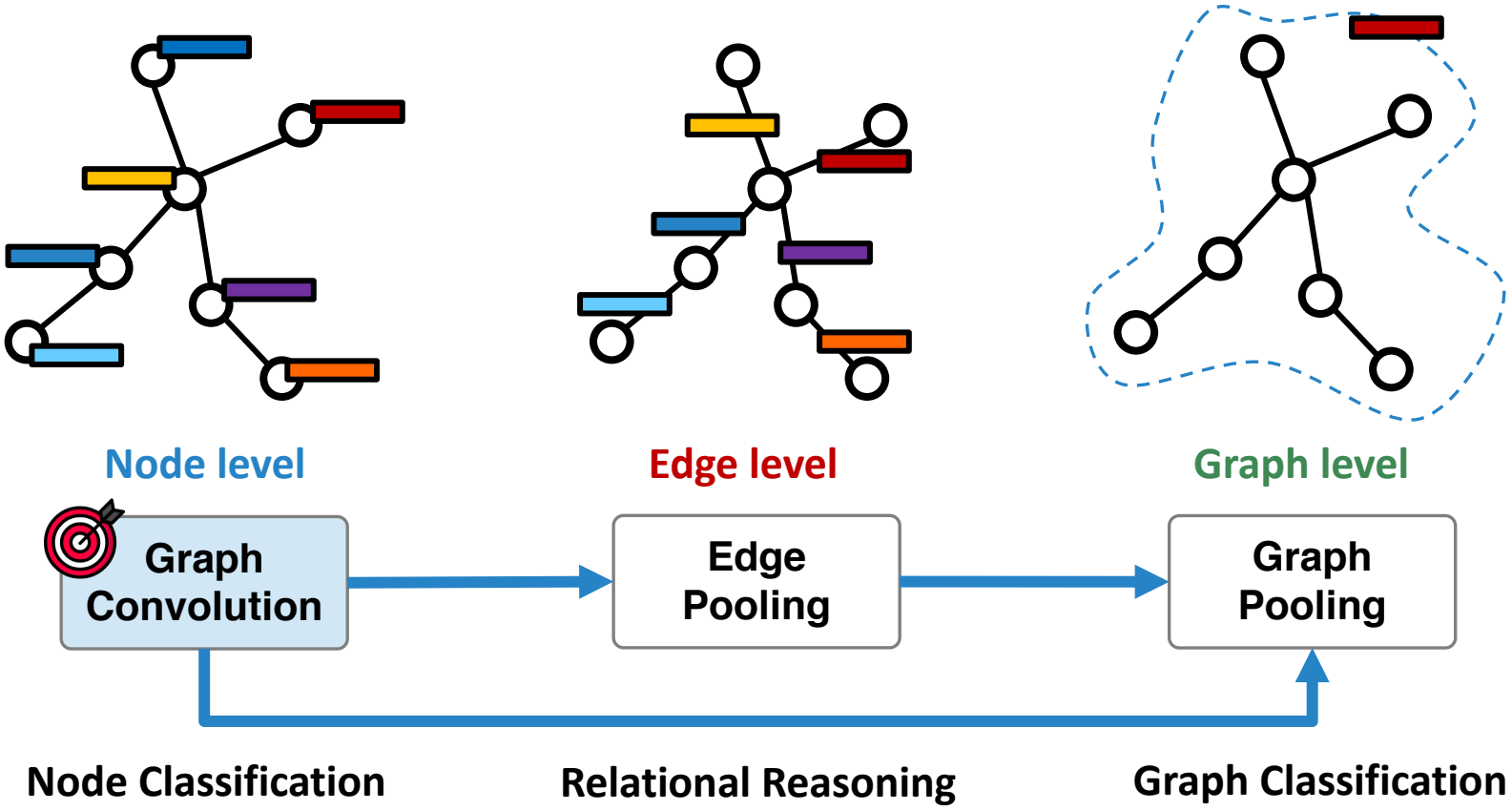
Question

- **How can we perform various ML tasks on graph structured data?**
  - Like CNN, how can we jointly learn *latent features* and *a supervised task* in a graph?
  - ⇒ **Graph convolutional networks (GCN) can do this!**



Representation Level

- Three representation levels in graph learning
  - Latent feature vector for a **node**, an **edge**, or a **graph**



Applications of GCN

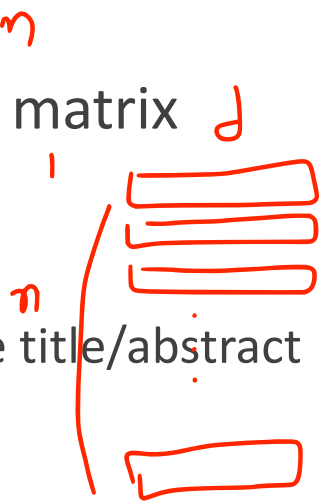
- **Numerous applications in various research fields!**

3.1 Physics	3.2 Chemistry and Biology
3.3 Knowledge Graph	3.4 Recommender Systems
3.5 Computer Vision	3.6 Natural Language Processing
3.7 Generation	3.8 Combinatorial Optimization
3.9 Adversarial Attack	3.10 Graph Clustering
3.11 Graph Classification	3.12 Reinforcement Learning
3.13 Traffic Network	3.14 Few-shot and Zero-shot Learning
3.15 Program Representation	3.16 Social Network
3.17 Graph Matching	3.18 Computer Network

https://github.com/thunlp/GNNPapers

Problem Definition

- **Input**
  - Undirected graph $G = (\mathcal{V}, \mathcal{E})$ where $\mathbf{A}$ is adj. matrix
  - Initial node feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$
    - n: number of nodes & d: feature dimension
    - e.g., bag-of-words feature vector extracted from the title/abstract of papers in citation networks



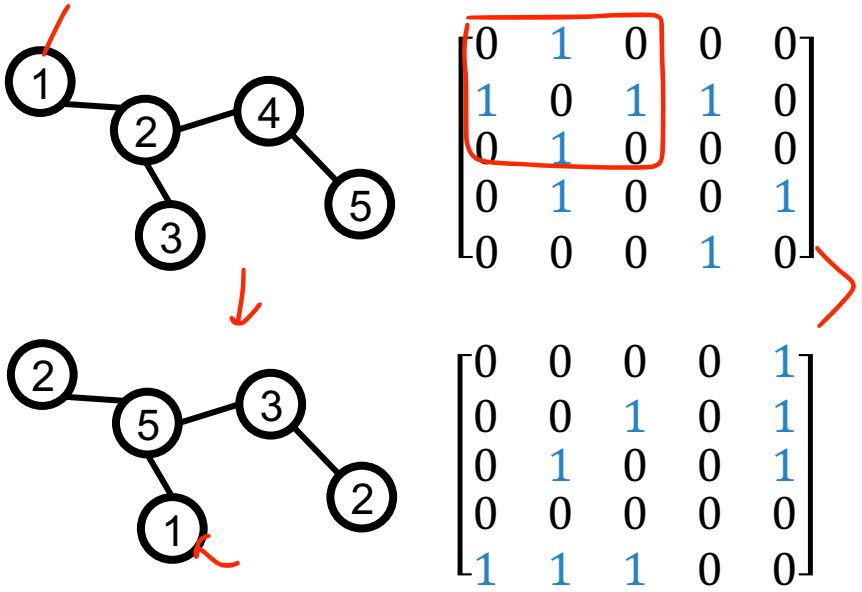
- **Output**
  - Final node feature matrix $\mathbf{Z} \in \mathbb{R}^{n \times d}$ through multiple layers

- **Target task: semi-supervised node classification**
  - Predict the label of a node among multiple classes
    - e.g., paper category (stat.ML, cs.LG, ...)
  - Small number of training nodes is given

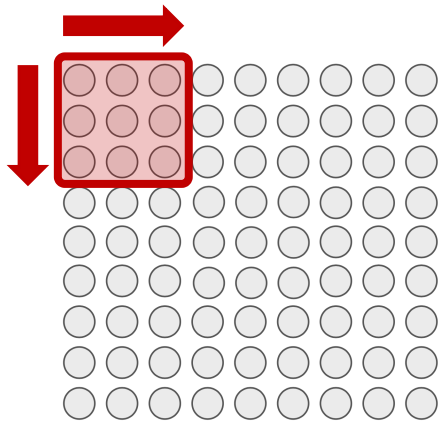
Challenges for Graph Conv.

- **C1. No fixed node ordering**
  - ⇒ Result in different convolution for the same graph
- **C2. Arbitrary number of node neighbors**
  - ⇒ Complex topological structure compared to grid

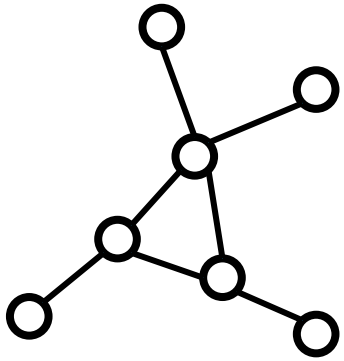
GCN



Different node ordering produces different adjacency matrix



Convolution in a grid structure



How to do convolution in a graph?

Taxonomy and Coverage

Covered!

- Graph Convolutional Networks
  - Node-level
    - Basic GCN
      - GCN (ICLR 2017)
      - GraphSAGE (NIPS 2017)
      - GAT (ICLR 2018)
    - Advanced GCN
      - JK (ICML 2018)
      - APPNP (ICLR 2019)
      - GDC (NeurIPS 2019)
  - Edge-level
  - Graph-level

[Wu et. al. 2020]

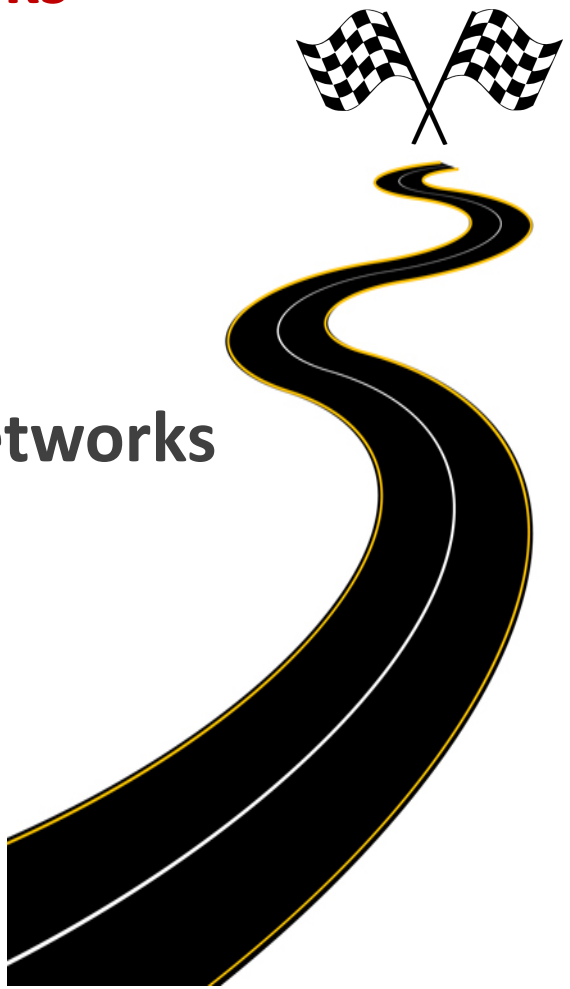
Node-level GCN models

Notations

- **Graphs**
  - $G = (\mathcal{V}, \mathcal{E})$; $\mathcal{V}$ & $\mathcal{E}$: sets of nodes and edges, resp.
  - $\mathcal{N}_i$: set of neighboring nodes of node i
- **Matrices and vectors**
  - $\mathbf{A} \in \mathbb{R}^{n \times n}$: adjacency matrix of graph G
    - $\widetilde{\mathbf{A}} \in \mathbb{R}^{n \times n}$: normalized adjacency matrix (e.g., $\widetilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$)
    - $\mathbf{D} \in \mathbb{R}^{n \times n}$: diagonal degree matrix
  - $\mathbf{X} \in \mathbb{R}^{n \times d}$: input node feature matrix
  - $\mathbf{H} \in \mathbb{R}^{n \times d}$: trained node embedding matrix
    - $\mathbf{h}_i$: node embedding of node i (i.e., i-th row vector of $\mathbf{H}$)
  - $\mathbf{W} \in \mathbb{R}^{d \times d}$: trainable weight matrix
- **Scalars**
  - n & m: # of nodes and edges, resp.; d: feature dimension
  - C: number of classes (or labels)

Roadmap

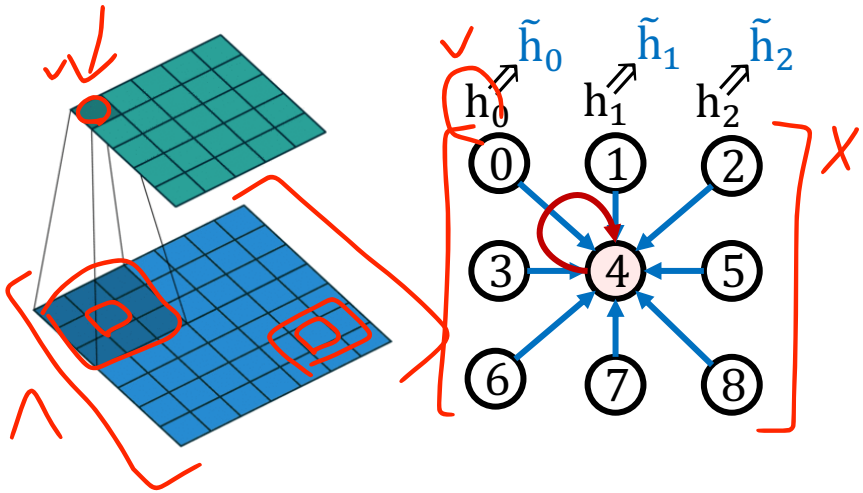
- **Basic Graph Convolutional Networks**
  - **GCN [ICLR 2017]**
  - GraphSAGE [NIPS 2017]
  - GAT [ICLR 2018]
- **Advanced Graph Convolutional Networks**
  - JK [ICML 2018]
  - APPNP [ICLR 2019]
  - GDC [NuerIPS 2019]



GCN [ICLR'17]

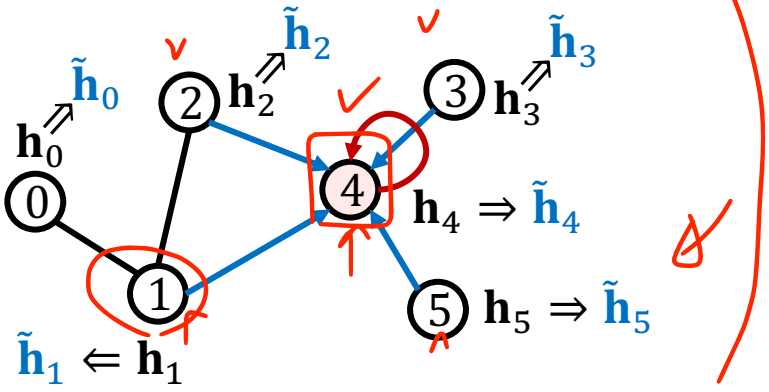
- **Main idea: Let's generalize convolution to graph**
  - Transform input information with trainable weights
  - Propagate the information of neighboring nodes
  - Aggregate the information at each node

Standard convolution



h_i: input activation (scalar) at pixel i
$\tilde{h}_i$: transformed activation at pixel i

Graph convolution



$\mathbf{h}_i$: input feature (vector) at node i
$\tilde{\mathbf{h}}_i$: transformed feature at node i

Update Rule for GCN

- **Graph convolution**
  - Transform input information with trainable weights
  - Propagate the information of neighboring nodes
  - Aggregate the information at each node

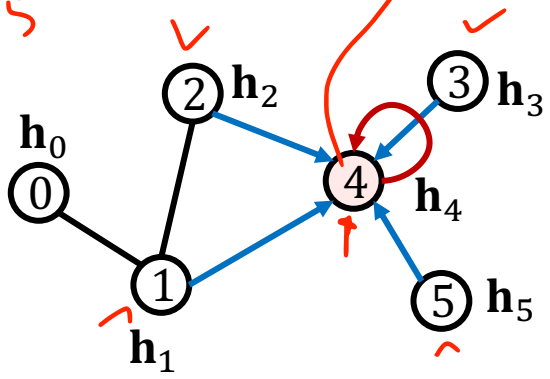
$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

Feature transformation ⇒ shared over all nodes

⇒ invariance to node permutation

Non-linear activator (e.g., ReLU)

Neighborhood aggregation



$\mathbf{h}_i^{(l)} \in \mathbb{R}^{1\times d}$: node feature at layer l

$\mathbf{W}_*^{(l)} \in \mathbb{R}^{d\times d}$: trainable weight matrix

$\mathcal{N}_i$: set of neighboring nodes of node i

c_{ij}: normalization constant

Vectorized From for GCN (1)

- **Update rule for all nodes**

$$\mathbf{H}^{(l+1)} = \sigma\left(\mathbf{H}^{(l)}\mathbf{W}_0^{(l)} + \widetilde{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}_1^{(l)}\right)$$

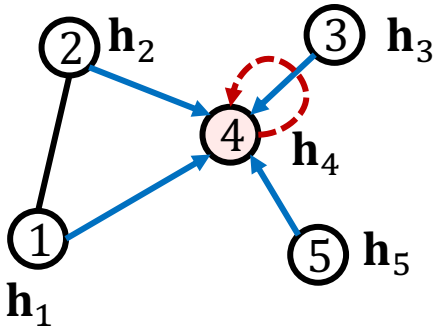
  - $\mathbf{H}^{(l)} = \left[\mathbf{h}_1^{(l)}; \ldots; \mathbf{h}_n^{(l)}\right] \in \mathbb{R}^{n \times d}$: node feature matrix
  - $\widetilde{\mathbf{A}} \in \mathbb{R}^{n \times n}$: (sparse) normalized adj. matrix (i.e., $\widetilde{\mathbf{A}}_{ij} = c_{ij}^{-1}$)

- **Choice of normalization constant**
  - $\widetilde{\mathbf{A}}$ such that largest eigenvalue $\lambda_{\max} = 1$
    - Why? Guarantees $\mathbf{H}^{(l)}$ is not exploded after consecutive graph convolutions
  - In [ICRL'17], $c_{ij} = \sqrt{|\mathcal{N}_i|}\sqrt{|\mathcal{N}_j|} \Rightarrow \widetilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ Graph normalization
    - Why? Derived from 1st-order approx. of graph spectral filters

Vectorized From for GCN (2)

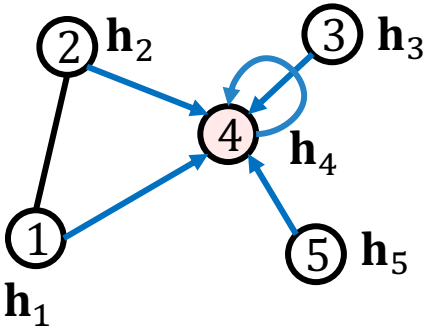
- **Compact update rule for all nodes**
  - Add a self-loop edge explicitly at each node

- $\mathbf{H}^{(l+1)} = \sigma\left(\mathbf{H}^{(l)}\mathbf{W}_0^{(l)} + \tilde{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}_1^{(l)}\right)$
  - With $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$
  - $\mathbf{D}$ is a diagonal degree matrix of the original graph



1-hop propagation

- $\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$
  - With $\hat{\mathbf{A}} = \hat{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})\hat{\mathbf{D}}^{-\frac{1}{2}}$
  - $\hat{\mathbf{D}}$ is a diagonal degree matrix of the self-loop added graph

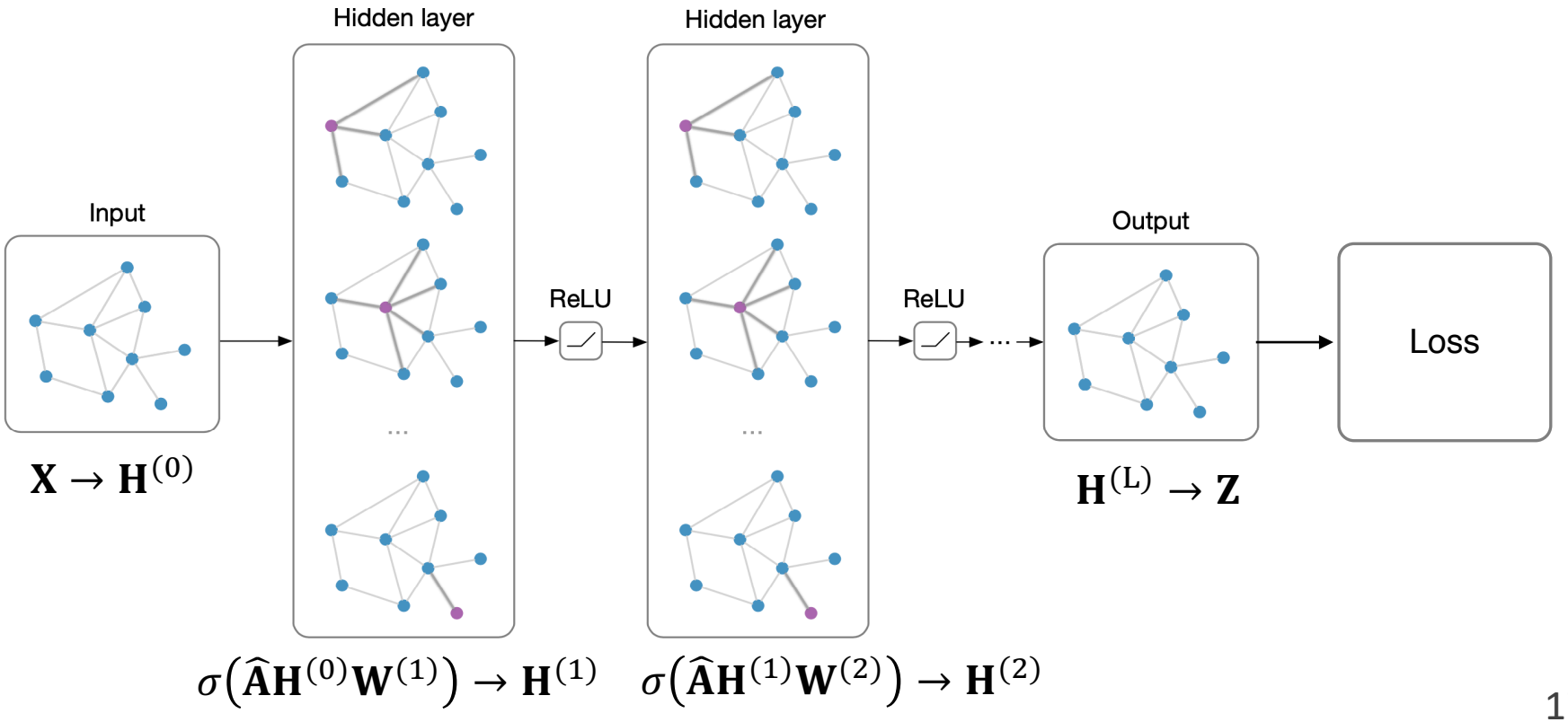


GCN Architecture (L layers)

- **Input:** raw node feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$

  normalized adjacency matrix $\hat{\mathbf{A}} \in \mathbb{R}^{n \times n}$

- **Output:** final node embeddings $\mathbf{Z} \in \mathbb{R}^{n \times C}$



Node Classification with GCN

- **Node classification is multi-class classification**
  - Predict a node's label among multiple classes
  - $\mathcal{V}_L$: set of training nodes having labels
  - Y_{ic}: 1 if node i has label c, or 0 otherwise.
  - C: number of classes
- Given the final node embedding $\mathbf{Z} \in \mathbb{R}^{n \times C}$ of GCN, compute row-wise softmax, i.e, $\mathbf{F} = \text{softmax}(\mathbf{Z})$
- Minimize ***multi-class cross entropy loss***

$$\mathcal{L} = -\sum_{i \in \mathcal{V}_L} \sum_{c=1}^{C} Y_{ic} \log \mathbf{F}_{ic}$$

Experimental Result

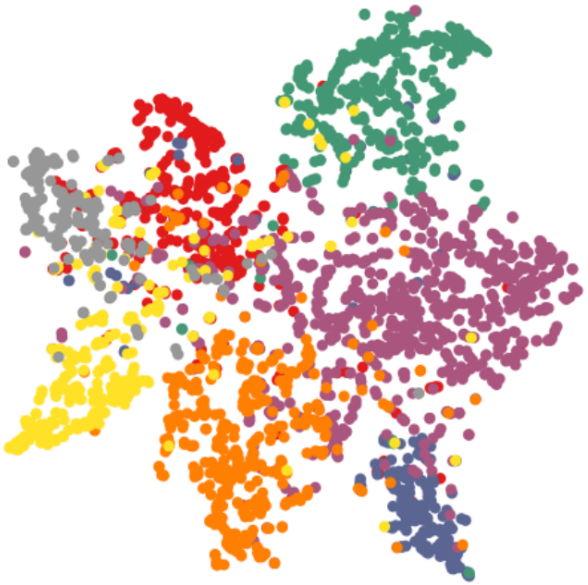
- **Semi-supervised node classification**
  - 2-layer GCN, i.e., $\mathbf{F} = \mathrm{softmax}(\mathrm{GCN}_2(\mathrm{GCN}_1(\mathbf{X})) \in \mathbb{R}^{n \times c}$

Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Method	Citeseer	Cora	Pubmed	NELL
ManiReg	60.1	59.5	70.7	21.8
SemiEmb	59.6	59.0	71.1	26.7
LP	45.3	68.0	63.0	26.5
DeepWalk	43.2	67.2	65.3	58.1
Planetoid*	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)

Learned Representations

- **Visualization of hidden node embeddings**
  - Where a color indicates a class



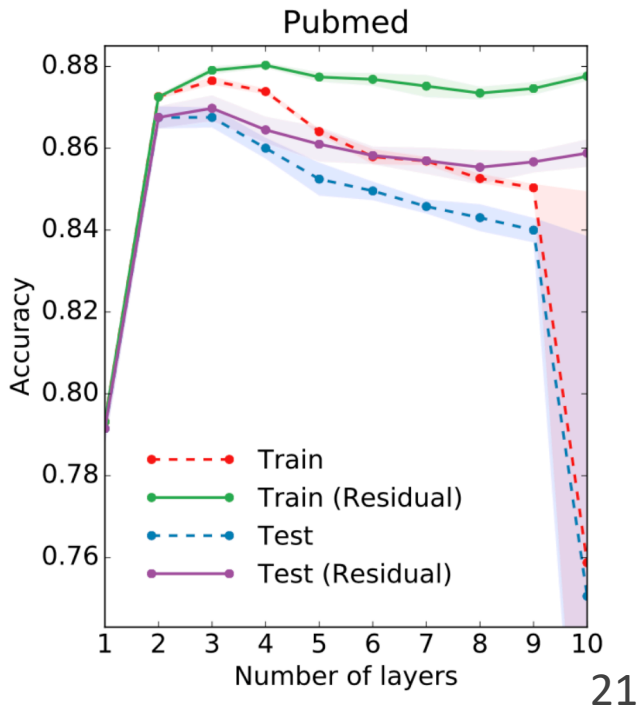
Trained Representations
(t-SNE embedding of hidden layer activations)

Discussion on GCN

- **Pros**
  - **P1.** Make ML tasks on graphs easy
    - Enable us to do end-to-end learning on graphs
  - **P2.** Invariance to node permutation
    - Due to weight sharing over all nodes
  - **P3.** Efficient computation
    - Time $\propto O(m)$ where m is # of edges
    - Model param. $\propto O(d^2)$
      feature dimension
- **Cons**
  - **C1.** Shallow propagation ($2\sim3$ layers)
    - *Small # of neighboring nodes is used*
    - *Residual can be helpful*
  - **C2.** Only indirect support for edge features (limited domain)
  - **C3.** Naïve neighborhood aggregation

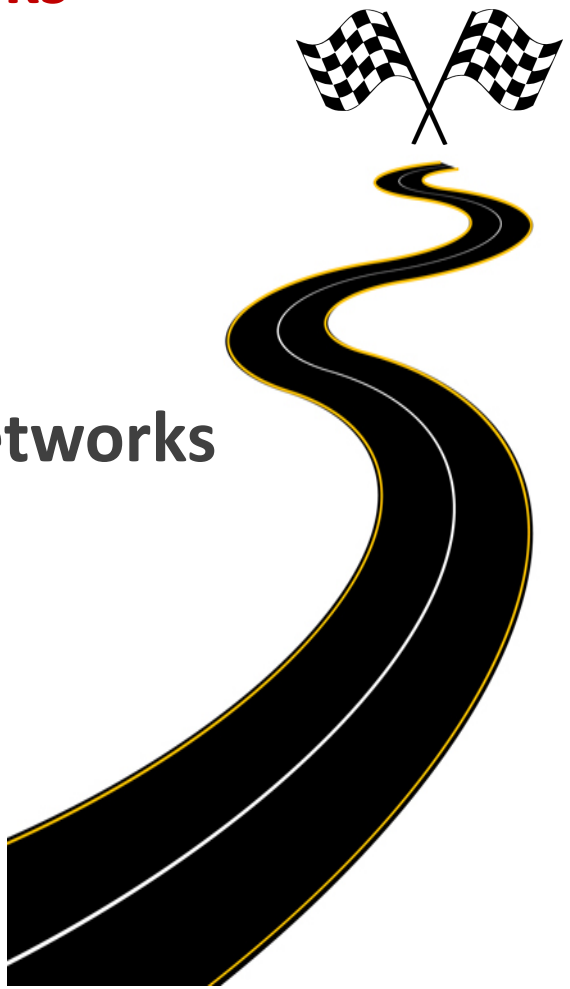
Residual connection ↓

$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right) + \mathbf{H}^{(l)}$$



Roadmap

- **Basic Graph Convolutional Networks**
  - GCN [ICLR 2017]
  - **GraphSAGE [NIPS 2017]**
  - GAT [ICLR 2018]
- **Advanced Graph Convolutional Networks**
  - JK [ICML 2018]
  - APPNP [ICLR 2019]
  - GDC [NuerIPS 2019]



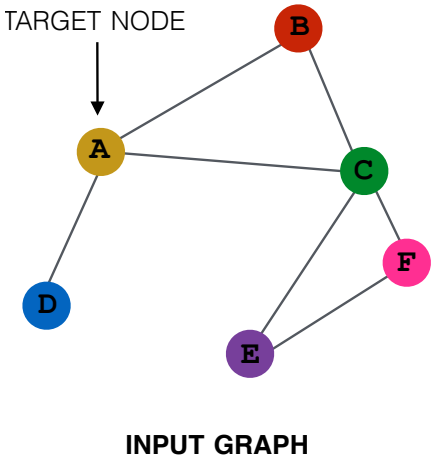
GraphSAGE [NIPS'17]

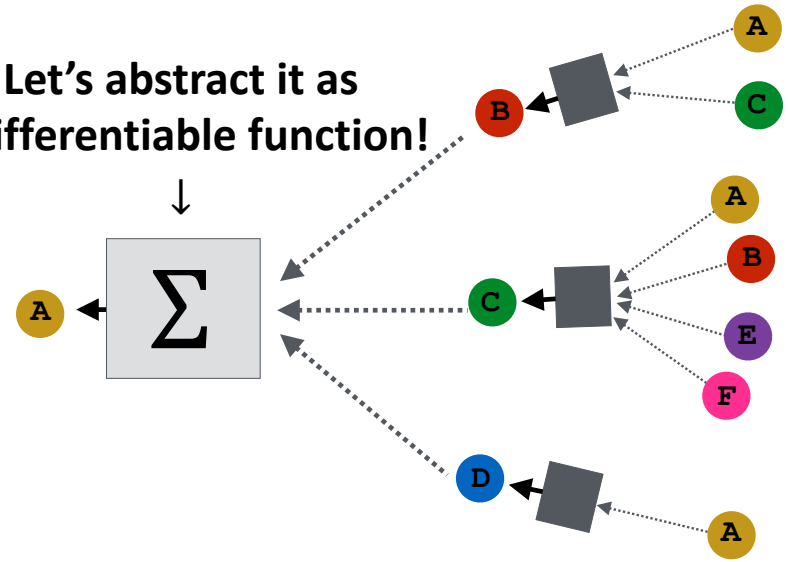
- **Assumption of GCN's neighborhood aggregation**
  - Just simple weighted average of neighbors' features.
    It this the best? Can we do better?

weight

$$\text{GCN's aggregation: } \mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i \cup i} \frac{1}{c_{ij}} \mathbf{h}_j^{(l)} \mathbf{W}^{(l)}\right)$$

- **Main idea: Let's generalize the aggregation!**





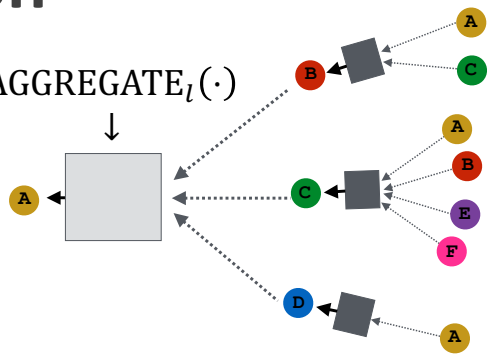
GraphSAGE

- GCN's (basic) neighborhood aggregation

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i \cup i} \frac{1}{c_{ij}} \mathbf{h}_j^{(l)} \mathbf{W}^{(l)}\right)$$

- GraphSAGE's neighborhood aggregation

$$\mathbf{h}_{\mathcal{N}_i}^{(l+1)} = \text{AGGREGATE}_l\left(\left\{\mathbf{h}_j^{(l)} | \forall j \in \mathcal{N}_i\right\}\right)$$



$$\mathbf{h}_i^{(l+1)} = \sigma\left(\text{CONCAT}\left(\mathbf{h}_i^{(l)}, \mathbf{h}_{\mathcal{N}_i}^{(l+1)}\right)\mathbf{W}^{(l)}\right)$$

GraphSAGE's Algorithm

- **Input & output formats are the same as GCN**

$$\mathbf{h}_i^{(0)} \leftarrow \mathbf{x}_i \; \forall i \in \mathcal{V}$$

For $l = 0 \cdots L-1$ **do**

  For $i \in \mathcal{V}$ **do**

$$\mathbf{h}_{\mathcal{N}_i}^{(l+1)} = \text{AGGREGATE}_l\left(\left\{\mathbf{h}_j^{(l)} | \forall j \in \mathcal{N}_i\right\}\right)$$

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\text{CONCAT}\left(\mathbf{h}_i^{(l)}, \mathbf{h}_{\mathcal{N}_i}^{(l+1)}\right)\mathbf{W}^{(l)}\right)$$

$$\mathbf{h}_i^{(l+1)} \leftarrow \mathbf{h}_i^{(l+1)} / ||\mathbf{h}_i^{(l+1)}||_2 \; \forall i \in \mathcal{V}$$

$$\mathbf{z}_i \leftarrow \mathbf{h}_i^{(L)} \; \forall i \in \mathcal{V}$$

⇑ **Explicit normalization is added since graph normalization could be omitted in the aggregation!**

GraphSAGE Variants

- **How to define $\mathrm{AGGREGATE}_l(\cdot)$?**
  - **Mean aggregator**
    - Nearly equivalent to GCN's aggregation

$$\mathrm{MEAN}\left(\left\{\mathbf{h}_j^{(l)} | \forall j \in \mathcal{N}_i\right\}\right) = \frac{1}{|\mathcal{N}_i|}\sum_{j\in\mathcal{N}_i}\mathbf{h}_j^{(l)}$$

  - **LSTM aggregator**
    - Apply LSTM to random permutation π of neighbors

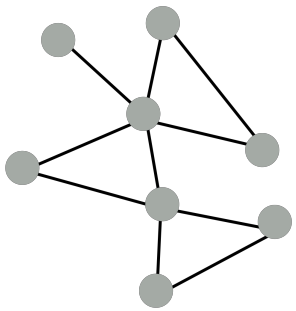
$$\mathrm{LSTM}\left(\left\{\mathbf{h}_j^{(l)} | \forall j \in \pi(\mathcal{N}_i)\right\}\right)$$

  - **Pool aggregator**
    - Apply elementwise max-pooling over neighbors' features

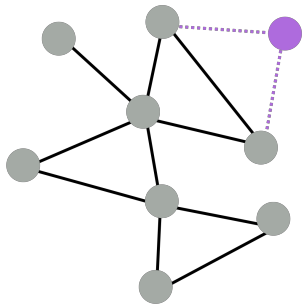
$$\max\left(\left\{\sigma\left(\mathbf{h}_j^{(l)}\mathbf{W}_{\mathrm{pool}}\right) | \forall j \in \pi(\mathcal{N}_i)\right\}\right)$$

Inductive Capability

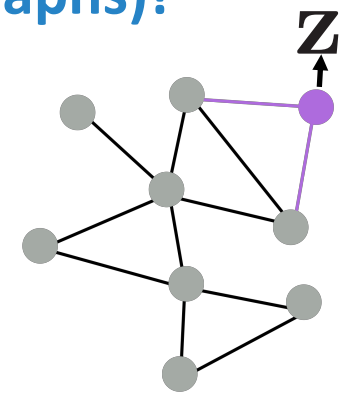
- **Transductive: training algorithm sees the features of all nodes, including test nodes (but, not labels)**
- **Inductive: it doesn't have access to test nodes**
  - Test nodes are dynamically inserted into training graphs
  - Test graphs are disjoint and completely unseen
- Same aggregation parameters are shared for all nodes
  - **Can generalize to unseen nodes (or graphs)!**



train with snapshot



new node arrives



generate embedding for new node

Experimental Result

- **Node classification performance of GraphSAGE**
  - Performed in inductive setting (i.e., test nodes are unseen at training step)
  - *Unsupervised GraphSAGE*: uses an unsupervised loss as in DeepWalk (similar embeddings for connected nodes)
  - *Supervised GraphSAGE*: uses the supervised loss for node classification

	Citation		Reddit		PPI	
Name	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1
Random	0.206	0.206	0.043	0.042	0.396	0.396
Raw features	0.575	0.575	0.585	0.585	0.422	0.422
DeepWalk	0.565	0.565	0.324	0.324	—	—
DeepWalk + features	0.701	0.701	0.691	0.691	—	—
GraphSAGE-GCN	0.742	0.772	**0.908**	0.930	0.465	0.500
GraphSAGE-mean	0.778	0.820	0.897	0.950	0.486	0.598
GraphSAGE-LSTM	0.788	0.832	**0.907**	**0.954**	0.482	**0.612**
GraphSAGE-pool	**0.798**	**0.839**	0.892	0.948	**0.502**	0.600
% gain over feat.	39%	46%	55%	63%	19%	45%

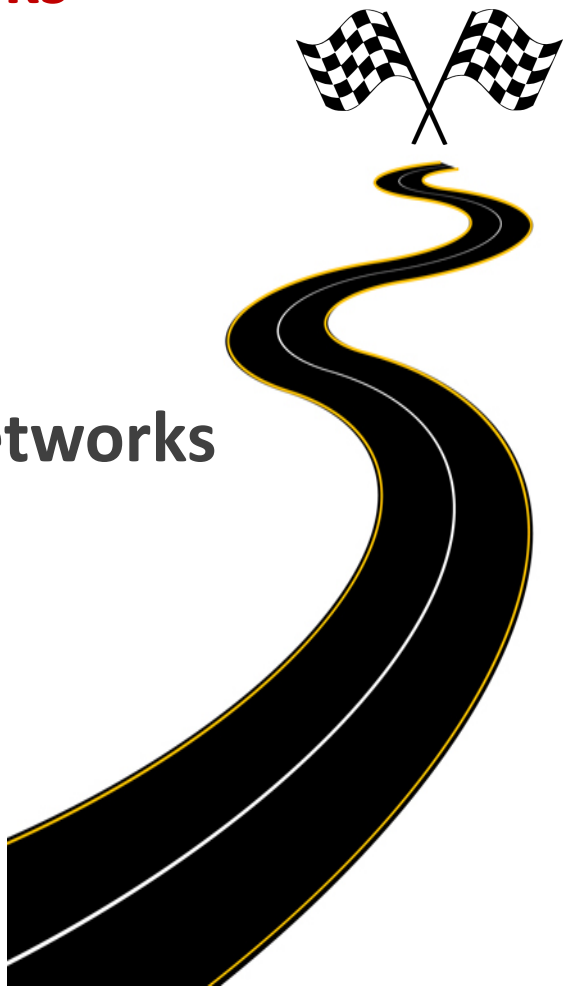
Discussion on GraphSAGE

- **Pros**
  - **P1.** Generalized neighborhood aggregators
    - Mean, LSTM, and max-pool
  - **P2.** Complex aggregator is better than simple one
    - LSTM/max-pool > GCN/Mean
  - **P3.** Effective for inductive node classification
- **Cons**
  - **C1.** Shallow propagation (2 layers)
  - **C2.** Do not know which aggregator is proper
    - Require a lot of experiments to search for it
  - **C3.** Fixed importance of edges for neighborhood feature aggregation

Roadmap

- **Basic Graph Convolutional Networks**
  - GCN [ICLR 2017]
  - GraphSAGE [NIPS 2017]
  - **GAT [ICLR 2018]**

- **Advanced Graph Convolutional Networks**
  - JK [ICML 2018]
  - APPNP [ICLR 2019]
  - GDC [NuerIPS 2019]



GAT [ICLR'17]

- **Assumption of previous models**
  - Weights of edges are fixed by graph normalization
  - Can we learn edge weights more suitable for a task?

Fixed weight

$$\text{GCN's aggregation: } \mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i} \frac{1}{c_{ij}} \mathbf{h}_j^{(l)} \mathbf{W}^{(l)}\right)$$

- **Main idea: let's learn edge weights via attention**
  - **Graph attention networks (GAT)**

Graph Attention

- **Suppose we compute attention on edge** (i, j)
  - Node i has hidden (activation) feature $\mathbf{h}_i$
- **Step 1.** transform features with weight matrix $\mathbf{W}$

$$\tilde{\mathbf{h}}_i = \mathbf{h}_i \mathbf{W}$$

- **Step 2.** compute attention coefficient
  - $\boldsymbol{a} \in \mathbb{R}^{2d}$ is a shared trainable weight vector

$$e_{ij} = \boldsymbol{a}^{\mathsf{T}} \left[\mathrm{CONCAT}(\tilde{\mathbf{h}}_i, \tilde{\mathbf{h}}_j) \right]$$

- **Step 3.** compute attention probability

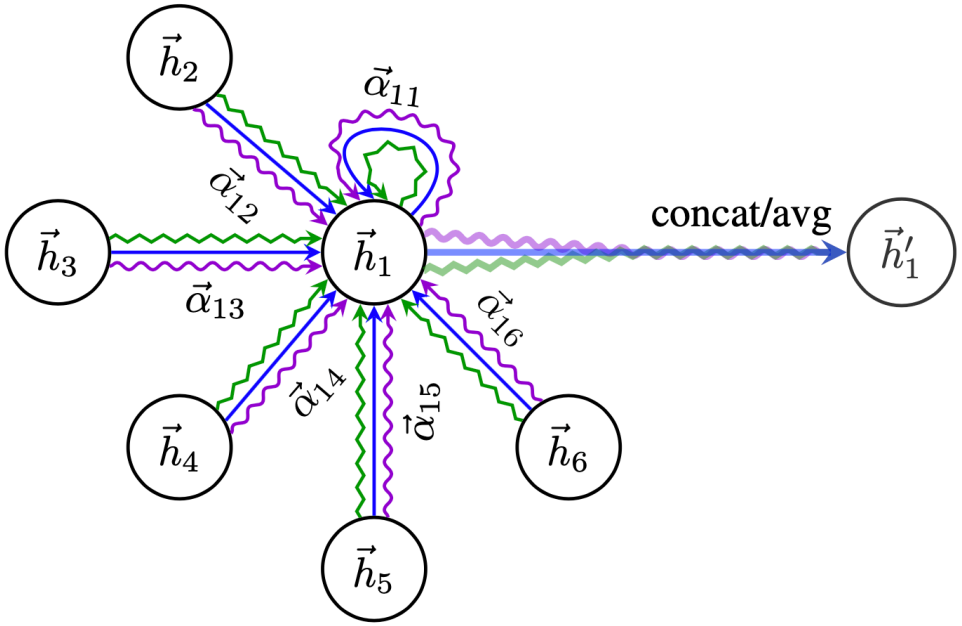
$$\alpha_{ij} = \mathrm{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$$

Multi-head Attention

- **Aim to obtain multiple attention scores per edge**

$$\mathbf{h}_i' = \underset{1\le k\le K}{\text{CONCAT}}\,\sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{h}_j\mathbf{W}_k\right)$$

  - k-th head attention α_{ij}^k is computed with $\mathbf{W}_k$ by graph attention



GAT with 3-head attention

Experimental Result

- **Node classification performance of GAT**
  - Transductive setting (2-layer GAT with 8 heads)

Method	Cora	Citeseer	Pubmed
MLP	55.1%	46.5%	71.4%
ManiReg (Belkin et al., 2006)	59.5%	60.1%	70.7%
SemiEmb (Weston et al., 2012)	59.0%	59.6%	71.7%
LP (Zhu et al., 2003)	68.0%	45.3%	63.0%
DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
ICA (Lu & Getoor, 2003)	75.1%	69.1%	73.9%
Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%
Chebyshev (Defferrard et al., 2016)	81.2%	69.8%	74.4%
GCN (Kipf & Welling, 2017)	81.5%	70.3%	**79.0%**
MoNet (Monti et al., 2016)	81.7 ± 0.5%	—	78.8 ± 0.3%
GCN-64*	81.4 ± 0.5%	70.9 ± 0.5%	**79.0** ± 0.3%
GAT (ours)	**83.0** ± 0.7%	**72.5** ± 0.7%	**79.0** ± 0.3%

Experimental Result

- **Node classification performance of GAT**
  - Inductive setting (3-layer GAT with (4,6) heads)

Method	PPI
Random	0.396
MLP	0.422
GraphSAGE-GCN (Hamilton et al., 2017)	0.500
GraphSAGE-mean (Hamilton et al., 2017)	0.598
GraphSAGE-LSTM (Hamilton et al., 2017)	0.612
GraphSAGE-pool (Hamilton et al. 2017)	0.600
GraphSAGE*	0.768
Const-GAT (ours)	0.934 ± 0.006
GAT (ours)	**0.973** ± 0.002

Discussion on GAT

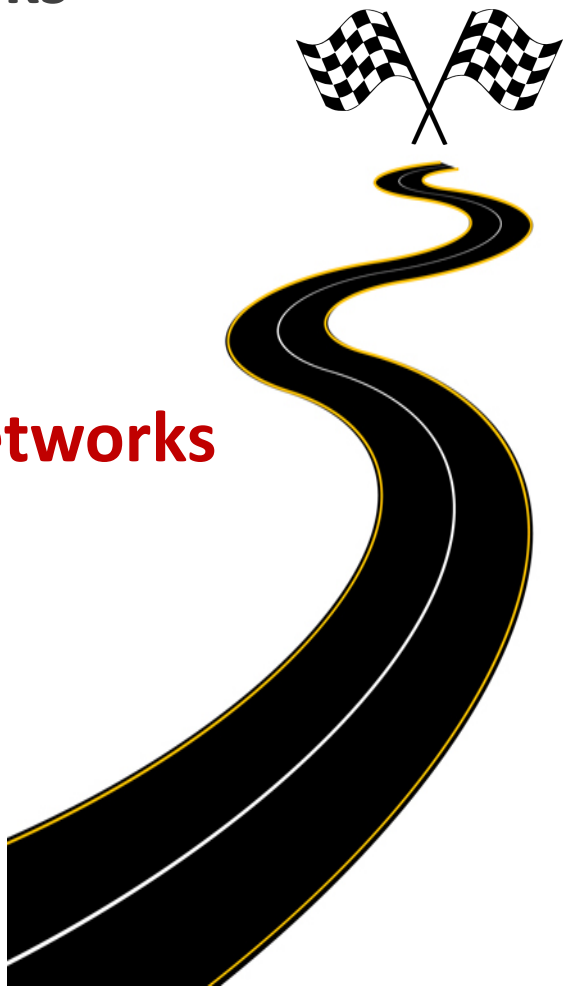
- **Pros**
  - **P1.** Increased learning power via multi-head attention
  - **P2.** Effective for both transductive and inductive settings

- **Cons**
  - **C1.** Shallow propagation ($2 \sim 3$ layers)
  - **C2.** Larger model complexity
  - **C3.** Slower than GCN or GraphSAGE
    - Due to multi-head attention

Roadmap

- **Basic Graph Convolutional Networks**
  - GCN [ICLR 2017]
  - GraphSAGE [NIPS 2017]
  - GAT [ICLR 2018]

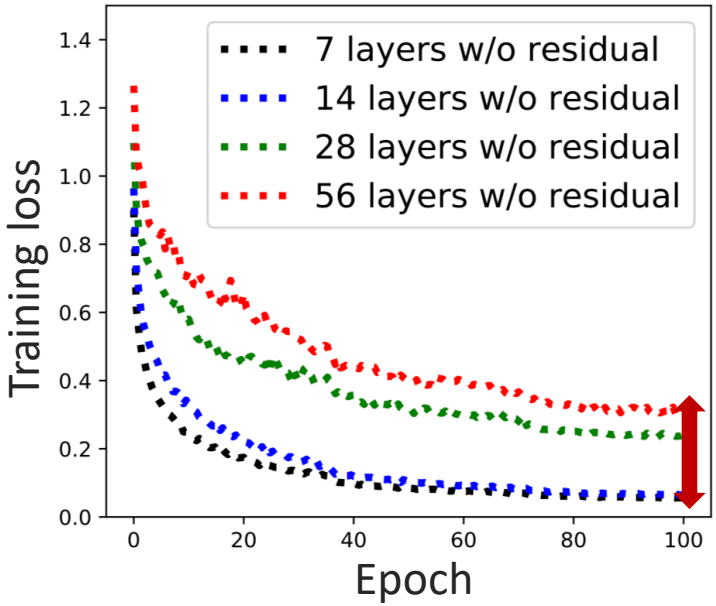
- **Advanced Graph Convolutional Networks**
  - **JK [ICML 2018]**
  - APPNP [ICLR 2019]
  - GDC [NuerIPS 2019]



Limitation of Basic GCNs

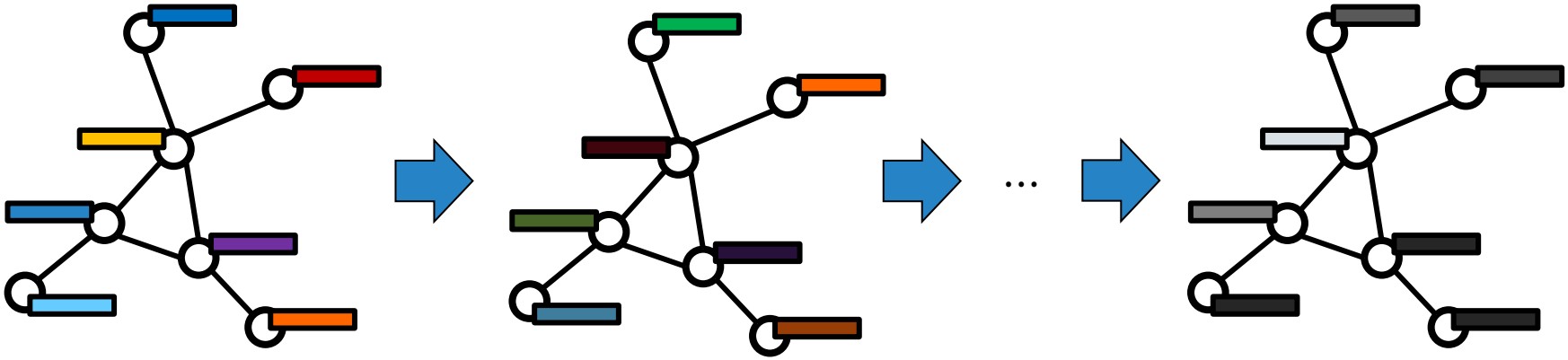
- Basic GCNs shows the best performance **when their depth is shallow**
  - GCN [ICLR'17]: $\mathbf{2}$ layers
  - GraphSAGE [NIPS'17]: $\mathbf{2}$ layers
  - GAT [ICLR'18]: $\mathbf{2}\sim\mathbf{3}$ layers
  - With two GCN layers, only 2-hop neighbors are considered
- **Deep layers degrade the GCNs' performance**
  - Learning power of GCN is limited
  - Why?

[Le et al., ICCV'19]



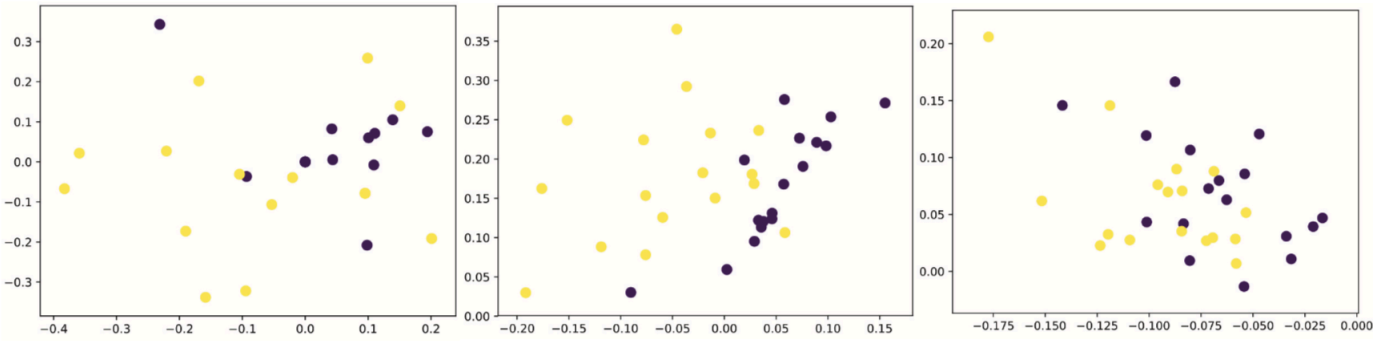
Over-smoothing Problem (1)

- **Node representations are over-smoothed as the layer depth increases**
  - ⇒ Become undistinguishable as the depth increases
- **Main cause: averaging aggregation**
  - It loses its focus on the local neighborhood

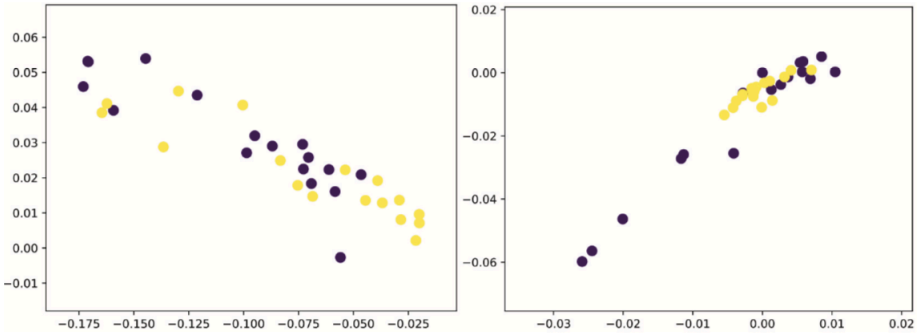


Over-smoothing Problem (2)

- **Visualization on node representation of GCN**
  - Zachary's karate club network (2 labels)
  - Point = node embedding t-SNE projected onto 2D space



(a) 1-layer (b) 2-layer (c) 3-layer



(d) 4-layer (e) 5-layer

⇒ Become similar to each other

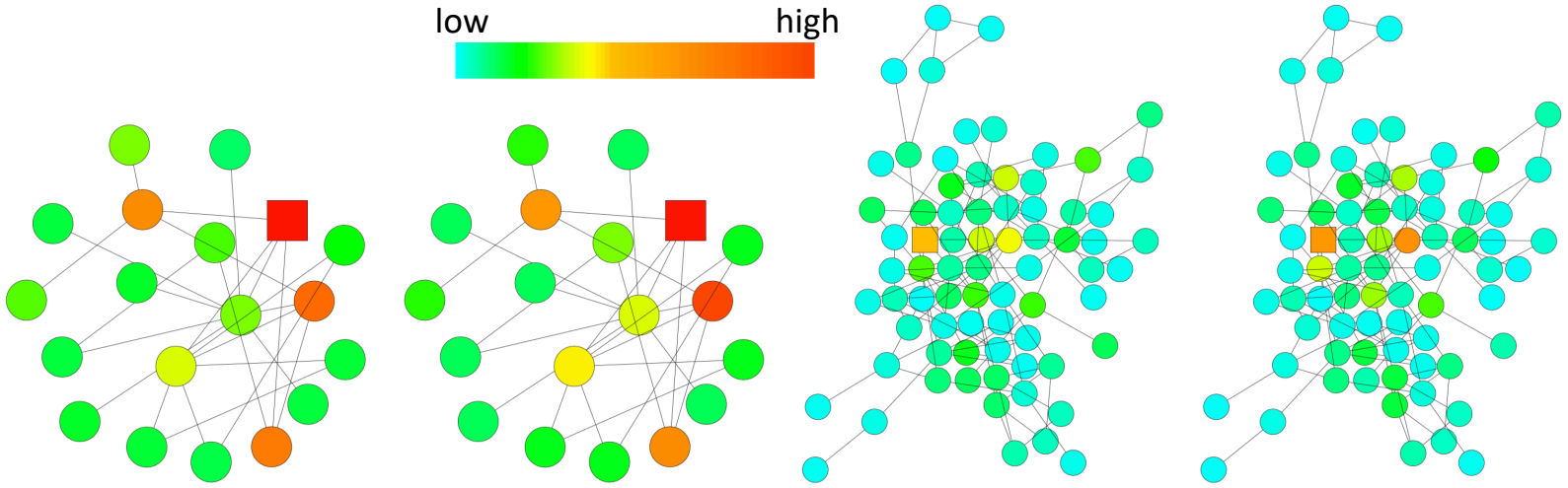
[Li et al., AAAI'18]

Analysis on GCN (1)

- **Analyzed the behavior of neighborhood aggregation**
  - Why does deep GCN produce over-smoothen embeddings?
- **Influence distribution of a node**
  - $\forall y \in \mathcal{V}$, ***influence score*** of node y on node x
    - How much a change in $\mathbf{h}_y^{(0)}$ affects $\mathbf{h}_x^{(k)}$ in the last layer
- **Random walk distribution of a node**
  - $\forall y \in \mathcal{V}$, probability that a surfer visits node y through k steps starting at node x

Analysis on GCN (2)

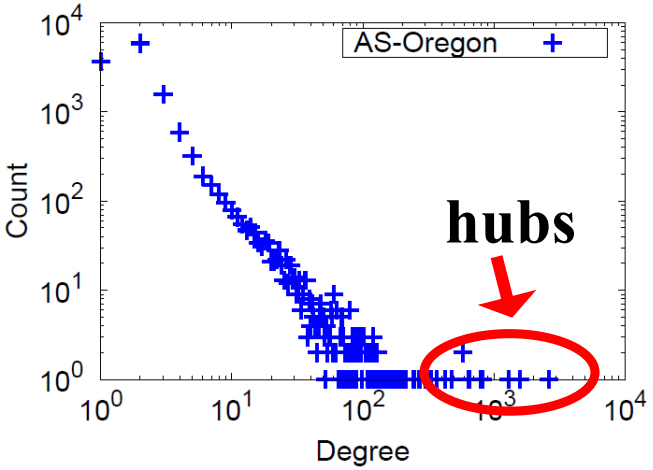
- **Influence distribution** of node x in k-layer GCN
  $\Leftrightarrow$ k-step **random walk distribution** of the node
  - Example where *a square node* is target node x
  - **GCN's feature modeling can be described by the random walk in graphs!**



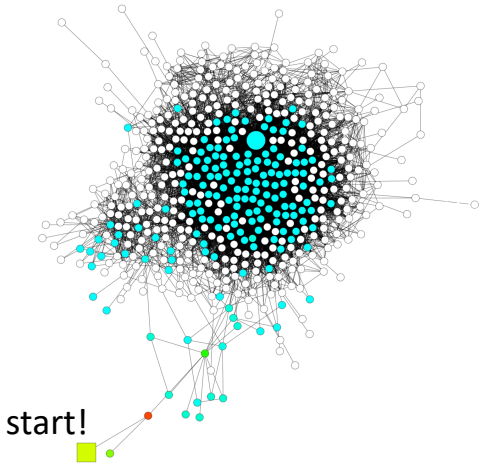
(a) 2 layer GCN (b) 2 step r.w. (c) 4 layer GCN (d) 4 step r.w.

Why Over-smoothing?

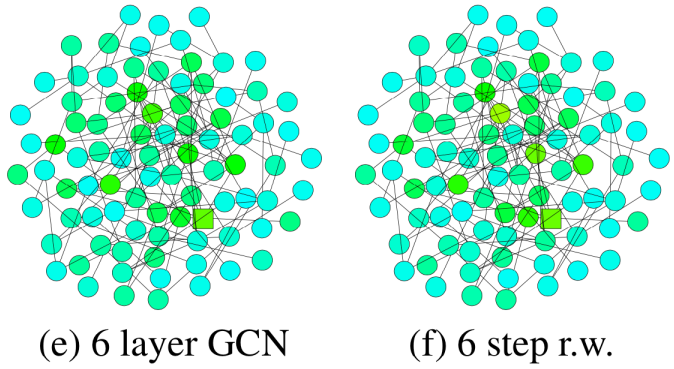
- **Real-world graphs are scale-free!**
  - There are a few ***hub*** nodes in real-world graphs
    - Disconnected nodes are reachable by a small number of steps (via hubs)
    - Random walks converge rapidly to an almost-uniform distribution
  - Small k is enough for ***smoothing RW distribution***!
  - k-layer GCN is closely related to k-step random walk
    - Node representations will be representative of the **global graph (*degree*)**



Power-law degree distribution



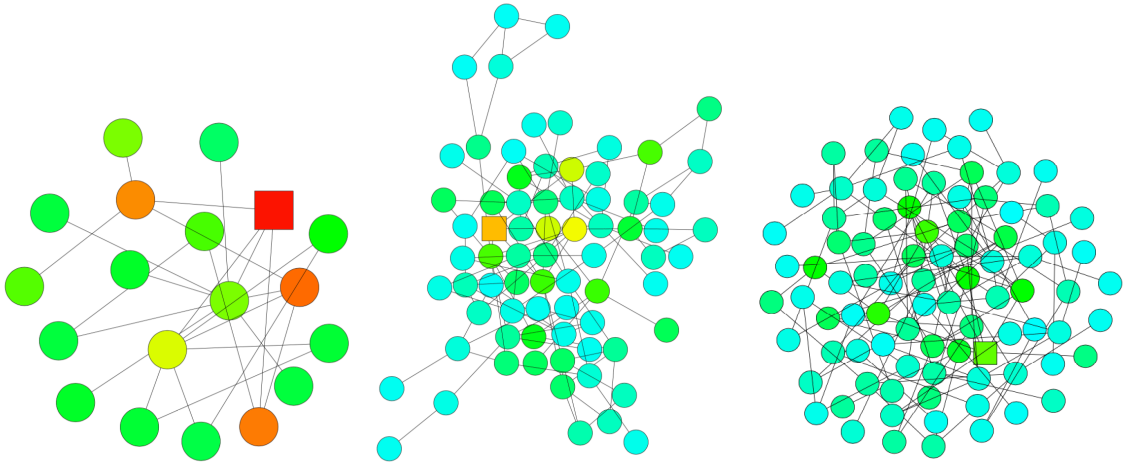
6 step random walks



Over-smoothness

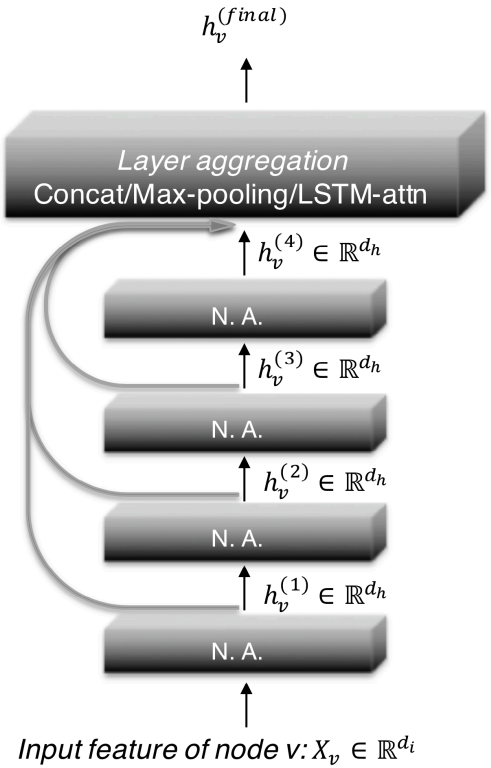
JK [ICML'18]

- **Propose JK (Jumping Knowledge) networks**
  - **Observation.** Small k will capture local structure while large k will capture global structure
  - **Main idea:** **Let's adaptively mix the result of each neighborhood aggregation!**



(a) 2 layer GCN (c) 4 layer GCN (e) 6 layer GCN

Local ⟷ Global



Experimental Result

- **Node classification performance of JK**
  - *Transductive setting*

Model	Citeseer	Model	Cora
GCN (2)	77.3 (1.3)	GCN (2)	88.2 (0.7)
GAT (2)	76.2 (0.8)	GAT (3)	87.7 (0.3)
JK-MaxPool (1)	77.7 (0.5)	JK-Maxpool (6)	**89.6** (0.5)
JK-Concat (1)	**78.3** (0.8)	JK-Concat (6)	89.1 (1.1)
JK-LSTM (2)	74.7 (0.9)	JK-LSTM (1)	85.8 (1.0)

  - *Inductive setting*

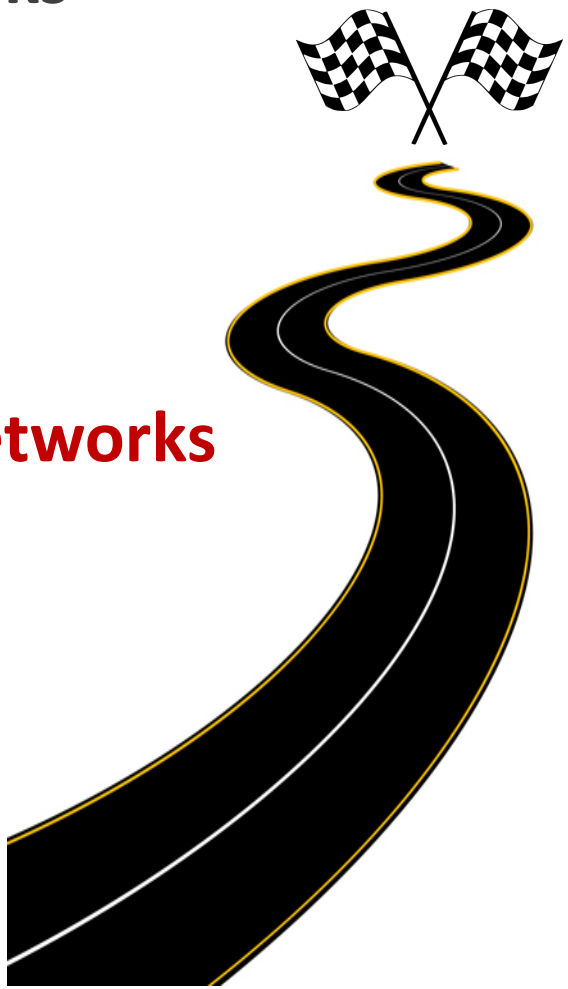
Model	PPI
MLP	0.422
GAT	0.968 (0.002)
JK-Concat (2)	0.959 (0.003)
JK-LSTM (3)	0.969 (0.006)
JK-Dense-Concat (2)*	0.956 (0.004)
JK-Dense-LSTM (2)*	**0.976** (0.007)

Discussion on JK

- **Pros**
  - **P1.** Theoretical analysis on GCN's over-smoothing
    - Connected with random walk in graphs
  - **P2.** Capture various structural information
    - From local to global, via aggregation at the last layer
- **Cons**
  - **C1.** Do not know which aggregator is proper
    - Require a lot of experiments to search for it
  - **C2.** Number of layers is still limited
    - Up to 6 layers, i.e., 6-hop neighbors are considered
    - Hard to know which # of layers is suitable (needs experiments)

Roadmap

- **Basic Graph Convolutional Networks**
  - GCN [ICLR 2017]
  - GraphSAGE [NIPS 2017]
  - GAT [ICLR 2018]
- **Advanced Graph Convolutional Networks**
  - JK [ICML 2018]
  - **APPNP [ICLR 2019]**
  - GDC [NuerIPS 2019]



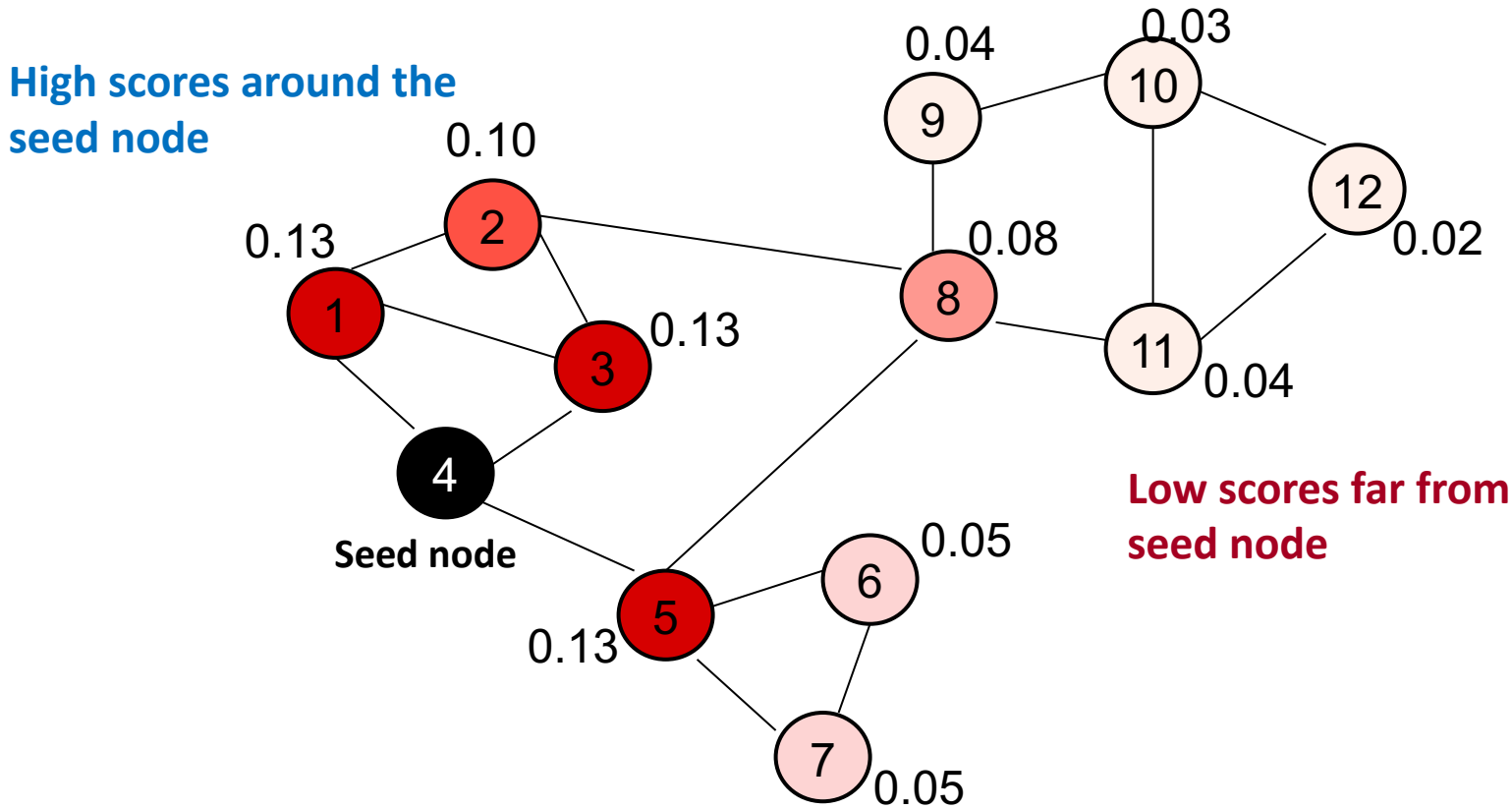
APPNP [ICLR'19]

- **GCN's propagation ∝ simple random walk in graphs**
  - Produce probability distribution representing ***global properties*** such as node degree
    - e.g., PageRank (similar to simple RW) obtains *global importance* over all nodes
  - **Personalized PageRank** is a random walk model for computing *node scores personalized to given seed nodes*

- **Main idea**: Let's propagate information like **Personalized PageRank**
  - To preserve local information during propagation

Personalized PageRank (1)

- **Personalized node-to-node scores**
  - **Random walk**: randomly moves to a neighboring node
  - **Restart**: goes back to the seed node and restart



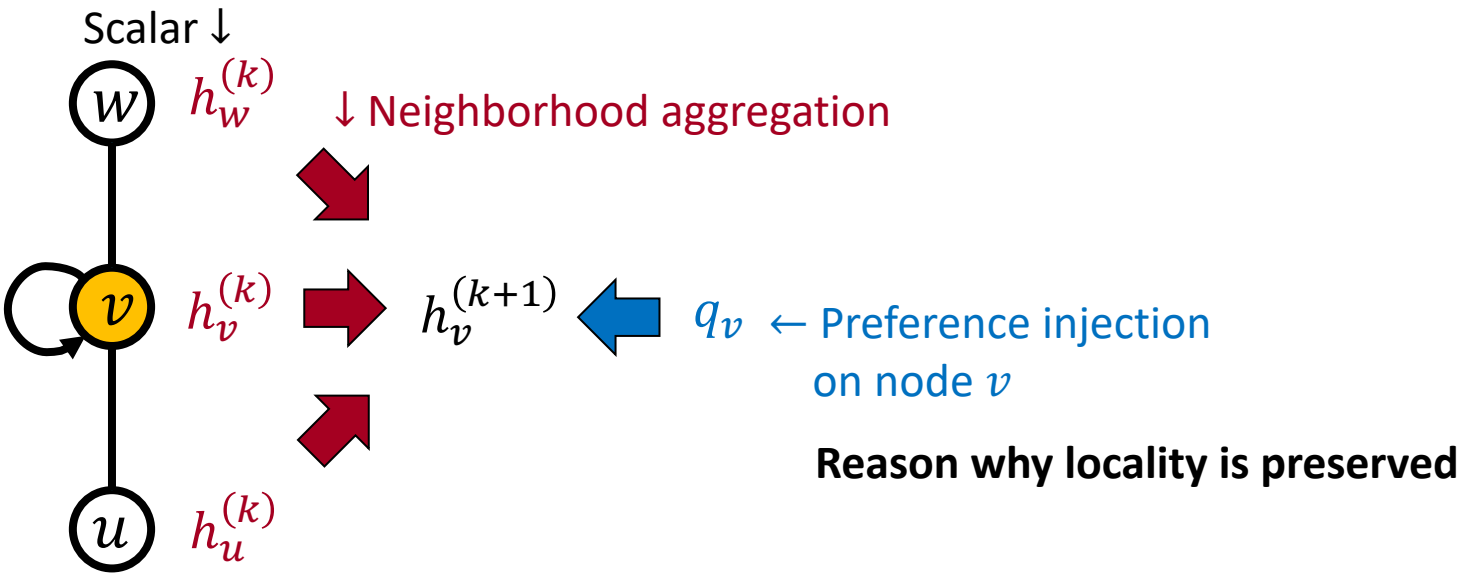
Personalized PageRank (2)

- **Interpretation of PPR equation**

$\boldsymbol{q}$ is a preference vector

c is a preference prob.

$$\underbrace{\mathbf{h}^{(k+1)}}_{\text{Scores at } k+1\text{-th step}} = \underbrace{(1-c)\tilde{\mathbf{A}}\mathbf{h}^{(k)}}_{\text{Neighborhood aggregation}} + \underbrace{c\mathbf{q}}_{\text{Preference injection}}$$



Reason why locality is preserved

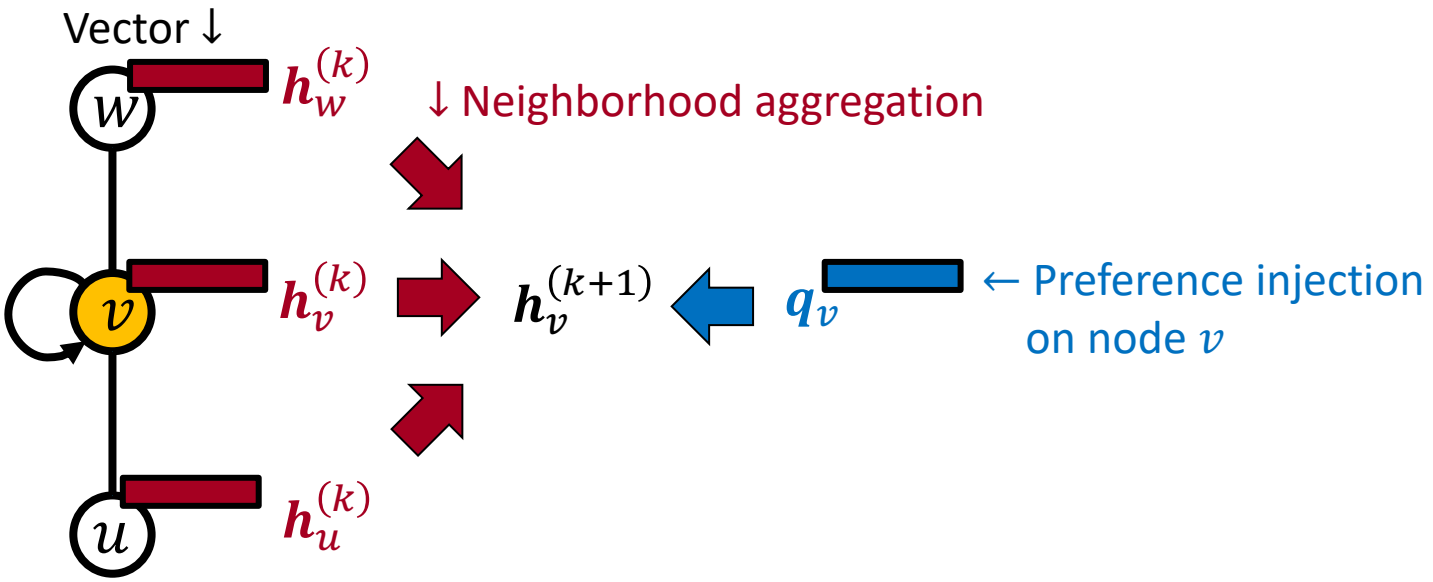
APPNP [ICLR'19]

- **Personalized Propagation like PPR**

$$\boldsymbol{Q} = f_\theta(\boldsymbol{X}) = \boldsymbol{XW}$$

Input feature matrix

$$\underbrace{\mathbf{H}^{(k+1)}}_{\text{Feature matrix at } k+1\text{-th step}} = \underbrace{(1-c)\tilde{\mathbf{A}}\mathbf{H}^{(k)}}_{\text{Neighborhood aggregation}} + \underbrace{c\mathbf{Q}}_{\text{Preference injection}}$$



Overall Procedure of APPNP

- **Step 1. Transform features with learnable weights**
  - Input feature matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$
  - Learnable weight matrix $\boldsymbol{W} \in \mathbb{R}^{d \times d}$

$$\mathbf{H}^{(0)} \leftarrow \mathbf{Q} \leftarrow \mathbf{XW}$$

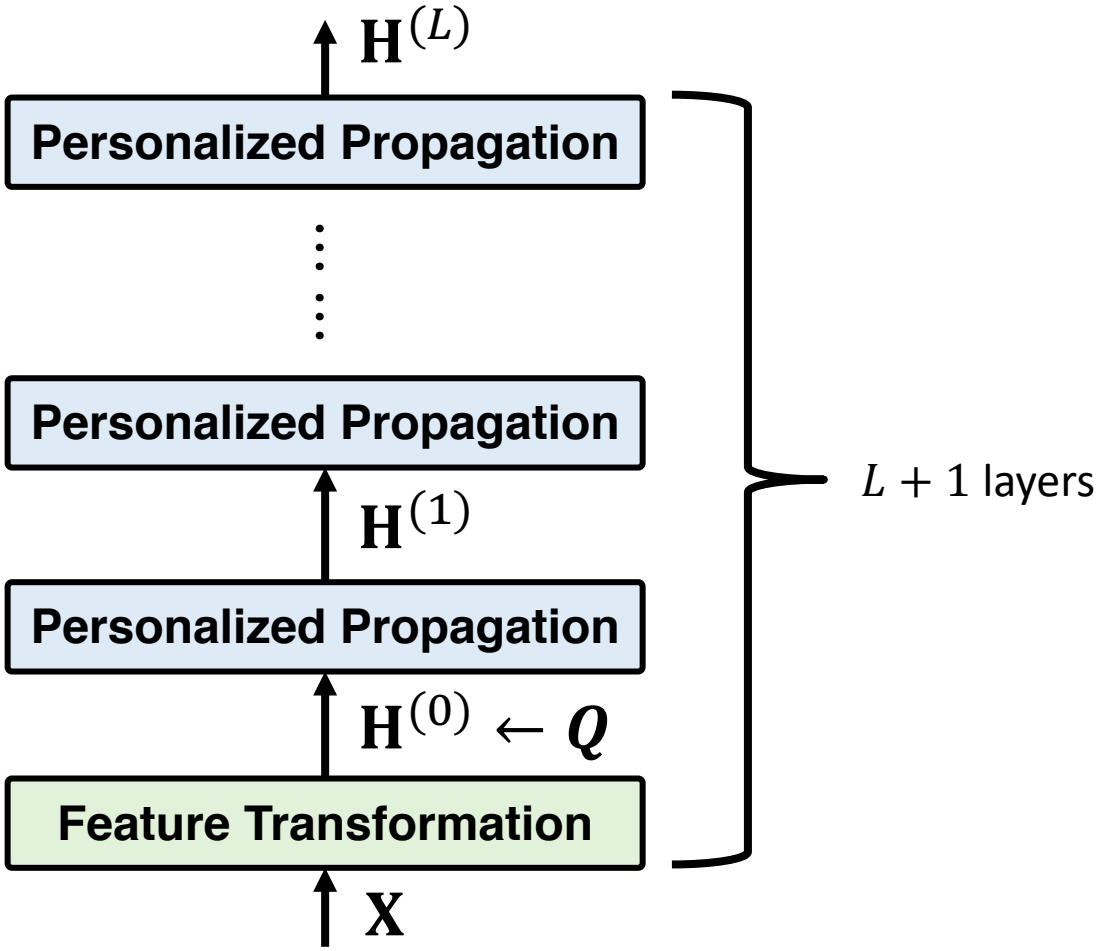
- **Step 2. Repeat personalized propagation L times**

$$\mathbf{H}^{(k+1)} = (1-c)\widetilde{\mathbf{A}}\mathbf{H}^{(k)} + c\mathbf{Q}$$

- **Return $\mathbf{H}^{(L)}$ as the final node representation**

Network Architecture of APPNP

- APPNP consists of a feature transform layer and multiple propagation layers



Comparison

- **Between GCN and APPNP**

- **Simple GCN**
  - **Step 1.** Transform features
    - $\widetilde{\mathbf{H}}^{(k)} = \mathbf{H}^{(k)}\mathbf{W}^{(k)}$
  - **Step 2.** Neighborhood aggregation
    - $\mathbf{H}^{(k+1)} = \sigma\left(\widetilde{\mathbf{A}}\widetilde{\mathbf{H}}^{(k)}\right)$

Repeat L times

- Transform & propagation are **coupled**
- Needs L weight matrices
- **Shallow transform**
- **Shallow propagation** ($L = 2 \sim 3$)

- **APPNP [ICLR'19]**
  - **Step 1.** Transform features
    - $\mathbf{H}^{(0)} = \mathbf{Q} = \mathbf{X}\mathbf{W}^{(k)}$
  - **Step 2.** Personalized propagation
    - $\mathbf{H}^{(k+1)} = (1-c)\widetilde{\mathbf{A}}\mathbf{H}^{(k)} + c\mathbf{Q}$

Repeat L times

↔ • Transform & propagation are **split**

↔ • Needs one weight matrix

↔ • **Shallow transform**

↔ • **Deep personalized propagation** ($L \geq 10$)

Analysis of APPNP

Details

- **Exact version: PPNP**
  - Infinite personalized propagation converges to

$$\lim_{k\to\infty} \mathbf{H}^{(k)} = \lim_{k\to\infty} \left(\sum_{i=0}^{k-1} (1-c)^i \, \tilde{\mathbf{A}}^i \right) = c\underbrace{\left(\mathbf{I} - (1-c)\tilde{\mathbf{A}}\right)^{-1}}_{\text{PPR score matrix}} \mathbf{Q}$$

  - PPNP works on only small graphs due to scalability
  - Interpreted as weighted aggregation with PPR scores and initial prediction $\mathbf{Q}$

Exponentially decaying

$$\mathbf{H}^{(k)} = \mathbf{Q} + (1-\mathrm{c})\tilde{\mathbf{A}}\mathbf{Q} + \cdots + (1-\mathrm{c})^k \tilde{\mathbf{A}}^k \mathbf{Q}$$

Local ⟵ ⟶ Global

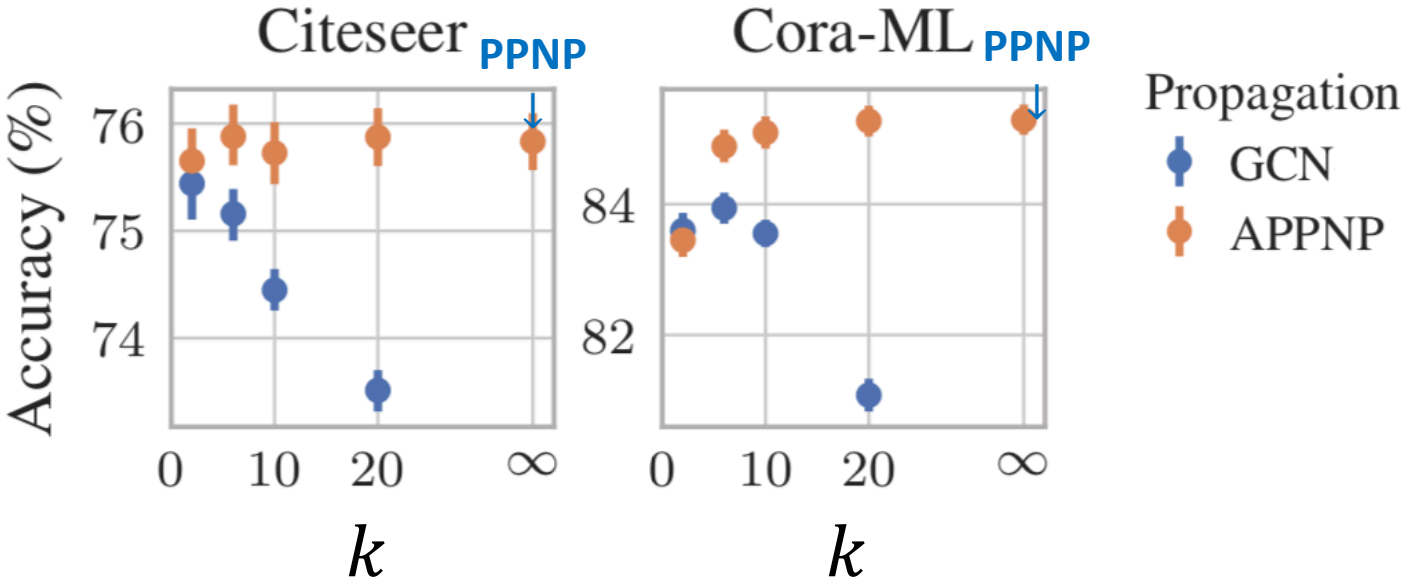
Experimental Results (1)

- **Node classification performance of APPNP**
  - Transductive setting
  - **PPNP**: the exact method computing matrix inversion ⇒ o.o.m. for large graphs

Model	Citeseer	Cora-ML	PubMed	MS Academic
V. GCN	73.51 ± 0.48	82.30 ± 0.34	77.65 ± 0.40	91.65 ± 0.09
GCN	75.40 ± 0.30	83.41 ± 0.39	78.68 ± 0.38	92.10 ± 0.08
N-GCN	74.25 ± 0.40	82.25 ± 0.30	77.43 ± 0.42	92.86 ± 0.11
GAT	75.39 ± 0.27	84.37 ± 0.24	77.76 ± 0.44	91.22 ± 0.07
JK	73.03 ± 0.47	82.69 ± 0.35	77.88 ± 0.38	91.71 ± 0.10
Bt. FP	73.55 ± 0.57	80.84 ± 0.97	72.94 ± 1.00	91.61 ± 0.24
PPNP*	**75.83 ± 0.27**	**85.29 ± 0.25**	Out-of-memory	
APPNP	75.73 ± 0.30	85.09 ± 0.25	**79.73 ± 0.31**	**93.27 ± 0.08**

Experimental Results (2)

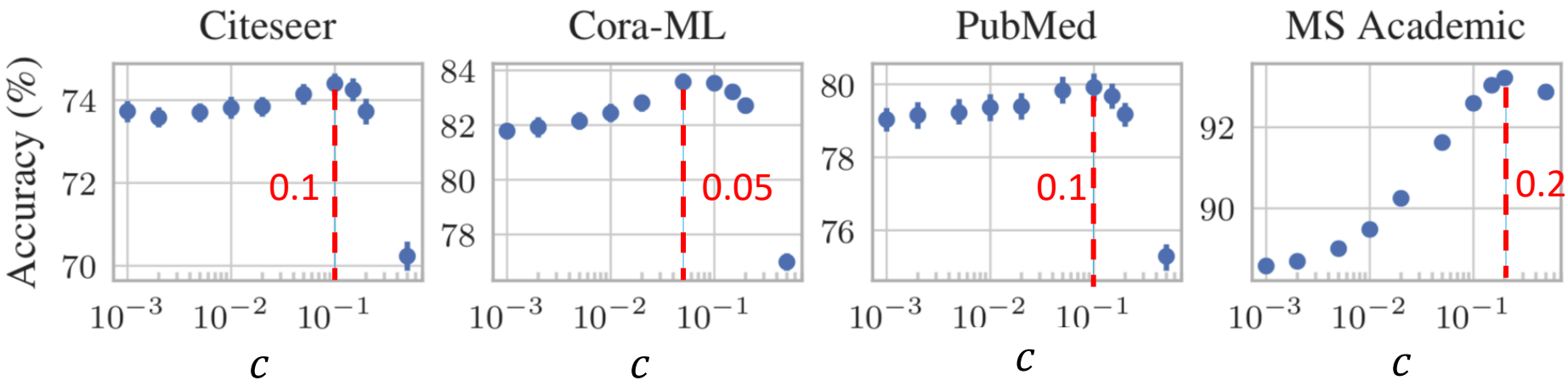
- **Effect of propagation steps k**



  - As k increases, APPNP's performance becomes **improved**, and converges to **PPNP!**
  - GCN's performance becomes **bad!**

Experimental Results (3)

- **Effect of preference probability c**



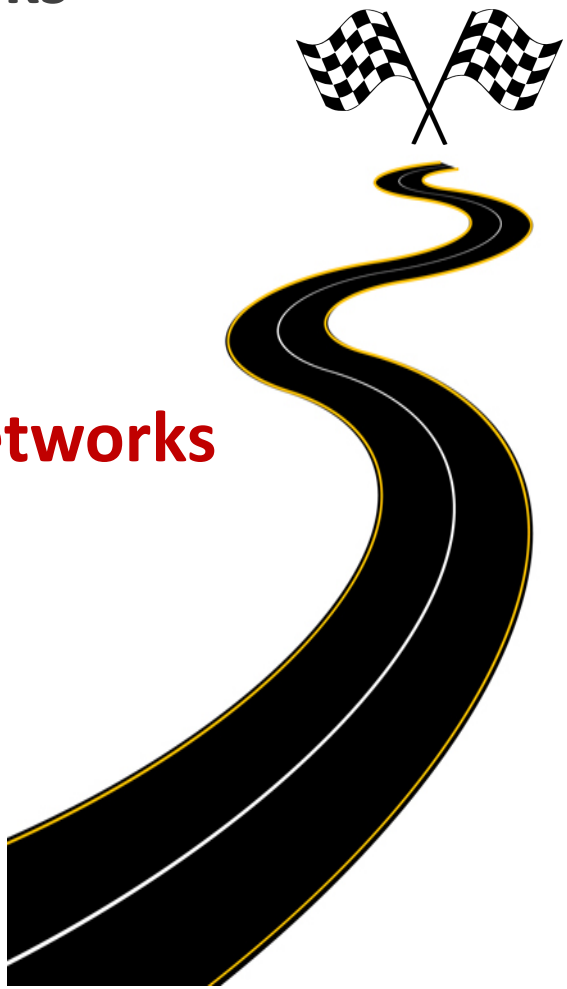
- Within [0.05, 0.2], the proposed model shows the best predictive performance
- However, *it depends on datasets, and should be manually tuned*

Discussion on APPNP

- **Pros**
  - **P1.** Carefully consider features from more nodes via PPR
    - Theoretically, it can consider all of nodes in the graph
  - **P2.** Split feature transformation and propagation
    - Increase the flexibility of feature propagation
  - **P3.** APPNP is efficient and very lightweight model
- **Cons**
  - **C1.** Additional hyperparameters to be tuned
    - e.g., # of propagation steps & preference probability c
  - **C2.** PPNP suffers from scalability issue
  - **C3.** Shallow transform with a trainable weight matrix
    - Its performance will be limited in complicated ML tasks such as graph clustering

Roadmap

- **Basic Graph Convolutional Networks**
  - GCN [ICLR 2017]
  - GraphSAGE [NIPS 2017]
  - GAT [ICLR 2018]
- **Advanced Graph Convolutional Networks**
  - JK [ICML 2018]
  - APPNP [ICLR 2019]
  - **GDC [NuerIPS 2019]**



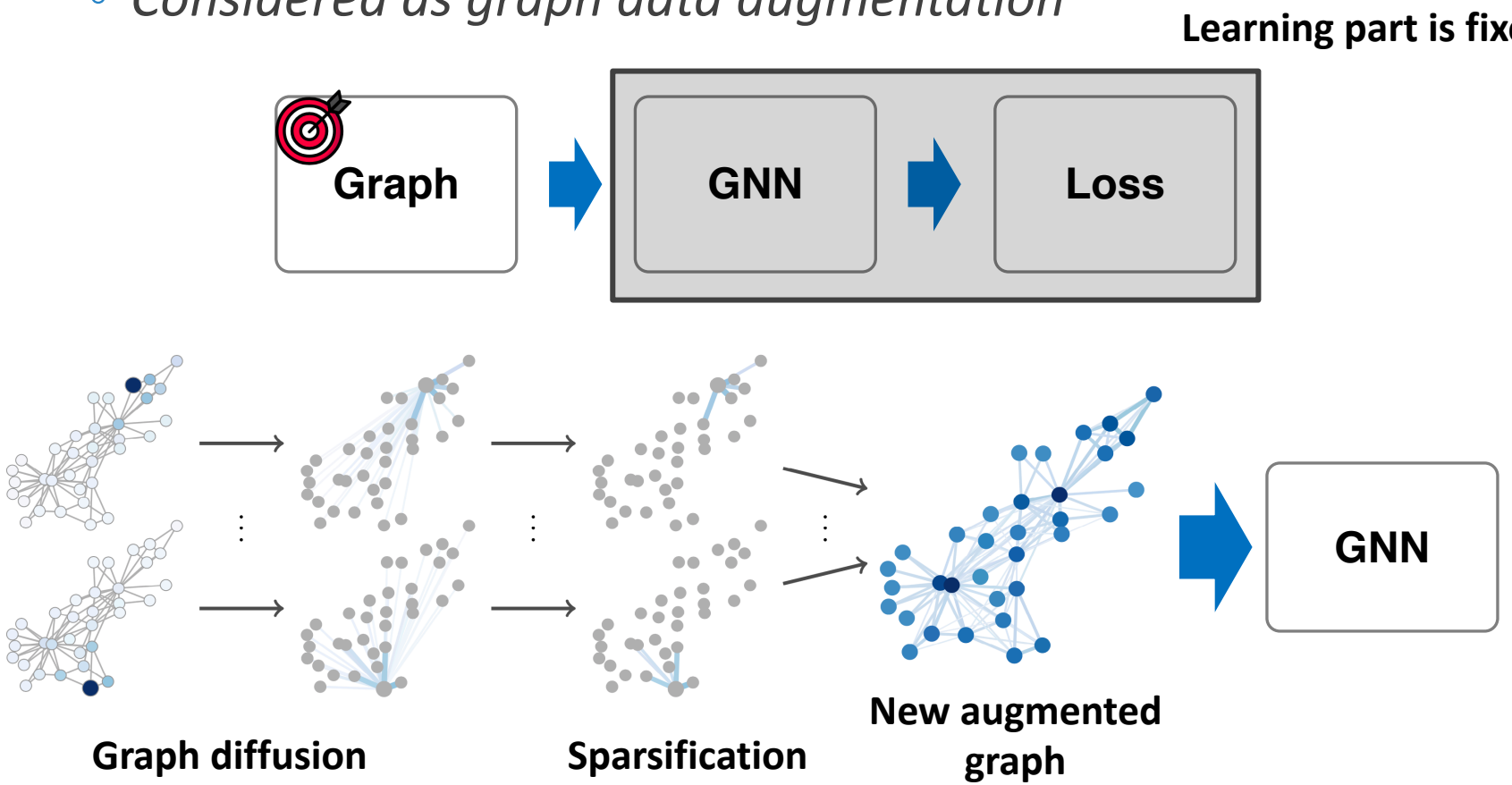
GDC [NeurIPS'19]

- **Assumptions behind previous approaches**
  - Only pass features between *direct neighboring nodes* in each step
    - *Why should we limit the propagation to one-hop neighbors?*
  - Researchers have assumed that ***the given graph is ideal***
    - *What if the graph contains missing or noisy edges?*

- **Main idea**: Let's augment the input graph so that any GNN models improve through ***graph diffusion***!

GDC [NeurIPS'19]

- **GDC** (Graph Diffusion Convolution)
  - *Considered as graph data augmentation*



GDC is applicable to any GNN models!

Proposed Method

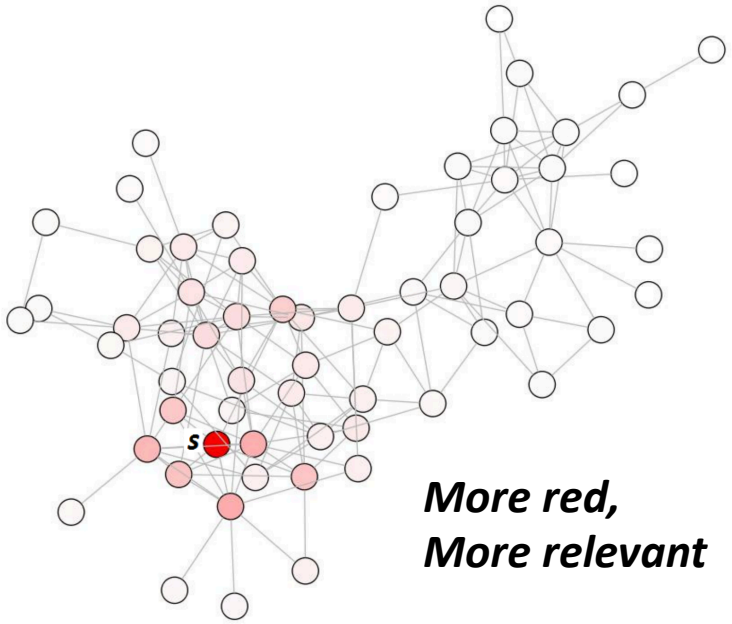
- **GDC (Graph Diffusion Convolution)**
  - **Input:** undirected & **unweighted** graph G
  - **Output:** new augmented **weighted** graph G'

  - **Step 1.** compute transition matrix $\mathbf{T}$
  - **Step 2.** compute graph diffusion matrix $\mathbf{S}$ based on $\mathbf{T}$
  - **Step 3.** sparsify the diffusion matrix $\mathbf{S} \Rightarrow \tilde{\mathbf{S}}$

  - **Return** $\tilde{\mathbf{S}}$ as *adjacency weighted matrix* of G'

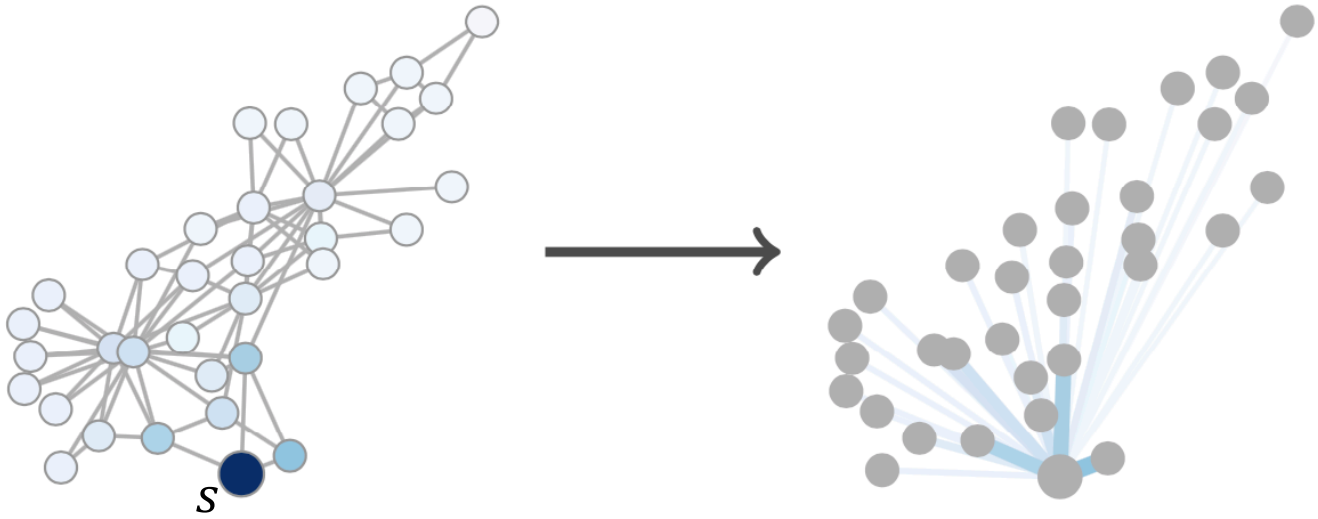
Graph Diffusion

- **What is Graph Diffusion?**
  - Stochastic process for *spreading (or propagating) information* along the edges in a graph
  - Starting from a seed (or source) node s
  - Obtain diffusion score vector for all nodes w.r.t. s
    - Typically ***localized*** to seed node s
  - Representative graph diffusions
    - **Personalized PageRank**
      - *Spread a random surfer over graphs*
    - **Heat Kernel**
      - *Spread heat over graphs*



Graph Diffusion

- **Graph diffusion matrix $S \in \mathbb{R}^{n \times n}$**
  - Contains diffusion scores of all pairs of nodes
    - S_{ij}: diffusion score on node i starting at node i
  - Let's use the diffusion matrix as the adjacency weighted matrix of a new graph
    - **Aim to use more nodes beyond 1-hop neighbors**



Diffusion scores w.r.t. seed node s

Diffusion scores ⇒ edge weights

Generalized Graph Diffusion

- **Generalized Graph diffusion matrix $\mathbf{S} \in \mathbb{R}^{n \times n}$**

$$\mathbf{S} = \sum_{k=0}^{\infty} \theta_k \mathbf{T}^k$$

  - k: diffusion step
  - θ_k: diffusion weight at step k
  - $\mathbf{T}^k$: transition (or stochastic) matrix after k steps
    - Movement of information from where to where

- Two models: **Personalized PageRank** and **Heat Kernel**
  - Personalized PageRank's diffusion matrix

$$\mathbf{T} = \tilde{\mathbf{A}}$$
$$\theta_k = c(1-c)^k$$

$$\mathbf{S} = \sum_{k=0}^{\infty} c(1-c)^k \tilde{\mathbf{A}}^k = c\left(\mathbf{I} - (1-c)\tilde{\mathbf{A}}\right)^{-1}$$

  - See the paper for Heat Kernel's diffusion matrix

Sparsification

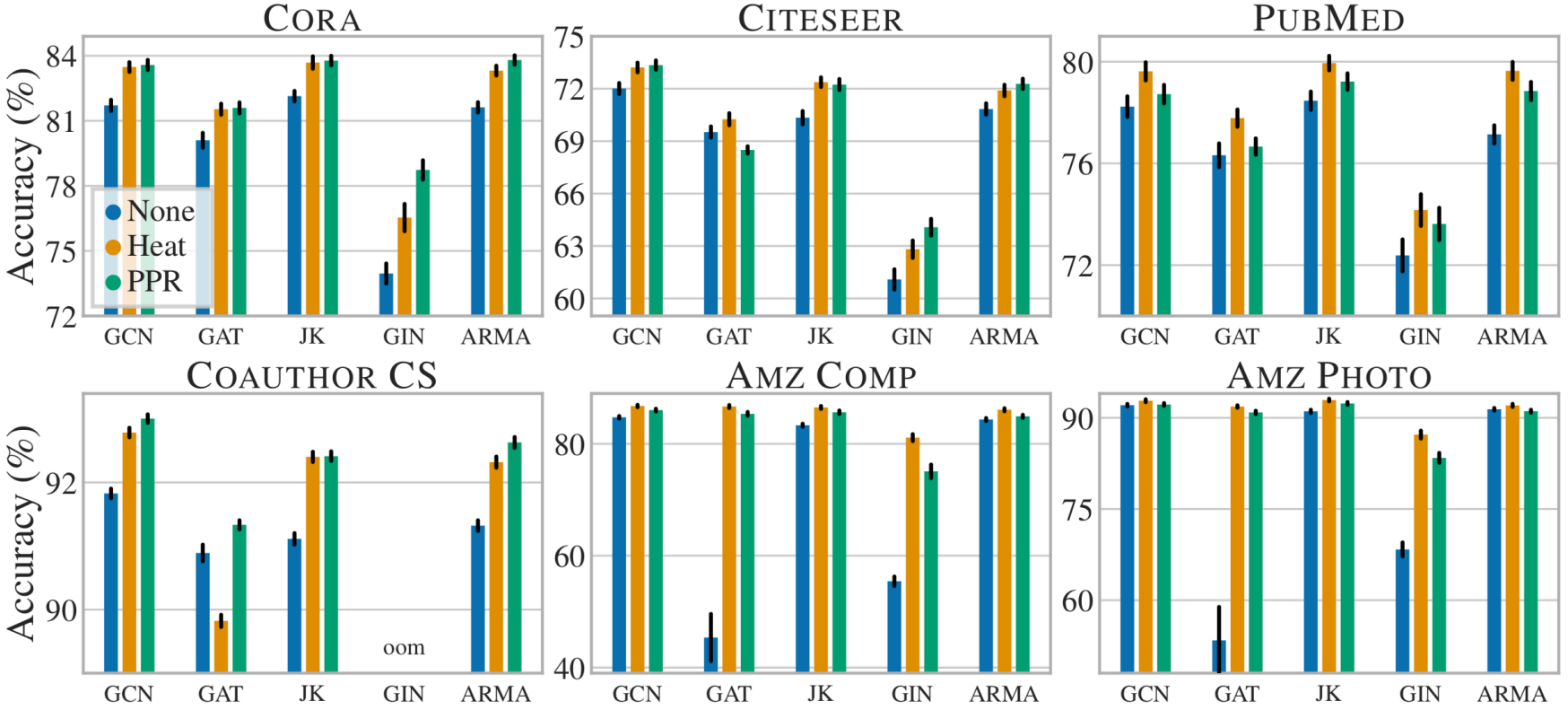
- **Diffusion matrix S is fully dense**
  - **Why?** Infinite sum of power of the transition matrix
  - Require $O(n^2)$ space ⇒ GNNs using naïve $\mathbf{S}$ will be inefficient

- **Solution.** To sparsify the diffusion matrix $\mathbf{S}$
  - i.e., let's reduce the number of non-zero entries in $\mathbf{S}$
  - **Option 1.** select top-k relevant nodes for each node
    - In order of diffusion scores w.r.t. each node
  - **Option 2.** remove entries below a threshold ϵ
  - *Choice between the strategies is another hyperparameter*

Experimental Result

- **Node classification performance of GDC**

None: the original graph, Heat/PPR (proposed): GDC with Heat/PPR



GDC with Heat or PPR improves the performance of all GNN models in the task

Discussion on GDC

- **Pros**
  - **P1.** Augmentation technique for graph data
  - **P2.** Use features from more nodes when modeling latent node embedding
  - **P3.** Applicable to any GNN models
    - Improve the performance of all tested GCN models
- **Cons**
  - **C1.** Need to carefully tune hyperparameters
    - Which diffusion we should use? (PPR or Heat Kernel)
    - Which sparsification we should use? (Top-k selection or threshold)
  - **C2.** Not scalable for computing diffusion matrices
    - Approximation is possible, but its effect is not revealed

Conclusion

- **Graph convolution makes ML tasks on graphs easy (as feature extractor on graphs)**
- **Various techniques for graph convolution**
  - Basic models (shallow): **GCN**, **GraphSAGE**, and **GAT**
  - Advanced models (deep): **JK**, **APPNP**, and **GDC**
- **Future research directions**
  - ***Unstudied graph structures***: heterogeneous graphs, signed graphs, and hypergrpahs
  - ***Dynamic graphs***: nodes, edges, and features can change over time
  - ***Interpretability***: how to interpret and reason the results of graph neural networks
  - ***Lightweight models***: how to accelerate GCN models without loss of accuracy

Q & A

- Email: jinhongjung@snu.ac.kr
- Web: jinhongjung.github.io
- **Must-read papers on GNN**
  - Contributed by Jie Zhou, Ganqu Cui, Zhengyan Zhang and Yushi Bai, https://github.com/thunlp/GNNPapers
- **Related tutorials on GNN**
  - "*Representation Learning on Networks*" by William L. Hamilton et al., WWW2018
  - "*Graph Neural Networks: Models and Applications*" by Yao Ma et al., AAAI2020
- **Geometric deep learning library for PyTorch**
  - Contributed by Matthias Fey and Jan E. Lenssen
  - https://github.com/rusty1s/pytorch_geometric

References

[1] Kipf, Thomas N., and Max Welling. "*Semi-supervised classification with graph convolutional networks*", ICLR 2017

[2] Hamilton, Will, Zhitao Ying, and Jure Leskovec. "*Inductive representation learning on large graphs*", NIPS 2017

[3] Veličković, Petar, et al. "*Graph attention networks*", ICLR 2018

[4] Xu, Keyulu, et al. "*Representation Learning on Graphs with Jumping Knowledge Networks*", ICML 2018

[5] Klicpera, Johannes, Aleksandar Bojchevski, and Stephan Günnemann. "*Predict then Propagate: Graph Neural Networks meet Personalized PageRank*", ICLR 2019

[6] Klicpera, Johannes, Stefan Weißenberger, and Stephan Günnemann. "*Diffusion improves graph learning*", NuerIPS 2019

[7] Li, Guohao, et al. "*Deepgcns: Can gcns go as deep as cnns?*", ICCV 2019

[8] Li, Qimai, Zhichao Han, and Xiao-Ming Wu. "*Deeper insights into graph convolutional networks for semi-supervised learning*", AAAI 2018