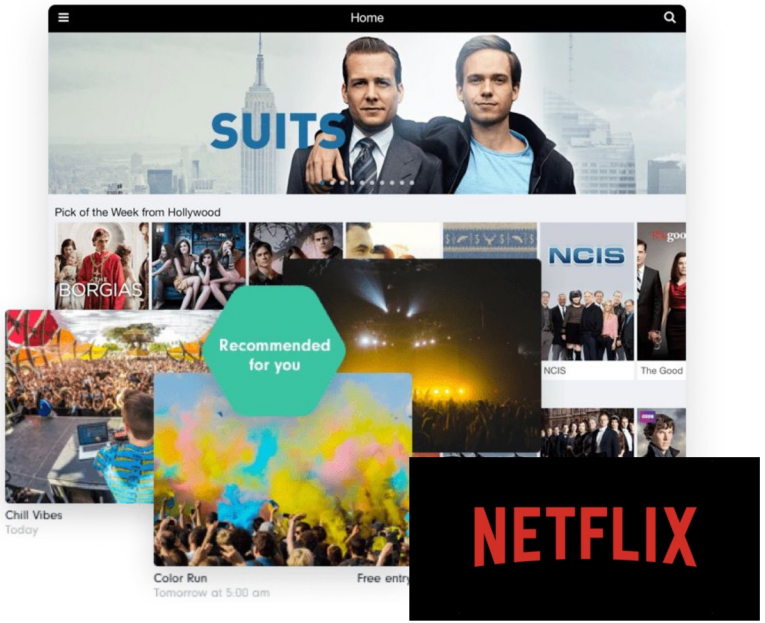
Alleviating Over-Smoothing in LightGCN for Improved Graph Representation Learning in Recommender Systems

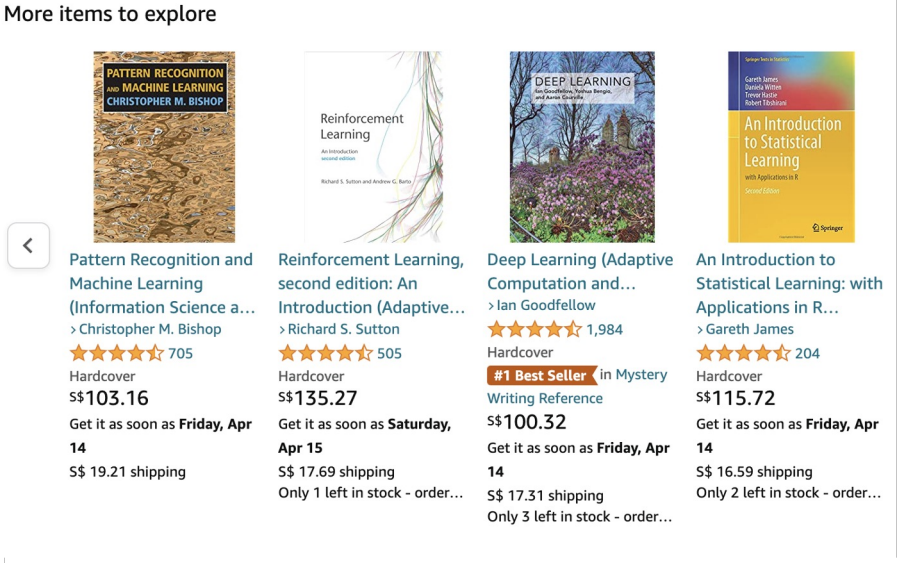
ST5188 Final Project Presentation (Project Group 14)

Yeo Ming Jie, Jonathan *Chen Pu* *Shi Qin*

Recommender Systems: Quick Intro

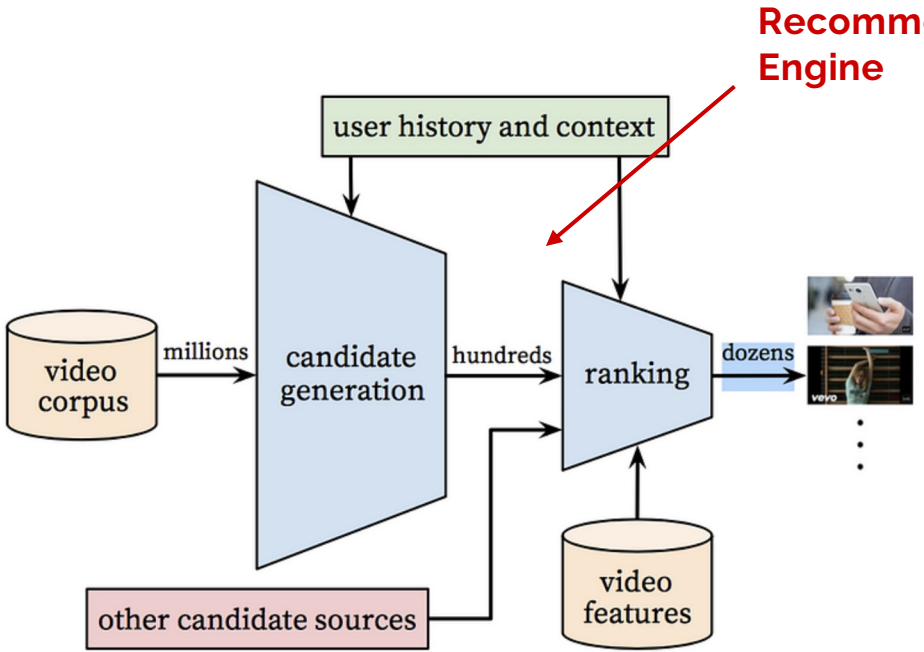
- Recommendation is everywhere - but how does it work?

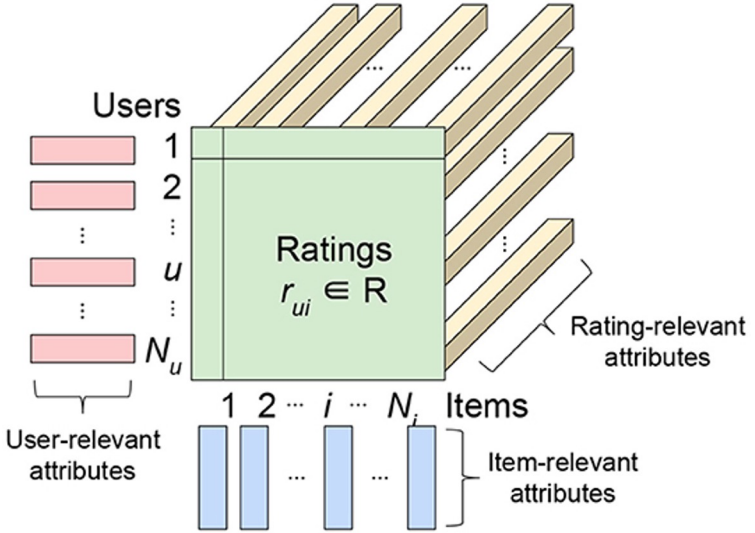




Recommender Systems: Quick Intro

- Recommendation is everywhere - but how does it work?



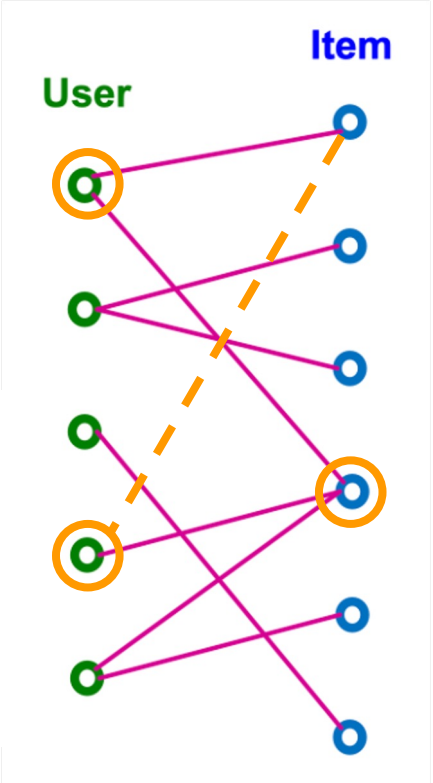


Frame as a representation learning problem

Recommender Systems as Graphs

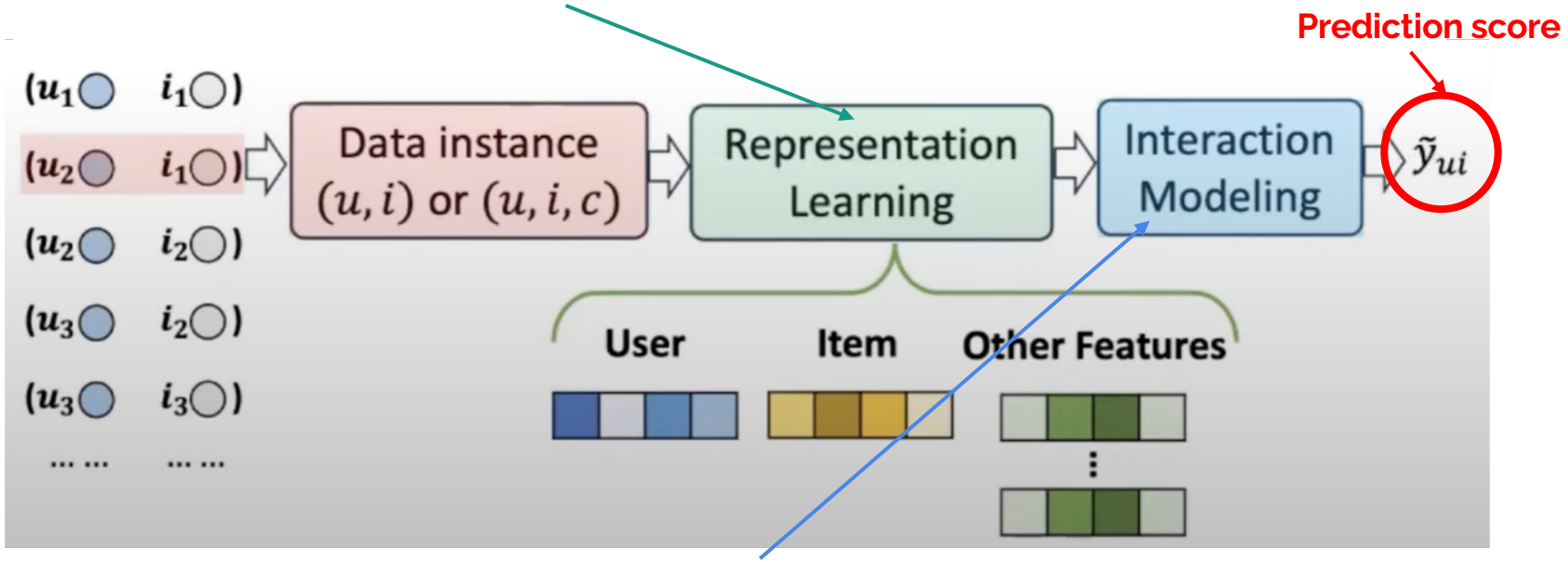
Collaborative Filtering (CF): Behaviourally similar users have similar item preferences

- **Recommender systems can be naturally modelled as a bipartite graph**
  - **Nodes: Users and Items**
  - **Edges: User-item interactions**
- **Goal: Recommend items users might like**
  - Cast as **link prediction** problem
    i.e. Given past edges, predict new user-item interaction edges



Recommendation as Representation Learning Task

Representation Learning: Generate representations for each input feature



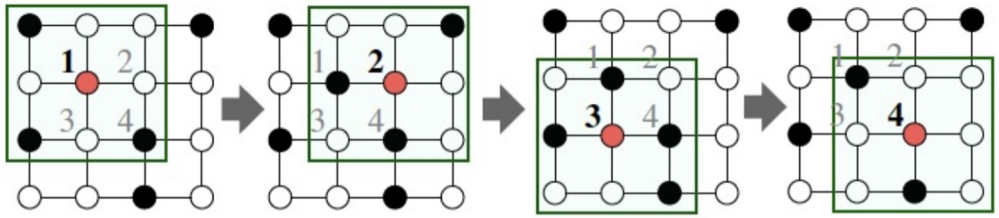
Interaction Modelling: Perform prediction based on user-item interactions

Use a Neural Network that can operate on graphs to perform Representation Learning

Graph Neural Networks (GNNs)

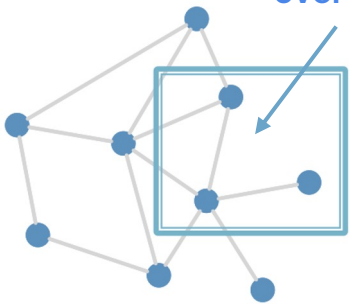
- Intuitively, GNNs can be viewed as a generalization of CNNs

CNNs: Convolution as message aggregation over a 2D grid



Feature Learning

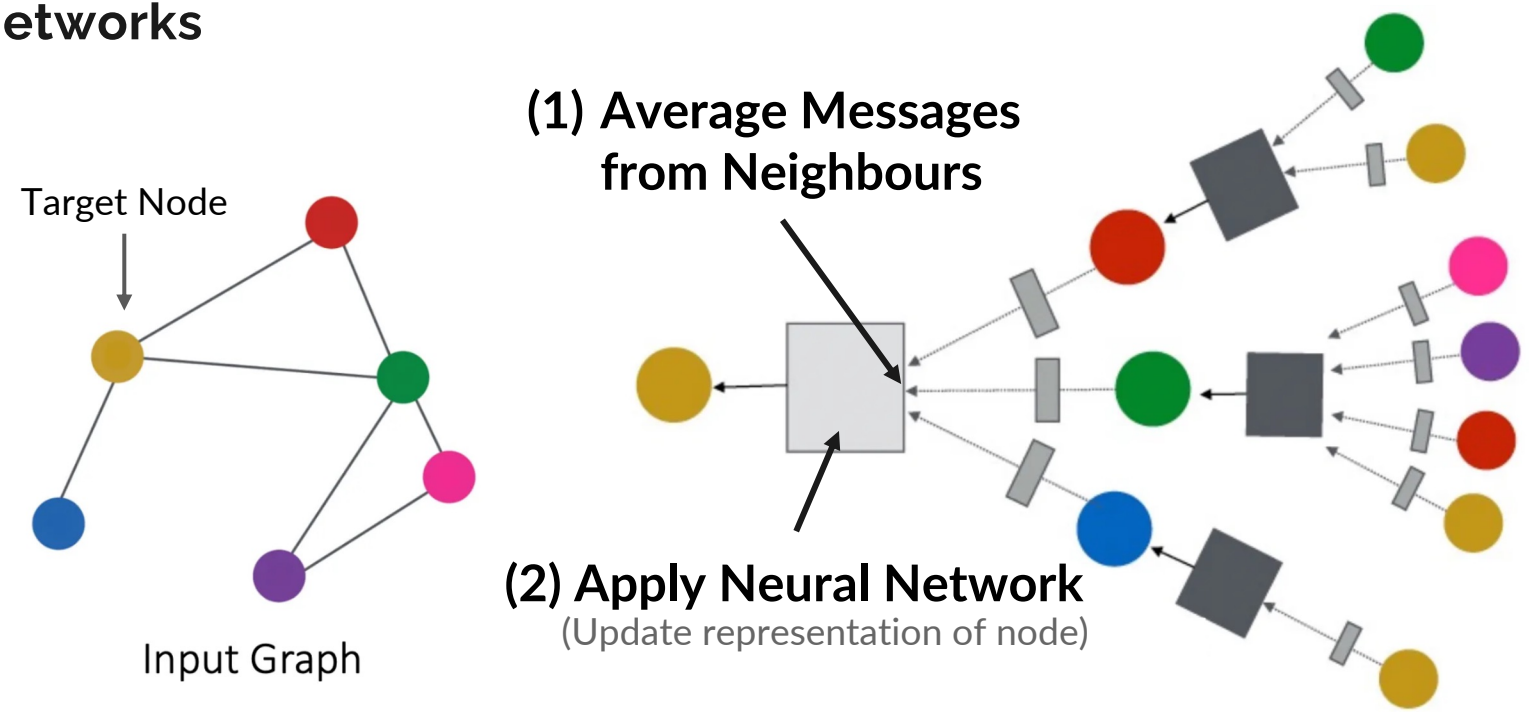
Message Aggregation over graphs?



GNNs consist of multiple permutation invariant operations

GNNs: Message Aggregation Framework

- **Nodes aggregate information from their neighbours using neural networks**



GNN Layer = Message + Aggregation

LightGCN Model for Recommendation

[He et al. 2020]

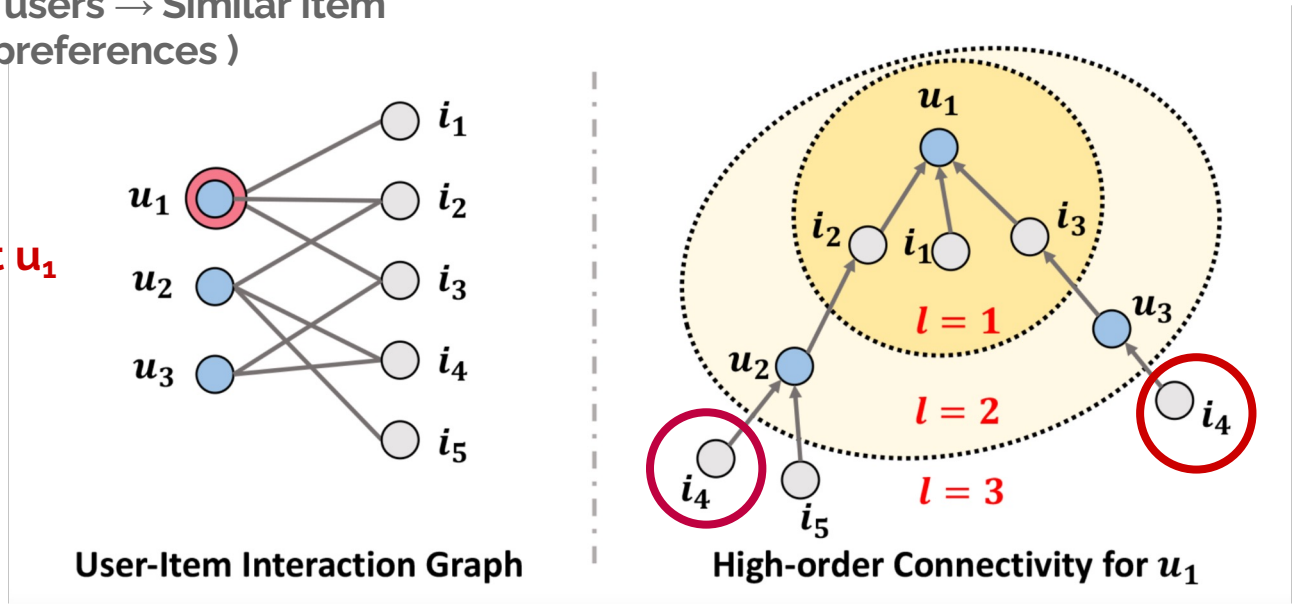
LightGCN Overview

- **Revisiting CF via Higher-Order Connectivity**

(Similar users → Similar item preferences)

Obtain the collaborative signal via Message Passing

Why might u_1 like i_4?



Higher-Order Connectivity as a natural way to **encode collaborative signal** in the interaction graph structure

LightGCN: Embedding Propagation Layers

- **Propagation Rule:**

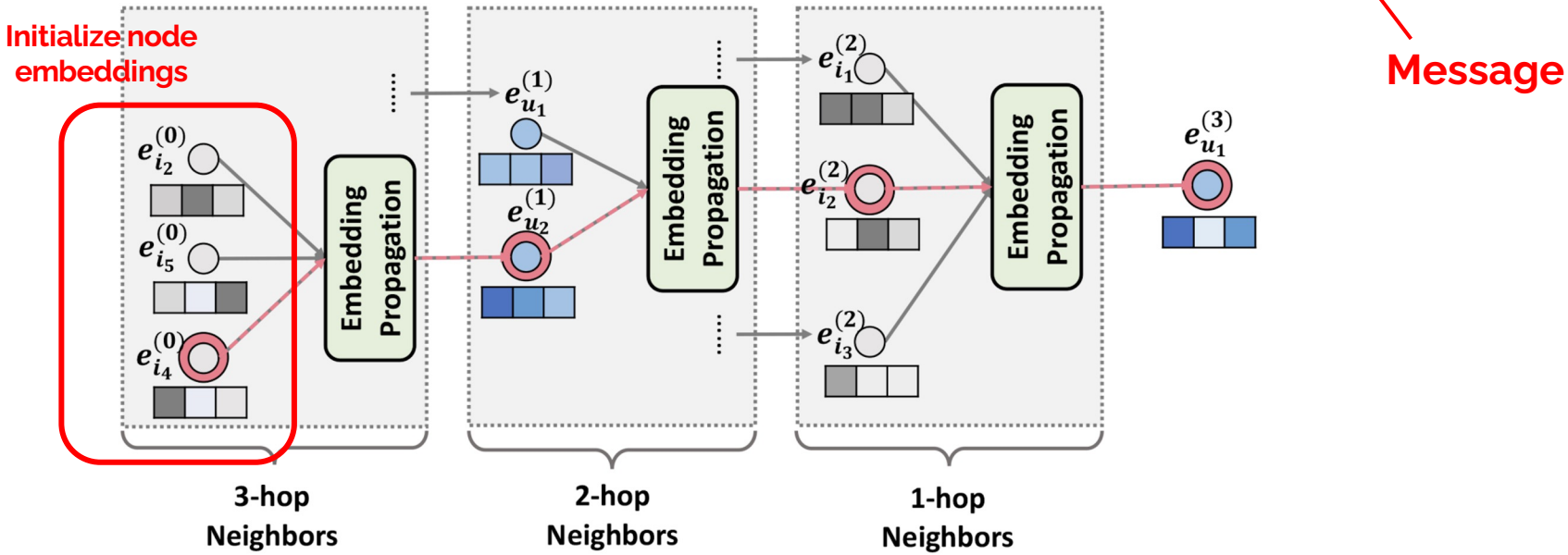
Aggregating over node neighbours

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

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

Message

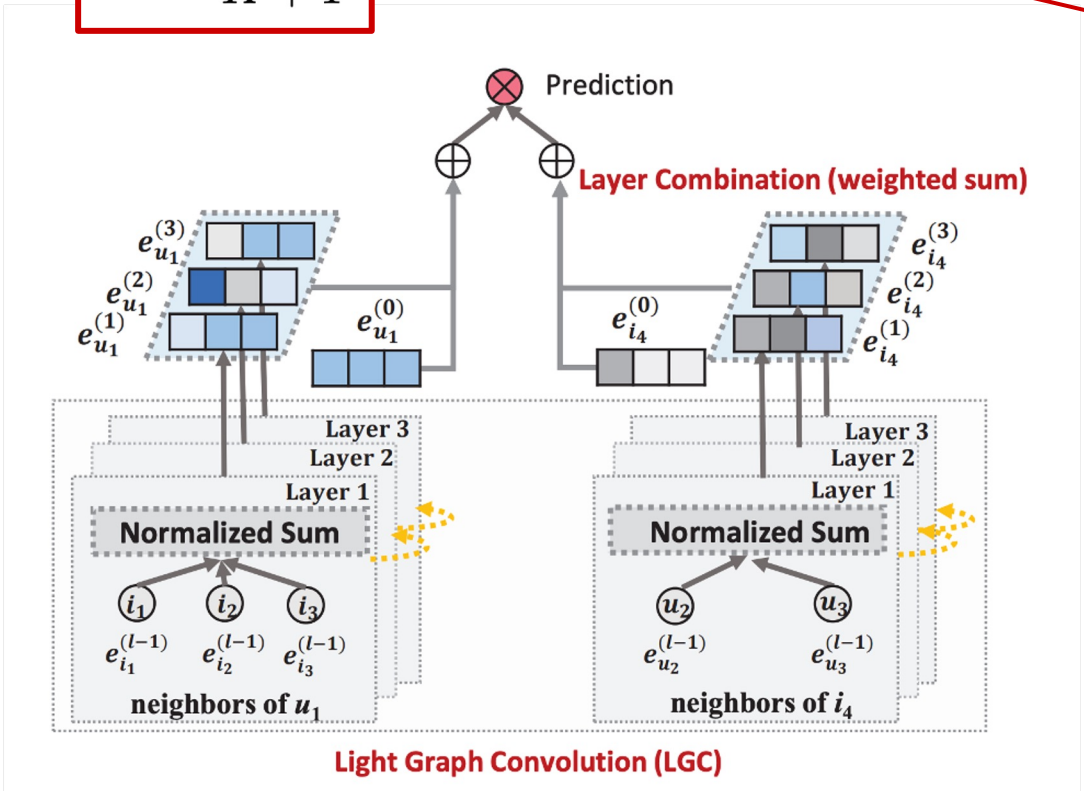
Illustration of Higher-order Propagation



LightGCN: Model Prediction

$$\alpha_k = \frac{1}{K+1}$$

Weighted linear combination of embeddings at each layer



- Final User Embedding:

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

- Final Item Embedding:

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

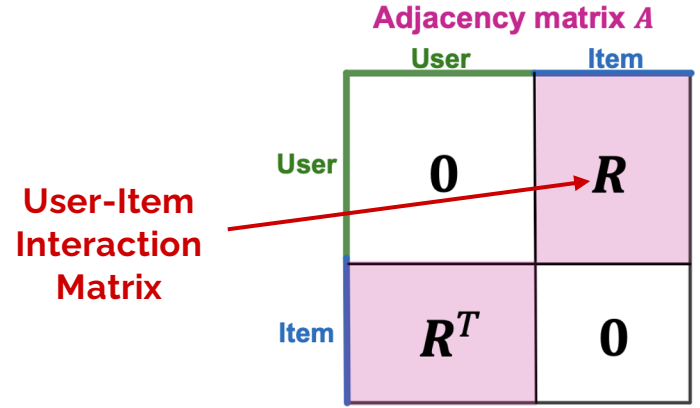
- Prediction: **(Taking inner product)**

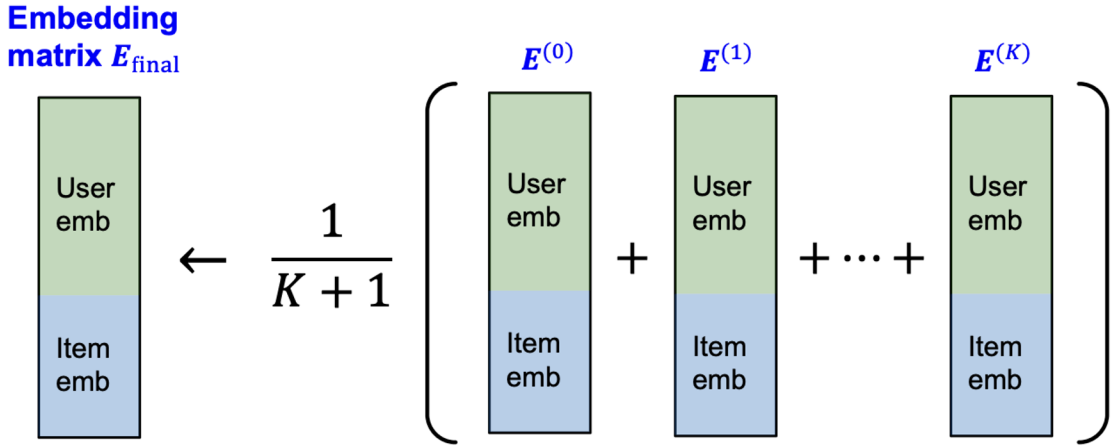
$$\hat{y}_{ui} = \mathbf{e}_u^T \mathbf{e}_i$$

LightGCN: Matrix Formulation

- **Propagation Rule (Matrix Form):**

$$\mathbf{E}^{(k+1)} = \underbrace{\left(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\right)}_{\text{Symmetrically normalized matrix } \tilde{\mathbf{A}}} \mathbf{E}^{(k)}$$





$$\leftarrow \frac{1}{K+1}\left(\mathbf{E}^{(0)} + \mathbf{E}^{(1)} + \cdots + \mathbf{E}^{(K)}\right)$$

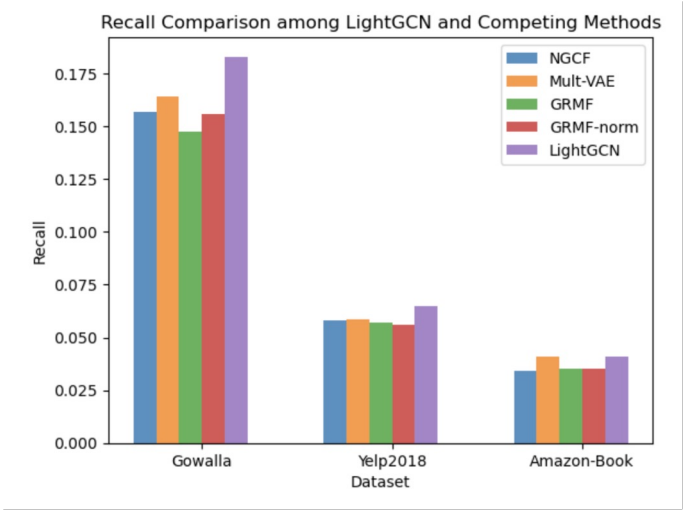
Final Embedding Matrix (Expansion)

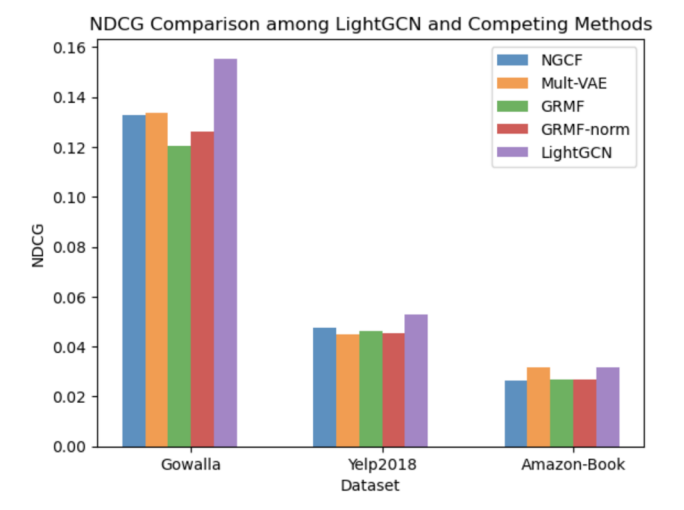
$$\mathbf{E} = \alpha_0\mathbf{E}^{(0)} + \alpha_1\mathbf{E}^{(1)} + \alpha_2\mathbf{E}^{(2)} + \ldots + \alpha_K\mathbf{E}^{(K)}$$

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

LightGCN: Successes and Limitations

LightGCN demonstrates strong performance over traditional GCNs on recommendation evaluation metrics.





Propagation Rule is simple, easily adaptable to various other tasks e.g. social recommendation systems (SocialLGN)

LightGCN: Successes and Limitations

LightGCN demonstrates strong performance over traditional GCNs on

- **High Computational Costs → Poor Scalability**
  - Passing the full graph through each propagation layer
  - Core component of neighborhood aggregation requires repeated multiplication of an adjacency matrix
  - e.g. 3 layer LightGCN requiring more than 700 epochs for convergence.

- **Over-Smoothing Problem**
  - Empirically, observed that LightGCN performance degrades after 2-3 layers.

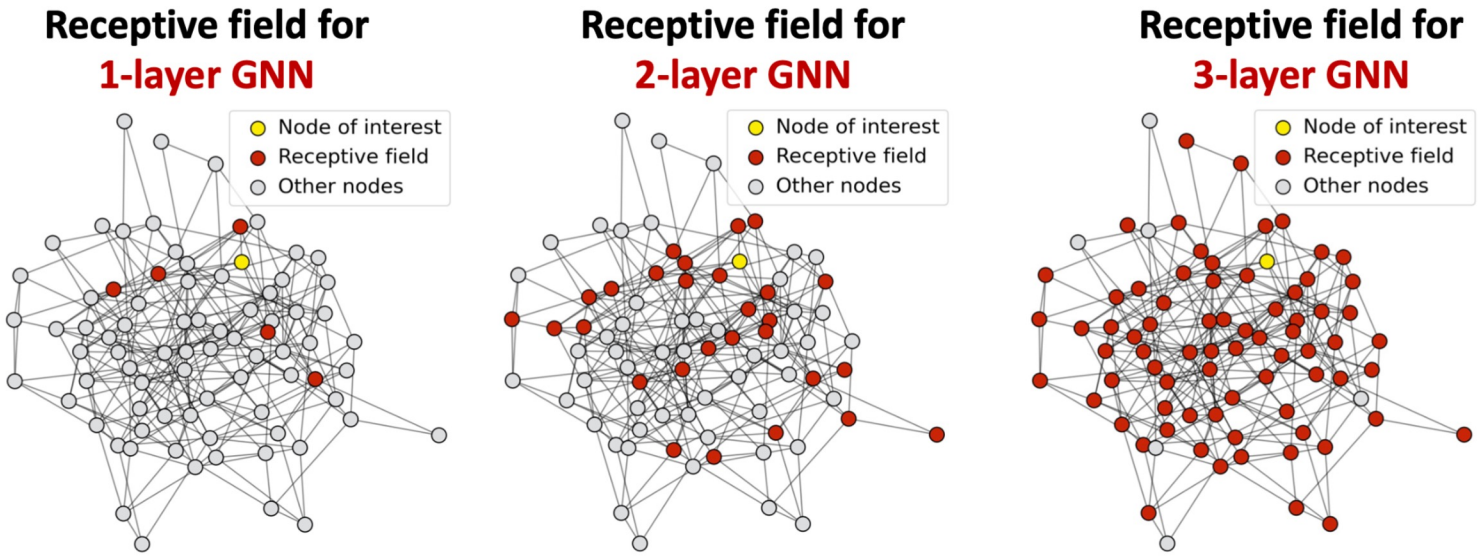
tasks e.g. social recommendation systems (SocialLGN)

The Over-Smoothing Problem

Over-Smoothing in GNNs (In General)

- Stacking too many GNN layers ⟶ All node embeddings converge to the same value

In a K-layer GNN, each node has a receptive field of K-hop neighbourhood



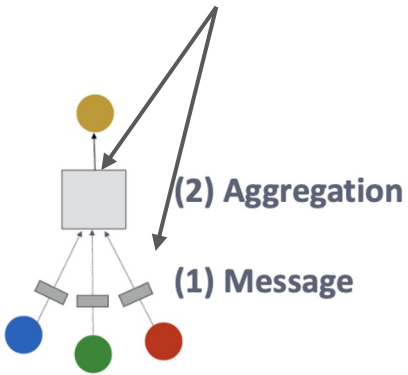
Receptive Field Overlap: Shared Neighbours Grows Rapidly

Traditional Approaches to Over-smoothing Problem

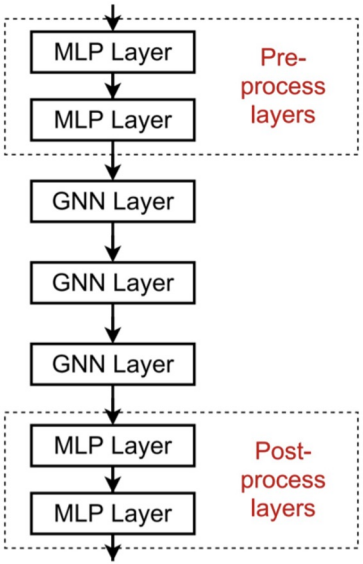
- **Main Lesson: Adding more GNN layers does not always help!**

Solution: Make a shallow GNN more expressive

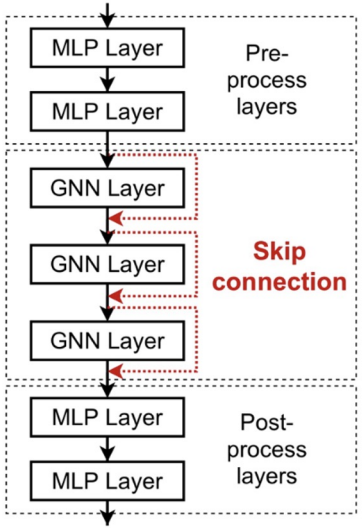
Each box (aggregation/transformation) could become a deep neural network



Increase expressive power <u>within</u> each GNN layer



Add non-message passing layers



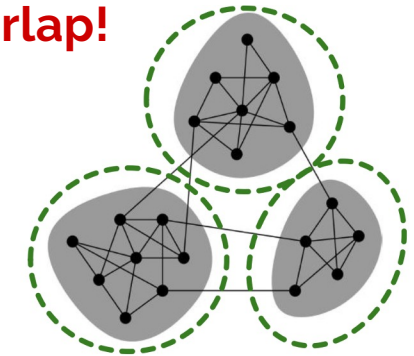
Adding skip connections

Proposed Solutions

Solution 1: Cluster-Based Sampling (Chiang et al, 2019)

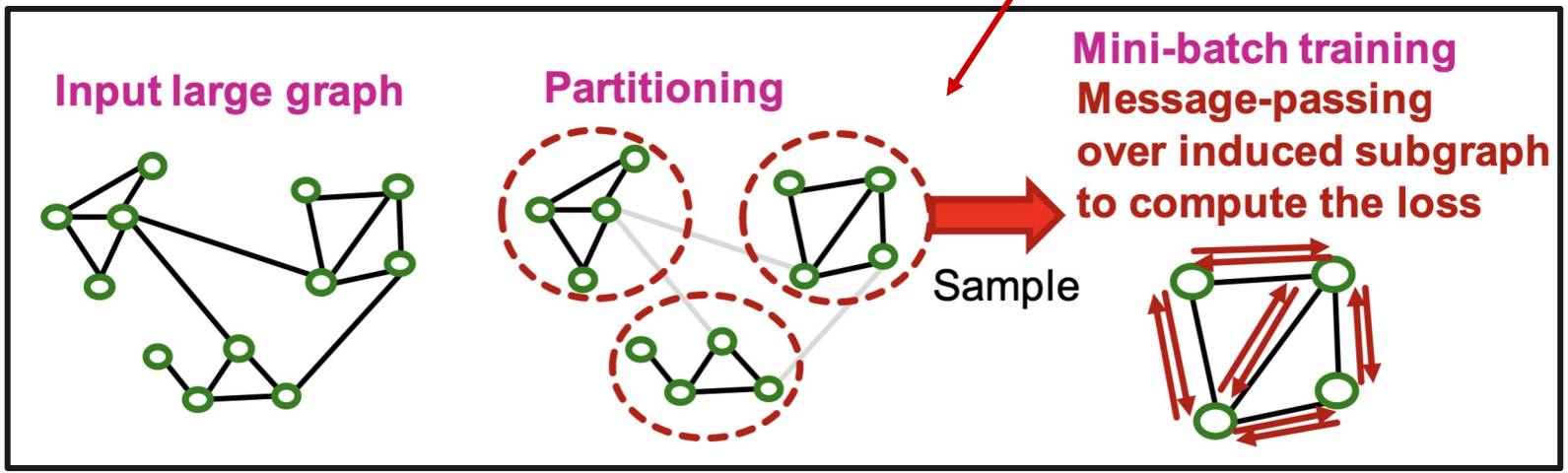
- In full-batch GNN, all node embeddings are updated together using embeddings of the previous layer

Minimize Receptive Field Overlap!



Real-world graphs exhibit community structure

Key Insight: Sample a community as a subgraph. Each subgraph retains essential connectivity patterns of the original graph



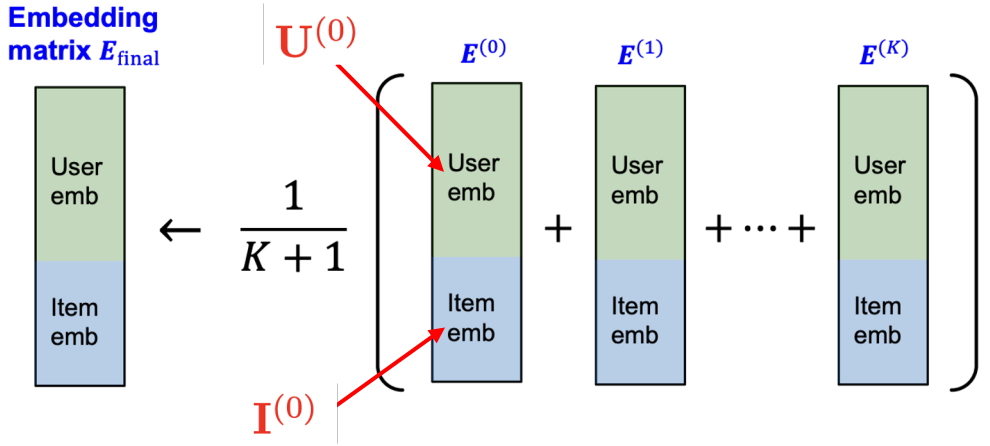
Solution 2: Collapsing User-Item Representation

Final Embedding Matrix (Expansion)

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

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

Consider 4-layer LGCN, let $\alpha_k = 1$ for all k



User-Item Embedding Matrices (By Block Matrix Multiplication)

$$\mathbf{U} = \mathbf{U}^{(0)} + \tilde{\mathbf{R}}\mathbf{I}^{(0)} + \tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\mathbf{U}^{(0)} + \tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\mathbf{I}^{(0)} + \tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\mathbf{U}^{(0)}$$

$$\mathbf{I} = \mathbf{I}^{(0)} + \tilde{\mathbf{R}}^T\mathbf{U}^{(0)} + \tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\mathbf{I}^{(0)} + \tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\mathbf{U}^{(0)} + \tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\mathbf{I}^{(0)}$$

$$\text{Set} \quad \mathbf{U}^{(0)} = \tilde{\mathbf{R}}\mathbf{I}^{(0)}$$

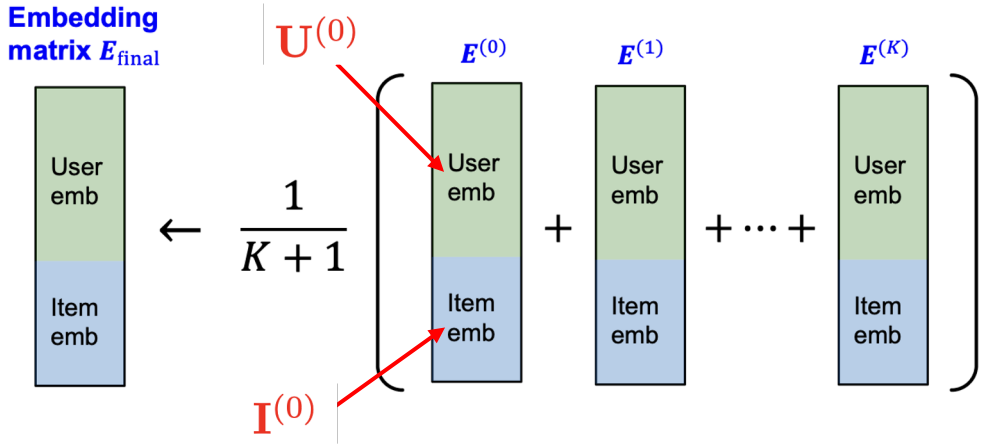
Solution 2: Collapsing User-Item Representation

Final Embedding Matrix (Expansion)

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

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

Consider 4-layer LGCN, let $\alpha_k = 1$ for all k



User-Item Embedding Matrices (SimpleLGN)

Step 1:

$$\mathbf{U} = 2\mathbf{R}\mathbf{I}^{(0)} + 2\mathbf{R}\mathbf{R}^T\mathbf{R}\mathbf{I}^{(0)} + \mathbf{R}\mathbf{R}^T\mathbf{R}\mathbf{R}^T\mathbf{R}\mathbf{I}^{(0)}$$

$$\mathbf{I} = \mathbf{I}^{(0)} + 2\mathbf{R}^T\mathbf{R}\mathbf{I}^{(0)} + 2\mathbf{R}^T\mathbf{R}\mathbf{R}^T\mathbf{R}\mathbf{I}^{(0)}$$

Note that the 4-Layer LGCN has reduced to 3-Layers of Propagation!

Final User Embedding matrix can be expressed fully in terms of the Item Embedding matrix!

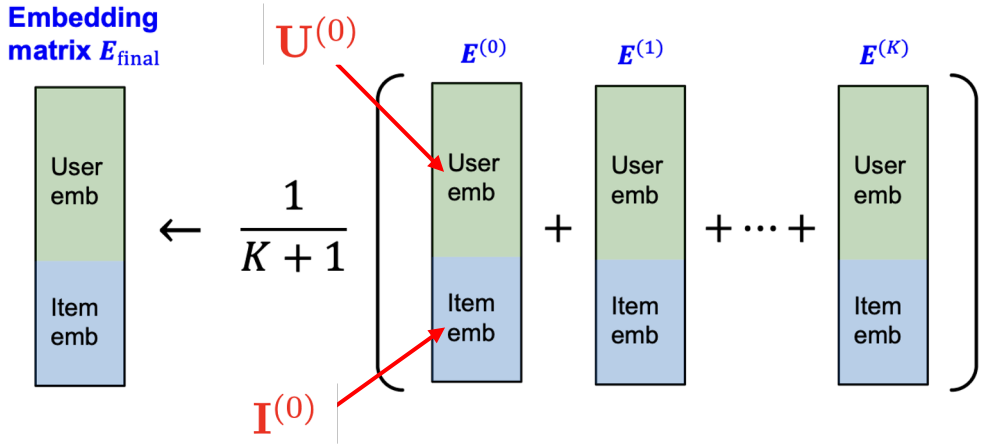
Solution 2: Collapsing User-Item Representation

Embedding Matrix (Update)

$$\mathbf{U} = 2\mathbf{R}\mathbf{I}^{(0)} + 2\mathbf{R}\mathbf{R}^T\mathbf{R}\mathbf{I}^{(0)} + \mathbf{R}\mathbf{R}^T\mathbf{R}\mathbf{R}^T\mathbf{R}\mathbf{I}^{(0)}$$

$$\mathbf{I} = \mathbf{I}^{(0)} + 2\mathbf{R}^T\mathbf{R}\mathbf{I}^{(0)} + 2\mathbf{R}^T\mathbf{R}\mathbf{R}^T\mathbf{R}\mathbf{I}^{(0)}$$

Step 2: Consider more simple LGCN, let $\alpha_k = \beta_k = 1$ or U(k)=RI(k) for all k



User-Item Embedding Matrices (Weight Sharing SimpleLGN)

Step 3: (Trainable)

$$\mathbf{U} = \alpha_0\mathbf{R}\mathbf{I}^{(0)} + \alpha_1\mathbf{R}\mathbf{R}^T\mathbf{R}\mathbf{I}^{(0)} + \alpha_2\mathbf{R}\mathbf{R}^T\mathbf{R}\mathbf{R}^T\mathbf{R}\mathbf{I}^{(0)} = \mathbf{R}\mathbf{I}$$

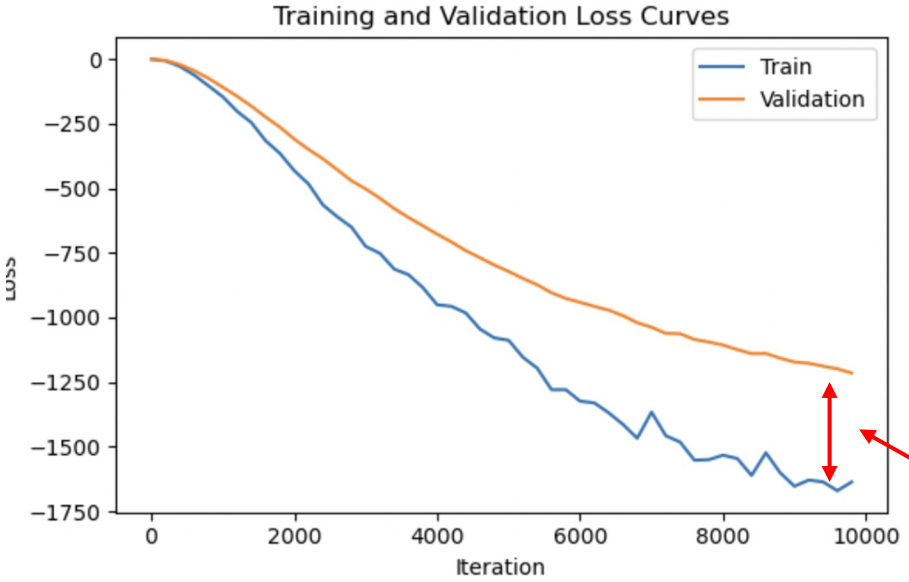
$$\mathbf{I} = \alpha_0\mathbf{I}^{(0)} + \alpha_1\mathbf{R}^T\mathbf{R}\mathbf{I}^{(0)} + \alpha_2\mathbf{R}^T\mathbf{R}\mathbf{R}^T\mathbf{R}\mathbf{I}^{(0)}$$

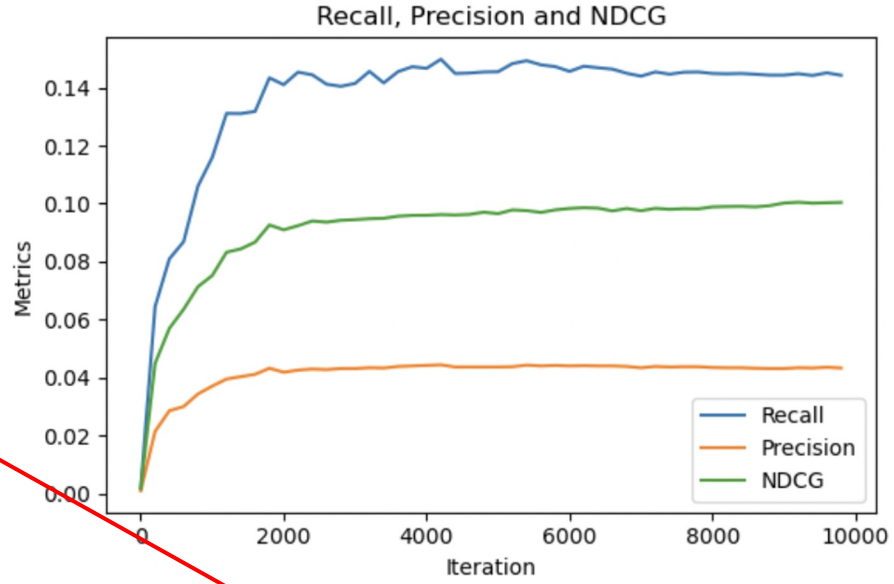
If we make the coefficients of user embedding same as the item embedding, then the final user embedding can be expressed as U=RI

Experimental Results: Cluster-Based Sampling

Experiment 1: Original LightGCN Model

- **Ran on own implementation of LightGCN model - MovieLens100K**





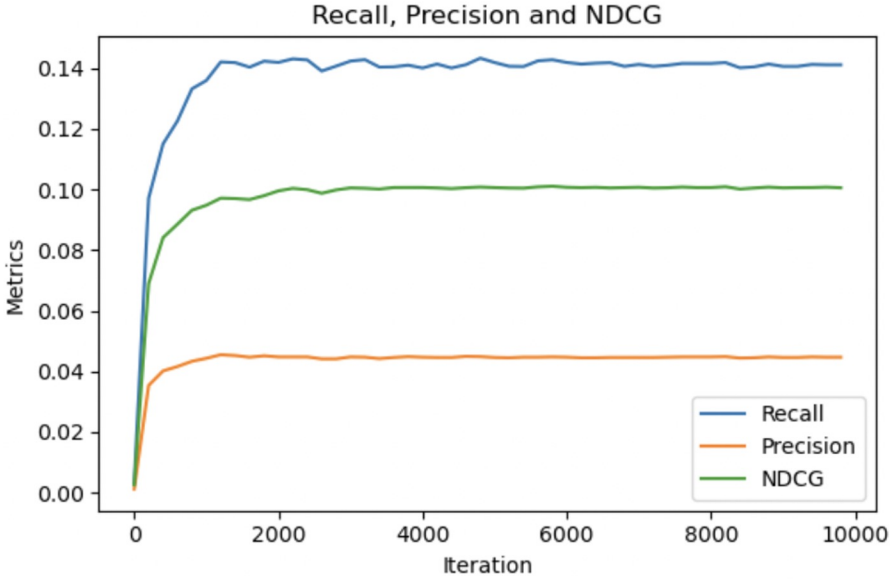
Training Time: 11 mins 47 secs

Overfitting observed

Note: 80 / 10 / 10 Training-Validation-Test Split

Experiment 1: Cluster-Based Sampling

Interesting result: Validation curve lies below training curve



Training Time: 6 mins 00 secs **(Halved Training Time)**

Experiment 1: Cluster-Based Sampling

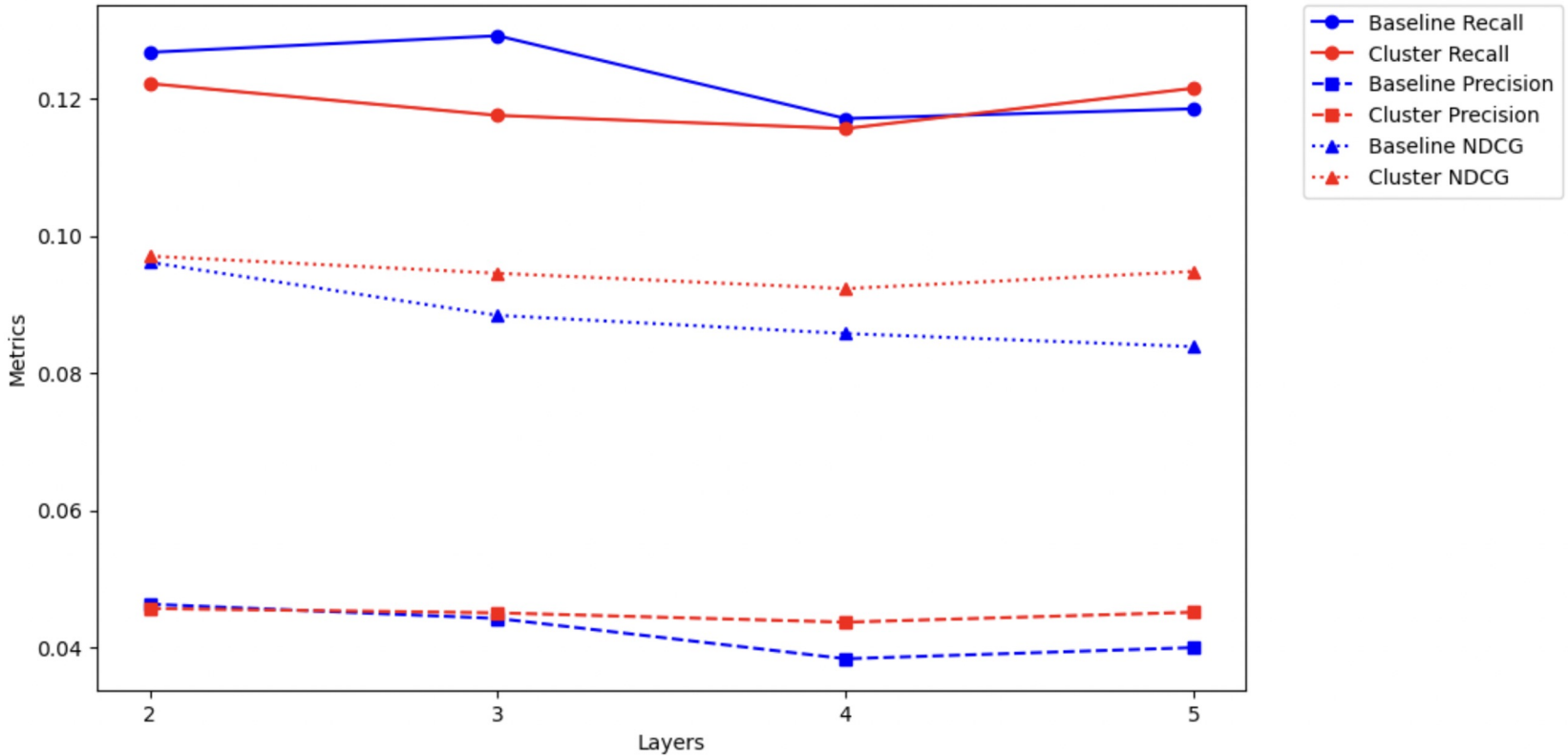
- **Training Time + Results over Test Data:**

Cluster-LightGCN has better Precision@20 and NDCG@20 scores over Baseline

Model	Layers	Training Time (s)	Recall@20	Precision@20	NDCG@20
Baseline LightGCN	3	707.76	0.12919	0.04420	0.08841
Cluster-LightGCN	3	360.55	0.11758	0.04502	0.09455
Baseline LightGCN	4	774.28	0.11712	0.03832	0.08576
Cluster-LightGCN	4	380.65	0.11566	0.04366	0.09231

Training time is halved!

Experiment 1: Cluster-Based Sampling



Summary of Results

1. **Training Time: Cluster-LightGCN models** consistently have **faster training** times compared to their corresponding Baseline LightGCN models.

1. **Recall@20: Baseline performs better** at lower layers, but **Cluster-LightGCN** has **more consistent trend**

1. **Precision@20: Cluster-LightGCN models** have slightly **higher precision** values compared to Baseline LightGCN models across all layer configurations

1. **NDCG@20: Cluster-LightGCN models** have slightly **higher NDCG values** compared to Baseline LightGCN models across all layer configurations

Cluster-LightGCN Model offers significantly faster training times and consistent, better recommendation quality compared to Baseline LightGCN

Experimental Results: Collapsing User-Item Representation

Experiment 2: Collapsing User-Item Representation

Ran on modified original source code of LightGCN on Gowalla benchmark dataset

Step1: Set U(0) = RI(0)

Best performance

	Recall@20	Precision@20	NDCG@20
4-L LightGCN	0.1794	0.05486	0.1526
3-L SimpleLGN	0.174	0.0516	0.1435

Step 2: Sharing weight (as fixed parameters)

3-L share-weight SimpleLGN	0.1688	0.04963	0.1398

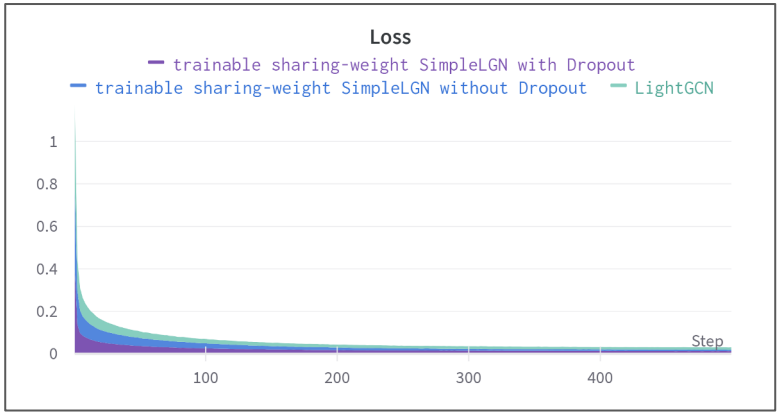
Moderate across all metrics

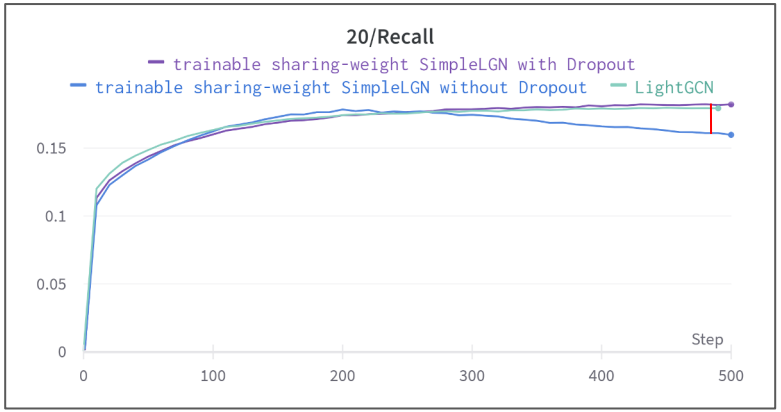
Step 3: Sharing weight (as trainable parameters)

3-L trainable share-weight SimpleLGN	**0.1822**	0.0536	0.1476

Experiment 2: Collapsing User-Item Representation

Overfitting in Trainable Sharing-weight SimpleLGN model



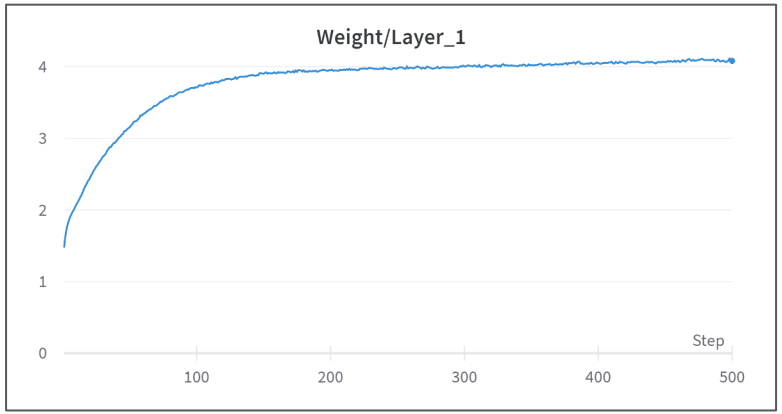


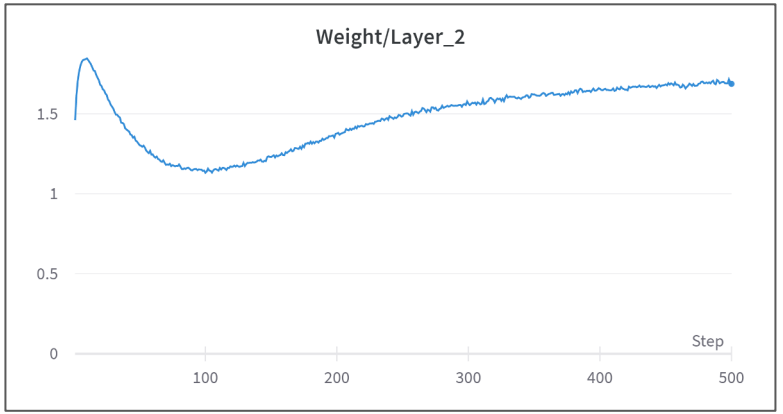
Converging faster

Recall@20: 0.1598 ——> 0.1822

Experiment 2: Collapsing User-Item Representation

Trainable Sharing-Weights





The weights/coefficients are converged to
α_1=4, α_2=1.7 (fixed α_0=1)

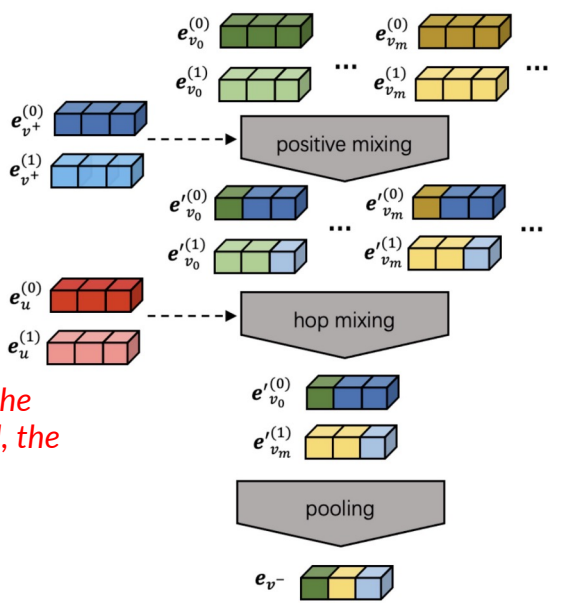
Improving Training

Tuned-MixGCF Overview

Tuned-MixGCF : A Refined Negative Sampling Model based on MixGCF

- **Phase 1: Positive Mixing**
  - Inject positive information of the target item into the *k* negative samples
  - The injection ratio of positive information is **trainable**

which is also the biggest difference between the original and refined model — in original model, the injection ratio is randomly selected

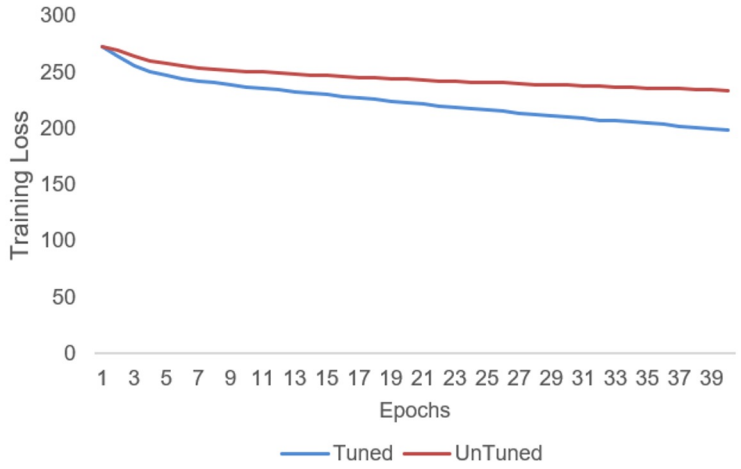


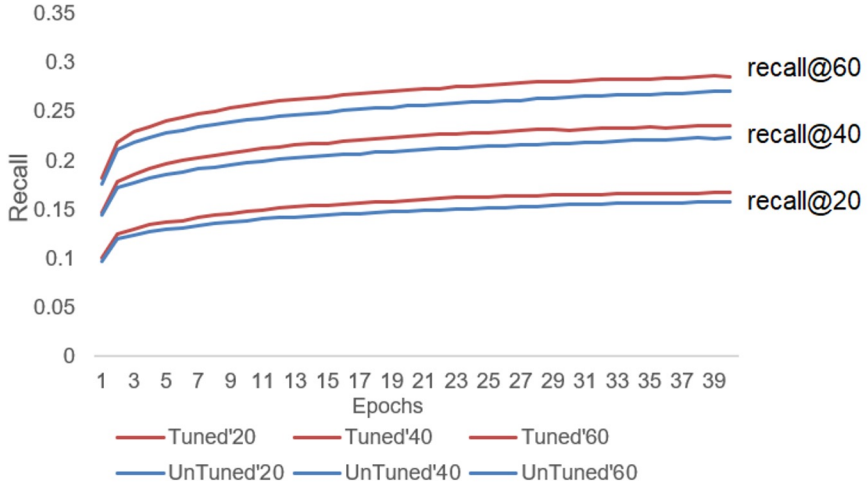
- **Phase 2: Hop Mixing**
  - For each layer *l* in LightGCN model, return *K* negative samples based on the Highest Product Score principle

* To alleviate the overfitting problem and improve the robustness of the model, we also add a regularization term in the final BPR loss function.

Experiment 3: Effectiveness of Tuned model based on LGN

Comparison: training results between Tuned-MixGCF and original MCGF





* Dataset: Gowalla, we only display 40 epochs because the test recall converges in about 30 epochs.

Experiment 3: Effectiveness of Tuned model based on LGN

Comparison: training results between Tuned-MixGCF and original LGN

	Model	Recall@20	Precision@20	NDCG@20	Training time
Dataset - Gowalla	4-L LightGCN	0.1794	0.05486	0.1526	22h
	4-L Tuned MGCF	0.1823	0.05621	0.1545	41min
Dataset - Yelp2018	4-L LightGCN	0.0649	…	0.0530	…
	4-L Tuned MGCF	0.0707	0.0318	0.0581	1h

Large improvement on both training result and training time!

Experiment 4: SimpleLGN + Tuned-MixGCF

Yelp2018 dataset

	Recall@20	Precision@20	NDCG@20
3-L LightGCN	0.0628	0.0281	0.0514
3-L LightGCN+Tuned-MGCF	0.0696	0.0313	0.0569
2-L SimpleLGN+Tuned-MGCF	0.0711	0.0320	0.0585
2-L share-weight SimpleLGN+Tuned-MGCF	0.07055	0.0318	0.0583
2-L trainable share-weight SimpleLGN+Tuned-MGCF	0.0717	0.0323	0.0589

+ 14.2%↑

Summary of Achievements/Notable Challenges

In Conclusion

Summary of Achievements:

1. **Demonstrated the effectiveness of Cluster-Based Sampling Approach in reducing training time and alleviating over-smoothing.**

1. **Introduced a novel approach of Collapsing the User-Item Representation in the LightGCN expansion, to obtain a more simplified form.**

1. **Tuned MixGCF Training Method can be used to boost both training speed and performance easily.**

In Conclusion

Challenges and Further Works:

1. **Integration of Cluster-Based Sampling on original source code for LightGCN model, deploy and experiment on Gowalla, Yelp benchmark datasets**

    - We were unable to integrate the Cluster approach in the limited research time, as this required extensive modification of the data preprocessing pipeline and mini-batch training pipeline.

1. **Exploring the joint effects of combining Cluster-Based Sampling with SimpleLGN (Collapsing user-item representation approach), Tuned-MixGCF**

Thank you!