Redefining Neighbor for Improved Knowledge Graph Based Recommendations

CS 224W Course Project

Ruge Zhao
rugezhao@stanford.edu

Meixian Zhu
mxzhu@stanford.edu

Jing Bo Yang
jingboy@stanford.edu

1 Introduction

Recommender systems are ubiquitous in today's customer facing services. Traditionally, researchers have modelled recommender system as a collaborative filtering problem. Data prepared for this type of problem can be represented by a tuple (c, r, i), indicating customer c has interacted through relation r with item i. These tuples can also be stored in matrix form, with each array element representing the user's prior interactions with the items (e.g. watched, rated, bought the item) for each user-item pair. The downside of traditional collaborative filtering method is that it only makes use of information in the utility matrix but not metadata available for items. In other words, the problem formulation only looks for similar users who also bought the item, or similar items bought by the same user, ignoring other similarity in items such as movies starred by the same actor or books authored by the same person. Inclusion of knowledge graph addresses this issue, but introduces a rather sophisticated set of new relations and millions more entities. Combining user-item interactions with knowledge graph results in a hybrid graph that requires more advanced handling and training methods. Current state-of-the-art models which build on graph convolutional networks retain existing neighborhood structure as-is. While this approach respects the given network structure, it leaves out potential correlations among similar items and structures in the network. As pointed out by Wu (1), distant neighbors bring little information, thus decreasing the utility of additional convolution layers, yet no existing work can be found on alternative local neighborhood definition. More intuitively, instead of growing graph convolution in depth, we would like to change its grows horizontally. In this work, we bridge the gap by experimenting with a number of neighborhood definitions, hoping to improve embedding propagation, a step critical for utilizing internal structures and similarities within the knowledge graph. Neighborhood definitions experimented in our work include grouping by cluster, choosing top-k similar nodes and pooling 2-hop neighbors. To test our ideas, we simultaneously developed a more generic framework for `Pytorch Geometric` to support processing of different types of graphs with interchangeable neighbor discovery and definition function. Through our experiments, we are able to show that using nodes from 2-hops away yields the improved result, leading to recall@20 of 0.1495 and ndcg@20 of 0.1011 for `Amazon Book` dataset and ndcg@20 of 0.1327 for `Last-FM` dataset, higher than that achived by vanilla KGAT-implementation. More significant improvements are observed when the models are trained without pre-trained embeddings under the same experiment condition.

2 Related Work

Recommender systems require algorithms that identify similarities among users and evaluate user preferences by analyzing connection between items of interest. Previsouly researchers have used memory-based algorithms and matrix factorization based algorithms (2) with limited success. These systems face the cold start problem (3) and sparse information links between users and items (4). They are difficult issues to tackle without prior knowledge of user or items, hence the quest for obtaining informative embeddings for entities of concern.

Graph neural networks combined with knowledge graph has been used in conjunction to ensure effective propagation of information carried by embeddings and existence of prior knowledge, along with a propagation network. At first, only user-item interactions are used for embedding generation (5) (6) (7). Graph neural network methods used by these works can largely be categorized into one of TransE (8), TransH (9), TransR (10) and TransD (11).

Building on the success of the above neural collaborative filter methods, information from knowledge graph was introduced. Knowledge graph is a directed graph whose nodes correspond to entities such as properties and characteristics, whereas edges correspond to relations, like "type" and "location". Most importantly, knowledge graph, commonly sourced from Freebase (12), has rich existing metadata from which initial item embeddings can be trained. Researchers have taken two ways to synthesize information from knowledge graph. 1) Embedding-based methods that aims to shape embeddings to follow certain desired properties through various embedding comparison metrics (13). 2) Path-based methods which regard knowledge graph as a heterogeneous information network and bases training on latent relations discovered along graph traversal paths (14). A "compromise" between these two approaches is using attention. Since attention-modified message is $\pi(h, r, t) = (W_r e_t)^T \cdot tanh(W_r e_h + e_r)$, node embeddings can be propagated more efficiently, thus behaving like weights for multiple paths. Wang et al's work (15) is a great example of applying attention method to GNN along with embedding-only training methods, and has achieved 0.0712 recall@20 compared to 0.0664 from other state-of-the-art baseline such as RippleNet (16).

3 Dataset

We present data analysis on 3 real-world datasets in this section. In section 3.1, we summarize statistics on types of nodes and relation edges in the 3 directed graphs. Section 3.2 shows a direct comparison of clustering coefficient on the 3 datasets to contrast the difference in interaction sparsity. Section 3.3 shows our preliminary clustering analysis using various clustering algorithms.

3.1 Dataset description

To minimize data preprocessing efforts on mapping items to entities in knowledge graph (extracted from Freebase (17)) and ensure fair performance comparison, we decided to use the same datasets as the KGAT paper (15). These datasets have defined the mappings between items and entities in Freebase. To be more specific, these datasets are `Amazon book` dataset (18) (19), `Last-FM` dataset (20) and `Yelp2018` dataset (21). Table 1 presents statistics of these 3 datasets that we plan to use. Figure 5 (See Appendix) shows the distribution of different relation types present in each of the 3 datasets.

	Type	Amazon Book	Last-FM	Yelp2018
User-Item Interaction	# Users	70,679	23,566	45,919
	# Items	24,915	48,123	45,538
	# Interactions	84,733	3,034,796	1,185,068
Knowledge Graph	# Entities	88,572	58,266	90,961
	# Relation Types	39	9	42
	# Triplets	2,557,746	464,567	1,853704

Table 1: Statistics of prepossessed datasets (15)

3.2 Modified clustering coefficient

Since user-item interactions form a directed bipartite graph, we cannot use the simple definition of clustering coefficient $\frac{3\times\#\text{of triangles}}{\#\text{of triplets}}$. In other words, only a user and an item can be connected, and it is impossible to see an edge between two users or two items. Instead, similar to one of our neighborhood definition methods in Section 4.1.1, we recognize 2-hop (treat the directed graph as undirected) neighbors as connected. With this new definition, we calculated the clustering coefficients of user-item relation graphs for the three datasets and summarize the clustering coefficients in the table below:

	Amazon Book	Last-FM	Yelp2018
Clustering Coefficient	0.250	0.433	0.251

Table 2: Modified clustering coefficient for reference datasets

We observe that `Yelp2018` and `Amazon Book` datasets have similar sparsity and are both more sparse than `Last-FM`. This observation is consistent with the fact that KGAT performs better on sparse datasets than regular methods (15) since it incorporates high-order connectivity. One possible reason being preference of users with too many interactions is too vague to capture. This observation motives us to expand embedding propagation set to contain lower-order connectivity as well as the representations of inactive users and similar users via neighborhood expansion.

3.3 Preliminary clustering analysis

We performed preliminary clustering analysis with the pre-trained embeddings on the `Yelp2018` dataset. First, we generated a t-SNE plot for all pre-trained embeddings colored by whether they are item or user embeddings in Figure 1. Here, we see that user and item embeddings mostly overlap, with only small shifts representing a user-item interaction, showing that embedding learned for this interaction is effective. This observation also provides support for potential success of training phase 2 of KGAT, during which triplets are learnt based on the existence of such interactions. Moreover, we observe that there are around 13 clusters, a fact to be used as a benchmark of the number of clusters for re-defining neighborhood based on clustering.

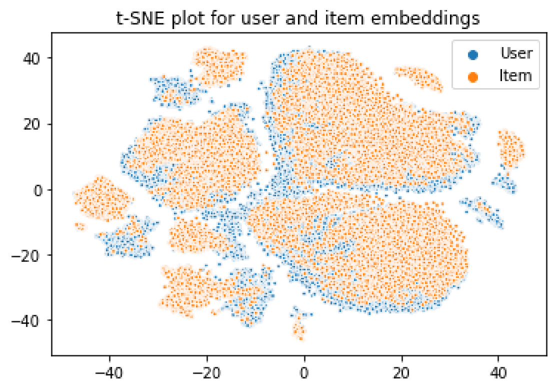
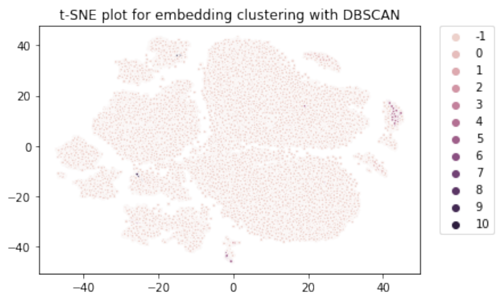
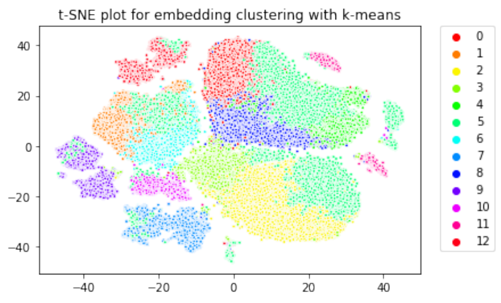


Figure 1: t-SNE plot of `Yelp2018` user-item interaction data

We then experimented with two clustering methods: DBSCAN (22) and K-Means clustering(23). DBSCAN is a density based clustering algorithm, with the advantage of identifying optimal number of clusters on its own. While DBSCAN is great at separating high density clusters from low density ones, it struggles with clusters of similar density, which happens to be a characteristic of our dataset. We plot the clustering result using DBSCAN as in Figure 2a, with minimum distance between clustered tuned at 0.3. This gives us 11 clusters (cluster assignment of -1 means it is noise and not a real cluster). However, almost all embeddings are clustered into a big group due to them having similar densities. In contrast, as shown in Figure 2b, result from K-Means is significantly better using $k = 13$ because K-Means is not density based.



(a) Clustering result using DBSCAN



(b) Clustering result using K-Means

Figure 2: Comparison of `Yelp2018` embedding clustering results

Although result from direct clustering algorithms vary, graphically, this comparison gives a reasonable result as we see each t-SNE clusters are of reasonable size with clear boundaries. Moving on, we will use K-Means with $k = 13$ for our clustering algorithm in *Node Clustering Phase* of our algorithm (see Section 4.1).

4 Methods

We adopted a 3-phase alternating training algorithm which is elaborated in section 4.1. We will introduce each of the 3 training phases in section 4.1.1 to 4.1.3. Evaluation metrics and baselines are also introduced later in the section.

4.1 Training methods

We modified the existing 2-stage KGAT (15) training scheme to a 3-phase training algorithm for a neighborhood-optimized recommender system: 1) node neighbor definition phase 2) embedding learning phase 3) collaborative attentive propagation phase. These 3 phases will be executed in an alternating fashion for each epoch at training time. The nature of this training setup limits meaningful evaluation to once per epoch, as all nodes' embeddings must be updated first before being used for evaluation/testing.

4.1.1 Neighborhood definition phase

Three additional node neighbor definitions were experimented with during our evaluation. These neighbor definitions focus on different ways of associating nodes together. Details of these methods are outlined below.

Clustering Re-defining neighbors of a node via clustering breaks down the existing graph structure. Since a relatively small number of clusters is observed across the entire graph, as illustrated in Section 3.3, forcing a node's neighbor to be within the same cluster lets those within the cluster to behave in a similar pattern. We experimented with $n_{cluster} = 13$ and $n_{cluster} = 25$ for our purpose. Clustering is performed once every epoch and for nodes in the same cluster, an edge is added from the original adjacency matrix with probability $p_{keep} = 0.01$. In results section, we will use "Clustering" to represent this neighborhood definition.

Choosing top-k similar user nodes To bridge the gap between a few selected groups and huge number of meaningless clusters, we introduce the method of adding top-k similar nodes as neighbors. For every user, we will look up its most similar K user nodes in terms of Euclidean distance. Then interactions between this user and the items purchased by the most similar K users. This is similar to RippleNet (16), where propagation of embeddings start at multiple locations. In our implementation, we experimented with $K = 10$. "Closest K" will be used to refer to this neighborhood definition.

Pooling 2-hop neighbors The original KGAT implementation included only 3 layers. While restricting the total number of layers reduces computational cost, it fails to include additional information from further neighbors. Reducing 2-hop neighbors to 1-hop can be expressed in manipulation of corresponding adjacency matrix. This setup also helps with the sparsity problem in recommendation system by promoting more nodes as neighbors. "2-hop" will be used in short to represent this method.

Let A be the original directed adjacency matrix (dimension: $(n_{user} + n_{item}) \times (n_{user} + n_{item})$) and $\tilde{A}$ be the adjacency matrix if we convert the graph to an undirected graph.

$$A_{2-hop} := \mathbb{1}(\tilde{A}\tilde{A} > 0)$$

where $\mathbb{1}()$ is an element-wise indicator function on a matrix.

4.1.2 Embedding learning phase

TransR (10) was used on collaborative knowledge graph to get embeddings and we optimize for $\mathcal{L}_{KG} = \sum_{(h,r,t,t')\in\mathcal{T}} -ln\sigma(g(h,r,t') - g(h,r,t))$. $\mathcal{T}$ are the broken triplets constructed by replacing one tail entity in a valid triplet randomly. Here $g(h,r,t) = \|W_r e_h + e_r - W_r e_t\|_2^2$ is a plausibility score to optimize the translation.

4.1.3 Collaborative attentive propagation phase

- **Attention mechanism (24)**: $e_{\mathcal{N}_h} = \sum_{(h,r,t)\in\mathcal{N}_h} \pi(h,r,t)e_t$ where $\pi(h,r,t) = (W_r e_t)^T tanh(W_r e_h + e_r)$ is the attention score that depends on the distance between e_h and e_t in the relation r's space. Note that $\pi(h,r,t)$ is then softmax normalized: $\pi(h,r,t) =$

$\frac{exp(\pi(h,r,t))}{\sum_{(h,r',t')\in\mathcal{N}_h} exp(\pi(h,r',t'))}$. The attention mechanism allows the model to give more attention to some neighbor nodes than the others.

- **Aggregation**: After all these calculations, we aggregate the entity representation e_h and its egonet's representation $e_{\mathcal{N}_h}$: $e_h^{(1)} = f(e_h, e_{\mathcal{N}_h})$. For example, $f = LeakyReLu(W(e_h + e_{\mathcal{N}_h})$ (GCN (25)), $f = LeakyReLu(W(e_h || e_{\mathcal{N}_h})$ (GraphSage (26)) and $f = LeakyReLU(W_1(e_h + e_{\mathcal{N}_h})) + LeakyReLU(W_2(e_h + e_{\mathcal{N}_h}))$ (Bi-interaction).
- **Higher-order propagation**: in l-th step $e_h^{(l)} = f(e_h^{(l-1)}, e_{\mathcal{N}_h}^{(l-1)})$
- **Predictions**: For a pair of user u and item i, we concatenate the node embeddings from all orders:

$$e_u^* = e_u^{(0)} || \dots e_u^{(L)} \tag{1}$$

$$e_i^* = e_i^{(0)} || \dots e_i^{(L)} \tag{2}$$

  The predicted interaction is $\hat{y}(u,i) = e_u^{*T} e_i^*$
- **Loss**: In this phase, we optimize for $L_{CF} = \sum_{(u,i,j)\in O} -ln(\sigma(\hat{y}(u,i) - \hat{y}(u,j)))$. $\mathcal{O}$ denotes the data used for training which comprises of observed positive (i.e. existent) interactions (u,i) and simulated negative (i.e. the interaction does not exist in our dataset) pair (u,j).

4.2 Evaluation metrics

We evaluate recommendation performance the same way as other state-of-the-art models. These evaluation metrics include recall, hit and normalized discounted cumulative gain (ndcg) for *top-K* items. Recall is defined as the proportion of correct predictions out of number of items the user truly interacted with. Hit rate is defined as an average of whether at least 1 correct prediction is present across all users, written as $\sum_{user} \mathbb{1}(\vec{y} \cdot \vec{y_{pred}} \geq 1)$. Discounted Cumulative Gain (dcg) is defined as $r_1 + \sum_{i=2}^{K} \frac{r_i}{log_2(i)}$. ndcg is calculated by normalizing DCG at rank n by the dcg value at rank n of the ideal ranking.

4.3 Baselines

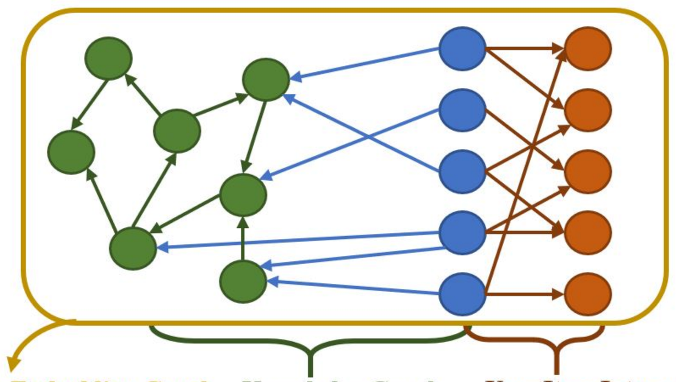
We have implemented Singular Value Decomposition (SVD) (27) and replicated the Knowledge Graph Attention Network model (KGAT) (15) as our baseline models. SVD is a dimension reduction technique and used in traditional collaborative recommender systems. We use SVD as a weak baseline. KGAT is the state-of-the-art and the setup is similar to section 4.1 without the neighborhood expansion methods. Performance of these baseline methods are compared against our custom neighborhood definitions in Section 5.2. From this point onwards, we will refer to KGAT algorithm as “vanilla KGAT” and short forms defined in section 4.1.1 to refer to our models.

5 Results and Analysis

In this section, we describe our efforts in implementing our algorithms with some implementation details and compare different models' performance.

5.1 Implementation

The original KGAT implementation is written using recent versions of `Tensorflow v1`. This implementation uses pre-defined neighborhood (vanilla KGAT) during preprocessing, conflicting with our goal of using an alternative neighborhood definition dynamically during model training. We therefore decided to overhaul the original structure and migrate it to follow an existing `Pytorch` framework originally used for time-series processing (our previous implementation from CS341). We spent substantial amount of time modifying its `Dataset` and `DataLoader` functionality, including a more flexible neighborhood sampler and sharing node embeddings for different “batch" definition used by knowledge graph and user-item interaction bipartite graph. `Tensorboard` has also been setup for proper experiment logging along with monitoring of training progress powered by live transferring of necessary files to `Google Cloud Storage Buckets` to avoid keeping a `Google Filestore` NAS (network attached storage) instance, which costs \$250+ per month. In a sharp contrast, with small amount of change in logging infrastructure, cost of keeping experiments are reduced to dozens of dollars with our new setup. Structure of this new general graph neural network training implementation is shown in Figure 3.



Major Components	Generic Embedding Graph	Knowledge Graph	User-Item Interaction
Propagation	TransR (no propagation)	Attention & GraphSage/GCN	None
Loss Function	Triplet embedding loss	Triplet embedding loss	None
Evaluation	None (loss/clustering only)	None (loss/clustering only)	Top-K NDCG/Recall

Figure 3: Structural breakdown of a modular implementation for joint user-item interaction with knowledge entities graph

Clarification *Although we have implemented the above infrastructure, its performance is far worse than the existing `Tensorflow` code base (3x as slow in per-step time using the same batch size). We estimated that using our `Pytorch` implementation will not allow us to conduct all required experiments and finish the analysis on time. Results obtained in the following sections are still obtained using a modified version of the original KGAT implementation. We ported the `Google Cloud Bucket` storage and logging infrastructure to enable more efficient experiment logging on Tensorboard. We will make public our Github repository for this new infrastructure.*

5.2 Comparison of neighbor definitions

We present a comparison of our methods using different neighborhood expansion definitions and the baseline models (SVD and Vanilla KGAT(15)) in Table 3 and Table 4 on all 3 datasets. Table 3 is obtained by training KGAT model weights using embeddings initialized by pretrained embeddings using matrix factorization. Here, 2-hop performs better than baseline on `Amazon-Book` dataset for both recall@20 and ndcg@20. On `Last-FM` dataset, 2-hop outperforms baseline for ndcg@20. Our custom neighborhood definitions are not able to beat the baseline model for `Yelp2018` dataset.

We believe the improvement in ndcg scores are due to the fact that 2-hop neighbor definition pools similar users together, thus increasing similarity across common items. This corresponds to a decrease in prediction quality for user-specific interactions, therefore preventing an improvement for recall scores.

These improvements are relatively small because we are fine-tuning on an already high-performing set of embeddings provided by authors of KGAT. We hypothesize that since the existing embeddings are trained with default neighbors, they could conflict with our new neighborhood definition. Training from ground up should be a better representation of performance of our new neighborhood definitions.

	Amazon Book		**Last-FM**		**Yelp2018**	
	recall	ndcg	recall	ndcg	recall	ndcg
SVD[1]	0.0918	0.0686	0.0538	0.0942	0.0533	0.0622
Vanilla KGAT	0.1489	0.1006	**0.087**	0.1325	**0.0712**	**0.0867**
Clustering	0.1328	0.0913	0.0851	0.1312	0.0645	0.0806
Closet K	0.0381	0.0315	0.0154	0.046	0.0144	0.0252
2-hop	**0.1495**	**0.1011**	0.086	**0.1327**	0.0684	0.0839

Table 3: Comparison of recall@20 and ndcg@20 for various neighborhood definition functions *with* pretrained embeddings

5.3 Performance without pretrained embeddings

Table 4 shows the result when we initialize our models using random embeddings and train the models from scratch. Although we were unable to reproduce performance claimed for KGAT in their original paper (15), values presented here are obtained using the same setup (learning rate, dropout ratio, batch size, etc.) for fair comparison. From this comparison, we see that again 2-hop neighbor definition achieves superior performance in ndcg@20 against all other KGAT-based implementations. This verifies our previous analysis that pooling 2-hop neighbors provides more emphasis on items that are already well-aligned with user preference. Moreover, a larger gap is observed for `Last-FM` dataset, because `Last-FM` has the highest clustering coefficient. Given its less-sparsed nature, modifying neighbors of user nodes likely increased similarity of neighboring nodes.

	Amazon Book		**Last-FM**		**Yelp2018**	
	recall	ndcg	recall	ndcg	recall	ndcg
Vanilla KGAT	**0.1462**	0.0684	0.0296	0.0602	**0.0542**	0.0706
Clustering	0.0693	0.0250	0.0299	0.0596	0.0305	0.0416
2-hop	0.107	**0.0710**	**0.0314**	**0.0631**	0.0154	**0.07883**

Table 4: Comparison of recall@20 and ndcg@20 for various neighborhood definition functions *without* pretrained embeddings

5.4 Further analysis of ndcg performance for 2-hop neighborhood definition

It is worth exploring the specific property of ndcg because we observed increase in performance for ndcg for multiple scenarios. As explained in Section 4.2, ndcg imposes a larger gain for a correctly predicted item at a higher rank, while recall treats all items in our prediction with equal weight. The fact that our model only consistently outperform vanilla KGAT in ndcg indicates our predictions on higher ranked items are more accurate, while having less than ideal performance on lower ranking items. This is verified in Table 5 for `Last-FM` dataset. We clearly see that our new 2-hop neighborhood definition has poorer performance for higher k metrics. The difference at top $k = 100$ items is in fact quite significant, far greater in values than the improvement we observe for $k = 20$. These behaviors can be explain as pooling 2-hop neighbors increases similarity across users, thus prediction on popular items are more accurate than those specific to individual users. From this analysis, we see that pooling 2-hop neighbors is better suited for identifying top ranking items and improving their alignment with user preference. Therefore, our 2-hop neighbor pooling method should be used in conjunction with vanilla methods to ensure across the board performance gain.

[1]SVD is not based on trained embeddings. SVD's model performance is shown in both Table 3 and Table 4 just as a baseline reference.

	recall@k		**hit@k**		**ndcg@k**	
	k=20	k=100	k=20	k=100	k=20	k=100
Vanilla KGAT	0.0296	**0.0892**	0.1907	**0.4264**	0.0602	**0.1265**
2-hop	**0.0314**	0.0772	**0.1955**	0.4137	**0.0631**	0.1207

Table 5: Comparison of recall, hit and ndcg for 2-hop neighborhood definition *without* pretrained embeddings on `Last-FM` dataset

5.5 Time cost for performance

Since an additional neighborhood definition step has been added as explained in Section 4.1, it is worth analyzing performance impact of the additional step of re-computing neighbors of a graph and the associated change in computation time. For our implementation, we have amortized the neighborhood search time into each training phase. In Table 6, we present step time for each training phase for all 4 methods using the same hardware (Google Cloud Compute Instance, 16 CPUs, 54GB RAM, Nvidia K80 GPU) for `Yelp2018` dataset with batch size of 1024. The observed step time is consistent with expectation because clustering and computing node-similarities are expensive operations, as hundreds of thousands of pairwise embedding comparisions are needed. In contrast, hop-skipping only requires computing graph confusion matrix, for which efficient compressed sparse row matrix multiplication algorithms exist. The maximum amount of increase in step time is from 0.681s to 0.699s, representing a merely 2% change. Although our current neighborhood computations only sees limited improvement for some metrics, we are able to verify that such neighborhood computation is much cheaper than the main task of computing graph convolutions. Researchers should continue exploring other neighborhood definitions without having too much concern on overall runtime.

	Baseline	**Clustering**	**Closest-K**	**2-hop**
Phase 1 (TransR)	0.544	0.564	0.542	0.550
Phase 2 (Attention)	0.137	0.135	0.120	0.135
Total Step Time	0.681	0.699	0.662	0.685

Table 6: Comparison of average step time for various neighborhood definitions

5.6 Analysis of trained embeddings

Changing neighborhood definition has an impact on trained embeddings. Initially, we plotted pretrained user and item embeddings from `Yelp2018` dataset in Figure 1 for a basic understanding of what user-item embeddings look like. As mentioned before, these embeddings overlap, and seems to only differ by a vector represent the embedding of "interacted" relation between users and items. In Figure 4 we compare generated user and item embeddings for our custom neighborhood definitions. From these figures, we see that embeddings from vanilla KGAT, clustering and 2-hop neighbors look similar to those in Figure 1 because there are clear boundaries separating groups that likely represent particular types of user preferences. In fact, the number of such small groups are similar for these graphs as well, likely indicating that the same groups are identified. In contrast, embeddings generated from closest-K neighbors form a contiguous large cluster. Since these embeddings do not form groups in the latent space, poor performance is expected, corresponding to what we presented in Table 3, in which Closest-K neighbors lead to the worst performance. This result suggests that closest-K neighbors further blurred the lines across users, overdoing what we planned to achieve by introducing alternative neighborhood definitions.

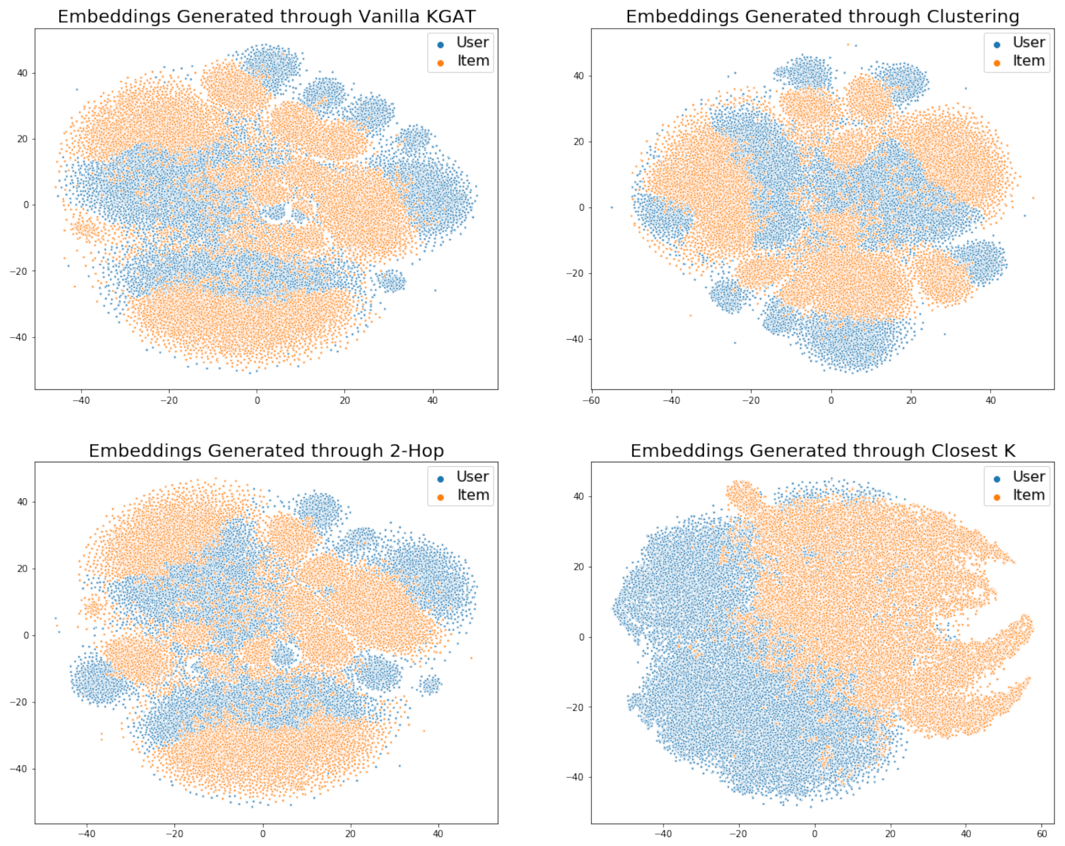


Figure 4: Comparison of user-item embeddings for `Yelp2018` dataset for various neighbor definitions

6 Conclusion and Additional Thoughts

For this project, we analyzed three alternative definitions of node neighborhood. These alternative definitions are achieved through clustering, node similarity, and pooling neighbors from two hops. Performance of these methods are analyzed using pretrained embeddings provided by author of vanilla KGAT implementation and without pretrained embeddings. Small improvement in the range of 0.001 to 0.007 are achieved for ndcg@20 and recall@20 using `Amazon Books` and `Last-FM` dataset with pretrained embeddings. Larger improvements in the range of 0.003-0.008 are achieved for all datasets without pretrained embeddings. Since the most improvement is observed for ndcg@20, we performed further analysis to show that our custom neighborhood definition can successfully identify items with high similarities and further improve their alignment with users.

Moreover, we have also implemented a general purpose graph neural network structure by extending functionalities provided by `Pytorch Geometric`. We believe this structure is more organized and modular, allowing future researchers to easily swap out various components needed to train a graph neural network.

A significant limitation faced for this project is that reference datasets are very large in size. A detrimental effect is that parameter tuning, cannot be conducted efficiently given lengthy experiments. Running `nvidia-smi` reveals that although GPU is running close to 100%, its VRAM utility is far lower than that of CNNs. It could be that dependencies specified by the graph acts much like dependencies of recurrent neural network, except that GNNs naturally have a lot more dependencies than RNNs. Our custom `Pytorch` implementation was an attempt to alleviate this problem, but is limited by the lack of efficient `Python` multiprocessing infrastructure, preventing efficient drawing of subgraphs from a large shared knowledge/user-item hybrid graph. We hope more research will be conducted in the field of improving efficiency of graph neural networks so expensive tasks like parameter tuning can be conducted without prohibiting overhead.

7 Contributions

Jingbo: Implementation of `Pytorch` framework, running various experiments, report writing (motivation and results)

Ruge: Implementation of neighborhood definition in modified KGAT framework and propagation methods in `Pytorch` framework, running various experiments, report writing (methods and theoretical background)

Meixian: Dataset analysis (clustering coefficients, embedding analysis (t-SNE plots), statistics, etc.), report and poster creation (graphs, charts, organizations)

References

[1] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv:1901.00596*, 2019.

[2] E. Frolov and I. Oseledets, "Fifty shades of ratings: How to benefit from a negative feedback in top-n recommendations tasks," in *Proceedings of the 10th ACM Conference on Recommender Systems*, pp. 91–98, ACM, 2016.

[3] B. Lika, K. Kolomvatsos, and S. Hadjiefthymiades, "Facing the cold start problem in recommender systems," *Expert Systems with Applications*, vol. 41, no. 4, pp. 2065–2073, 2014.

[4] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge & Data Engineering*, no. 6, pp. 734–749, 2005.

[5] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th international conference on world wide web*, pp. 173–182, International World Wide Web Conferences Steering Committee, 2017.

[6] X. He, Z. He, J. Song, Z. Liu, Y.-G. Jiang, and T.-S. Chua, "Nais: Neural attentive item similarity model for recommendation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 12, pp. 2354–2366, 2018.

[7] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural graph collaborative filtering," *arXiv preprint arXiv:1905.08108*, 2019.

[8] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Advances in neural information processing systems*, pp. 2787–2795, 2013.

[9] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *Twenty-Eighth AAAI conference on artificial intelligence*, 2014.

[10] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," in *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[11] G. Ji, S. He, L. Xu, K. Liu, and J. Zhao, "Knowledge graph embedding via dynamic mapping matrix," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 687–696, 2015.

[12] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1247–1250, AcM, 2008.

[13] H. Wang, F. Zhang, X. Xie, and M. Guo, "Dkn: Deep knowledge-aware network for news recommendation," in *Proceedings of the 2018 World Wide Web Conference*, pp. 1835–1844, International World Wide Web Conferences Steering Committee, 2018.

[14] H. Zhao, Q. Yao, J. Li, Y. Song, and D. L. Lee, "Meta-graph based recommendation fusion over heterogeneous information networks," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 635–644, ACM, 2017.

[15] X. Wang, X. He, Y. Cao, M. Liu, and T. Chua, "KGAT: knowledge graph attention network for recommendation," in *KDD*, pp. 950–958, 2019.

[16] H. Wang, F. Zhang, J. Wang, M. Zhao, W. Li, X. Xie, and M. Guo, "Ripplenet: Propagating user preferences on the knowledge graph for recommender systems," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pp. 417–426, ACM, 2018.

[17] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1247–1250, AcM, 2008.

[18] B. K. Iwana, S. T. R. Rizvi, S. Ahmed, A. Dengel, and S. Uchida, “Judging a book by its cover,” *arXiv preprint arXiv:1610.09204*, 2016.

[19] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel, “Image-based recommendations on styles and substitutes,” in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 43–52, ACM, 2015.

[20] O. Celma, *Music Recommendation and Discovery: The Long Tail, Long Fail, and Long Play in the Digital Music Space*. Springer, 2010.

[21] Yelp.com, “Yelp dataset challenge.” `https://www.yelp.com/dataset/challenge`.

[22] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, “A density-based algorithm for discovering clusters in large spatial databases with noise.,” in *Kdd*, vol. 96, pp. 226–231, 1996.

[23] D. Arthur and S. Vassilvitskii, “k-means++: The advantages of careful seeding,” in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027–1035, Society for Industrial and Applied Mathematics, 2007.

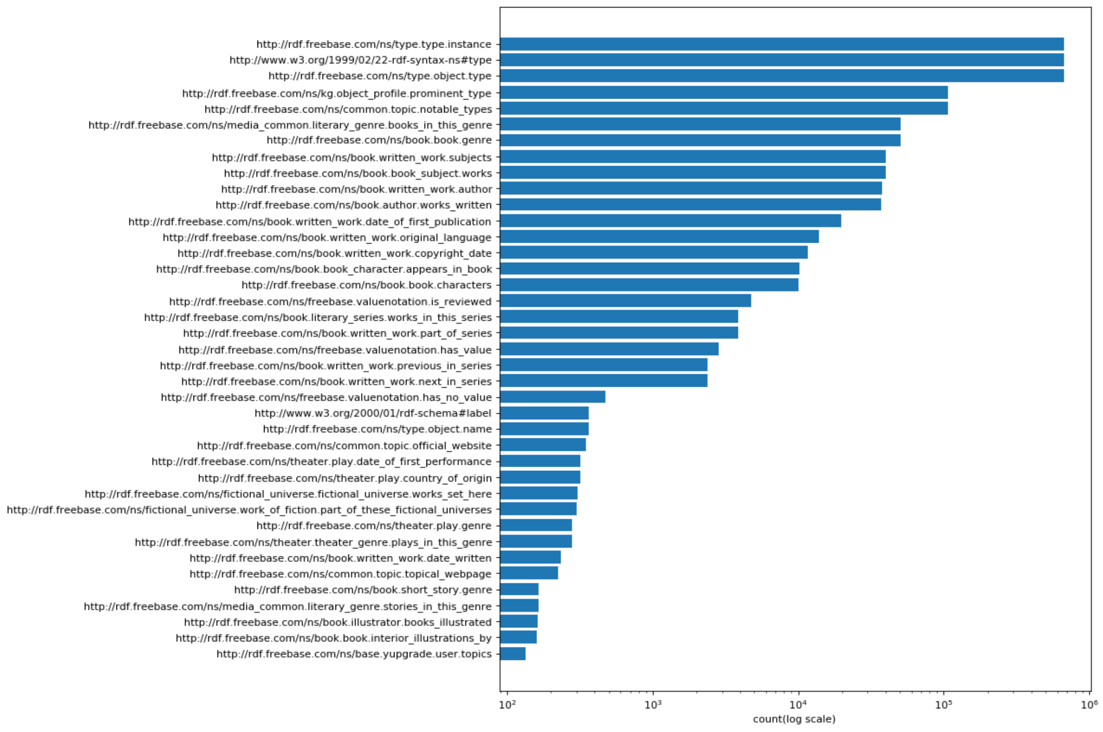
[24] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” *arXiv preprint arXiv:1710.10903*, 2017.

[25] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.

[26] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

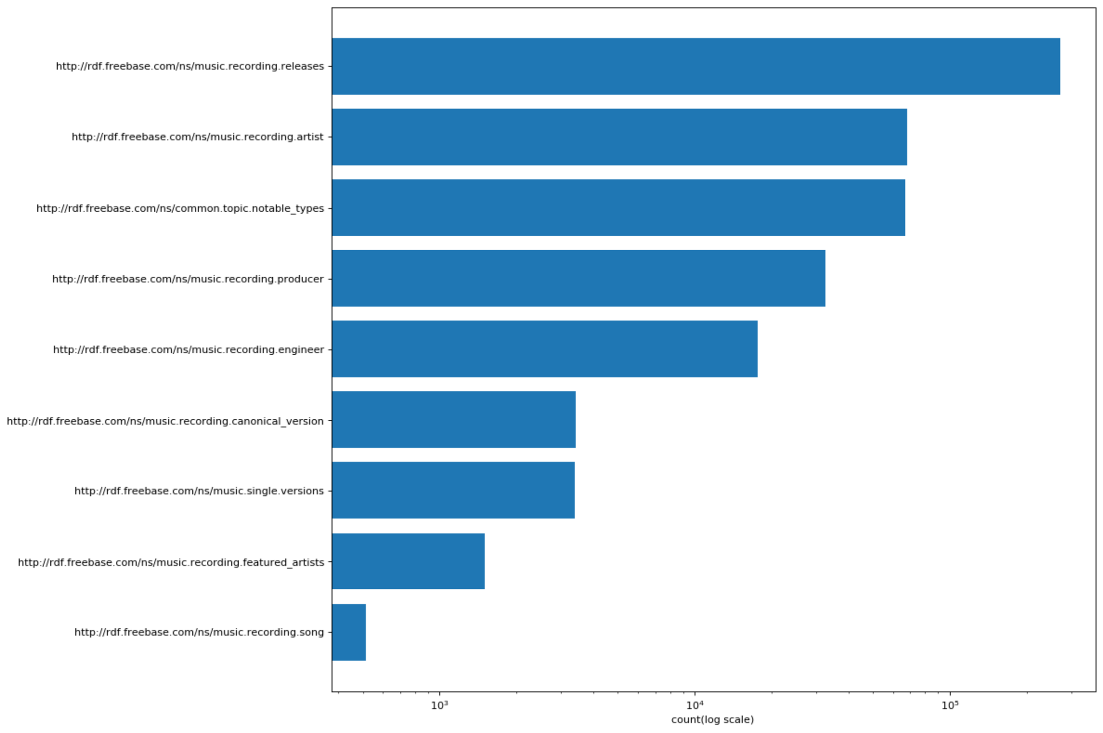
[27] D. Billsus and M. J. Pazzani, “Learning collaborative information filters.,” in *Icml*, vol. 98, pp. 46–54, 1998.

Appendix A Appendix

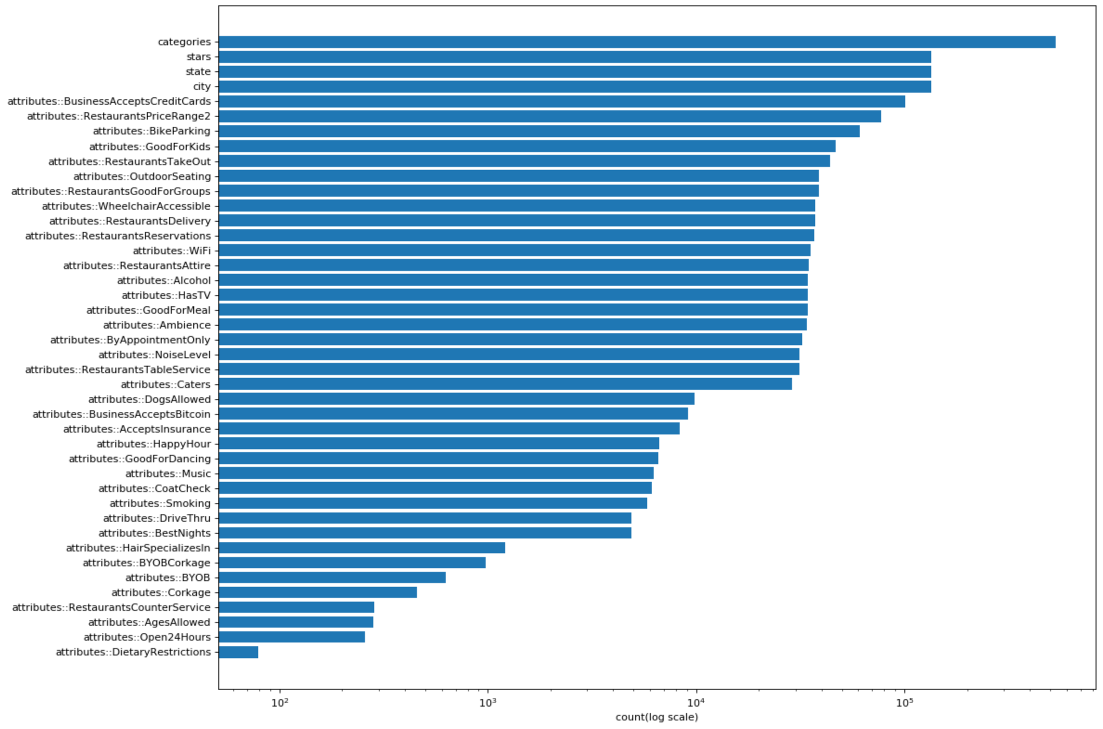


(a) Amazon Book

Figure 5: Distribution of relation types in knowledge graph.



(b) Last-FM



(c) Yelp2018

Figure 5: Distribution of relation types in knowledge graph (Continued).