Semi-Supervised Classification with Graph Convolutional Networks

Thomas N.Kipf, Max Welling

ICLR 2017

InHyeok Jeong
School of Computer Science and Engineering
Chung-Ang University
DMAIS Lab

2025-02-25

Index

Motivation

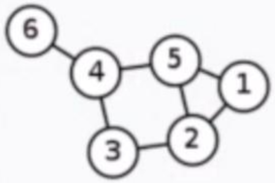
- **Graph-based semi-supervised learning**
  - $L = L_0 + \lambda L_{reg}$, with $L_{reg} = \sum_{i,j} A_{ij} \left|\left|f(X_i) - f(X_j)\right|\right|^2 = f(X)^T \Delta f(X)$
  - L_0 : supervised loss w.r.t the labeled part of the graph
  - L_{reg} : Induce the predictions of nearby nodes in the graph to be similar

    → *Restrict modeling capacity!*

Background

□ Laplacian Matrix

- $L = D - A$
- D : Degree Matrix, A : Adjacency Matrix

Labeled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$

- $x \times L$: Differences in values between one node and its neighbors

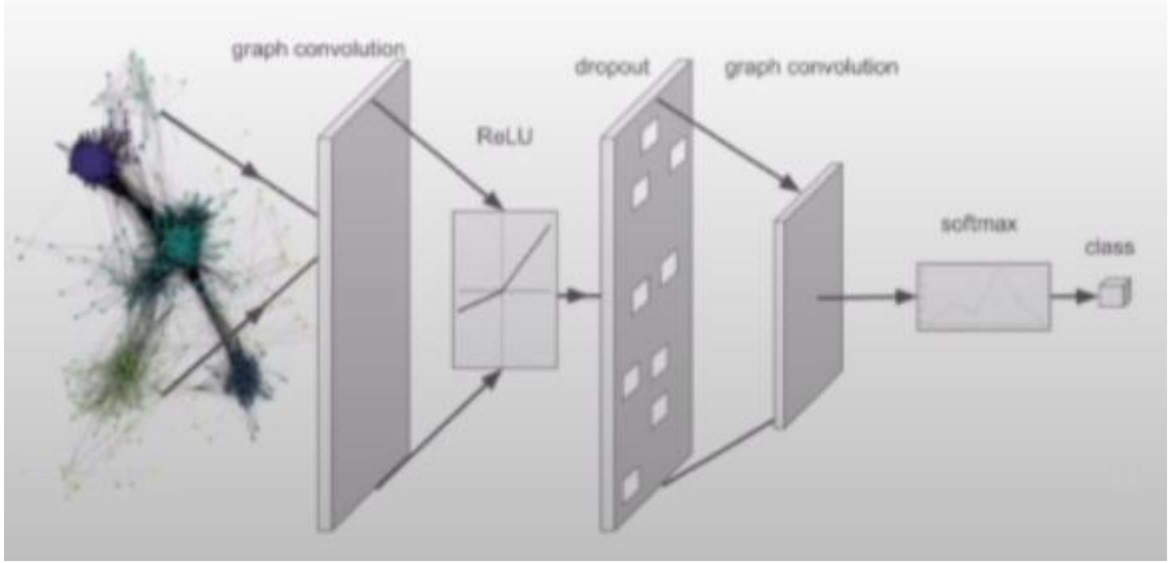
$$(x_1, x_2, x_3)\begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix},\ 2x_1 - x_2 - x_3 = (x_1 - x_2) + (x_1 - x_3)$$

Goal

- **Encode the graph structure directly using $f(X, A)$**
  - Train on a supervised target L_0 for all nodes with labels
  - Avoid explicit graph-based regularization in the loss function
  - Learn representations of nodes with and without labels

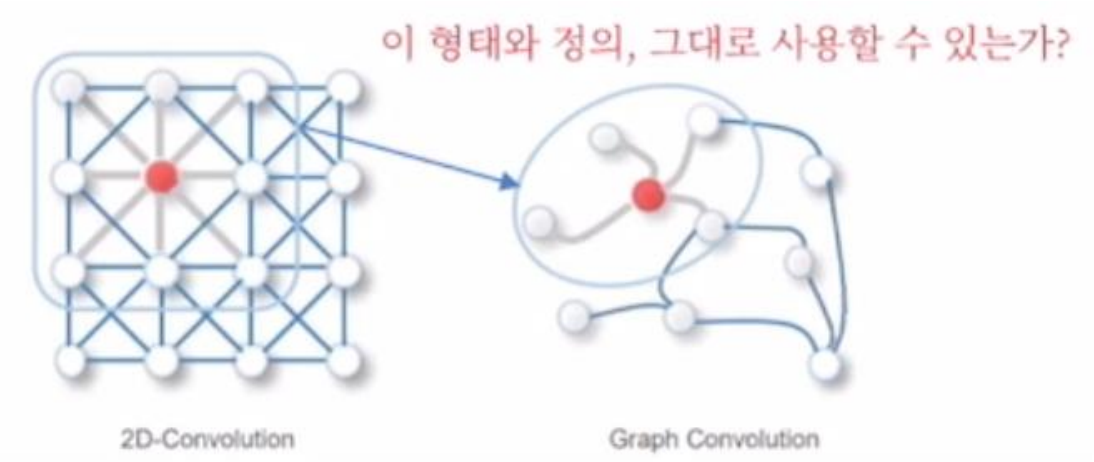
Graph Convolutional Network

- **Two-layer GCN for semi-supervised node classification on a graph**
  - $Z = f(X, A) = softmax(\hat{A} ReLU(\hat{A} X W^{(0)}) W^{(1)})$



Graph Convolutional Network

- **Apply a convolutional filter to the graph**
  - Useful for extracting local feature from entire data



→ *General convolution not applicable!*

Graph Convolutional Network

- **How to define graph convolution?**
  - Apply Convolutional theorem
    - Fourier transform of the convolution of two functions is the pointwise product of their Fourier transforms

$$\mathcal{F}(w * h) = \mathcal{F}(w) \odot \mathcal{F}(h)$$

$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

signal h on graph *Graph Fourier transform*

$$\underset{filter/kernel}{w} * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

$$\underset{(n \times n)}{\boldsymbol{\Phi}} \quad \underset{(n \times n)(n \times 1)}{\boldsymbol{\Phi}^* \boldsymbol{w} = \widehat{\boldsymbol{w}}} \quad \underset{(n \times n)\ (n \times 1)}{\boldsymbol{\Phi}^* \boldsymbol{h}}$$

learning parameterss/filter

$$= \boldsymbol{\Phi}(\widehat{\boldsymbol{w}} \odot \boldsymbol{\Phi}^* \boldsymbol{h})$$

$$(\boldsymbol{n} \times \boldsymbol{n})\ (\boldsymbol{n} \times \mathbf{1}) \quad (\boldsymbol{n} \times \mathbf{1})$$

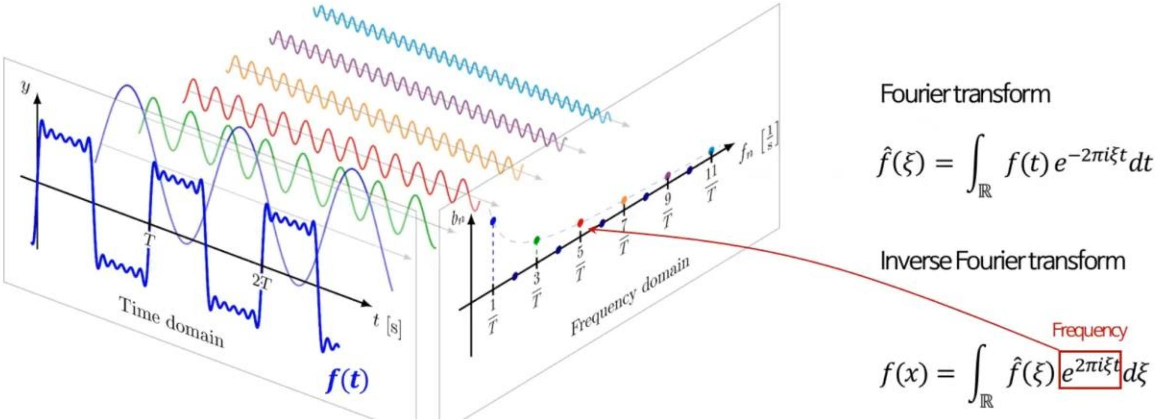
$$= \boldsymbol{\Phi}\ \widehat{\boldsymbol{w}}(\boldsymbol{\Lambda})\ \boldsymbol{\Phi}^* \boldsymbol{h}$$

$$(\boldsymbol{n} \times \boldsymbol{n})\ (\boldsymbol{n} \times \boldsymbol{n}) \quad (\boldsymbol{n} \times \mathbf{1})$$

Graph Convolutional Network

Fourier transform

- An arbitrary input signal is represented by decomposing it into the sum of periodic functions with various frequencies

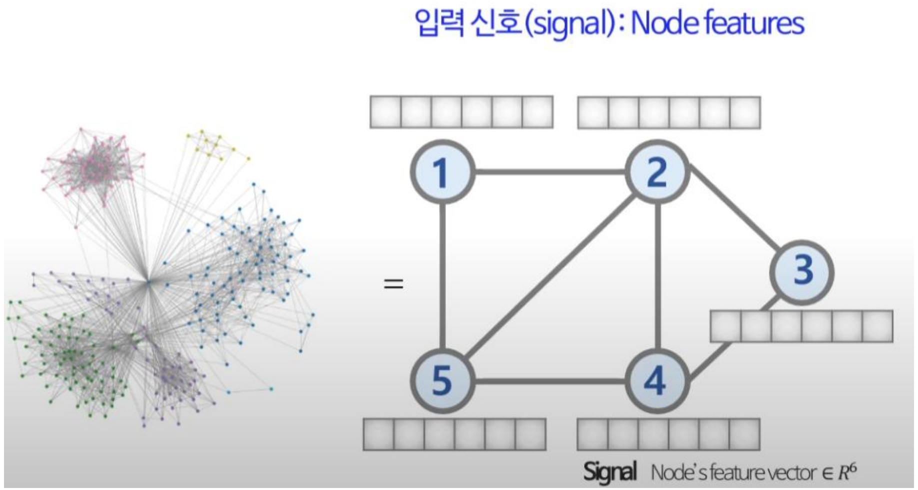


→ How to apply it to graph?

Graph Convolutional Network

☐ How to define Fourier transforms for graphs?

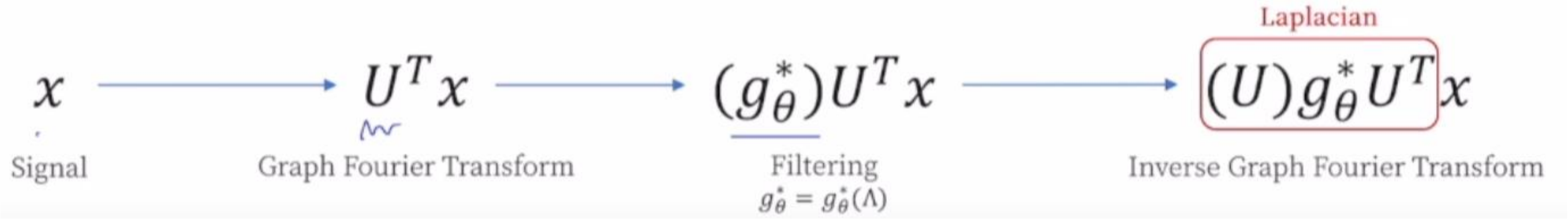
- An arbitrary input signal is represented by decomposing it into the sum of periodic functions with various frequencies



→ *Define the node as a sum of signals from neighbor nodes*

Graph Convolutional Network

- **How to define Fourier transforms for graphs?**
  - $g_\theta \star x = U g_\theta U^T x$
  - U : the matrix of eigenvectors of the normalized graph Laplacian $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^T$
  - g_θ : function of the eigenvalues of graph Laplacian $L, i.e.\, g_\theta(\Lambda)$



Computationally expensive!

Graph Convolutional Network

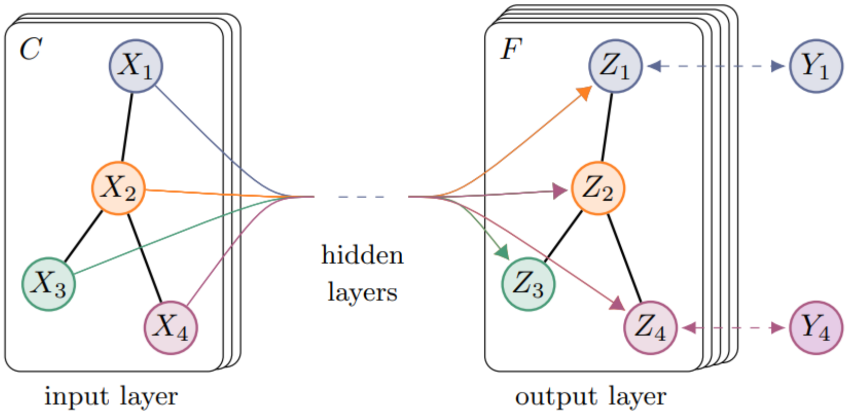
- **How to define Fourier transforms for graphs?**
  - $g_\theta(\Lambda)$ can be approximate!
  - $g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x$, $\tilde{L} = \frac{2}{\lambda_{max}} L - I_N$,
  - $T_k(x) = 2xT_{k-1}x - T_{k-2}(x), T_0(x) = 1$ and $T_1(x) = x$
  - It depends only on nodes that are at maximum K steps away from the central node(K^{th} order neighborhood)

Graph Convolutional Network

- **Layer-wise Linear Model**
  - Suppose that $K = 1, \lambda_{max} \approx 2, \theta = \theta_0' = -\theta_1'$
  - $g_{\theta'} \star x \approx \theta\left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}\right) x$

Semi-Supervised Node Classification

- Schematic depiction of multi-layer Graph Convolutional Network



(a) Graph Convolutional Network

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

Experiments

- **Semi-Supervised Node Classification**
  - GCN outperforms other methods

Table 2: Summary of results in terms of classification accuracy (in percent).

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

Experiments

Evaluation of Propagation Model

- The trick they proposed (i.e., $K = 1, \lambda_{max} \approx 2, \theta = \theta_0' = -\theta_1'$) had a good effect on the accuracy

Table 3: Comparison of propagation models.

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
1^{st}-order model (Eq. 6)		$X\Theta_0 + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} X \Theta_1$	68.3	80.0	77.5
Single parameter (Eq. 7)		$(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) X \Theta$	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**
1^{st}-order term only		$D^{-\frac{1}{2}} A D^{-\frac{1}{2}} X \Theta$	68.7	80.5	77.8
Multi-layer perceptron		$X\Theta$	46.5	55.1	71.4

Limitations

- **Memory requirement**
  - Use Full Batch Gradient Descent(include all Neighbors)
- **Directed edges and edge features**
  - Do not support directed graph
- **Limiting assumptions**
  - $\tilde{A} = A + \lambda I_N$

Conclusions

- **Graph-based semi-supervised learning**
  - The assumption that connected nodes in the graph are likely to share the same label
- **Graph Convolutional Network**
  - Import graph convolution of spectral domain into spatial domain
- **Experiments**
  - Outperform other methods
  - Renormalization trick for computational efficiency shows good results
- **Limitation**
  - Memory requirement, Directed edges and edge features, Limiting assumptions

LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation

2025-02-25

Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, Meng Wang

SIGIR '20: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval

Index

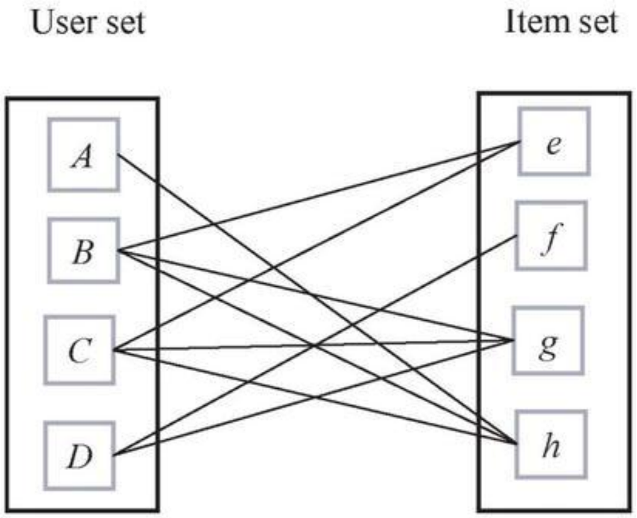
- **Introduction**
  - What is a Recommender System?
  - GCN in Recommender Systems
- **Ablation study on NGCF**
- **Goal**
- **LightGCN**
- **Experiments**
- **Conclusion**

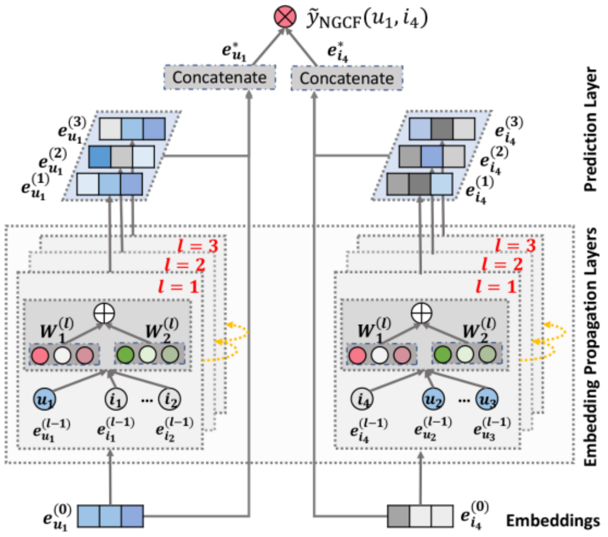
What is a Recommender System (RecSys)?

- **Predict whether a user will interact with an item**
- **Based on Collaborative filtering**
  - Method of making automatic predictions about a user's interests by utilizing information collected from many users
  - Parameterize users and items as embeddings and learn

GCN in Recommender Systems

- **Neural Graph Collaborative filtering (NGCF)**
  - Suppose that the user-item graph is bipartite graph
  - $e_u^{(k+1)} = \sigma(W_1 e_u^{(k)} + \sum_{i \in N_u} \frac{1}{\sqrt{|N_u||N_i|}} (W_1 e_i^{(k)} + W_2 \left(e_i^{(k)} \odot e_u^{(k)}\right)))$
  - $e_i^{(k+1)} = \sigma(W_1 e_i^{(k)} + \sum_{u \in N_i} \frac{1}{\sqrt{|N_u||N_i|}} (W_1 e_u^{(k)} + W_2 \left(e_u^{(k)} \odot e_i^{(k)}\right)))$



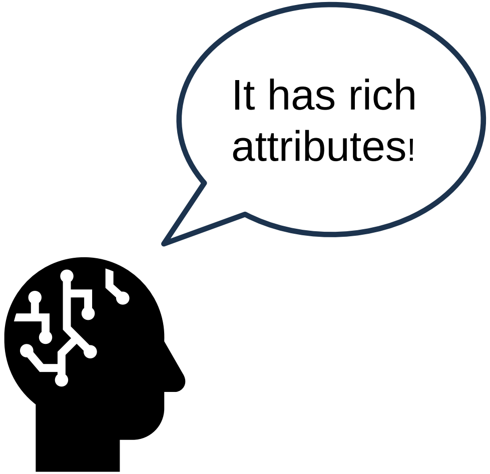


GCN in Recommender Systems

- **Each node in user-item graph is only described in a *one-hot ID***

  → *No concrete semantics in ID embeddings*

  → *Feature transformation and nonlinear activation has no contribution!*

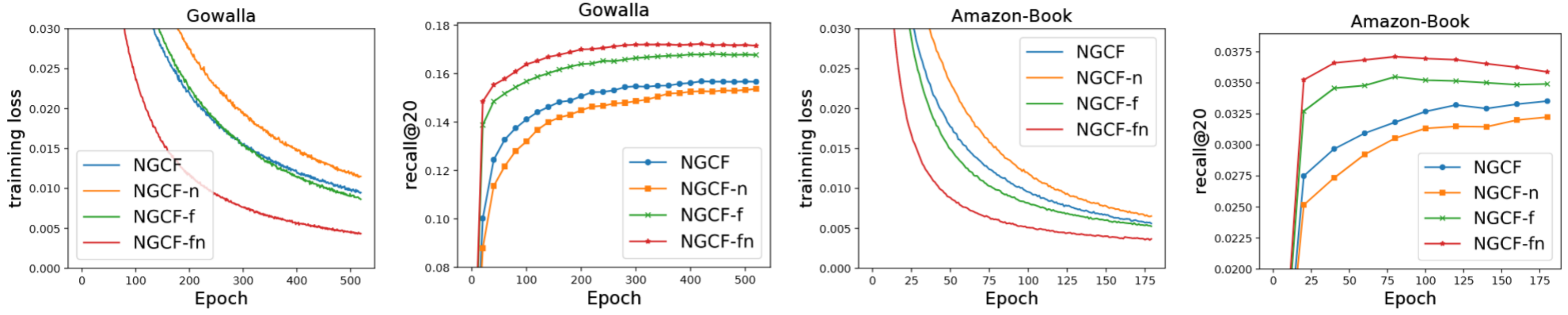


$$\begin{bmatrix} 0 \\ 0 \\ \mathbf{1} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Just 3rd item….

Ablation study

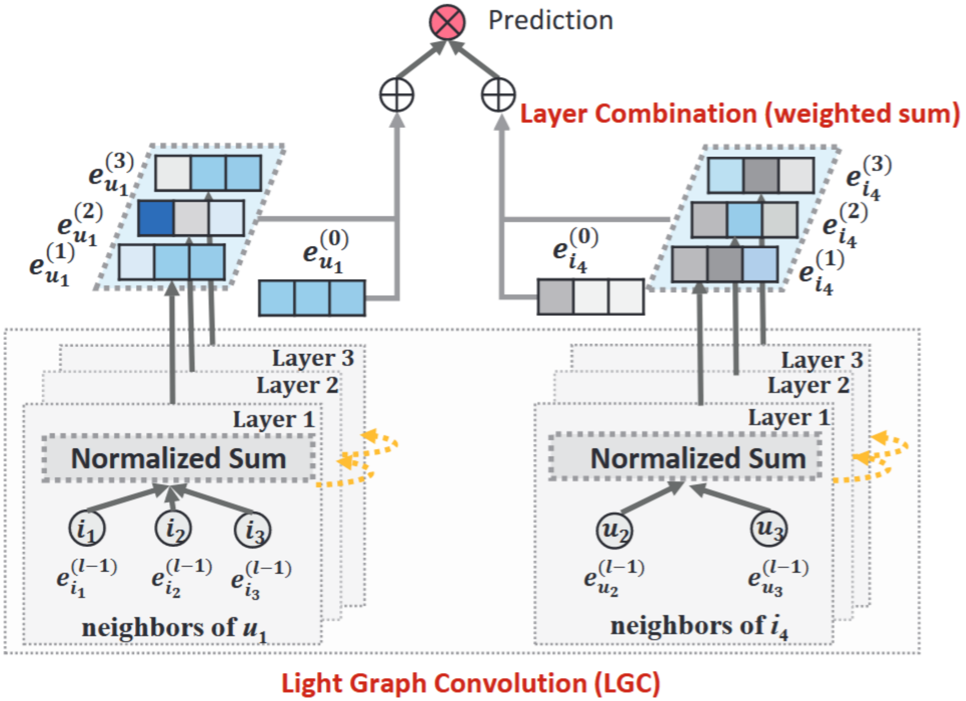
- Removing two operations leads to significant accuracy improvements



(a) Training loss on Gowalla (b) Testing recall on Gowalla (c) Training loss on Amazon-Book (d) Testing recall on Amazon-Book

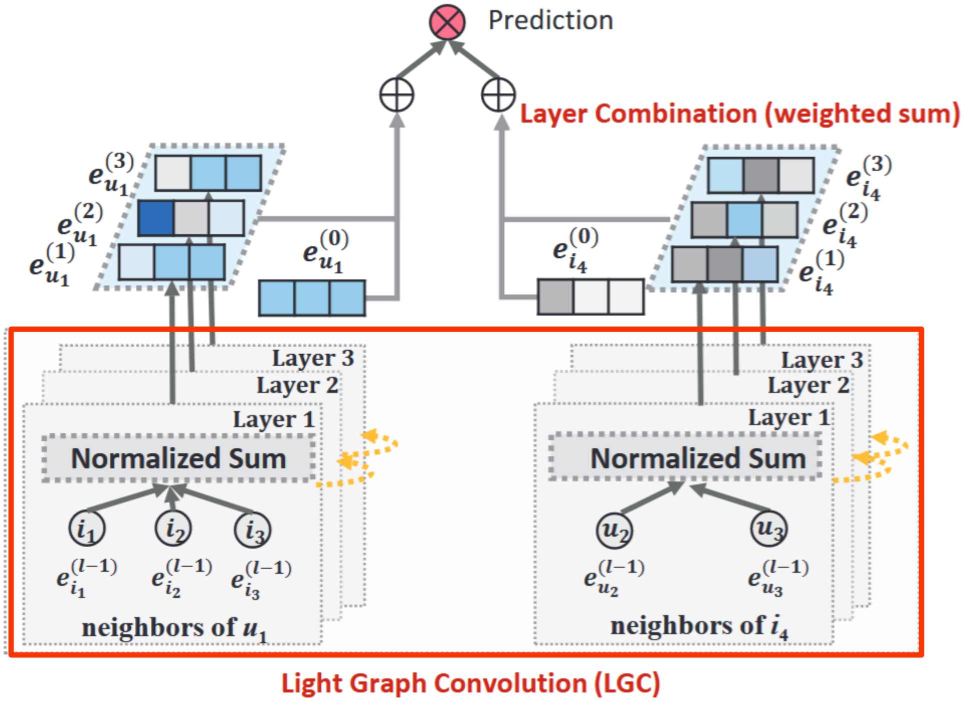
Goal

- Develop a light yet effective model by including the most essential ingredients of GCN for recommendation
  - More interpretable, practically easy to train and maintain, etc.



LightGCN

- **Light Graph Convolution (LGC)**
  - Same as the basic idea of GCN (i.e., neighborhood aggregation)
  - Remove unnecessary operations

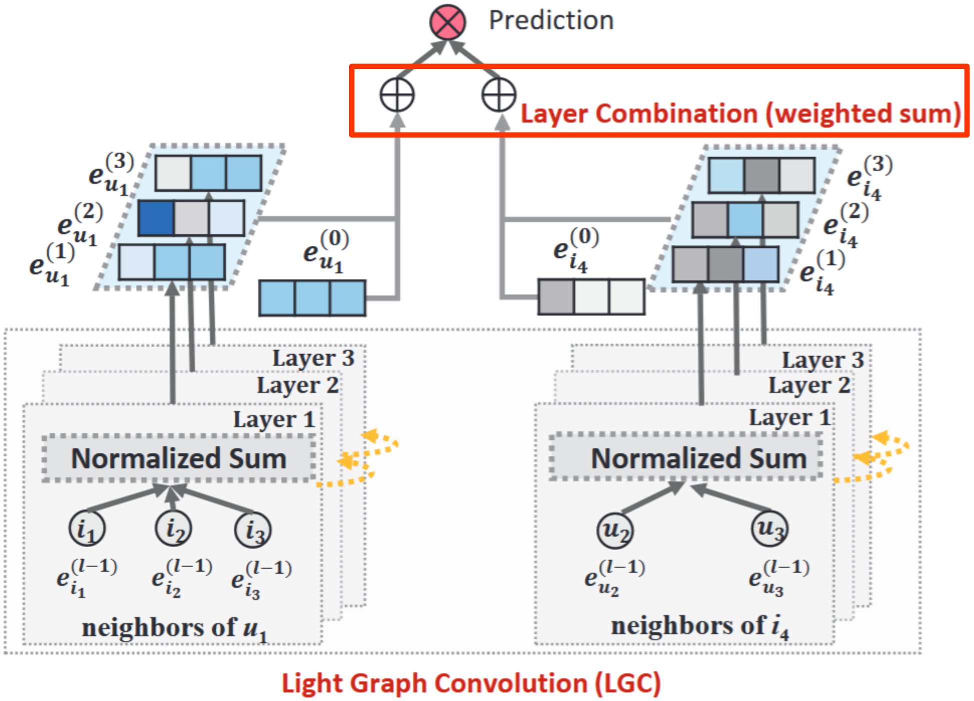


$$e_u^{(k+1)} = \sum_{i \in N_u} \frac{1}{\sqrt{|N_u|}\sqrt{|N_i|}} e_i^{(k)}$$

$$e_i^{(k+1)} = \sum_{u \in N_i} \frac{1}{\sqrt{|N_i|}\sqrt{|N_u|}} e_u^{(k)}$$

LightGCN

- **Layer Combination**
  - α_k : the importance of the k-th layer embedding
    - (1) Prevent oversmoothing (2) Capture different semantics (3) self-connection



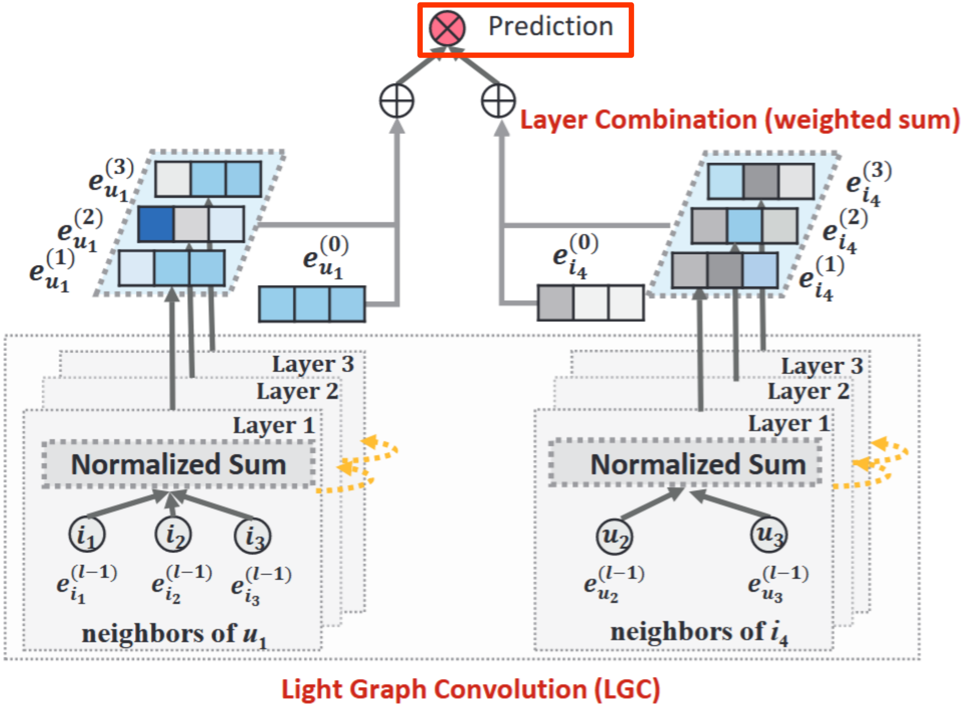
$$e_u = \sum_{k=0}^{K} \alpha_k e_u^{(k)}$$

$$e_i = \sum_{k=0}^{K} \alpha_k e_i^{(k)}$$

LightGCN

Model Prediction

- The ranking score for recommendation generation



$$\hat{y}_{ui} = \mathbf{e}_u^T \mathbf{e}_i$$

LightGCN

Matrix Form

- User-item interaction matrix $\boldsymbol{R} \in R^{M \times N}$
- Adjacency matrix $A = \begin{bmatrix} 0 & R \\ R^T & 0 \end{bmatrix}$
- $E^{(k+1)} = \left(D^{-\frac{1}{2}} A D^{-\frac{1}{2}}\right) E^{(k)}$
- Final embedding matrix used for model prediction

$$E = \alpha_0 E^{(0)} + \alpha_1 E^{(1)} + \cdots + \alpha_K E^{(k)} = \alpha_0 E^{(0)} + \alpha_1 \tilde{A} E^{(0)} + \cdots + \alpha_K \tilde{A}^K E^{(0)}$$

$\tilde{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$: symmetrically normalized matrix

LightGCN

- **Self-Connection**
  - $E^{(k+1)} = (D+I)^{-\frac{1}{2}}(A+I)(D+I)^{-\frac{1}{2}}E^{(k)}$
  - $(D+I)^{-\frac{1}{2}}$ terms for simplicity, since they only re-scale embeddings

$$\mathbf{E}^{(K)} = (\mathbf{A}+\mathbf{I})\mathbf{E}^{(K-1)} = (\mathbf{A}+\mathbf{I})^K\mathbf{E}^{(0)}$$
$$= \binom{K}{0}\mathbf{E}^{(0)} + \binom{K}{1}\mathbf{A}\mathbf{E}^{(0)} + \binom{K}{2}\mathbf{A}^2\mathbf{E}^{(0)} + \ldots + \binom{K}{K}\mathbf{A}^K\mathbf{E}^{(0)}$$

LightGCN

- **Alleviate over-smoothing (APPNP)**
  - Staying close to the root node
  - Leveraging the information from a large neighborhood

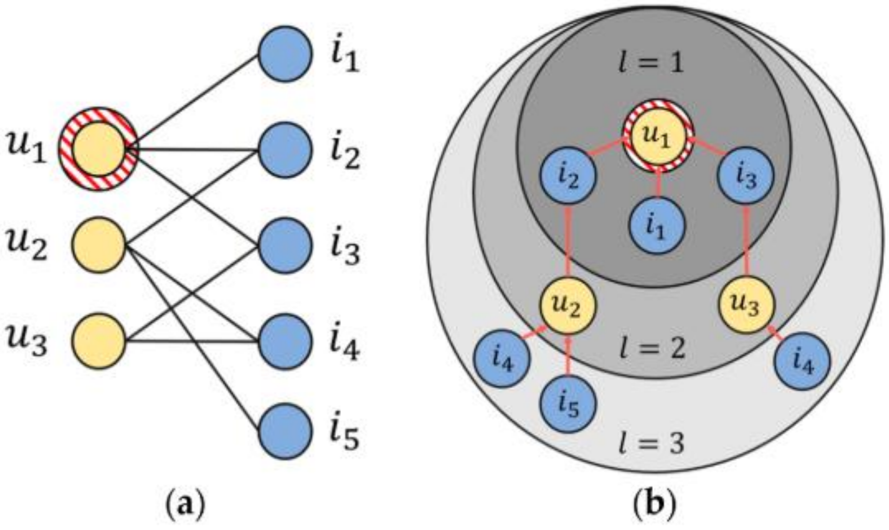
$$E^{(k+1)} = \beta E^{(0)} + (1-\beta)\tilde{A}E^{(k)}$$

$$\begin{aligned}
\mathbf{E}^{(K)} &= \beta\mathbf{E}^{(0)} + (1-\beta)\tilde{\mathbf{A}}\mathbf{E}^{(K-1)}, \\
&= \beta\mathbf{E}^{(0)} + \beta(1-\beta)\tilde{\mathbf{A}}\mathbf{E}^{(0)} + (1-\beta)^2\tilde{\mathbf{A}}^2\mathbf{E}^{(K-2)} \\
&= \beta\mathbf{E}^{(0)} + \beta(1-\beta)\tilde{\mathbf{A}}\mathbf{E}^{(0)} + \beta(1-\beta)^2\tilde{\mathbf{A}}^2\mathbf{E}^{(0)} + \ldots + (1-\beta)^K\tilde{\mathbf{A}}^K\mathbf{E}^{(0)}
\end{aligned}$$

Can use a large K for long-range modeling with controllable oversmoothing!

LightGCN

- **Second-Order Embedding**
  - $e_u^{(2)} = \sum_{i \in N_u} \frac{1}{\sqrt{|N_u|}\sqrt{|N_i|}} e_i^{(1)} = \sum_{i \in N_u} \frac{1}{|N_i|} \sum_{v \in N_i} \frac{1}{\sqrt{|N_u|}\sqrt{|N_v|}} e_v^{(0)}$
  - First layer : smoothness on users and items that have interactions
  - Second layer : users that have overlap on interacted items



LightGCN

- **Model Training**
  - The trainable parameter : embeddings of the 0-th layer $\theta = \{E^{(0)}\}$
  - Loss function

$$L_{BPR} = -\sum_{u=1}^{M} \sum_{i \in N_u} \sum_{j \notin N_u} \ln \sigma(\hat{y}_{ui} - \hat{y}_{uj}) + \lambda ||\mathbf{E}^{(0)}||^2$$

  - Make prediction score for interacting items and users larger and otherwise smaller

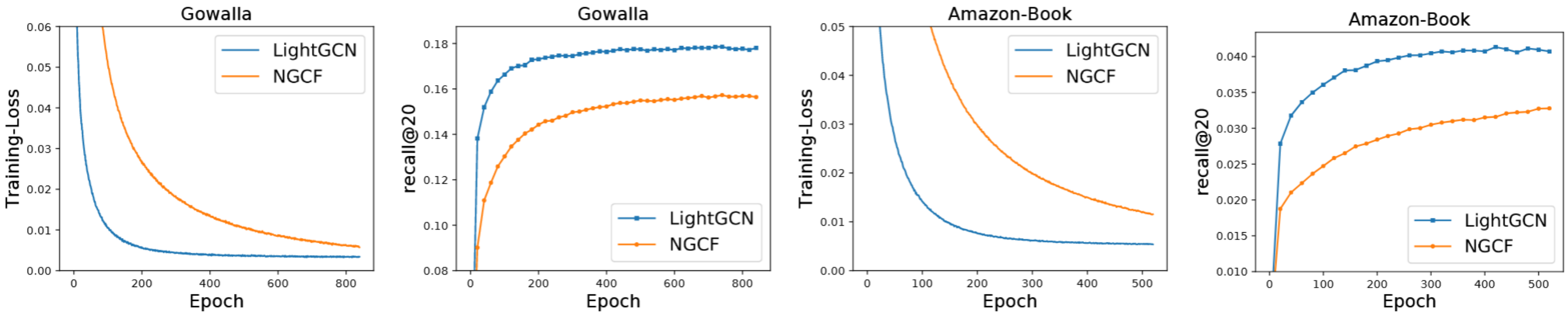
Experiments

Performance Comparison with NGCF

- LightGCN outperforms NGCF by a large margin

Dataset		Gowalla		Yelp2018		Amazon-Book	
Layer #	**Method**	**recall**	**ndcg**	**recall**	**ndcg**	**recall**	**ndcg**
1 Layer	NGCF	0.1556	0.1315	0.0543	0.0442	0.0313	0.0241
	LightGCN	0.1755(+12.79%)	0.1492(+13.46%)	0.0631(+16.20%)	0.0515(+16.51%)	0.0384(+22.68%)	0.0298(+23.65%)
2 Layers	NGCF	0.1547	0.1307	0.0566	0.0465	0.0330	0.0254
	LightGCN	0.1777(+14.84%)	0.1524(+16.60%)	0.0622(+9.89%)	0.0504(+8.38%)	0.0411(+24.54%)	0.0315(+24.02%)
3 Layers	NGCF	0.1569	0.1327	0.0579	0.0477	0.0337	0.0261
	LightGCN	0.1823(+16.19%)	0.1555(+17.18%)	0.0639(+10.38%)	0.0525(+10.06%)	0.0410(+21.66%)	0.0318(+21.84%)
4 Layers	NGCF	0.1570	0.1327	0.0566	0.0461	0.0344	0.0263
	LightGCN	0.1830(+16.56%)	0.1550(+16.80%)	0.0649(+14.58%)	0.0530(+15.02%)	0.0406(+17.92%)	0.0313(+18.92%)

*The scores of NGCF on Gowalla and Amazon-Book are directly copied from Table 3 of the NGCF paper (https://arxiv.org/abs/1905.08108)



Experiments

Performance Comparison with State-of-the-Arts

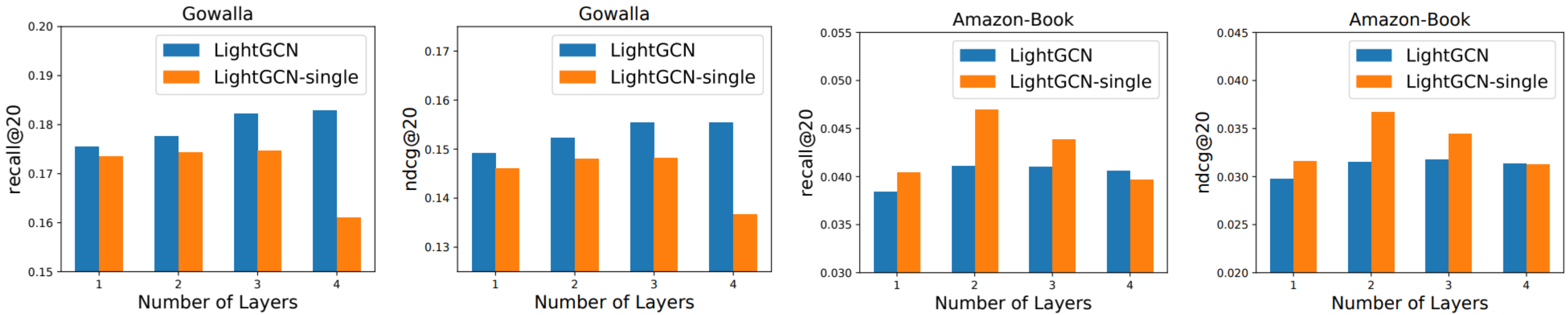
- LightGCN consistently outperforms other methods on all three datasets

Dataset	Gowalla		Yelp2018		Amazon-Book	
Method	**recall**	**ndcg**	**recall**	**ndcg**	**recall**	**ndcg**
NGCF	0.1570	0.1327	0.0579	0.0477	0.0344	0.0263
Mult-VAE	0.1641	0.1335	0.0584	0.0450	0.0407	0.0315
GRMF	0.1477	0.1205	0.0571	0.0462	0.0354	0.0270
GRMF-norm	0.1557	0.1261	0.0561	0.0454	0.0352	0.0269
LightGCN	**0.1830**	**0.1554**	**0.0649**	**0.0530**	**0.0411**	**0.0315**

Experiments

Comparison of LightGCN and LightGCN-single

- We can see the effectiveness of weighted sum (i.e., α_k)



Conclusion

- **GCN in Recommender System (NGCF)**
  - No contribution of two operations to accuracy

- **LightGCN**
  - Remove unnecessary operations
  - Light graph convolution & Layer Combination

- **Model Analysis**
  - Weighted sum(i.e., α_k) can lead to self-connection, alleviating oversmoothing, etc.,

- **Experiments**
  - Outperform other methods