SIGIR 2020
XI'AN·CHINA

LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation

Xiangnan He
Kuan Deng
Yongdong Zhang

Xiang Wang

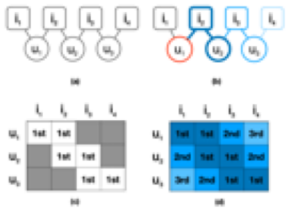
Yan Li

Meng Wang

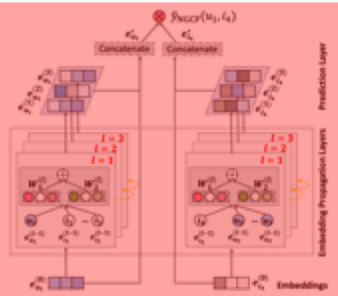
Outline

- Background: NGCF
  - SIGIR 2019. Neural Graph Collaborative Filtering
- Model: LightGCN
- Fast Loss for LightGCN
- Conclusion & Future Work

Representation Learning in CF



Hop-Rec [2018]
GRMF [2015]



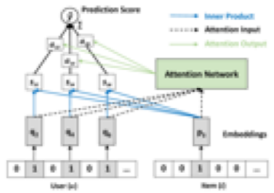
GC-MC [2017]
NGCF [2019]
LightGCN [2020] …

Model Holistic Interaction Graph

- Apply embedding smooth constraints on connected nodes
- Perform embedding propagation via graph neural networks

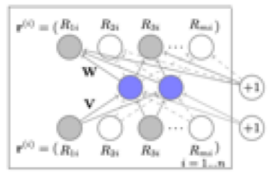
Model Personal History as User Feature

- Integrate embeddings of historical items as user embeddings
- Or use autoencoders to generate user behaviors



SVD++ [2008]
FISM [2013]
NAIS [2018], …

ACF [2017]



Mult-VAE [2018]
AutoRec [2015]
CDAE [2016]

MF [2009]
BPRMF [2009]

NCF [2017]

CML [2017],
LRML [2018]

Model Single User-Item Pairs

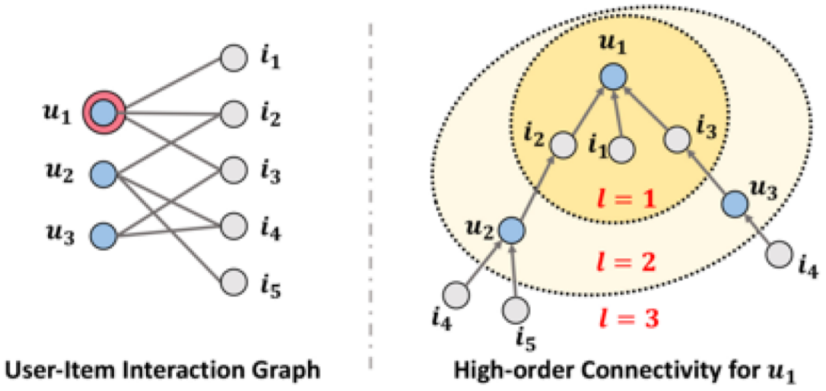
- Project each user/item ID into an embedding vector

Recap: NGCF [Wang et al, SIGIR'19]

High-level Idea:

- Organize historical interactions as a user-item bipartite graph
- Capture CF signal via high-order connectivity
  - Definition: the paths that reach u_1 from any node with the path length l larger than 1.

- E.g., why u_1 may like i_4?
  - $u_1 \leftarrow i_2 \leftarrow u_2 \leftarrow i_4$
  - $u_1 \leftarrow i_3 \leftarrow u_3 \leftarrow i_4$



NGCF's contribution: explicitly modeling high-order connectivity in representation space via GNN.

Wang et al. Neural Graph Collaborative Filtering. SIGIR 2019

NGCF: First-order Connectivity Modeling

Embedding Propagation, inspired by GNNs

- Propagate embeddings recursively on the graph → high-order connectivity
- Construct information flows in the embedding space → embed CF signal

➢ **First-order Propagation**

➢ **Message Construction:** generate message from one neighbor

message passed from i to u

$$\mathbf{m}_{u\leftarrow i} = \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}}\left(\mathbf{W}_1\mathbf{e}_i + \mathbf{W}_2(\mathbf{e}_i \odot \mathbf{e}_u)\right)$$

discount factor

- Make message dependent on the affinity,
- Pass more information to similar nodes

➢ **Message Aggregation:** update ego node's representation by aggregating message from all neighbors

$$\mathbf{e}_u^{(1)} = \text{LeakyReLU}\left(\mathbf{m}_{u\leftarrow u} + \sum_{i\in\mathcal{N}_u} \mathbf{m}_{u\leftarrow i}\right)$$

self-connections

all neighbors of u

Wang et al. Neural Graph Collaborative Filtering. SIGIR 2019

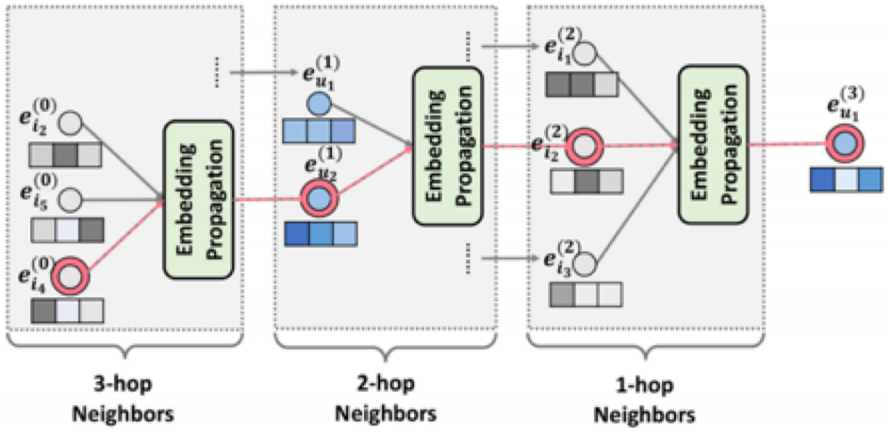
NGCF: Higher-order Connectivity Modeling

- Stack more embedding propagation layers to explore high-order connectivity

$$\mathbf{e}_u^{(l)} = \text{LeakyReLU}\left(\mathbf{m}_{u\leftarrow u}^{(l)} + \sum_{i\in\mathcal{N}_u} \mathbf{m}_{u\leftarrow i}^{(l)}\right),$$

representation of u at the l-th layer

- The collaborative signal like $u_1 \leftarrow i_2 \leftarrow u_2 \leftarrow i_4$ can be captured in the embedding propagation process.



- **Final embedding: concatenate the embedding from all layers**

Wang et al. Neural Graph Collaborative Filtering. SIGIR 2019

Our Argument

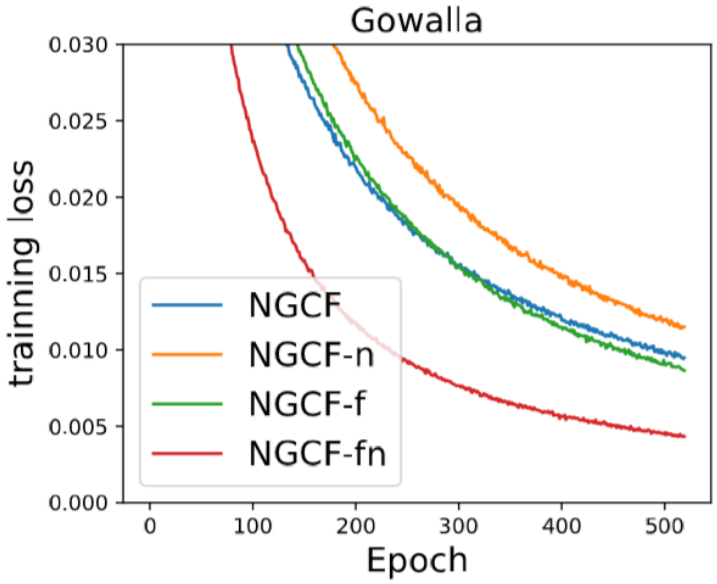
- Designs of NGCF are rather heavy and burdensome
  - Many operations are directly inherited from GCN without justification.

	GNNs	NGCF
Original task	Node classification	Collaborative filtering
Input data	Rich node features • Attributes, text, image data	Only node ID • One-hot encoding
Feature transformation	Distill useful information	Generate ID embeddings
Neighborhood aggregation	Pass messages from neighbors to the egos	Pass messages from neighbors to the egos
Nonlinear activation	Enhance representation ability	Negatively increases the difficulty for model training

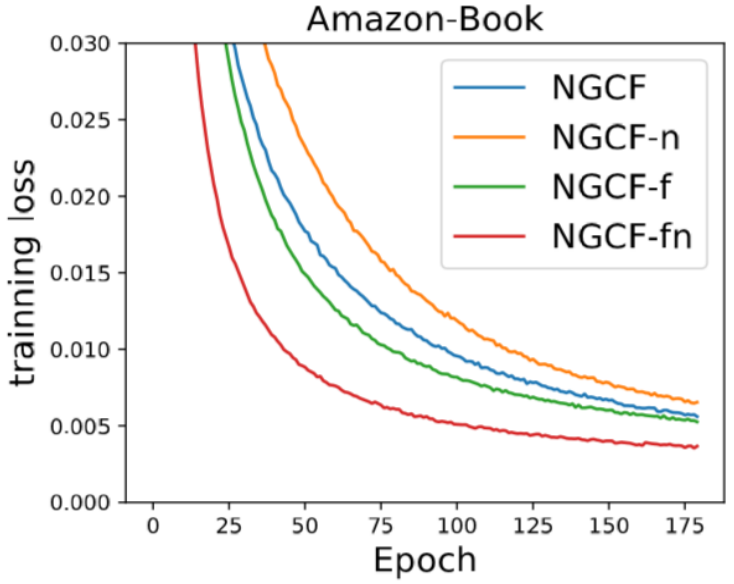
He et al. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. SIGIR 2020

Empirical Evidence on Training Difficulty

- Removing feature transformation (NGCF-f) → decrease training loss
- Removing nonlinear activation (NGCF-n) → increase training loss
- But, removing nonlinear activation & feature transformation (NGCF-fn) → significantly decrease training loss



(a) Training loss on Gowalla

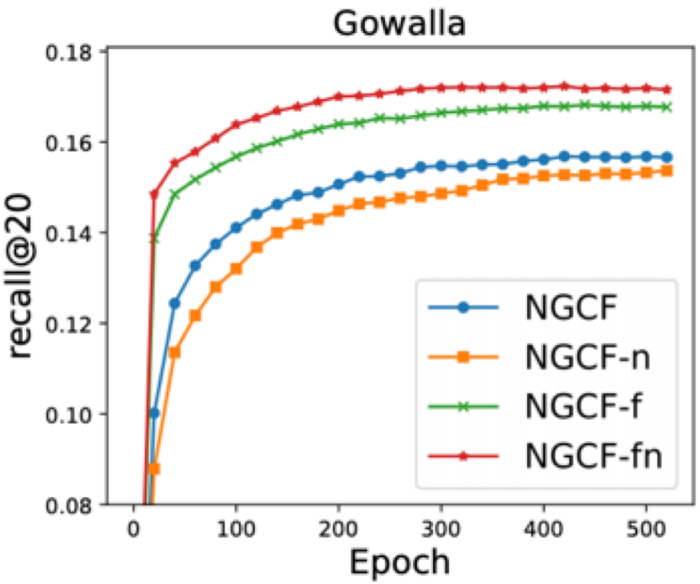


(c) Training loss on Amazon-Book

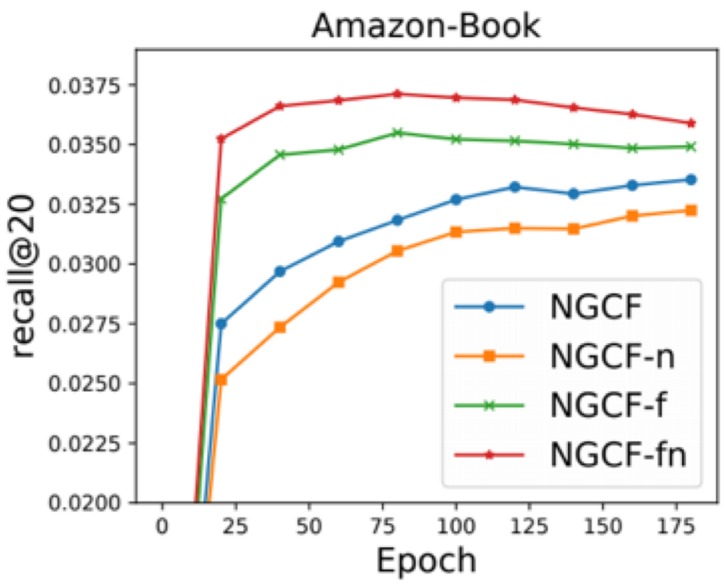
He et al. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. SIGIR 2020

Empirical Evidence on Training Difficulty

- Removing feature transformation (NGCF-f) → improve testing accuracy
- Removing nonlinear activation (NGCF-n) → hurt testing accuracy
- Removing nonlinear activation & feature transformation (NGCF-fn) → significantly improve testing accuracy



(b) Testing recall on Gowalla



(d) Testing recall on Amazon-Book

He et al. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. SIGIR 2020

Light Graph Convolution

NGCF

- Graph Convolution Layer

$$\mathbf{e}_u^{(l)} = \text{LeakyReLU}\left(\mathbf{m}_{u\leftarrow u}^{(l)} + \sum_{i\in\mathcal{N}_u} \mathbf{m}_{u\leftarrow i}^{(l)}\right)$$

- Layer Combination

$$\mathbf{e}_u^* = \mathbf{e}_u^{(0)} \| \cdots \| \mathbf{e}_u^{(L)}$$

- Matrix Form

$$\mathbf{E}^{(l)} = \text{LeakyReLU}\left((\mathcal{L} + \mathbf{I})\mathbf{E}^{(l-1)}\mathbf{W}_1^{(l)} + \mathcal{L}\mathbf{E}^{(l-1)} \odot \mathbf{E}^{(l-1)}\mathbf{W}_2^{(l)}\right)$$

LightGCN

- **Light** Graph Convolution Layer

$$\mathbf{e}_u^{(k+1)} = \sum_{i\in\mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}} \mathbf{e}_i^{(k)}$$

- Layer Combination

$$\mathbf{e}_u = \sum_{k=0}^{K} \alpha_k \mathbf{e}_u^{(k)}$$

- Matrix Form

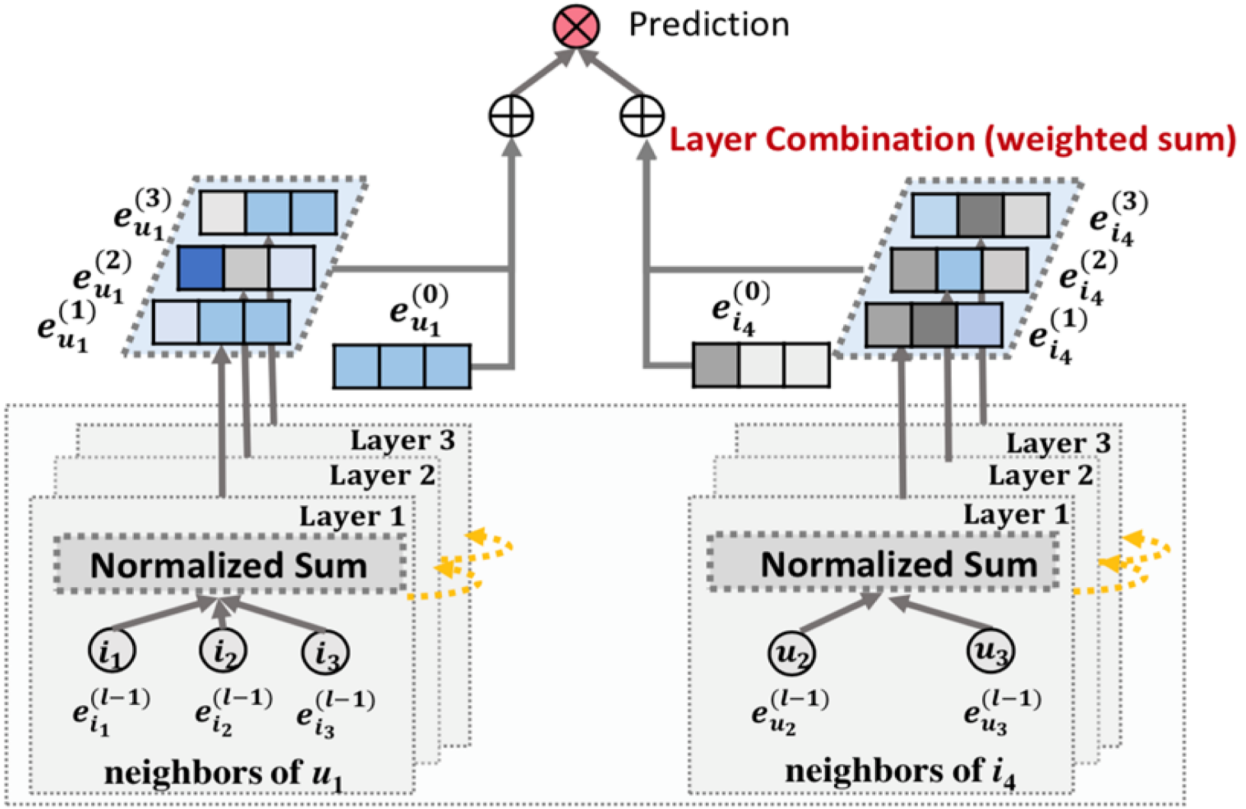
$$\mathbf{E}^{(k+1)} = (\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{E}^{(k)}$$

Only simple weighted sum aggregator is remained

- No feature transformation
- No nonlinear activation
- No self connection

He et al. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. SIGIR 2020

Overall Framework



Light Graph Convolution (LGC)

$$\mathbf{E} = \alpha_0\mathbf{E}^{(0)} + \alpha_1\mathbf{E}^{(1)} + \alpha_2\mathbf{E}^{(2)} + \ldots + \alpha_K\mathbf{E}^{(K)}$$
$$= \alpha_0\mathbf{E}^{(0)} + \alpha_1\tilde{\mathbf{A}}\mathbf{E}^{(0)} + \alpha_2\tilde{\mathbf{A}}^2\mathbf{E}^{(0)} + \ldots + \alpha_K\tilde{\mathbf{A}}^K\mathbf{E}^{(0)}$$

importance of the k-th layer embedding in constituting the final embedding

He et al. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. SIGIR 2020

Model Analysis

- Relation with **SGCN** [Wu et al. ICML 2019]:
  - By doing layer combination, LightGCN subsumes the effect of self-connection → **no need to add self-connection in adjacency matrix.**

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

- Relation with **APPNP** [Klicpera et al. ICLR 2019]:
  - By setting α_k properly, LightGCN can recover APPNP → **use a large K for long-range modeling with controllable oversmoothing.**

$$\mathbf{E}^{(K)} = \beta\mathbf{E}^{(0)} + \beta(1-\beta)\tilde{\mathbf{A}}\mathbf{E}^{(0)} + \beta(1-\beta)^2\tilde{\mathbf{A}}^2\mathbf{E}^{(0)} + \ldots + (1-\beta)^K\tilde{\mathbf{A}}^K\mathbf{E}^{(0)}.$$

He et al. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. SIGIR 2020

Experiment Settings

Datasets:

Gowalla, Yelp2018, Amazon-Book

Evaluation Metrics:

recall@20, ndcg@20

Dataset partition: randomly select 80% data for training set, and 20% data for testing set.

Table 2: Statistics of the experimented data.

Dataset	User #	Item #	Interaction #	Density
Gowalla	29, 858	40, 981	1, 027, 370	0.00084
Yelp2018	31, 668	38, 048	1, 561, 406	0.00130
Amazon-Book	52, 643	91, 599	2, 984, 108	0.00062

He et al. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. SIGIR 2020

Compared Methods

- LightGCN:
  - BPR optimizer, Uniform layer combination weights
  - Tuning L2 regularizer coefficient only.
- NGCF[Wang et al. SIGIR 2019]
  - Using the paper results
- Mult-VAE[Liang et al. WWW 2018]
  - Parameter setting: dropout ratio $\in\{0, 0.2, 0.5\}$, $\beta \in\{0.2, 0.4, 0.6, 0.8\}$
    model architecture : 600→200→600
- GRMF[Rao et al. NIPS 2015]: smooth embeddings with Laplacian regularizer
  - Parameter setting: $\lambda_g \in \{1e^{-5}, 1e^{-4}, \dots, 1e^{-1}\}$

He et al. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. SIGIR 2020

Experiment Results

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**

- LightGCN achieves significant improvements over the state-of-the-art baselines → **outstanding performance**

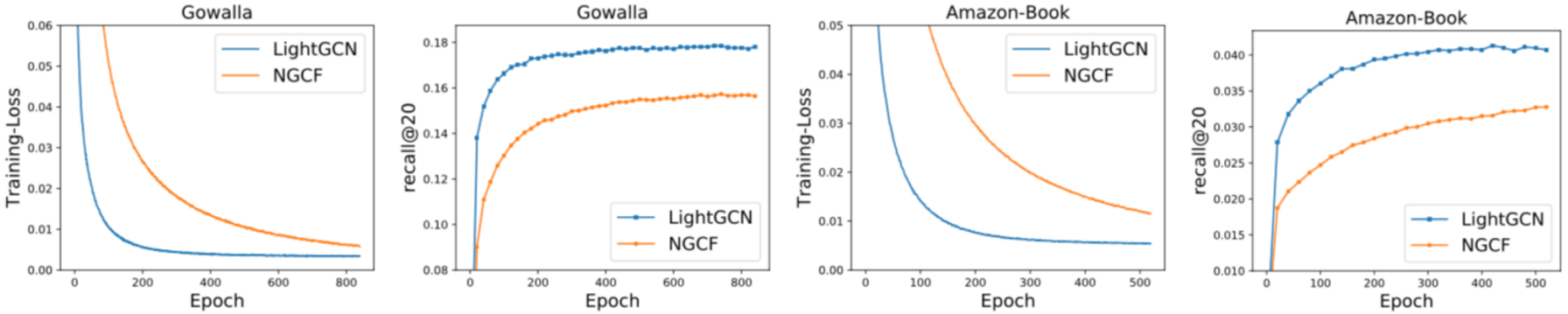
He et al. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. SIGIR 2020

Experiment Results

- Performance comparison between NGCF and LightGCN at different layers :

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%)

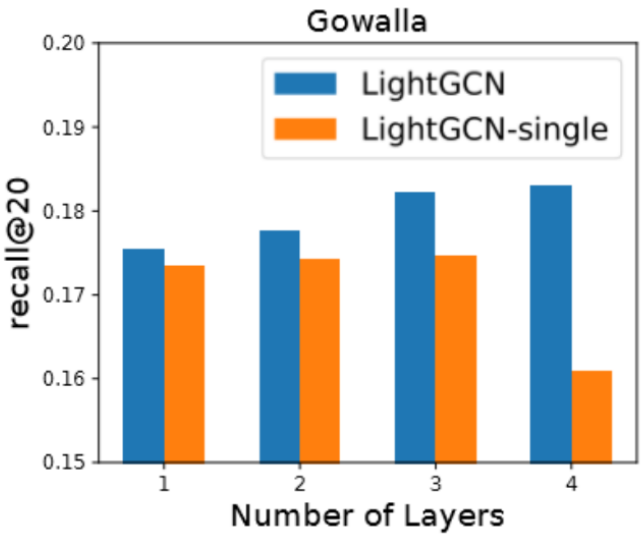
- Training curves of LightGCN and NGCF :

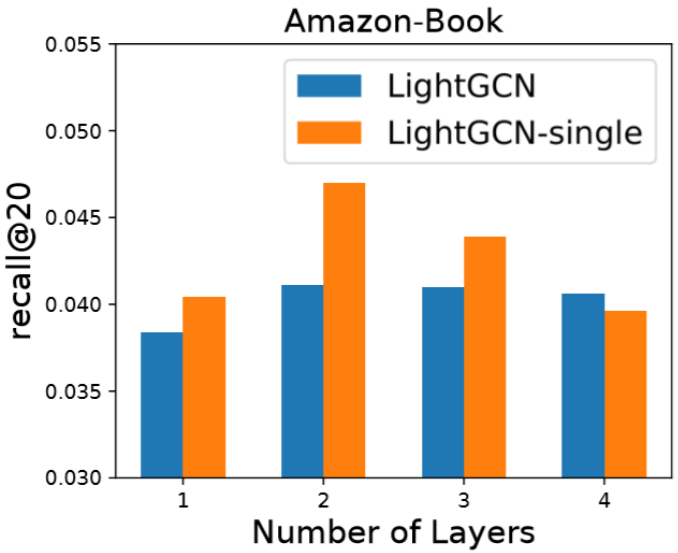


He et al. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. SIGIR 2020

Experiment Results

- LightGCN - single : Only use the final layer's output





- LightGCN-single performs better than LightGCN on sparser datasets → **further simplified**
- It implies that the layer combination weights are important to tune for datasets of different properties => better to learn automatically (future work)

He et al. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. SIGIR 2020

Embedding Smoothness

- 2-layers Light Graph Convolution:

$$\mathbf{e}_u^{(2)} = \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}} \mathbf{e}_i^{(1)} = \sum_{i \in \mathcal{N}_u} \frac{1}{|\mathcal{N}_i|} \sum_{v \in \mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_v|}} \mathbf{e}_v^{(0)}$$

  - Coefficient of $e_v^{(0)}$: $c_{v->u} = \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_v|}} \sum_{i \in \mathcal{N}_u \cap \mathcal{N}_v} \frac{1}{|\mathcal{N}_i|}$.

→ **similarity between user u and user v**

- Definition of user smoothness of user embeddings :

$$S_U = \sum_{u=1}^{M} \sum_{v=1}^{M} c_{v \to u} \left(\frac{\mathbf{e}_u}{||\mathbf{e}_u||^2} - \frac{\mathbf{e}_v}{||\mathbf{e}_v||^2} \right)^2,$$

  - user and item smoothness between LightGCN-single and MF:

Dataset	Gowalla	Yelp2018	Amazon-book
	Smoothness of User Embeddings		
MF	15449.3	16258.2	38034.2
LightGCN-single	12872.7	10091.7	32191.1
	Smoothness of Item Embeddings		
MF	12106.7	16632.1	28307.9
LightGCN-single	5829.0	6459.8	16866.0

  - LightGCN-single has lower smoothness than MF → **smoother embeddings are more suitable for recommendation**

He et al. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. SIGIR 2020

Fast Loss

Inefficiency of BPR Loss

- Bayesian Personalized Ranking (BPR) is a widely-used pairwise loss to optimize recommender models.

- However, BPR randomly samples user-item interactions to form a mini-batch → **failing to fully leverage parallel computing ability of GPU.**

	C++ (CPU)	TensorFlow (GPU)
time/epoch	1.1s	55s

- BPR samples the interactions to form a mini-batch, and the data cannot form a well-structured matrix

- We run BPRMF with amazon-book dataset using C++ on CPU(i9 9000kf) and TensorFlow on GPU(2080Ti)

Fast Loss on LightGCN

$$\sum_{u \in U} \sum_{i \in V} c_{ui}(y_{ui} - \hat{y}_{ui})^2$$

- $c_{ui} = \alpha, y_{ui} = 1$; if u *and* i *have interaction*
- $c_{ui} = 1, y_{ui} = 0$; *otherwise*

All possible user-item pairs

Mathematical transformations

$$\sum_{u \in U} \sum_{i \in V_u^+} [(\alpha - 1)\hat{y}_{ui}^2 - 2\alpha \hat{y}_{ui}] + \sum_{s=1}^{d} \sum_{t=1}^{d} (\sum_{u \in U} p_{us} p_{ut})(\sum_{i \in V} q_{is} q_{it})$$

The s-th term of user embedding $\boldsymbol{eu}$

Historical user-item pairs

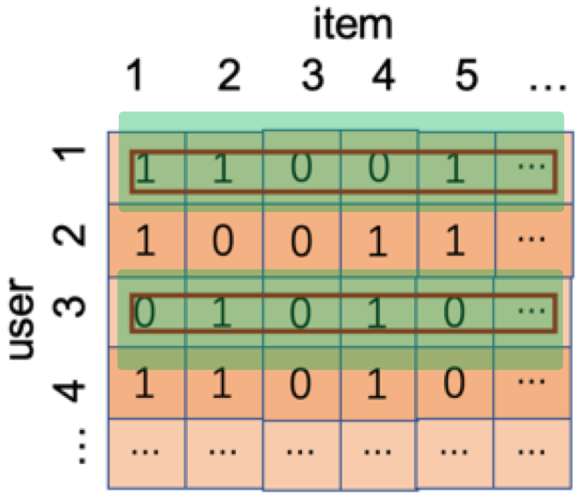
The s-th entry of item embedding of $\boldsymbol{ei}$.

Good Characteristics:

- Can support any model of inner product structure
- Time complexity is $O(|R|d + |N|d^2)$.
- Linear to the number of observed interactions

He et al. Fast Matrix Factorization for Online Recommendation with Implicit Feedback. SIGIR 2016

Good Characteristics of Fast Loss



- Distinct from BPR that samples interactions as a batch, Fast Loss samples **rows (users) as a batch**
  - The data of a batch is well-structured.
- Adv: allow better use of the speed-up of GPU/CPU, and the computation is linear to #observations.

Fast-loss brings **2~3 magnitude speed-up** compared with BPR

	BPR			Fast-loss		
	t	N	T	t	N	T
Gowalla	65s	840	15.1h	0.5s	860	86s
Yelp2018	115s	520	16.6h	0.8s	1250	16.7mim
Amazon-book	435s	340	41.8h	1.5s	870	21.8min

Able to train GNN on 100K users and 10M interactions in single GPU in 1 hour

Recommendation Accuracy

- LightGCN optimized with Fast Loss can achieve comparable performance to that with BPR loss.
  - Which loss is better depends on the data characteristics
  - Fast Loss seems better on long-tail users/items (we are still exploring)

	BPR		Fast-loss	
	recall@20	NDCG@20	recall@20	NDCG@20
Gowalla	0.1823	0.1547	0.1575	0.1358
Yelp2018	0.0640	0.0531	0.0636	0.0522
Amazon-book	0.0416	0.0311	0.0379	0.0294
Amazon-office	0.0822	0.0413	0.1164	0.0730
Amazon-cellphone	0.0520	0.0234	0.0590	0.0302

Fast loss are better

Conclusion & Future Work

- Conclusion
  - Feature transformation and nonlinear activation increase the training difficulty and hurt the model accuracy
  - Smoother embeddings are more suitable for recommendation
  - Fast-loss brings comparable performance and great efficiency improvement compared with BPR
- Future work
  - Personalize the layer combination weights α_k
  - Streaming LightGCN for online industrial scenarios

THANK YOU!

Code(Tensorflow): https://github.com/kuandeng/LightGCN

Code(Pytorch): https://github.com/gusye1234/LightGCN-PyTorch