정보검색 및 데이터 마이닝

Chapter3. Graph Neural Networks

1. Outline

A. What is a Graph

B. Machine Learning for Graphs

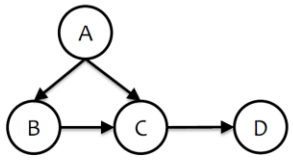
C. Graph Convolution Networks (GCNs)

D. Takeaways

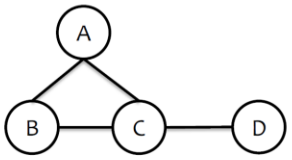
A. What is a Graph?

1) 정의

그래프는 Node(Vertex, Entity)와 Edge(Relation)로 정보가 저장된 형태를 말한다. Edge는 방향성이 있을 수도, 없을 수도 있다. 이 때, 방향성이 있으면 Directed Graph라고 하고, 방향성이 없으면 Undirected Graph라고 한다.



Directed graph (digraph):
a graph with directed edges



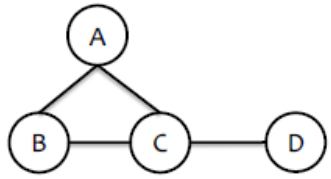
Undirected graph:
a graph with undirected edges

2) Graph Terminologies

End Points	두 노드가 Edge로 연결되어 있을 때, 노드가 Endpoint라 함.
Adjacent	인접. 두 노드 간의 연결인 Edge가 존재함을 말함.
Incident	주어진 노드가 어떤 노드와 Edge로 연결되어 있을 때, 해당 Edge는 그 노드에 대해 incident하다고 함. (노드가 Edge의 endpoint. Ex) (A, B) is incident to A and B)
Degree	deg(v): v의 incident 수, v와 연결된 노드 수

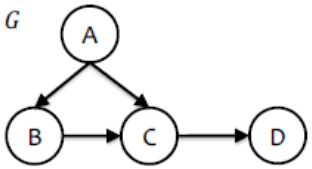
* Directed Graph

Directed Graph에서는 in-degree와 out-degree가 정의된다. In-degree는 노드를 기준으로 그 노드로 들어오는 incident의 수를, out-degree는 나가는 incident수를 말한다. (In-degree: incoming edges, Out-degree: outcoming edges)

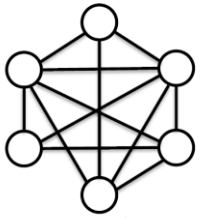


Path	인접 노드들의 sequence. Ex) A-B-C-D
Length	Path의 길이. Path의 Edge수! Ex) A-B-C-D -> Length=3
Cycle	Path중 시작과 끝 노드가 같은 path Ex) A-B-C-A

Directed Graph 중 Directed Cycle이 없는 그래프를 Directed Acyclic Graph (DAG)라고 한다.

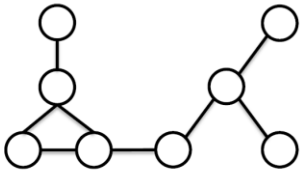


3) Complexity of Graph



Dense graphs: $|E| \approx |V|^2$

A large fraction of pairs of vertices are connected by edges

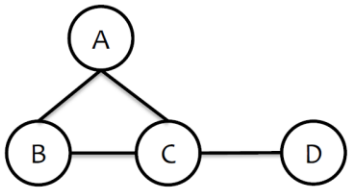


Sparse graph: $|E| \approx |V|$

The number of edges is small relative to the number of vertices.

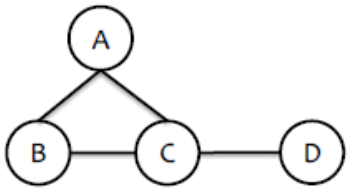
Entity 수와 Vertex 수에 의해 결정됨.

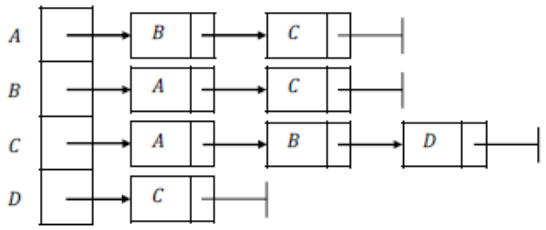
4) Adjacency Matrices



A	A	B	C	D
A	0	1	1	0
B	1	0	1	0
C	1	1	0	1
D	0	0	1	0

노드 사이 연결이 있으면 1, 없으면 0으로 표현. 각각의 노드는 인접 행렬을 리스트 형태로 보관한다. 각각의 노드는 인접 행렬의 리스트로 저장된다.





B. Machine Learning for Graph

1) Node Embedding

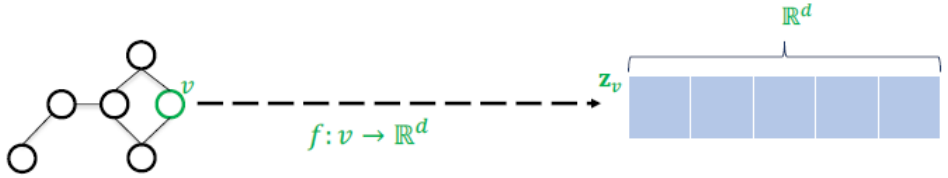
그래프로 Deep learning을 하는 방법은 크게 3가지 카테고리로 분류된다.

- Traditional Method: Graphlets, Graph Kernels
- Node Embedding: DeepWalk, node2vec
- GNN: GCN, GraphSAGE, GAT

Given a graph $G = (V, E)$:

Generate embeddings (or vectors) $\mathbf{z}_v \in \mathbb{R}^d$ for every node $v \in V$

d 차원 벡터 = Representation = Embedding



노드 임베딩은 각각의 노드를 고유한 임베딩 벡터로 mapping하는 방법이다. 즉, 각 노드는 각자 특정한 벡터로 임베딩된다. 각 노드들이 같은 차원수의 벡터로 mapping되기 때문에 비슷한 노드는 가깝게 임베딩된다.

similar nodes in the graph are embedded close to each other.

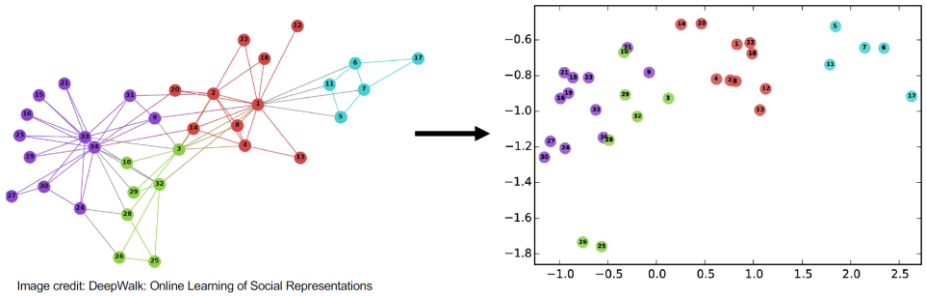


Image credit: DeepWalk: Online Learning of Social Representations

How do we define the 'Similarity' between vertices

: proximity, distance 또는 class로 두 노드의 유사도(Similarity)를 정의한다.

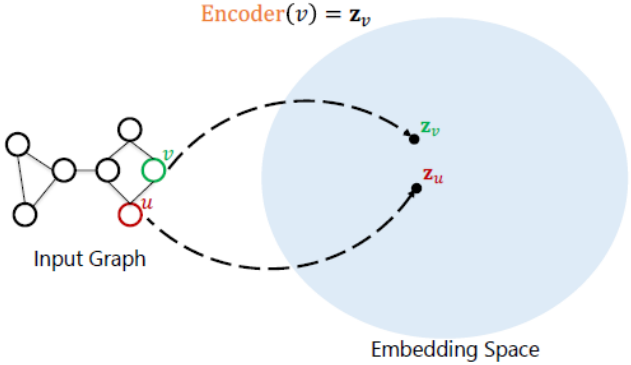
⇨ Similarity = sim(u, v) = $Z_u^T Z_v$: 벡터 내적(inner product)

Transductive vs Inductive

Output of training is node embeddings $\mathbf{z}_v$ (**Transductive**)

Output of training is an encoder (a function that generates embeddings) (**Inductive**)

$\text{Encoder}(v) = \mathbf{z}_v$



Transductive	학습의 결과가 node embedding Z_v 이다.
Inductive	학습의 결과가 Encoder인 경우(Encoder(v) = Z_v)

Transductive: 학습 중 unseen 노드에 대한 임베딩을 생성하지 못함.

Inductive: 학습 중 unseen 노드의 임베딩을 생성할 수 있음.

Drawbacks of Node Embedding

1) Transductive하다.

2) O(|V|d)의 파라미터가 필요

   a. 노드들의 paramter가 공유되지 않기 때문이다.

   b. 각 노드가 unique한 임베딩을 가진다.

3) Node Feature가 통합되지 못함

   a. 각 feature(age, gender, region, ...)이 따로 임베딩 벡터를 만듬.

2) GNN

(1) Task

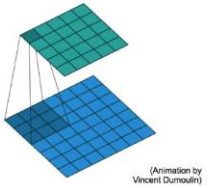
Graph Level: Graph Embedding, Graph Generation

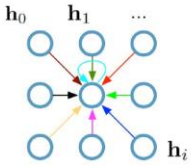
Node Level: Node Embedding, Node Classification

Edge Level: Link Prediction

(2) Recap of CNNs

Single CNN layer with 3x3 filter:





Update for a single pixel:

- Transform messages individually $\mathbf{W}_i\mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i\mathbf{h}_i$

$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

Full update:

$$\mathbf{h}_4^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \cdots + \mathbf{W}_8^{(l)}\mathbf{h}_8^{(l)}\right)$$

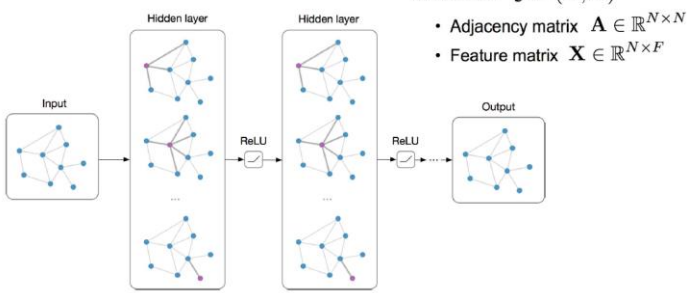
Irregular하고 Non-Euclidian한 Graph 데이터들을 CNN 아이디어를 이용해 일반화한 버전이 GCN이다.

Main Idea: Pass massages between pairs of nodes and agglomerate

Alternative Interpretation: Pass massages between nodes to refine node (and possibly edge) representations

Notation: $\mathcal{G} = (\mathbf{A}, \mathbf{X})$

- Adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$
- Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$

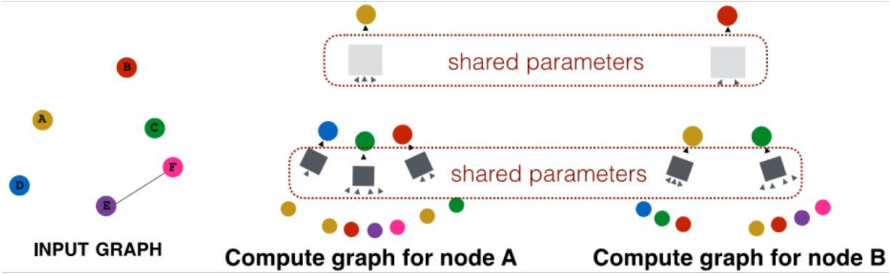


(3) GNNs의 구조

Target node의 임베딩을 반복해서 계산하며, 이웃 노드들의 정보를 취합하여 임베딩을 구한다. 각 Step별로 1-hop 단위로 정보를 취합(Message passing -> Aggregation)한다.

```
Each layer outputs a new embedding of a target node by aggregating the embeddings of its neighbors in the previous layer
```

Inductive Capability



모든 노드들이 같은 aggregation 파라미터를 공유하며, |V|의 모델 파라미터 수는 sublinear하다. 따라서 Unseen Node에 대해 일반화가 가능하다.

GNN Layer = Message passing + Aggregation (+ Combine [self-information])

Laplacian matrix

Degree matrix에서 adjacency matrix를 뺀 matrix.

Laplacian matrix(L) = Degree Matrix(D) − Adjacency Matrix(A)

L = D − A

GNN Layer = Message passing + Aggregation (+ Combine [self-information])

(1) Message Passing

a. message function: $\boldsymbol{m}_u^{(l)} = MSG^{(l)}(\boldsymbol{h}_u^{(l-1)})$

b. 각 노드가 message를 만들어 냄.

c. Linear layer $\boldsymbol{m}_u^{(l)} = MSG^{(l)}(\boldsymbol{h}_u^{(l-1)})$ 가 노드 feature를 나타내는 weight matrix $W^{(l)}$과 곱해짐 -> $\boldsymbol{m}_u^{(l)} = W^{(l)}\boldsymbol{h}_u^{(l-1)}$

(Node feature = node attribute)

(2) Aggregation

a. 이웃 정보를 취합

b. $\boldsymbol{a}_v^{(l)} = AGG^{(l)}(\{\boldsymbol{m}_u^{(l)}, u \in N(v)\})$

c. 취합을 하는 aggregator의 예로는 sum, mean, max등이 있음

(3) Combine

a. Purpose: Target node v의 자체 정보는 유실되기 때문에 포함시키기 위함.

b. 따라서, $\boldsymbol{h}_v^{(l)}$을 계산할 때 $\boldsymbol{h}_v^{(l-1)}$도 포함시켜야 함

c. $\boldsymbol{h}_v^{(l)} = Combine^{(l)}(\boldsymbol{a}_v^{(l)}\boldsymbol{m}_v^{(l-1)})$

i. $\boldsymbol{a}_v^{(l)} = AGG^{(l)}(\{\boldsymbol{m}_u^{(l)}, u \in N(v)\})$

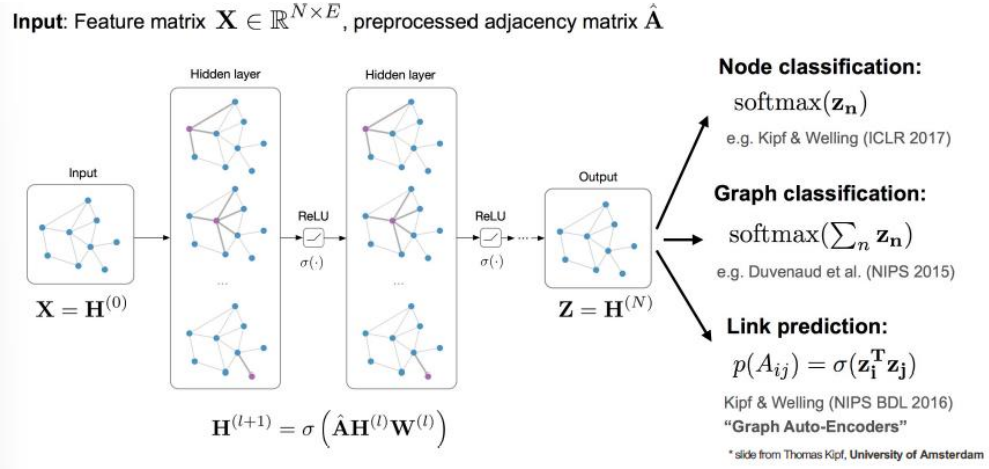
ii. $\boldsymbol{m}_u^{(l)} = W^{(l)}\boldsymbol{h}_u^{(l-1)}$

iii. $\boldsymbol{m}_v^{(l-1)} = B^{(l)}\boldsymbol{h}_v^{(l-1)}$

d. 만약 Combine이 필요 없다면 $\boldsymbol{h}_v^{(l)} = \boldsymbol{a}_v^{(l)}$로 두고 진행

GNN/GCNs 로 풀 수 있는 Task

- Input: Feature matrix & Adjacency matrix
- Output: Node Embedding



Loss for Noce Classification

(1) Cross-Entropy Loss

$$\mathcal{L} = \sum_{u \in \mathcal{V}_{\text{train}}} -\log(\text{softmax}(\mathbf{z}_u, \mathbf{y}_u)).$$

$$\text{softmax}(\mathbf{z}_u, \mathbf{y}_u) = \sum_{i=1}^{c} \mathbf{y}_u[i] \frac{e^{\mathbf{z}_u^\top \mathbf{w}_i}}{\sum_{j=1}^{c} e^{\mathbf{z}_u^\top \mathbf{w}_j}},$$

W = Weight vector(Learnable Parameter)

Node Classification은 Semi-supervised learning과 Supervised learning 방식 有

- Semi-supervised learning: Transductive
- Supervised learning: Inductive

(2) Transductive Test nodes Vtrans

Training Set에는 노드들이 label이 존재하지만 Test set에는 label이 없다. 하지만, Test Set이 Training Set Graph안에 존재하기 때문에 test set의 attribute들을 학습 시 message passing과 aggregation에 영향을 끼친다.

(3) Inductive test nodes Vind

Unseen data에 대해 classification을 하는 방식으로 Test set의 노드들이 Training set에 없다.

Loss for Graph-Level Task

(1) Graph Classification: Cross Entropy

(2) Graph Regression

$$\mathcal{L} = \sum_{\mathcal{G}_i \in \mathcal{T}} \|\text{MLP}(\mathbf{z}_{\mathcal{G}_i}) - y_{\mathcal{G}_i}\|_2^2,$$

Graph Pooling을 거친 후 얻은 ZG 임베딩이 있다할 때 위와 같은 Loss를 사용(MSE)

Ex) 분자 구조 예측

Loss for Link prediction

Pairwise node embedding loss function

$$\mathcal{L} = \sum_{(u,v) \in \mathcal{D}} \ell\left(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v), \mathbf{S}[u, v]\right),$$

- $DEC(\mathbf{z}_u, \mathbf{z}_v)$: similarity between $\mathbf{z}_u, \mathbf{z}_v$
- $\mathbf{S}[u, v]$: ground truth pairwise neighborhood (e.g., adjacency matrix **A**)

Recommendation systems, knowledge graph completion

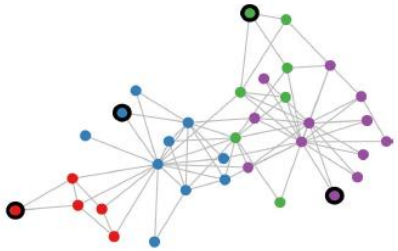
Semi-supervised learning GNN을 위한 Setting

특징은, 몇몇 노드들만 Labeling이 되어있다는 것이다. 나머지는 모두 Unlabeled.

- Task
  a. Labeled 노드들로부터 unlabeled node를 예측하자
  b. GCN with asymmetric normalization이 이에 해당

Setting:
Some nodes are labeled (black circle)
All other nodes are unlabeled

Task:
Predict node label of unlabeled nodes



Evaluate loss on labeled nodes only:

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}$$

$\mathcal{Y}_L$ set of labeled node indices
$\mathbf{Y}$ label matrix
$\mathbf{Z}$ GCN output (after softmax)

* slide from Thomas Kipf, **University of Amsterdam**

C.1 Graph Convolution Networks (GCNs)

1) 정의

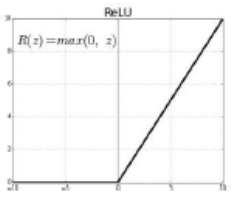
이미지, 텍스트, 정형 데이터는 격자 형태로 표현이 가능하다. 즉, 유클리디언 공간 상의 격자 표현할 수 있다.(좌표 평면상에서 벡터로 표현 가능)

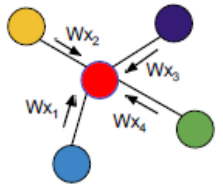
반면, social network와 분자 데이터 등은 유클리디언 공간 상에 표현하는 것이 어렵다. 유클리디언 공간이 아니므로 거리는 중요하지 않으며, '연결 여부'와 '연결 강도'가 중요하다.

GCN with asymmetric normalization

$$h_v = f\left(\frac{1}{|\mathcal{N}(v)|}\sum_{u \in \mathcal{N}(v)} W x_u + b\right), \quad \forall v \in \mathcal{V}.$$

Normalization
Neighborhood Aggregation
Bias (Model-parameter)
Non-Linearity
Filter matrix (Model-parameter)
Initial Node Features





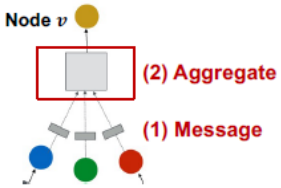
CNN에서 사용되는 Convolution은 결국 커널을 움직여가며 커널 크기에 해당하는 정보들을 한 곳에 모으는 과정이다. Graph에서는 Target 노드와 연결된 이웃 정보를 weighted average(가중합)함으로서 Convolution 효과를 만들어낸다.

Stacking K-GNN Layers for capturing K-hop neighborhood

$$h_v^{k+1} = f\left(\frac{1}{|\mathcal{N}(v)|}\sum_{u \in \mathcal{N}(v)} W^k h_u^k + b^k\right), \quad \forall v \in \mathcal{V}.$$

- GCN layer: $\mathbf{h}_v^{(l)} = \sigma(\Sigma_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|})$

1. Message
   - $\mathbf{m}_u^{(l)} = \frac{1}{|N(v)|}(\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)})$ for each neighbor $u \in N(v)$
   - Normalized by node degree
2. Aggregate
   - $\mathbf{a}_v^{(l)} = \text{sum}(\{\mathbf{m}_u^{(l)}, u \in N(v)\})$
   - Do not combine: a graph is assumed to have self-edges included in summation
- Followed by nonlinearity (activation) $\sigma(\cdot)$
   - $\mathbf{h}_v^{(l)} = \sigma\left(\text{sum}(\{\mathbf{m}_u^{(l)}, u \in N(v)\})\right)$



GNN Layer = Message passing + Aggregation (+ Combine [self-information])

GCN (Final Version) with Symmetric Normalization

- GCN aggregates and combines simultaneously with element-wise mean pooling.

$$\mathbf{h}_v^{(k)} = \sigma\left(\mathbf{W}_{k-1} \cdot MEAN\left\{\mathbf{h}_u^{(k-1)}, \forall u \in N(v) \cup \{v\}\right\}\right)$$

$$\mathbf{h}_v^{(k)} = \sigma\left(\mathbf{W}_{k-1} \cdot \Sigma_{u \in N(v) \cup \{v\}} \frac{\mathbf{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}}\right).$$

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

Same matrix for self and neighbor embeddings

Per-neighbor normalization

Hidden representation of all Graph

이 때문에 Symmetric Normalization이라 함

  - where σ is ReLU, $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ ($\mathbf{A}$ is an adjacency matrix), $\hat{\mathbf{D}}$ is a diagonal node degree matrix of $\hat{\mathbf{A}}$

Self-Loop

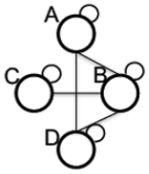
- Simple, but transductive (can only generate embeddings for a single fixed graph).
  - Transductive approach does not efficiently generalize to unseen nodes (e.g., in evolving graphs), and cannot learn to generalize across different graphs.

GCN에서는 앞의 식과는 다르게, Self-loop를 추가해 target node의 정보도 aggregation한다. 기존에는 Target Node인 v에 대해서만 Normalization을 해주었는데, 이제는 aggregation 과정에서 이웃과 Target노드 모두 가져오기 때문에 이웃에 대한 Normalization(Per-neighbor normalization) term을 추가해준다.

2) Asymmetric Normalization

A = Adjacency Matrix

D = Diagonal Degree 행렬



$\hat{D}^{-1}\hat{A}$

where $\hat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ ($\boldsymbol{A}$ is an adjacency matrix), $\hat{\boldsymbol{D}}$ is a diagonal degree matrix of $\hat{\boldsymbol{A}}$

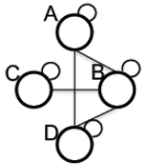
Asymmetric

$\hat{D}^{-1}$	a	b	c	d
a	1/3	0	0	0
b	0	1/4	0	0
c	0	0	1/2	0
d	0	0	0	1/3

$\hat{A}$	a	b	c	d
a	1	1	0	1
b	1	1	1	1
c	0	1	1	0
d	1	1	0	1

=

$\hat{D}^{-1}\hat{A}$	a	b	c	d
a	**1/3**	1/3	0	1/3
b	1/4	**1/4**	1/4	1/4
c	0	1/2	**1/2**	0
d	1/3	1/3	0	**1/3**



$\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$

Contribution
: target node와 neighbor 노드의 degree 모두 이용
즉, Target 노드의 정보와 Neighbor 정보 모두 이용

where $\hat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ ($\boldsymbol{A}$ is an adjacency matrix), $\hat{\boldsymbol{D}}$ is a diagonal degree matrix of $\hat{\boldsymbol{A}}$

Symmetric

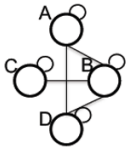
$$\Sigma_{u \in N(v) \cup \{v\}} \frac{h_u^{k-1}}{\sqrt{|N(u)||N(v)|}}$$

$\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$	a	b	c	d
a	$\frac{1}{3}$	$\frac{1}{2\sqrt{3}}$	0	$\frac{1}{3}$
b	$\frac{1}{2\sqrt{3}}$	$\frac{1}{4}$	$\frac{1}{2\sqrt{2}}$	$\frac{1}{2\sqrt{3}}$
c	0	$\frac{1}{2\sqrt{2}}$	$\frac{1}{2}$	0
d	$\frac{1}{3}$	$\frac{1}{2\sqrt{3}}$	0	$\frac{1}{3}$

$H^{(k-1)}$
$H_a^{(k-1)}$
$H_b^{(k-1)}$
$H_c^{(k-1)}$
$H_d^{(k-1)}$

=

$\Sigma_{u \in N(v) \cup \{v\}} \frac{h_u^{k-1}}{\sqrt{\lvert N(u)\rvert\lvert N(v)\rvert}}$
$D^{-1}AH^{(k-1)}$
$\frac{1}{3}H_a^{(k-1)} + \frac{1}{2\sqrt{3}}H_b^{(k-1)} + \frac{1}{3}H_d^{(k-1)}$
$\frac{1}{2\sqrt{3}}H_a^{(k-1)} + \frac{1}{4}H_b^{(k-1)} + \frac{1}{2\sqrt{2}}H_c^{(k-1)} + \frac{1}{2\sqrt{3}}H_d^{(k-1)}$
$\frac{1}{2\sqrt{2}}H_b^{(k-1)} + \frac{1}{2}H_c^{(k-1)}$
$\frac{1}{3}H_a^{(k-1)} + \frac{1}{2\sqrt{3}}H_b^{(k-1)} + \frac{1}{3}H_d^{(k-1)}$



$$\hat{D}^{-1}\hat{A}H^{(k-1)}$$

where $\hat{A} = A + I$ (A is an adjacency matrix), $\hat{D}$ is a diagonal degree matrix of $\hat{A}$

$\hat{D}^{-1}\hat{A}$	A	B	C	D
A	**1/3**	1/3	0	1/3
B	1/4	**1/4**	1/4	1/4
C	0	1/2	**1/2**	0
D	1/3	1/3	0	**1/3**

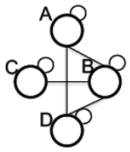
$H^{(k-1)}$

$H_A^{(k-1)}$
$H_B^{(k-1)}$
$H_C^{(k-1)}$
$H_D^{(k-1)}$

$=$

$D^{-1}AH^{(k-1)}$
$\frac{1}{3}H_A^{(k-1)} + \frac{1}{3}H_B^{(k-1)} + \frac{1}{3}H_D^{(k-1)}$
$\frac{1}{4}H_A^{(k-1)} + \frac{1}{4}H_B^{(k-1)} + \frac{1}{4}H_C^{(k-1)} + \frac{1}{4}H_D^{(k-1)}$
$\frac{1}{2}H_B^{(k-1)} + \frac{1}{2}H_C^{(k-1)}$
$\frac{1}{3}H_A^{(k-1)} + \frac{1}{3}H_B^{(k-1)} + \frac{1}{3}H_D^{(k-1)}$

Asymmetric



$$H^{(k)} = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(k-1)}W_{k-1}\right)$$

where σ is ReLU,
$\hat{A} = A + I$ (A is an adjacency matrix),
$\hat{D}$ is a diagonal node degree matrix of $\hat{A}$

σ

$\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(k-1)} \in R^{n\times d_{k-1}}$	$W_{k-1} \in R^{d_{k-1}\times d_k}$
$\frac{1}{3}H_A^{(k-1)} + \frac{1}{2\sqrt{3}}H_B^{(k-1)} + \frac{1}{3}H_D^{(k-1)}$	
$\frac{1}{2\sqrt{3}}H_A^{(k-1)} + \frac{1}{4}H_B^{(k-1)} + \frac{1}{2\sqrt{3}}H_C^{(k-1)} + \frac{1}{2\sqrt{3}}H_D^{(k-1)}$	
$\frac{1}{2\sqrt{2}}H_B^{(k-1)} + \frac{1}{2}H_C^{(k-1)}$	
$\frac{1}{3}H_A^{(k-1)} + \frac{1}{2\sqrt{3}}H_B^{(k-1)} + \frac{1}{3}H_D^{(k-1)}$	

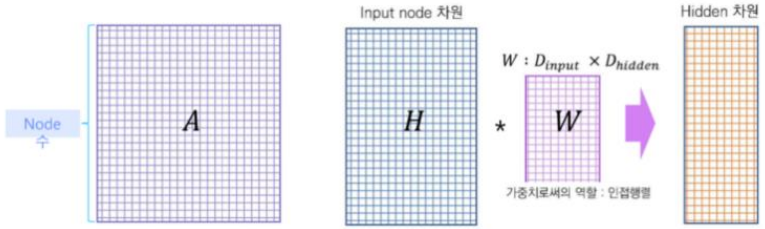
$=$

$H^{(k)} \in R^{n\times d_k}$

Symmetric

GCN은 결국 AH * W이다. (A = 인접 행렬, H = Hidden state, W = Learnable Weight)

GCN은 결국, $AH * W$

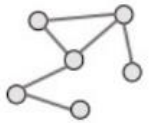
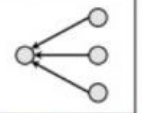
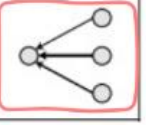


- 위 과정을 통해 계산되어진 output은 1번 업데이트된 임베딩이 될 것입니다.
- Shape
  - A: $N * N$
  - H: $N * h$
  - W: $H_{in} * H_{out}$
  - output: $N * H_{out}$

3) Drawbacks of GCN

Self-Loop가 추가된 $\tilde{A}$는 행렬을 사용하므로, 상호작용이 없는 유저-아이템에 대해서는 사용할 수 없습니다(Transductive)

C.2 Graph Attention Network (GAT)

Model	Graph type of GNN	GNN category	Aggregate
GCN		Spectral	
GAT		Spatial	

GAT는 Spatial한 방법론을 사용한다. 따라서 Adjacency matrix를 이용하지 않는다. Graph Attention network의 핵심은 단순히 합치는 것이 아니라, 노드 별 상이한 weight를 가지고 가중합 하는 방식이다. 즉, '나'한테 영향을 주는 '정도'까지 학습한다.

GCN의 경우 노드들이 $\alpha_{vu} = \frac{1}{|N(v)|}$로 같은 크기의 가중치를 가진다. 하지만, GAT에서는 이 값을 다르게 주기위해 α_{vu}를 Learnable Parameter로 만든다.

Attention coefficient

- unnormalized coefficient $e_{ij} = a(\mathbf{W}\mathbf{h}_i, \mathbf{W}\mathbf{h}_j)$
  where a is attention mechanism
- normalized coefficient $\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in N(u_i)}\exp(e_{ik})}$
  - the importance of node j's features to node i

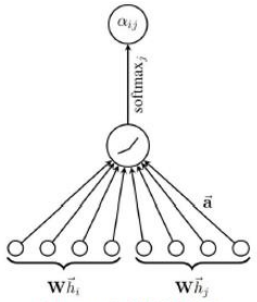


Image credit: GAT, ICLR 2018

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

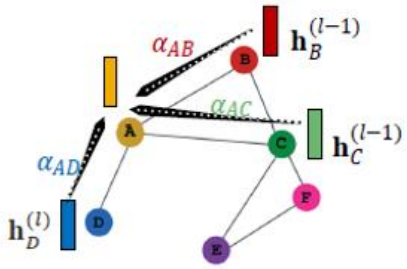
GAT layer: $\mathbf{h}_v^{(l)} = \sigma(\Sigma_{u \in N(v)} \alpha_{vu} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$

- where α_{vu}: attention weights

Weighted sum based on the final attention weight α_{vu}

Example

$$\mathbf{h}_A^{(l)} = \sigma\left(\alpha_{AB}\mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)} + \alpha_{AC}\mathbf{W}^{(l)}\mathbf{h}_C^{(l-1)} + \alpha_{AD}\mathbf{W}^{(l)}\mathbf{h}_D^{(l-1)}\right)$$

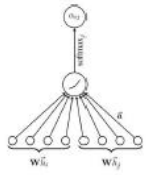


나한테 영향을 주는 정도를 학습하기 위해 Attention을 사용한다. $a_{ij}^{(l)}$ 는 나한테 영향을 주는 '정도' 즉, 가중치를 나타낸다. $e_{ij}^{(l)}$ 는 i와 j의 중요도를 나타낸 것으로 $softmax\left(e_{ij}^{(l)}\right) = a_{ij}^{(l)}$ 이다.

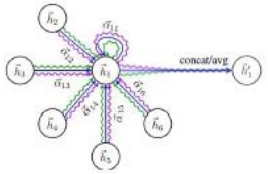
Multi Head Attention

- Uses self-attention for GCNs
  - Input features: $\mathbf{h} = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}, \vec{h}_i \in \mathbb{R}^F$,
  - Importance of vj to vi : $e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$ $\quad \alpha_{ij} = \mathrm{softmax}_j(e_{ij})$

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



x K
K = # Heads



$$\vec{h}_i' = \Big\|_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$ ←GAT Update (K=#heads)

$$\vec{h}_i' = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$ ←Final Layer : Mean 계산

GCN은 GAT는 Semi-Supervised Learning 방식이다. 반면 Unsupervised Learning방식은 채택한 모델은 GraphSAGE이다. labeling이 되지 않은 데이터를 이용하거나, 큰 사이즈의 graph를 학습해야 하는 모델들은 모두 unsupervised learning에 해당한다.

C.3 GraphSAGE

GraphSAGE는 노드의 임베딩을 학습하는 것이 아닌, aggregate function 그 자체를 학습한다. 이러한 방식을 채택한 이유는, 실제로 Graph에서 실시간으로 새로운 node가 추가되는 경우가 많기 때문이다. 이런 경우, graph의 label을 모르기 때문에 기존의 모델들의 한계가 드러난다.

GraphSAGE는 이러한 fixed graph에 대한 노드 임베딩을 학습하는 Transductive learning의 방식의 한계점을 지적하고, evolving graph에서 새롭게 추가된 노드에 대해서도 inductive node embedding을 산출할 수 있는 프레임워크를 제안한다.

- GraphSAGE learns aggregate function instead of training the node embeddings in GCN.

$$\mathbf{a}_v^{(k)} = \mathrm{AGG}_k(\{\mathbf{h}_u^{(k-1)}, u \in N(v)\})$$

where $N(v)$ is a set of nodes adjacent to v, e.g., $\max(\{\sigma\left(\mathbf{W}_{pool}\mathbf{h}_u^{(k-1)} + b\right), \forall u \in N(v)\})$

Concatenate self embedding and neighbor embedding

Generalized aggregation

$$\mathbf{h}_v^{(k)} = \sigma\left(\mathbf{W}^k \cdot \mathrm{concat}(\mathbf{h}_v^{(k-1)}, \mathbf{a}_v^{(k)})\right)$$

- Inductive: use node attribute information to efficiently generate representations on previously unseen data.

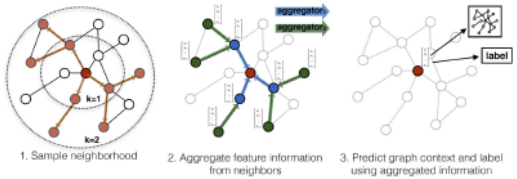


Figure 1: Visual illustration of the GraphSAGE sample and aggregate approach.

Image credit: GraphSage, NIPS 2017

GraphSAGE layer: $\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \cdot \text{concat}\left(\mathbf{h}_v^{(l-1)}, \text{AGG}\left(\left\{\mathbf{h}_u^{(l-1)}, \forall u \in N(v)\right\}\right)\right)\right)$

Message and Combining are computed within AGG(·)

Mean Aggregator:

$$\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W} \cdot \text{MEAN}(\{\mathbf{h}_v^{k-1}\} \cup \{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}).$$

LSTM Aggregator:

Applies LSTM to a random permutation of neighbors.

Pooling Aggregator:

$$\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\});$$
$$\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$$

$$\text{AGGREGATE}_k^{\text{pool}} = \max(\{\sigma\left(\mathbf{W}_{\text{pool}}\mathbf{h}_{u_i}^k + \mathbf{b}\right), \forall u_i \in \mathcal{N}(v)\}),$$

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

이웃 노드를 정의할 때 uniform random draw 방식으로 정해진 개수의 최 근접 노드를 샘플링한다. 이는 실제 학습 과정에서 N(u)를 모든 근접 노드로 적용한다면, 계산 복잡도를 제어할 수 없기 때문이다.

Loss function: close nodes get similar representations while distant nodes get different representations

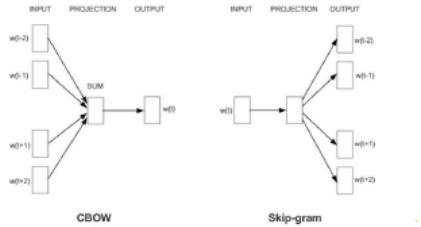
positive / Negative

$$J_{\mathcal{G}}(\mathbf{z}_u) = -\log\left(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)\right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log\left(\sigma(-\mathbf{z}_u^\top \mathbf{z}_{v_n})\right),$$

Nearby nodes → Similar representation Disparate nodes → Highly distinct representation

Inductive 하기 때문에 Class 를 고려할 필요 X

- $\mathbf{z}_u$: the representation of node u
- v: a node within the random walk from u
- v_n: a negative sample
- $P_n(v)$: negative sampling distribution
- Q: the number of negative samples



C.4 GIN(Graph Isomorphism Network)

Weisfeiler-Lehman test

- Polynomial-time heuristic for the graph isomorphism problem.
- Can (sometimes cannot) discriminates between two non-isomorphic graphs.
- False positive may exists.

GIN은 위의 Weisfeiler-Lehman Test 사이의 유사성에 기반하여 만들어졌다. WN Test는 여러 종류의 Graph를 구별할 수 있는 효과적인 테스트 방법이다. GNN과 비슷하게, WN Test도 노드가 주어졌을 때, 이 노드의 이웃들의 feature를 벡터로 통합하여 node feature를 반복적으로 업데이트한다. WL Test의 경우 다른 노드의 이웃들을 다른 feature 벡터로 mapping하는 Injective Aggregation Update의 존재로 인해 강력한 효과를 지니게 된다.

Node	Feature Vector
알라딘 서점	[1, 0, 0.5]
스타벅스 커피	[0, 1, 0.5]
이디야 커피	[0.5, 0.5, 0.5]

이 때 알라딘 서점과 스타벅스 커피는 사람 A의 이웃 Node이고, 이디야 커피는 사람 B의 이웃 Node라고 생각해보자. 그리고 사람 A와 사람 B의 Feature 벡터를 얻기 위해 GraphSAGE에서 제시한 Mean Aggregation 과정을 진행한다고 해보자.

그렇다면 사람 A의 이웃 Node의 Feature 벡터는 [0.5, 0.5, 0.5]가 될 것이고, 사람 B의 이웃 Node의 Feature 벡터 또한 [0.5, 0.5, 0.5]가 될 것이다. 물론 위 예시에 대해 애초에 Feature 값이 이들을 표현하기에 불충분하다고 생각할 수도 있지만, 충분히 실제 데이터 상에서 나타날 수 있는 예시일 것이다. 어쨌든 위 결과에 따르면 사람 A와 사람 B의 이웃 Feature 벡터는 똑같은 형상을 하게 될 것이고, 이는 학습에 있어서 데이터 소실 혹은 왜곡이라는 문제로 귀결되게 될 것이다.

본 논문에서는 GNN의 Aggregation Scheme이 굉장히 Expressive하고 Injective Function을 모델링할 수 있다면 GNN 또한 WL Test처럼 굉장히 강력한 Discriminative Power를 지니게 될 것이라고 이야기하고 있다.

수학적으로 위 인사이트를 공식화하기 위해, 본 논문의 Framework는 먼저 Node가 주어졌을 때, 이 Node의 이웃들의 Feature 벡터를 Multiset, 즉 repeating elements가 존재할 수 있는 집합으로 표현할 것이다. 그러면 GNN에 있는 Neighbor Aggregation이 Aggregation Function over the multiset으로 표현될 수 있을 것이다. 따라서 강력한 Representational Power를 가지기 위해서는 GNN은 다른 multiset을 다른 representation으로 표현할 수 있어야 한다.

본 논문에서는 몇몇 multiset function의 변형 버전과 그들의 discriminative power, 즉 aggregation function이 다른 multiset들을 얼마나 잘 구별할 수 있는지를 이론적으로 분석하였다. 더 discriminative 할 수록, GNN 속에 내재되어 있는 Representational Power는 더 강력할 것이다.

본 논문의 핵심적인 결과는 아래와 같다.

1) GNN은 Graph 구조를 판별하는데에 있어 WL Test 만큼이나 효과적이다.

2) 위 문장이 성립하기 위해서는 Neighbor Aggregation이나 Graph Readout Function에 관하여 조건들이 필요한데, 본 논문에서는 이를 제시하였다.

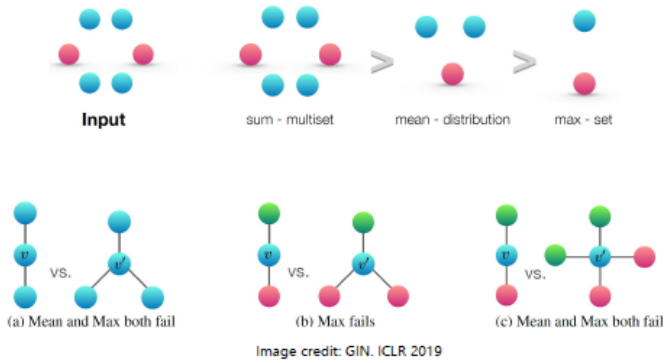
3) 본 논문에서는 GCN이나 GraphSAGE는 구분할 수 없는 Graph 구조에 대해 언급하고, GNN 기반의 모델들이 포착할 수 있는 Graph 구조들에 대해 정확하게 분석하였다.

4) 본 논문에서는 GIN이라고 부르는 간단한 신경망 구조를 고안하였고, 이 알고리즘이 WL Test 만큼이나 discriminative/representational power를 갖고 있음을 증명하였다.

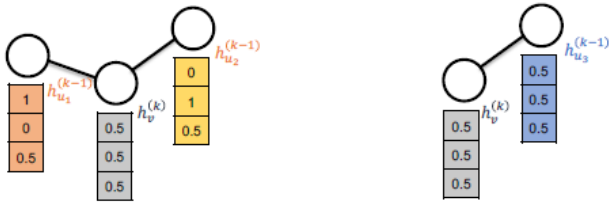
GIN adopts sum as the aggregation function instead of max or mean.

$$\mathbf{h}_v^{(k)} = \text{MLP}^{(k)}\left(\left(1+\epsilon^{(k)}\right)\cdot \mathbf{h}_v^{(k-1)} + \Sigma_{u\in N(v)}\mathbf{h}_u^{(k-1)}\right)$$

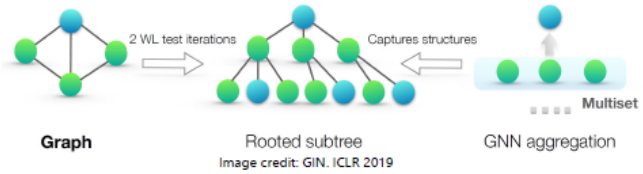
- $\epsilon^{(k)}$: a learnable parameter or a fixed scalar



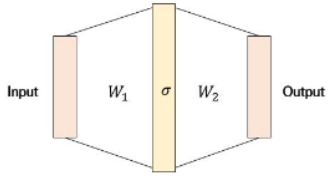
Problem: existing GNN cannot discriminate two non-isomorphic graphs



Idea: if a GNN's aggregation function captures the full multiset of node neighbors, the GNN can capture the rooted subtrees in a recursive manner and be as powerful as the WL test.



핵심 내용은 간단하다. 충분히 큰 Hidden Dimensionality와 적절한 비선형 함수를 갖는 1-hidden-layer MLP는 일정 수준의 정확도로 어떠한 연속형 함수도 근사할 수 있다는 것이 본 이론의 내용이다.



최초의 반복에서 만약 Input Feature가 One-hot 인코딩이면 그들의 합 또한 `injective` 할 것이기 때문에 합 연산 이전에 MLP가 필요하지는 않다. 물론 One-hot 인코딩 되어 있지 않거나 연속형 변수가 중간에 끼어 있다면 MLP가 필요할 것이다. ϵ 의 경우 학습 가능한 파라미터 혹은 고정된 스칼라로 둘 수 있다. 최종적으로 `GIN` 은 Node Representation을 아래와 같이 업데이트하게 될 것이다.

$$h_v^k = MLP^k((1+\epsilon^k)\cdot h_v^{k-1} + \Sigma_{u\in\mathcal{N}(v)} h_u^{k-1})$$

효과적인 GNN은 많이 존재하지만 `GIN` 은 굉장히 간단하면서도 굉장히 효과적인 GNN의 대표적인 예라고 할 수 있다.

D. Graph Neural Network's Drawbacks

1) Deep GNNs

GNNs의 층이 깊어질수록 성능이 급격하게 떨어짐.

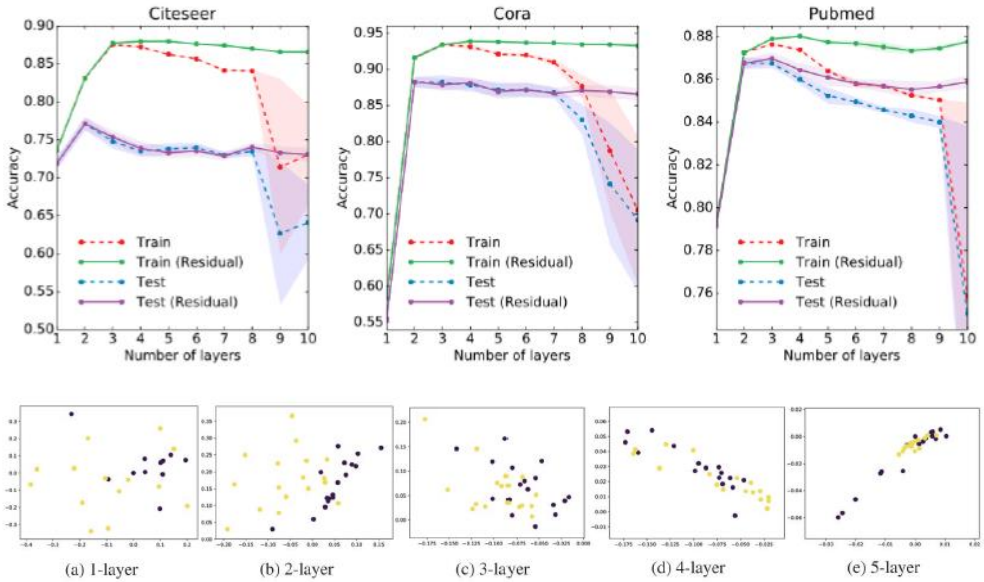
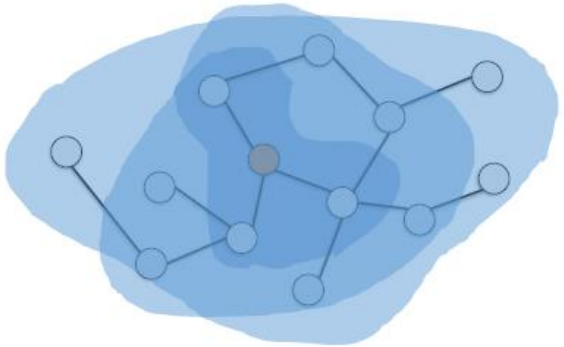


Figure 2: Vertex embeddings of Zachary's karate club network with GCNs with 1,2,3,4,5 layers.

- Complex structure
  - A variety of degrees and sizes
- No fixed node ordering or reference point
  - Unlike grids, sequences
- Heterogeneous graphs
  - Graphs that contain different types of nodes and edges.
- Dynamic graphs
  - Nodes/edges are inserted/deleted as time goes on.
- Over-smoothing, over-squashing
- Scalability

2) Over-Smoothing

Stacking GCN layers makes representation inseparable!



Over-smoothing은 노드들의 임베딩 값이 다 같아지는 현상이다. Receptive field가 GNN층이 깊어지면서 겹치게 되는데, 쉽게 말하면 GNN이 깊어지면서 더 넓은 범위의 이웃 정보를 가져온다. 하지만, GNN 층 수 만큼만 이웃 정보를 가져와야 하는데, path가 층 수를 넓히는 과정에서 더욱 더 길어질 수 있기 때문이다. 따라서, 이런 경우 모든 노드들의 임베딩이 이 길어진 path에 영향을 받아 값이 다 같아지는 현상이다.

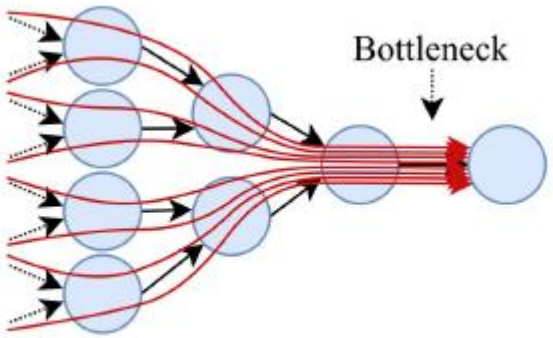
- Multiple propagation makes representation inseparable!
- **Theorem 1.** In a connected graph, given a propagation matrix P and node features $x \in \mathbb{R}^n$, we have

$$\lim_{k\to\infty} P^k x \propto u_1,$$

  where u_1 is the eigenvector of P corresponds to its largest eigenvalue.
- (1) When $P = D^{-1}A$, u_1 is $\mathbf{1}$
- (2) When $P = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, u_1 of is $D^{-\frac{1}{2}}\mathbf{1}$

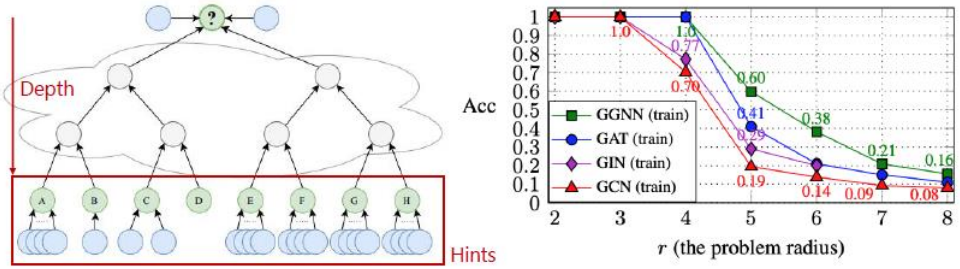
3) Over-Squashing

Over-smoothing은 모든 노드들의 관점에서 해석한 것이라면 Over-Squashing은 Target node에 대한 관점이다. 하나의 Target이 특정 Depth에 있는 정보들까지 aggregation할 때, 정보가 기하급수적으로 많아질 수 있다는 것이다.



(b) The bottleneck of graph neural networks

이처럼 너무 많은 정보가 message passing되고 aggregation되어 target node로 전달되기 때문에 일종의 정보 과잉으로 인한 bottleneck이 생긴다.



Takeaways

- Graphs are the powerful representations to model relationships of entities
  - Computation on abstract concepts.
  - Computation on different data types.
- Mining graphs
  - Many interesting problems on the boundary of algorithms & systems
  - Distributed systems allow handling massive graphs or many graphs.
- Learning from graphs (graph neural networks)
  - Diverse graph-related applications
  - Expressive GNNs that differentiate between different graphs
  - Scalable GNNs that deal with large-scale graphs