Semi-Supervised Classification With Graph Convolutional Networks

Thomas N. Kipf, Max Welling

University of Amsterdam

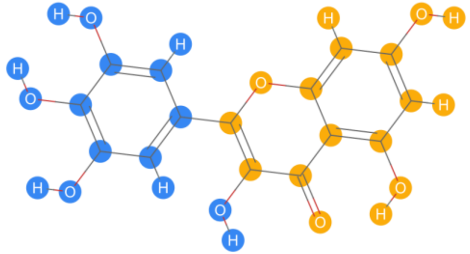
DMAIS@CAU

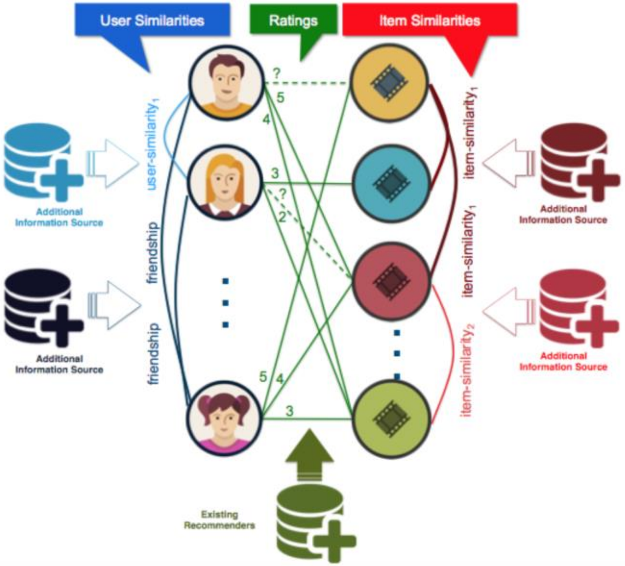
Joohee Cho

Contents

What Are Graphs Used?

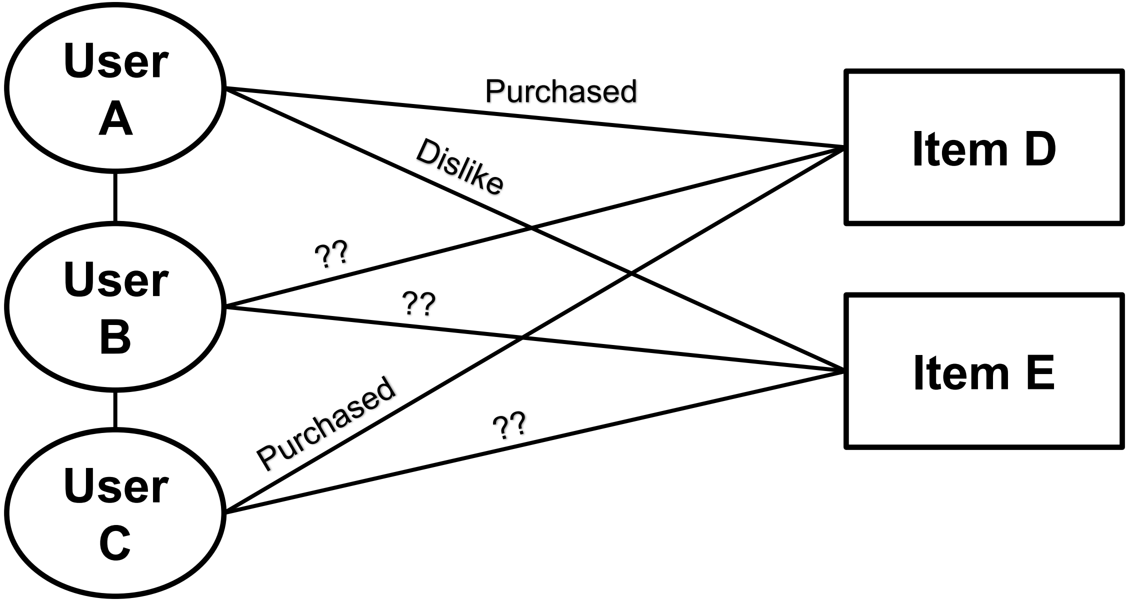
- Used in fields where relationships between entities are important
  - Social Media, Item Recommendation System, Bioinformatics etc.





What Problem Do They Have?

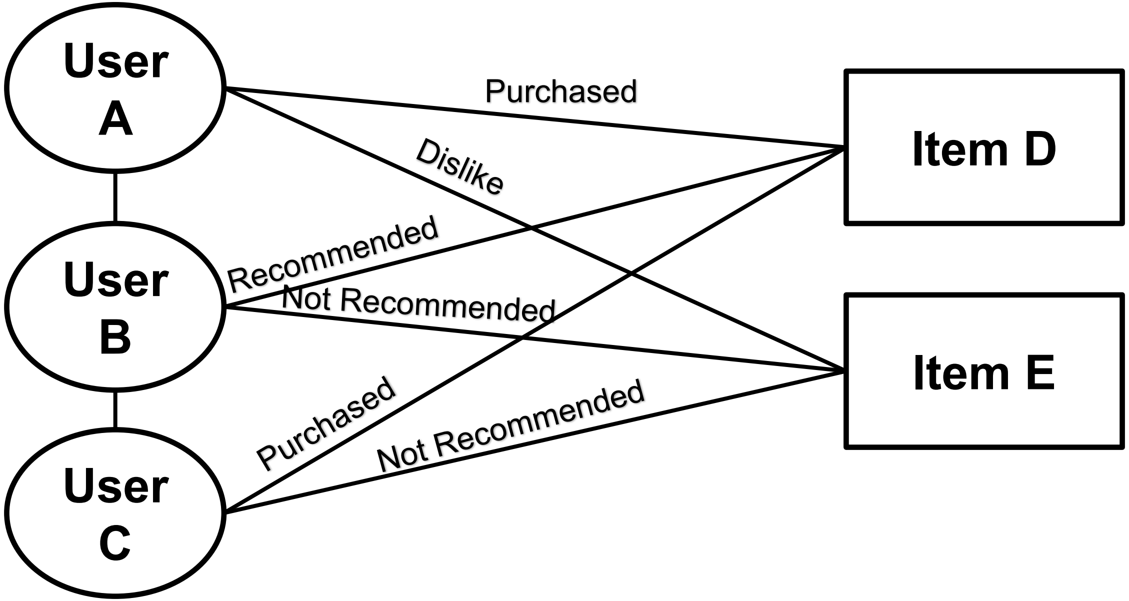
- We don’t have all the information of nodes in real world, but still need it



Example of Item Recommendation System

How Can We Solve It?

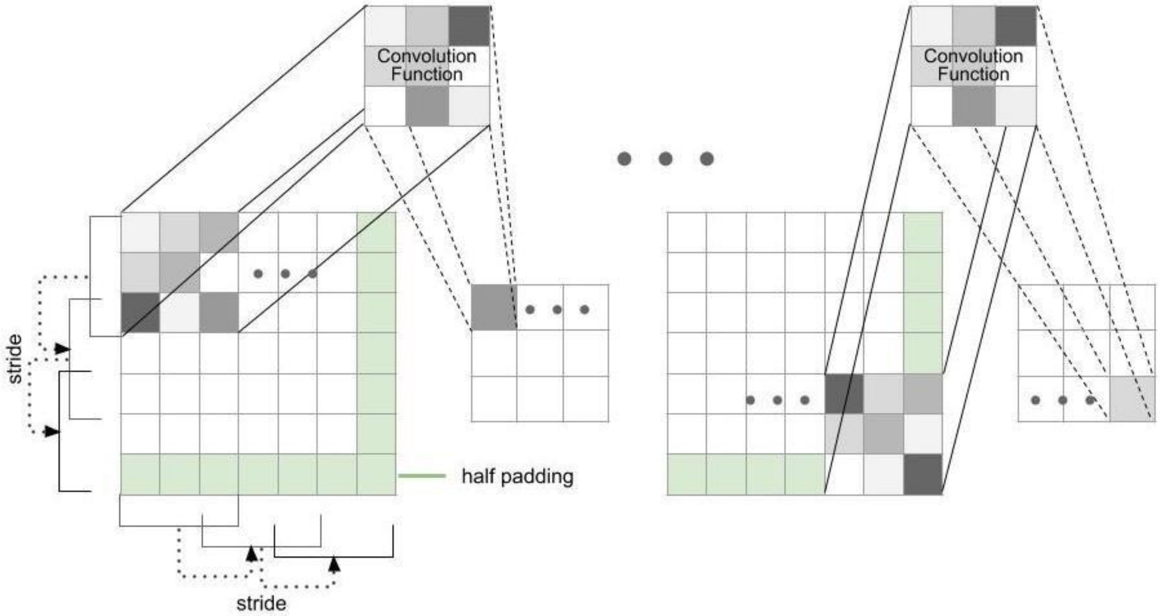
- Nodes that are connected share the similar characteristics



Example of Item Recommendation System

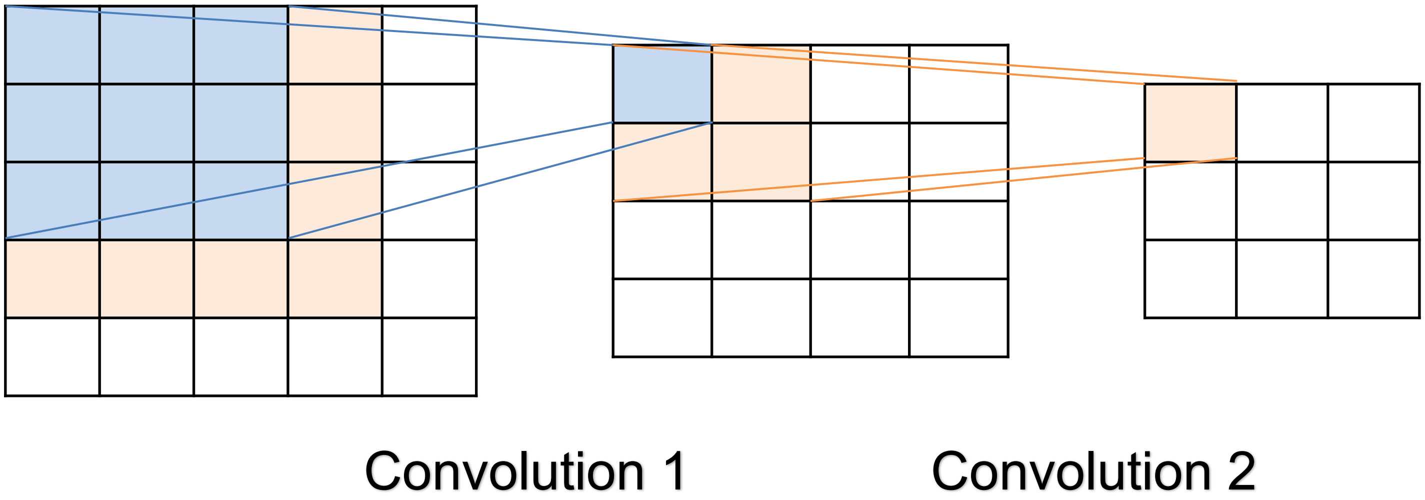
Reminder of Convolutions on Images

- ☐ **Operation that combines features using shared kernel with locality**



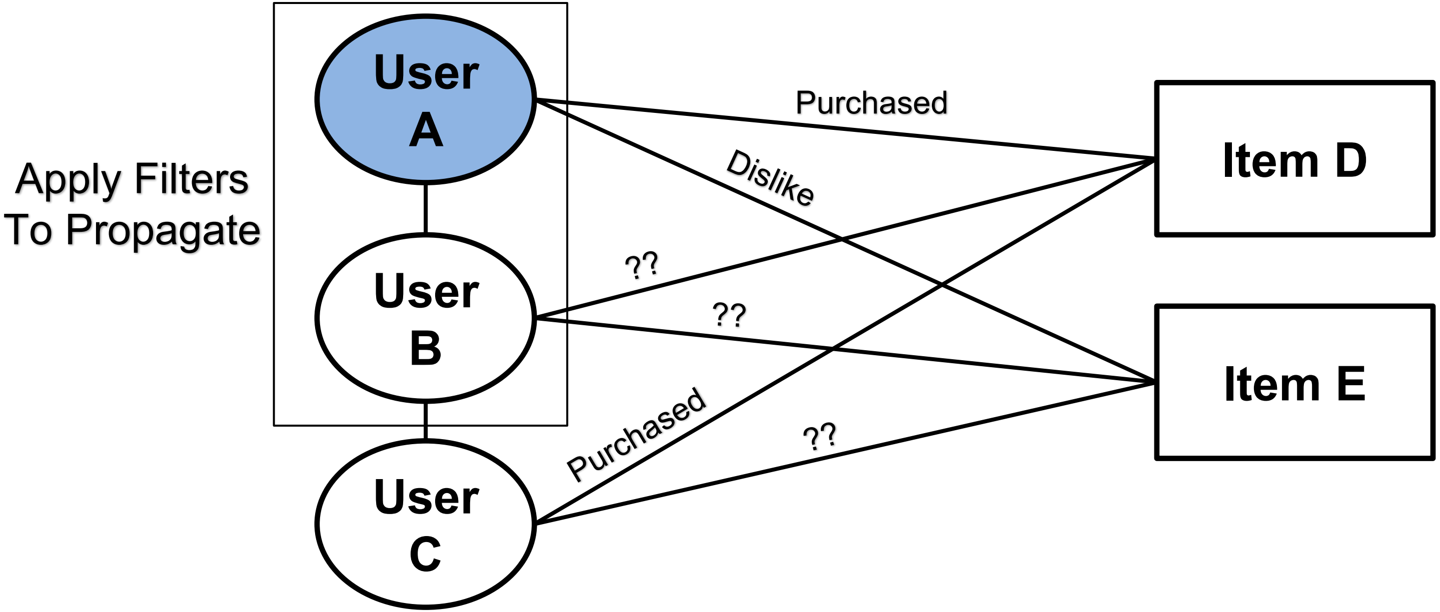
Reminder of Convolutions on Images (cont.)

- By adding layers, convolutions can aggregate features further away



Convolution on Graphs

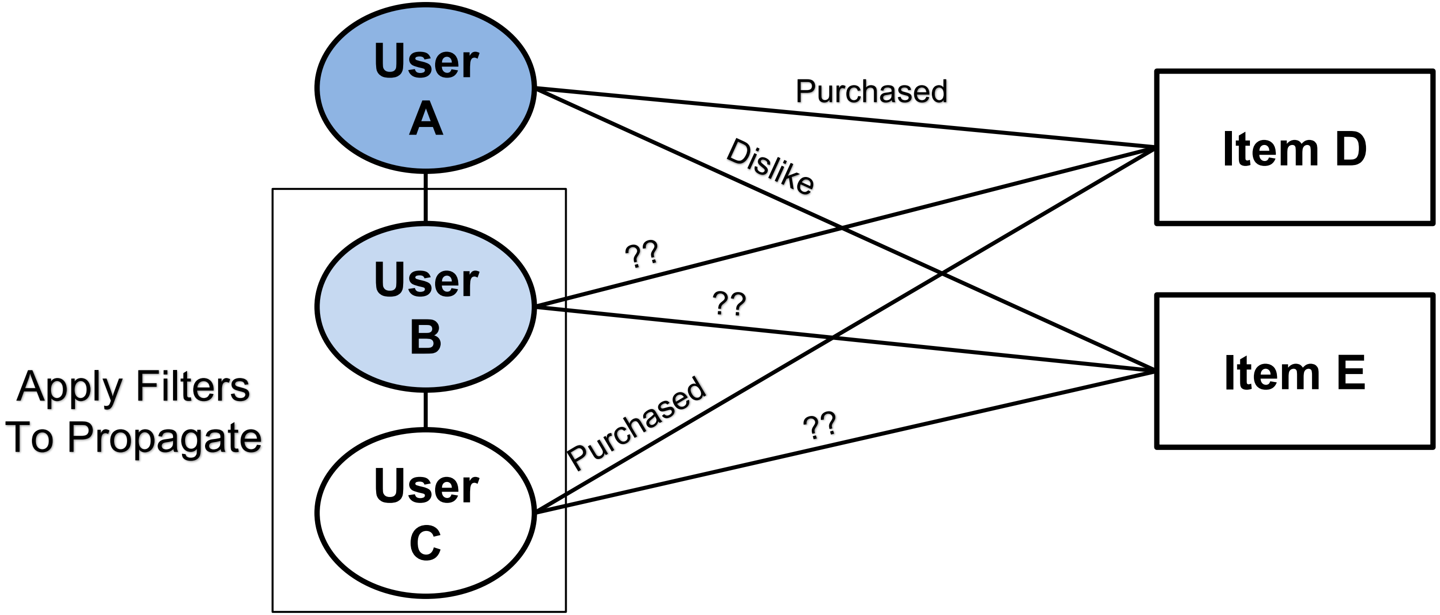
- Aggregates nodes nearby by edges instead of real distance



Example of Item Recommendation System

Convolution on Graphs (cont.)

- Add layers to reach further nodes



Example of Item Recommendation System

Convolution on Graphs (cont.)

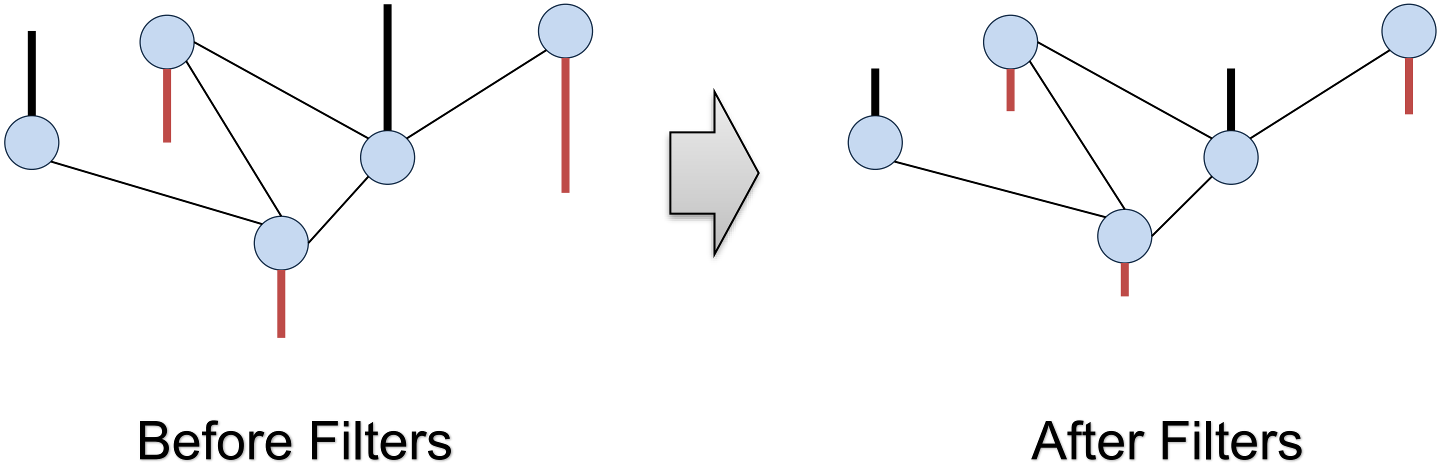
- ☐ **What do filters do?**

- ☐ **How to apply filters?**

- ☐ **How to reach further?**

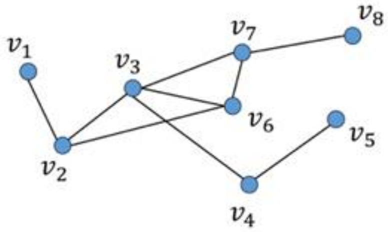
Goal of Applying Filters

- ☐ **Smoothen feature representations of graphs**
  - Make connected nodes close to each other in embedding space



About Graph Laplacian

☐ **Express how much nodes are different from their neighbors**



Adjacency Matrix: $A[i,j] = 1$ if v_i is adjacent to v_j
$A[i,j] = 0$, otherwise

Degree Matrix: $\mathbf{D} = \mathrm{diag}(degree(v_1), \ldots, degree(v_N))$

$$\underset{\text{Degree Matrix}}{\begin{pmatrix} 1&0&0&0&0&0&0&0\\ 0&3&0&0&0&0&0&0\\ 0&0&4&0&0&0&0&0\\ 0&0&0&2&0&0&0&0\\ 0&0&0&0&2&0&0&0\\ 0&0&0&0&0&4&0&0\\ 0&0&0&0&0&0&3&0\\ 0&0&0&0&0&0&0&1 \end{pmatrix}}_{\boldsymbol{D}} - \underset{\text{Adjacency Matrix}}{\begin{pmatrix} 0&1&0&0&0&0&0&0\\ 1&0&1&0&0&1&0&0\\ 0&1&0&1&0&1&1&0\\ 0&0&1&0&1&0&0&0\\ 0&0&0&1&0&1&0&0\\ 0&1&1&0&1&0&1&0\\ 0&0&1&0&0&1&0&1\\ 0&0&0&0&0&0&1&0 \end{pmatrix}}_{\boldsymbol{A}} = \underset{\text{Laplacian Matrix}}{\begin{pmatrix} 1&-1&0&0&0&0&0&0\\ -1&3&-1&0&0&-1&0&0\\ 0&-1&4&-1&0&-1&-1&0\\ 0&0&-1&2&-1&0&0&0\\ 0&0&0&-1&2&-1&0&0\\ 0&-1&-1&0&-1&4&-1&0\\ 0&0&-1&0&0&-1&3&-1\\ 0&0&0&0&0&0&-1&1 \end{pmatrix}}_{\boldsymbol{L}}$$

About Graph Laplacian (cont.)

- ☐ **Express how much nodes are different from their neighbors**

$$\boldsymbol{h} = \boldsymbol{L}\boldsymbol{f} = (\boldsymbol{D} - \boldsymbol{A})\boldsymbol{f} = \boldsymbol{D}\boldsymbol{f} - \boldsymbol{A}\boldsymbol{f}$$

$$h_i = \boldsymbol{L}_i\,\boldsymbol{f} = (\boldsymbol{D}_i - \boldsymbol{A}_i)\boldsymbol{f} = \boldsymbol{D}_i\boldsymbol{f} - \boldsymbol{A}_i\boldsymbol{f}$$

$$= \boldsymbol{D}_{i,i}f_i - \boldsymbol{A}_i\boldsymbol{f}$$

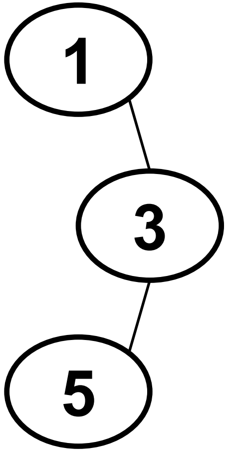
$$\boxed{h_i = \sum\nolimits_{v_j \in \mathcal{N}(v_i)} (f_i - f_j)}$$

v_i : i $th\ node$
$N(v_i)$: $set\ of\ the\ neighbors\ of\ v_i$
f : feature $matrix$
v_j : $neighbor\ node\ of\ v_i$

Loss Function of Graph Laplacian

□ Show the smoothness of the graph

- Might restrict model capacity as not all node features have similarities



Laplacian quadratic form:

$$
A = \begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}
$$

$$\boldsymbol{h} = \boldsymbol{L}\boldsymbol{f}, \quad h_i = \sum\nolimits_{v_j \in \mathcal{N}(v_i)} (f_i - f_j)$$

$$
\begin{aligned}
\boldsymbol{f}^T \boldsymbol{L} \boldsymbol{f} &= \sum\nolimits_i f_i h_i = \sum\nolimits_i \sum\nolimits_{v_j \in \mathcal{N}(v_i)} f_i (f_i - f_j) \\
&= \frac{1}{2} \sum\nolimits_{i,j} A(i,j) f_i (f_i - f_j) + \frac{1}{2} \sum\nolimits_{i,j} A(j,i) f_i (f_i - f_j) \\
&= \frac{1}{2} \sum\nolimits_{i,j} A(i,j) f_i (f_i - f_j) - \frac{1}{2} \sum\nolimits_{i,j} A(i,j) f_j (f_i - f_j) \\
&= \frac{1}{2} \sum\nolimits_{i,j} A(i,j) (f_i - f_j)^2
\end{aligned}
$$

➡ The smaller, the similar the connected nodes

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{\text{reg}}, \quad \text{with} \quad \mathcal{L}_{\text{reg}} = \sum_{i,j} A_{ij} \| f(X_i) - f(X_j) \|^2 = f(X)^\top \Delta f(X)$$

GCN Layer-wise Propagation Rule

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

$$Propagation\ Rule$$

$$(H^{0} = Feature\ Matrix)$$

Reminders of Fourier Transform

- Fourier transform is operation that decomposes signal into frequences using sinusoidal basis functions

Define

$$\omega = \exp\left(-\frac{2\pi}{N}i\right)$$

so that

$$\exp\left(-jk\frac{2\pi}{N}i\right) = \omega^{jk}$$

Then, we can define the $N \times N$ **Fourier matrix**

$$F = \begin{bmatrix} \omega^{0\cdot0} & \omega^{0\cdot1} & \omega^{0\cdot2} & \cdots & \omega^{0\cdot(N-1)} \\ \omega^{1\cdot0} & \omega^{1\cdot1} & \omega^{1\cdot2} & \cdots & \omega^{1\cdot(N-1)} \\ \omega^{2\cdot0} & \omega^{2\cdot1} & \omega^{2\cdot2} & \cdots & \omega^{2\cdot(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega^{(N-1)\cdot0} & \omega^{(N-1)\cdot1} & \omega^{(N-1)\cdot2} & \cdots & \omega^{(N-1)\cdot(N-1)} \end{bmatrix}$$

We can use F to write the DFT in matrix form

$$X = Fx$$

Inverse Fourier matrix

The inverse transform can be written as

$$x = F^{-1}X$$

The entries of the inverse Fourier matrix F^{-1} have already been derived above:

$$F^{-1} = \frac{1}{N}\begin{bmatrix} \omega^{-0\cdot0} & \omega^{-0\cdot1} & \omega^{-0\cdot2} & \cdots & \omega^{-0\cdot(N-1)} \\ \omega^{-1\cdot0} & \omega^{-1\cdot1} & \omega^{-1\cdot2} & \cdots & \omega^{-1\cdot(N-1)} \\ \omega^{-2\cdot0} & \omega^{-2\cdot1} & \omega^{-2\cdot2} & \cdots & \omega^{-2\cdot(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega^{-(N-1)\cdot0} & \omega^{-(N-1)\cdot1} & \omega^{-(N-1)\cdot2} & \cdots & \omega^{-(N-1)\cdot(N-1)} \end{bmatrix}$$

Fourier Transform on Graphs

- **Decompose using eigenvectors of graph Laplacian, and add a filter**

$Set\ g_\theta\ as\ a\ filter,$

$\mathrm{L}\ as\ a\ normalized\ graph\ Laplacian\ \ I\ -\ D^{\frac{1}{2}}AD^{\frac{1}{2}},$
$U\ as\ an\ eigenvectors\ of\ normalized\ graph\ Laplacian\ L,$
$x\ as\ a\ signal\ (feature\ vector)$

$Apply\ fourier\ transform:$	$\hat{x} = U^T x$
$Apply\ filter:$	$g_\theta\,\hat{x} = g_\theta U^T x$
$Apply\ Inverse\ fourier\ transform:$	$g_\theta \hat{x} = U g_\theta U^T x$

$Time\ complexity: O(N^2)$

Simplification for Less Computation

- Apply Chebyshev polynomials and limit k value to 1

$L = U^T \wedge U$

$Rescale\ L\ to\ fit\ range\ of\ [-1, 1] : \tilde{L} = \frac{2}{\lambda_{max}} \Lambda$

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda}) , \quad g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x$$

$\Downarrow$ Set k to 1

$$g_{\theta'} \star x \approx \theta'_0 x + \theta'_1 \left(L - I_N\right) x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x$$

Simplification for Less Computation (cont.)

- ☐ **Reduce parameters and avoid gradient exploding**

$$g_{\theta'} \star x \approx \theta'_0 x + \theta'_1 (L - I_N) x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x$$

$\Downarrow$ *Integrate* θ_0 *and* θ_1 *into one* θ

$$g_\theta \star x \approx \theta \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) x$$

$\Downarrow$ *Use renormalization trick to set the eigenvalue range to* $[0, 1]$

$$I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}, \quad \tilde{A} = A + I_N \text{ and } \tilde{D}_{ii} = \textstyle\sum_j \tilde{A}_{ij}$$

$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta$, Z : *convolved signal matrix*, Θ : *filter parameter*

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)} \right)$$

Implementation for Experiments

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

$$Z = f(X, A) = \mathrm{softmax}\left(\hat{A}\ \mathrm{ReLU}\left(\hat{A} X W^{(0)}\right) W^{(1)}\right)$$

Table 1: Dataset statistics, as reported in Yang et al. (2016).

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

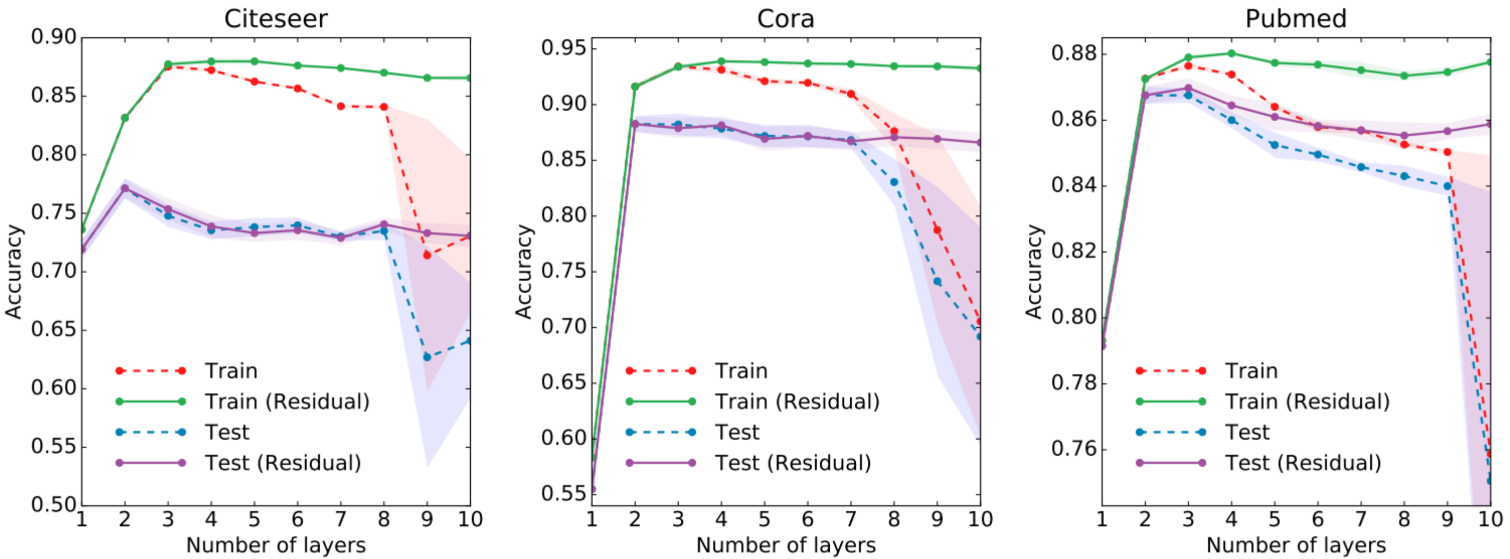
Comparison to Other Models

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

Evaluation of Propagation Model

Description		Propagation model	Citeseer	Cora	Pubmed
Chebyshev filter (Eq. 5)	$K=3$	$\sum_{k=0}^{K} T_k(\tilde{L})X\Theta_k$	69.8	79.5	74.4
	$K=2$		69.6	81.2	73.8
1st-order model (Eq. 6)		$X\Theta_0 + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}X\Theta_1$	68.3	80.0	77.5
Single parameter (Eq. 7)		$(I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})X\Theta$	69.3	79.2	77.4
Renormalization trick (Eq. 8)		$\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta$	**70.3**	**81.5**	**79.0**
1st-order term only		$D^{-\frac{1}{2}}AD^{-\frac{1}{2}}X\Theta$	68.7	80.5	77.8
Multi-layer perceptron		$X\Theta$	46.5	55.1	71.4

Limitation of Hops



Variational Graph Auto-Encoders

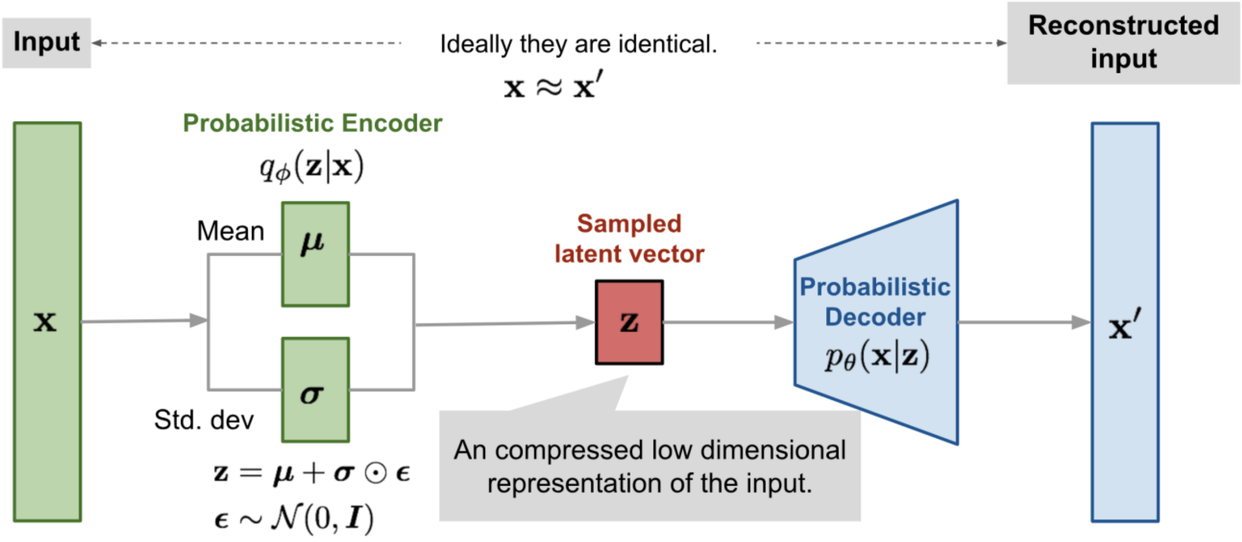
Thomas N. Kipf, Max Welling
University of Amsterdam

DMAIS@CAU
Joohee Cho

Contents

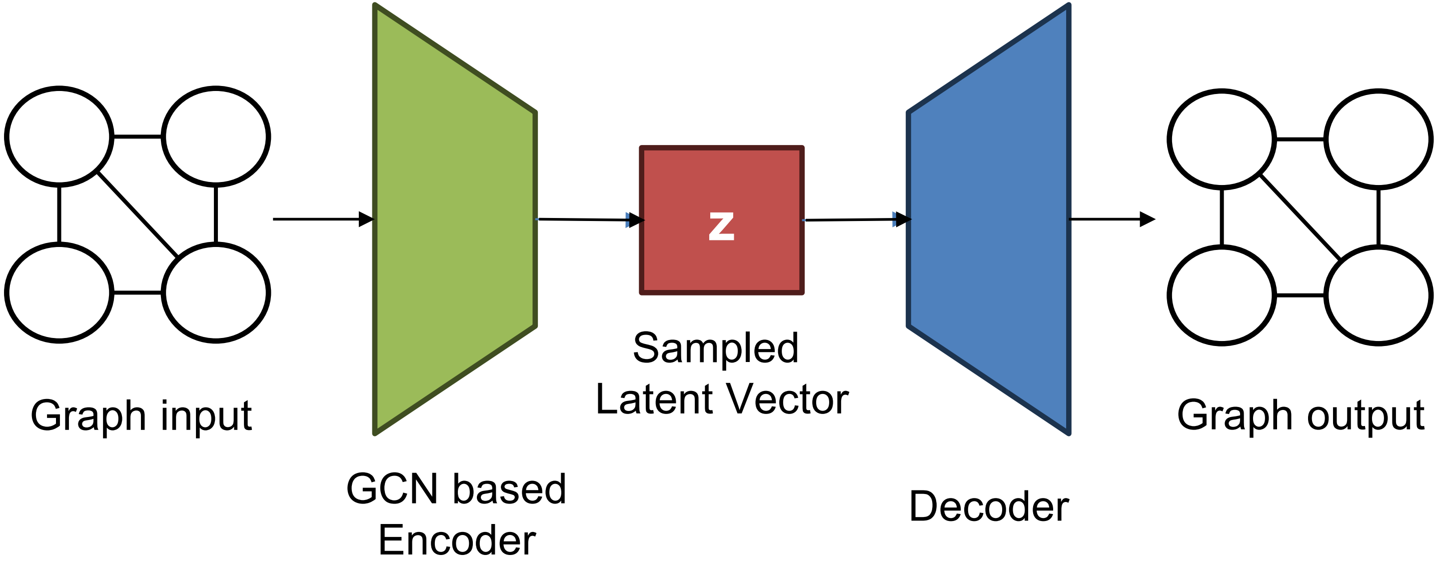
Reminders of Variational Auto-Encoders

- Generative model that allow sampling while encoding data into a latent space and decode back



VAE on Graphs

- ☐ **Apply Variational Auto-Encoder logics on graph structures, to generate better link predictions**



Inference Model of VGAE

- Sample based on probability distribution made from GCN

$$q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) = \prod_{i=1}^{N} q(\mathbf{z}_i \mid \mathbf{X}, \mathbf{A}), \quad \text{with} \quad q(\mathbf{z}_i \mid \mathbf{X}, \mathbf{A}) = \mathcal{N}(\mathbf{z}_i \mid \boldsymbol{\mu}_i, \operatorname{diag}(\boldsymbol{\sigma}_i^2))$$

$$\boldsymbol{\mu} = \mathrm{GCN}_{\boldsymbol{\mu}}(\mathbf{X}, \mathbf{A}) = \tilde{\mathbf{A}}\,\mathrm{ReLU}(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}_0)\mathbf{W}_1$$

$$\log \boldsymbol{\sigma} = \mathrm{GCN}_{\boldsymbol{\sigma}}(\mathbf{X}, \mathbf{A}) = \tilde{\mathbf{A}}\,\mathrm{ReLU}(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}_0)\mathbf{W}_2$$

Generative Model of VGAE

- Generate graph by an inner product between latent variables

$$p(\mathbf{A} \mid \mathbf{Z}) = \prod_{i=1}^{N} \prod_{j=1}^{N} p(A_{ij} \mid \mathbf{z}_i, \mathbf{z}_j) \ , \ \text{with} \quad p(A_{ij} = 1 \mid \mathbf{z}_i, \mathbf{z}_j) = \sigma(\mathbf{z}_i^\top \mathbf{z}_j)$$

$z \; using \; reparametrization \; trick$:

$$z_i = \mu_i + \sigma_i \odot \epsilon_i$$

$$z_j = \mu_j + \sigma_j \odot \epsilon_j$$

Learning of VGAE

- Learning is based on ELBO

$$\mathcal{L} = \underbrace{\mathbb{E}_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}\left[\log p\left(\mathbf{A} \mid \mathbf{Z}\right)\right]}_{\text{Show how well the model reconstructed the edges}} - \underbrace{\mathrm{KL}\left[q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) \,||\, p(\mathbf{Z})\right]}_{\text{KL divergence value}}$$

Graph Auto-Encoder

- ☐ **Non-probabilistic model just use the inner product between GCN latent vectors**

$$\hat{\mathbf{A}} = \sigma\left(\mathbf{Z}\mathbf{Z}^\top\right), \quad \text{with} \quad \mathbf{Z} = \mathrm{GCN}(\mathbf{X}, \mathbf{A})$$

$$\mathrm{GCN}(\mathbf{X}, \mathbf{A}) = \tilde{\mathbf{A}}\,\mathrm{ReLU}\left(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}_0\right)\mathbf{W}_1$$

Experiments

Method	Cora		Citeseer		Pubmed	
	AUC	AP	AUC	AP	AUC	AP
SC [5]	84.6 ± 0.01	88.5 ± 0.00	80.5 ± 0.01	85.0 ± 0.01	84.2 ± 0.02	87.8 ± 0.01
DW [6]	83.1 ± 0.01	85.0 ± 0.00	80.5 ± 0.02	83.6 ± 0.01	84.4 ± 0.00	84.1 ± 0.00
GAE*	84.3 ± 0.02	88.1 ± 0.01	78.7 ± 0.02	84.1 ± 0.02	82.2 ± 0.01	87.4 ± 0.00
VGAE*	84.0 ± 0.02	87.7 ± 0.01	78.9 ± 0.03	84.1 ± 0.02	82.7 ± 0.01	87.5 ± 0.01
GAE	91.0 ± 0.02	92.0 ± 0.03	89.5 ± 0.04	89.9 ± 0.05	$\mathbf{96.4} \pm 0.00$	$\mathbf{96.5} \pm 0.00$
VGAE	$\mathbf{91.4} \pm 0.01$	$\mathbf{92.6} \pm 0.01$	$\mathbf{90.8} \pm 0.02$	$\mathbf{92.0} \pm 0.02$	94.4 ± 0.02	94.7 ± 0.02

Inductive Representation Learning On Large Graphs

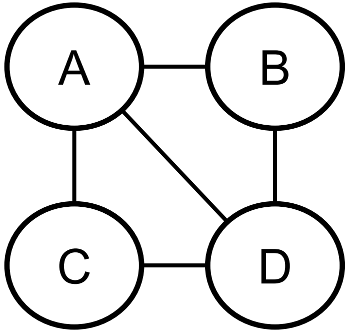
William L. Hamilton, Rex Ying, Jure Leskovec
Stanford University

DMAIS@CAU
Joohee Cho

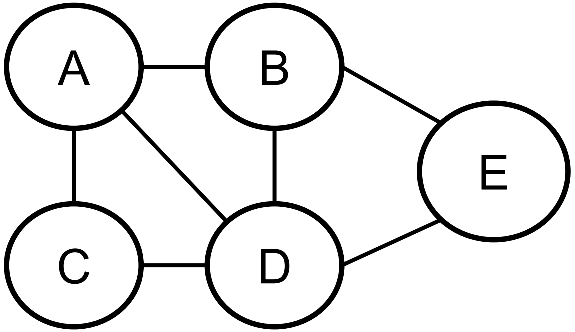
Contents

Limitation of Previous Models

☐ **Embed nodes from fixed graph and difficult to deal with unseen nodes**



GCN learned on a graph
of four nodes

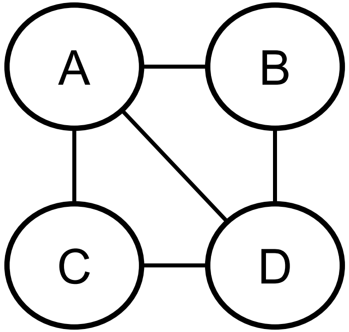


New node added to the graph

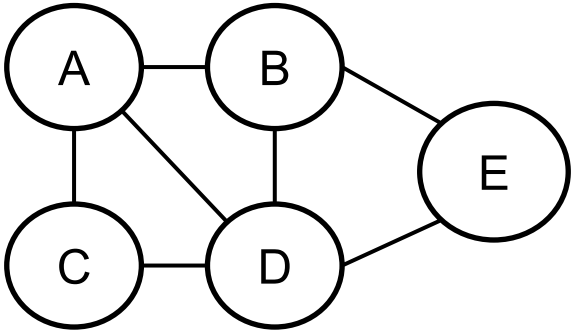
How to classify E?

How To Solve It?

- Make graph inductive by learning how to make relationship with its neighbors



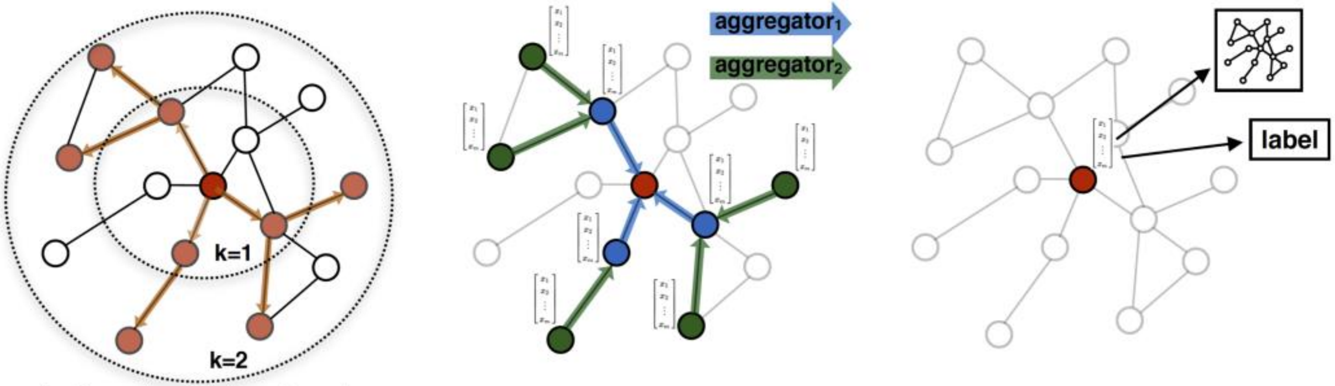
Learn how B collects features from its two neighbors



New node E can work like B

What is Graph SAGE

- A general framework to SAmple and aggreGatE Graph node neighbors



1. Sample neighborhood

2. Aggregate feature information from neighbors

3. Predict graph context and label using aggregated information

How to Embed Generations

- Aggregate neighborhood features, concat with current node, and normalize

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 $\mathcal{N} : v \to 2^{\mathcal{V}}$

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

```
h_v^0 ← x_v, ∀v ∈ V ;
for k = 1...K do
    for v ∈ V do
        h_N(v)^k ← AGGREGATE_k({h_u^{k-1}, ∀u ∈ N(v)});
        h_v^k ← σ(W^k · CONCAT(h_v^{k-1}, h_N(v)^k))
    end
    h_v^k ← h_v^k / ||h_v^k||_2, ∀v ∈ V
end
z_v ← h_v^K, ∀v ∈ V
```

Where:

$\mathbf{h}^k_{\mathcal{N}(v)} \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}^{k-1}_u, \forall u \in \mathcal{N}(v)\});$

$\mathbf{h}^k_v \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}^{k-1}_v, \mathbf{h}^k_{\mathcal{N}(v)})\right)$

$\mathbf{h}^k_v \leftarrow \mathbf{h}^k_v / \|\mathbf{h}^k_v\|_2, \forall v \in \mathcal{V}$

$\mathbf{z}_v \leftarrow \mathbf{h}^K_v, \forall v \in \mathcal{V}$

What is a Weisfeiler-Lehman Test

- A combinatory heuristic to test graph isomorphism

Given two graphs G_1, G_2:

1. In every graph, assign the same initial colour to every node (or start from given features)
2. Repeat until colors are stable:
   Update the colour of every node as

neigh. colours

$$c_{v,G_1}^{(t)} = \mathrm{HASH}(c_{v,G_1}^{(t-1)}, \{\!\{c_{w,G_1}^{(t-1)}\}\!\}_{w \in \mathcal{N}(v)})$$

prev. colour

$$c_{v,G_2}^{(t)} = \mathrm{HASH}(c_{v,G_2}^{(t-1)}, \{\!\{c_{w,G_2}^{(t-1)}\}\!\}_{w \in \mathcal{N}(v)})$$

If $\{\!\{c_{v,G_1}^{(t)}\}\!\}_{v \in G_1} \neq \{\!\{c_{v,G_2}^{(t)}\}\!\}_{v \in G_2}$ non-isomorphic.

What is a Weisfeiler-Lehman Test (cont.)

- A combinatory heuristic to test graph isomorphism

1. Assign the same initial colour to every node

2. Update the colour of every node

3. Different label multisets ⇒ non isomorphic!

Weisfeiler-Lehman Test and Graph-SAGE

- Graph-SAGE can be converted into WL-Test, and therefore can be used to learn topology structures

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

How to Aggregate - Mean Aggregator

- Add mean value of neighborhoods to current node

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**

        $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow Mean_k\left(\{h_u^{k-1}, \forall u \in N(v)\}\right)$

        $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$

    end

    $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$

end

$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Reminder of GCN

$$\tilde{A} = A + I_N$$

$$\Big\downarrow \text{ Normalize}$$

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

$$\Big\downarrow \text{ Get influenced by neighborhoods}$$

$$\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}$$

How to Aggregate - GCN Aggregator

☐ **A simplified version of GCN**

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**

    $\mathbf{h}_v^k \leftarrow Mean_k(\{h_u^{k-1}, \forall u \in N(v)\})$

  end

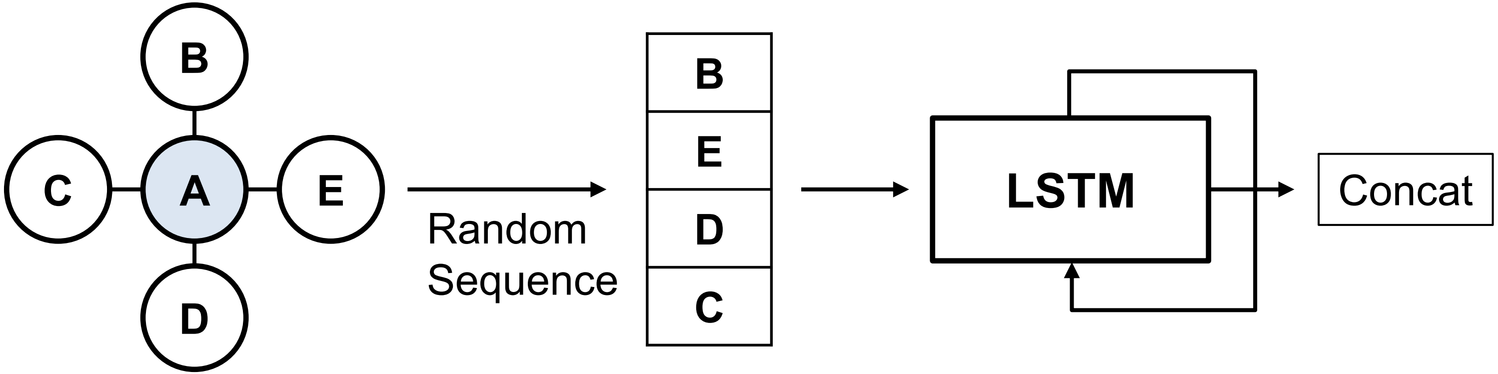
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$

end

$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

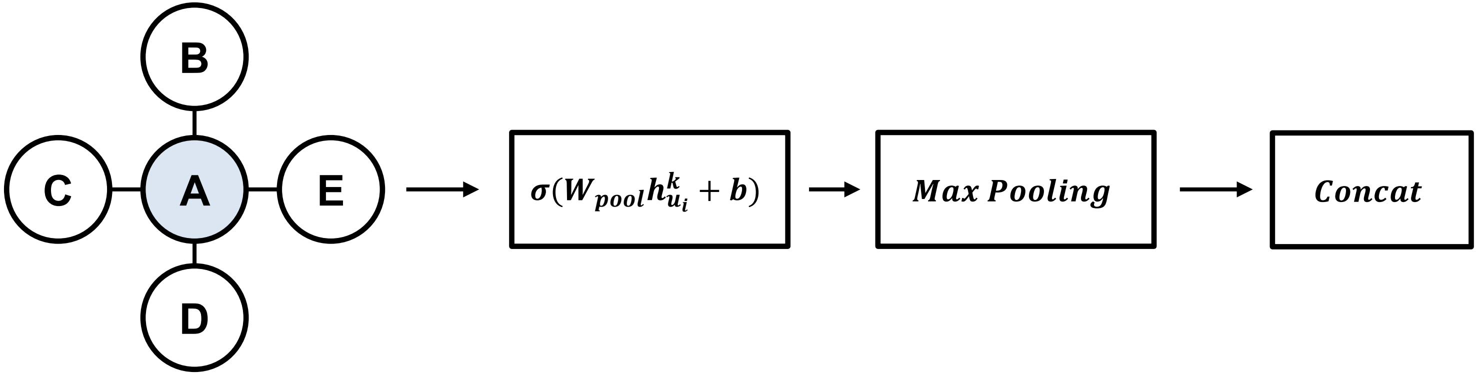
How to Aggregate - LSTM Aggregator

- ☐ **Randomly sample neighbors and apply LSTM**
  - As it is sequential, the aggregation is not permutation invariant



How to Aggregate - Pooling Aggregator

- **Aggregate through max pooling of the results of neural network**
  - Mean pooling can also be applied, however the test results are similar



Loss Function

- Keep nearby nodes to have similar representations, while setting disparate nodes to be distinct

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

How to Apply Mini-Batch

- Precompute which nodes need to be used and compute only the nodes related to the nodes in the mini-batch

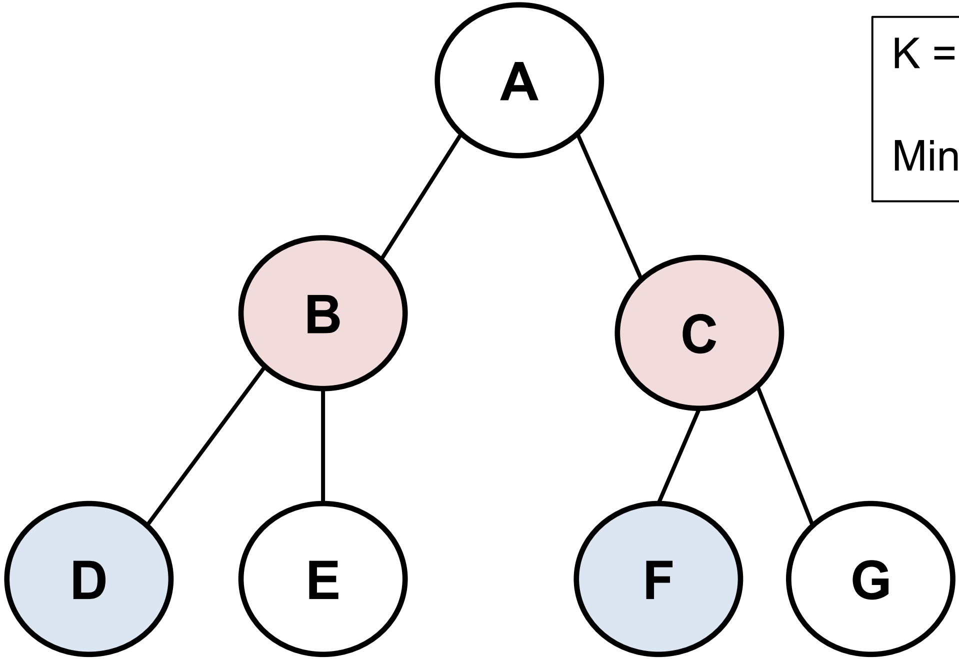
Algorithm 2: GraphSAGE minibatch forward propagation algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$;
input features $\{\mathbf{x}_v, \forall v \in \mathcal{B}\}$;
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 sampling functions, $\mathcal{N}_k : v \rightarrow 2^{\mathcal{V}}, \forall k \in \{1, ..., K\}$

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

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

How to Apply Mini-Batch (cont.)

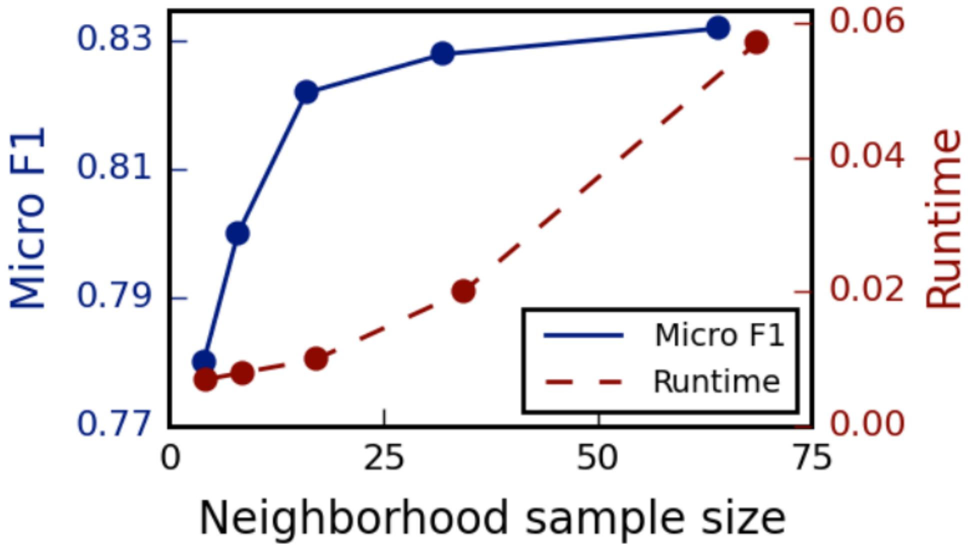


K = 1

Mini-Batch = [D, F]

Impact of the Number of Sampling Number

- Sampling is used for reducing time complexity
  - The bigger the sampling size is, the more accurate and time consuming



Comparison to Other Models

- GraphSAGE outperforms other models
  - LSTM and pool based models usually have the best performance

	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%

Inference Time Comparison

- ☐ **Training time is comparable, but takes far less inference time**