An End-to-End Neighborhood-based Interaction Model for Knowledge-enhanced Recommendation

Yanru Qu[1*], Ting Bai[2,3*], Weinan Zhang[1], Jianyun Nie[4], Jian Tang[5,6,7]
[1]Shanghai Jiao Tong University, [2]Beijing University of Posts and Telecommunications, [3]Renmin University of China
[4]Université de Montréal, [5]Mila-Quebec Institute for Learning Algorithms, [6]HEC Montréal, [7]CIFAR AI Research Chair
kevinqu@apex.sjtu.edu,baiting0317@gmail.com,wnzhang@sjtu.edu.cn
nie@iro.umontreal.ca,jian.tang@hec.ca

ABSTRACT

This paper studies graph-based recommendation, where an interaction graph is constructed built from historical records and is leveraged to alleviate data sparsity and cold start problems. We reveal an early summarization problem in existing graph-based models, and propose Neighborhood Interaction (NI) model to capture each neighbor pair (between user-side and item-side) distinctively. NI model is more expressive and can capture more complicated structural patterns behind user-item interactions. To further enrich node connectivity and utilize high-order structural information, we incorporate extra knowledge graphs (KGs) and adopt graph neural networks (GNNs) in NI, called Knowledge-enhanced Neighborhood Interaction (KNI). Compared with the state-of-the-art recommendation methods, *e.g.*, feature-based, meta path-based, and KG-based models, our KNI achieves superior performance in click-through rate prediction (1.1%-8.4% absolute AUC improvements) and outperforms by a wide margin in top-N recommendation on 4 real world datasets.

CCS CONCEPTS

• **Information systems → Information retrieval**; • **Computing methodologies → Neural networks**;

KEYWORDS

Knowledge Graph, Knowledge-enhanced Recommendation, Neighborhood-based Interaction

ACM Reference Format:
Yanru Qu[1*], Ting Bai[2,3*], Weinan Zhang[1], Jianyun Nie[4], Jian Tang[5,6,7]. 2019. An End-to-End Neighborhood-based Interaction Model for Knowledge-enhanced Recommendation. In *1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data (DLP-KDD'19), August 5, 2019, Anchorage, AK, USA*. 9 pages. https://doi.org/10.1145/3326937.3341257

1 INTRODUCTION

Recommender systems have become increasingly important in various online services for helping users find the information they want. However, existing recommender systems are challenged by the problems of data sparsity and cold start, *i.e.*, most items receive only a few feedbacks (*e.g.*, ratings and clicks) or no feedbacks at all (*e.g.*, for new items). To tackle these problems, the existing approaches usually utilize side information to learn better user/item representations [9, 14, 16], which then facilitate the learning of user-item interactions, and finally promote the recommendation quality. In many scenarios, knowledge graphs (KGs) can be used to provide general background knowledge as well as rich structural information [11, 26, 30].

Graph-based recommender systems build interaction graphs from historical feedbacks and side information, where the nodes can be users, items, or side information (*e.g.*, tag, genre), and two nodes are linked together based on relevance or co-occurrence. Recently, graph-based models are becoming more powerful with advanced graph neural networks. For example, graph convolution networks [23, 28] can integrate high-order neighborhood information in an end-to-end way, graph attention networks [26] can simulate user preferences on knowledge graphs. Graph-based models are more expressive than traditional feature-based models, because they take the local structures (of the users, items, and relevant nodes) into consideration.

However, due to an "early summarization" issue, the existing graph-based models cannot fully utilize the local structures: these models usually compress user- and item-neighborhoods into single user/item embeddings before prediction. In this case, only two nodes and one edge are activated, yet other nodes and their connections are mixed and relayed. We consider the meticulous local structures are valuable, and a good system should be able to capture useful patterns, and filter out other noise. Here is an example. A system is recommending a film to a user, where the user has rated 5 stars for "La La Land" (Land) and "Interstellar" (Inter), and the film has 2 tags, "romance" and "fiction". We know that "Land" is a romance film, and "Inter" is a science fiction film. Thus the connections between ("Land", "romance") and ("Inter", "fiction") could be helpful for recommendation, meanwhile, the connection between ("Land", "fiction") is nonsense and not expected, which is regarded as noise. We argue that the local structures are hidden and not fully utilized in previous graph-based methods.

To address the early summarization problem, we extend user-item interactions to their neighbors, and propose a unified Neighborhood Interaction (NI) model. More specifically, we propose a bi-attention network to make prediction on the local structures directly, instead of compressing them into user/item embeddings. We also utilize graph neural networks (GNNs) to integrate high-order neighborhood information, and introduce knowledge graphs to

[*] Equal contribution. This work was done when the first and the second authors were visiting Mila and Université de Montréal.

DLP-KDD'19, August 5, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6783-7/19/08.
https://doi.org/10.1145/3326937.3341257

increase the local connectivity. The final model, called Knowledge-enhanced Neighborhood Interaction (KNI), is evaluated on 4 real-world datasets and compared with 8 feature-based, meta path-based, and graph-based models. Our experiments show that KNI outperforms state-of-the-art models, including Wide&Deep, MCRec, PinSage and RippleNet, by 1.1%-8.4% of AUC in click-through rate prediction, and exceeds baseline models by a wide margin in top-N recommendation. We also provide a case study and statistical analysis to demonstrate our model.

The rest of this paper is organized as follows: we first define the problem and introduce our KNI model in Section 2. And then we demonstrate the experiments and discuss the results in Section 3. Related works are summarized in Section 4. Finally, Section 5 concludes this paper.

2 KNOWLEDGE-ENHANCED NEIGHBORHOOD INTERACTION

In this section, we introduce graph-based recommendation and the early summarization issue at first, and define Neighborhood Interaction (NI). Then, we extend NI with graph neural networks (GNNs) and knowledge graphs (KGs). Finally, we present the overall framework of Knowledge-enhanced Neighborhood Interaction (KNI) model, shown in Fig. 1.

2.1 Neighborhood Interactions

2.1.1 User-item Interaction Graph. The user-item interaction history can be represented by an interaction matrix, $Y \in \mathbb{R}^{|U|\times|V|}$, where $U = \{u_1, u_2, \ldots, u_n\}$ is the set of users, $V = \{v_1, v_2, \ldots, v_m\}$ the set of items. An element $y_{u,v}$ indicates the feedback of user u on item v. In this paper, we assume that $y_{u,v}$ takes a binary value (which could be easily extended to other values). Regarding the positive responses in Y as edges, we build the interaction graph for users and items, $\mathcal{G}_{rec} = \{(u, c, v)|u \in U, v \in V, c = 1\}$. In $\mathcal{G}_{rec}$, users' neighbors are items, and items' neighbors are users.

2.1.2 Graph-based Recommendation. Most existing graph-based recommender systems utilize the graph structures via summarizing the neighborhood information into user/item representations. And then these models learn user-item interactions from the representations, which is usually formulated as inner product. Denote N_u and N_v as user and item neighborhoods, $\mathbf{u}$ and $\mathbf{v}$ as their representations.

$$\mathbf{u} = \text{agg}(N_u) \tag{1}$$

$$\mathbf{v} = \text{agg}(N_v) \tag{2}$$

$$\hat{y}_{u,v} = \sigma(\langle \mathbf{u}, \mathbf{v} \rangle) \tag{3}$$

where agg() is an aggregation function which maps a set of neighbor nodes into a single embedding vector, $\sigma()$ is the sigmoid function. For simplicity, we omit σ in the following.

The main difference of graph-based models is how to learn user/item representations from the graph structures, in another word, designing aggregation functions. The most popular and general agg() include averaging, attention [26, 27, 30], etc.

$$\text{Average: } \mathbf{u} = \frac{1}{|N_u|} \sum_{i \in N_u} \mathbf{x}_i \tag{4}$$

$$\text{Attention: } \alpha_{u,i} = \text{softmax}_i(\mathbf{w}^\top [\mathbf{x}_u, \mathbf{x}_i] + b) \tag{5}$$

$$\mathbf{u} = \sum_{i \in N_u} \alpha_{u,i} \mathbf{x}_i \tag{6}$$

$\mathbf{x}_u$ is the embedding vector of user u, $\mathbf{x}_v$ is of item v, $\mathbf{x}_i$ is of node i, and $\alpha_{u,i}$ is the user-side attention score produced by an attention network. $\mathbf{w}$ and b[1] are attention network parameters. In this paper, we mainly employ the attention network structure in Eq. (5), where [,] means concatenation.

Most previous methods summarize user/item neighborhood information before learning their interactions, which compresses the local structures into only two nodes and one edge, yet other nodes and their connections are mixed and relayed. The behavior may restrict model from exploring the local structures and obstruct to distinguish useful patterns from noise. We call it the early summarization issue.

2.1.3 Neighbor-Neighbor Interaction. After expanding $\hat{y}_{u,v}$ (before taking σ),

$$\text{Average: } \hat{y}_{u,v} = \langle \frac{1}{|N_u|} \sum_{i \in N_u} \mathbf{x}_i, \frac{1}{|N_v|} \sum_{j \in N_v} \mathbf{x}_j \rangle \tag{7}$$

$$= \sum_{i \in N_u} \sum_{j \in N_v} \frac{1}{|N_u||N_v|} \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{8}$$

$$\text{Attention: } \hat{y}_{u,v} = \langle \sum_{i \in N_u} \alpha_{u,i} \mathbf{x}_i, \sum_{j \in N_v} \alpha_{v,j} \mathbf{x}_j \rangle \tag{9}$$

$$= \sum_{i \in N_u} \sum_{j \in N_v} \alpha_{u,i} \alpha_{v,j} \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{10}$$

we find there exists a general form of Eq. (8) and (10)

$$\hat{y} = \mathbf{A} \odot \mathbf{Z} \tag{11}$$

$$\text{s.t.} \sum_{i,j} \mathbf{A}_{i,j} = 1 \, , \mathbf{Z}_{i,j} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{12}$$

where $\mathbf{A} \in \mathbb{R}^{|N_u|\times|N_v|}$ is a weight matrix summing up to 1, $\mathbf{Z} \in \mathbb{R}^{|N_u|\times|N_v|}$ is a matrix of inner product terms, and $\odot$ denotes the sum of element-wise product. According to this form, graph-based models generally learn two things: (i) exploring the general interactions of all node pairs (as $\mathbf{Z}$ does), (ii) assigning proper weights for different interactions (as $\mathbf{A}$ does). We consider $\mathbf{Z}$ contains more global information because a node usually gets trained in different neighborhoods, yet $\mathbf{A}$ can better describe local structures, whose values indicate the confidence of certain connections being helpful. Therefore, this form, Eq. (11), provides a tool to convert a graph structure to a prediction.

[1] In general, the bias term is canceled because of softmax. In [24], the linear projection is followed by a LeakyReLU layer, therefore, the bias term is kept.

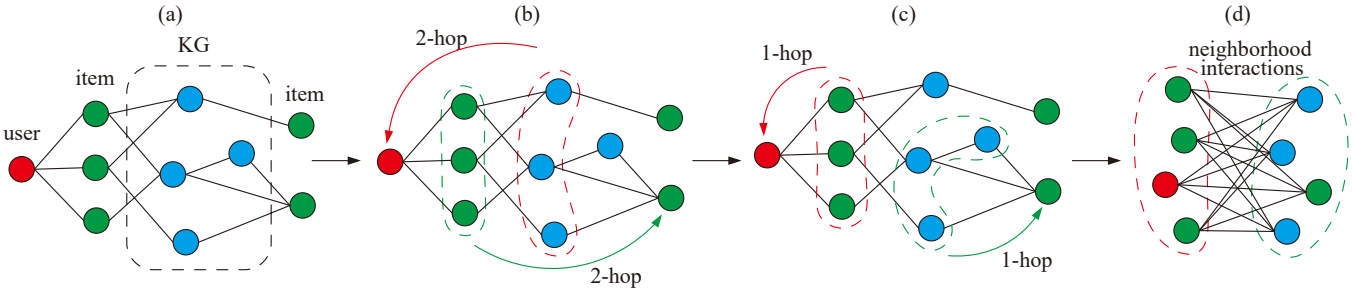

Figure 1: Model overview. *Note*: Red circles denote users. Green circles denote rated or unseen items. Blue circles denote non-item entities. Dash circles denote user and item neighborhoods. In this example, a KIG is constructed at first, and then higher hop neighborhood information is aggregated into local neighbors. Finally, the user and item neighborhoods are collected to compute neighborhood interactions.

We propose a bi-attention network to better utilize the neighborhood information, namely Neighborhood Interaction (NI).

$$\alpha_{i,j} = \text{softmax}_{i,j}(\mathbf{w}^\top[\mathbf{x}_u, \mathbf{x}_i, \mathbf{x}_v, \mathbf{x}_j] + b) \tag{13}$$

$$\hat{y}_{u,v} = \sum_{i \in N_u} \sum_{j \in N_v} \alpha_{i,j} \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{14}$$

Different from Eq. (5), Eq. (13) takes both user- and item-side neighbors into consideration. In Eq. (14), different interaction terms are weighted distinctively. From the discussion above, NI can better utilize the local structures for recommendation, therefore, NI is supposed to address the early summarization problem of graph-based models.

It is worth noting that the average form (Eq. (8)) and attention form (Eq. (10)) are special cases of NI. The average form sets the weight matrix as a constant $\mathbf{A} = 1/|N_u||N_v|$, asnd the attention form approximates the weight matrix via 1st rank matrix decomposition $\mathbf{A} \sim \alpha_u \alpha_v^\top$, where α_u and α_v are column vectors of the user-/item-side attention scores (Eq.(5)).

Besides, we include the user and the item in their neighborhoods, *i.e.*, $u \in N_u$, $v \in N_v$, thus the interactions between user and item (u, v), user and item neighbor (u, j), user neighbor and item (i, v), user neighbor and item neighbor (i, j) are all considered for prediction. The NI model is illustrated in Fig. 1 (d), where the edges represent interactions among two neighborhoods.

2.2 Integrating High-order Neighborhood Information

In last section, the user neighbors are ever-rated items, and the item neighbors are historical audience. The interaction graph $\mathcal{G}_{rec}$ also contains high-order neighborhood information, for example, a film is a 2-hop neighbor of another film if they share the same tag. Introducing high-order neighborhood information has shown effective [23, 25, 28] in graph-based recommendation, thus we introduce graph convolution network (GCN) [15] and graph attention network (GAT) [24] to encode high-order neighborhood information for NI model.

Graph convolution network computes high-order node representations by stacking several graph convolution layers. Each graph convolution layer computes a node representation according to its nearest neighbors and itself (equivalent to a self loop in the graph). For a node u, a 2-layer GCN computes

$$\mathbf{x}_i^1 = \sigma\left(\frac{1}{|N_i|} \sum_{j \in N_i} \mathbf{w}^1 \mathbf{x}_j + \mathbf{b}^1\right) \tag{15}$$

$$\mathbf{x}_u^2 = \sigma\left(\frac{1}{|N_u|} \sum_{i \in N_u} \mathbf{w}^2 \mathbf{x}_i^1 + \mathbf{b}^2\right) \tag{16}$$

where $\mathbf{x}_j$ is the feature vector or initial embedding of node j, $\mathbf{x}_i^1$ and $\mathbf{x}_u^2$ are outputs of the 1st and 2nd graph convolution layers, N_i and N_u are neighborhoods of i and u, and $\mathbf{w}$ and $\mathbf{b}$ are parameters to be learned. Successive graph convolution layers are separated by non-linear transformation $\sigma()$, which is usually ReLU.

Graph attention network is similar to GCN except that node embeddings are computed by multi-head self attention networks. For a node u, a 2-layer GAT computes (single head)

$$\mathbf{x}_i^1 = \sigma\left(\sum_{j \in N_i} \alpha_{i,j}^1 \mathbf{w}^1 \mathbf{x}_j + \mathbf{b}^1\right) \tag{17}$$

$$\mathbf{x}_u^2 = \sigma\left(\sum_{i \in N_u} \alpha_{u,i}^2 \mathbf{w}^2 \mathbf{x}_i^1 + \mathbf{b}^2\right) \tag{18}$$

where $\alpha_{i,j}^l$ is the attention score of node j to node i, produced by the l-th layer attention network[2]

$$\alpha_{i,j}^l = \frac{\exp(\text{LeakyReLU}(\mathbf{w}_a^{l\top}[\mathbf{x}_i^{l-1}, \mathbf{x}_j^{l-1}] + b_a^l))}{\sum_{k \in N_i} \exp(\text{LeakyReLU}(\mathbf{w}_a^{l\top}[\mathbf{x}_i^{l-1}, \mathbf{x}_k^{l-1}] + b_a^l))} \tag{19}$$

where $\mathbf{w}_a^l$ and b_a^l are parameters of the attention network, other notations are the same as GCN. Note that the above attention network structure is suggested in [24].

For any target node i, we can generate node embeddings $\mathbf{x}_i^l$ containing high-order neighborhood information with GCN or GAT [15, 24]. And $\{\mathbf{x}_i^l, i \in \mathcal{G}_{rec}\}$ can replace feature vectors or initial embeddings in NI model (Eq. (14)), where graph network serves as an encoder. This process is demonstrated in Fig. 1 (b) and (c). In (b) the 2-hop neighbors are propagated to 1-hop neighbors, and the 1-hop neighbors are concentrated to the central node.

Neighbor Sampling (NS) [7] is a sampling method to facilitate graph network computation on large graphs. The original graph networks, *e.g.*, GCN and GAT, traverse all neighbor nodes to generate

[2]GAT [24] utilizes LeakyReLU transformation before softmax in its original form.

Table 1: Statistics for the expanded datasets. *Note*: "entities" contain both items and non-item entities.

Datasets	C-Book	Movie-1M	A-Book	Movie-20M
# users	17,860	6,036	78,809	59,296
# items	14,967	2,445	32,389	11,895
# interactions	139,746	753,772	1,181,684	9,104,038
# entities	77,881	182,011	265,478	64,067
# relations	10	12	22	38
# triples	71,628	923,718	1,551,554	1,195,391

a node embedding, which is time consuming and not tractable for a very large graph. NS proposes to sample a fixed number (*e.g.*, K) of neighbors for each node in forward computation. Combining GCN and NS as an example

$$\tilde{\mathbf{x}}_i^1 = \sigma(\frac{1}{K}\sum_{j\in\tilde{N}_i}\mathbf{w}^1\mathbf{x}_j + \mathbf{b}^1) \tag{20}$$

where $\tilde{N}_i$ is drawn randomly from N_i, containing exactly K elements. NS controls the number of high-order neighbors directly, thus restrains model's complexity. There are other sampling methods, including random walk-based sampling [28], importance sampling [2], etc. In this work, we mainly adopt NS.

2.3 Integrating Knowledge Graphs

A knowledge graph consists of a large number of entity-relation-entity triples $\mathcal{G}_{kg} = \{(h, r, t)|h, t \in E, r \in R\}$, where E is the entity set, R is the relation set. Using the item set V as initial queries, we can map items to corresponding entities in knowledge graph. Using the newly added entities as queries, we repeat the expansion several times and obtain knowledge-enhanced interaction graph (KIG), $\mathcal{G} = \mathcal{G}_{rec} \cup \mathcal{G}_{kg}$. The resulting KIG is shown in Fig. 1 (a). In KIG, the users' and items' neighbors are extended to non-item entities, *e.g.*, a movie star. We can recklessly replace $\mathcal{G}_{rec}$ with $\mathcal{G}$ without modifying NI.

2.4 Model Overview

The training objective is log loss

$$\mathcal{L}(Y, \hat{Y}) = -\sum_{y_{u,v}=1}\log(\hat{y}_{u,v}) - \sum_{y_{u,v}=0}\log(1-\hat{y}_{u,v}) + \lambda\|\theta\|_2^2 \tag{21}$$

where $\lambda\|\theta\|_2^2$ is the L2 regularization term to control overfitting.

We then revisit the whole framework of KNI as shown in Fig. 1. (a): We first build knowledge-enhanced interaction graph (KIG) with user feedbacks and knowledge graphs. (b) and (c): We then apply graph neural networks (GNNs) to propagate high-order neighborhood information to user/item neighbors. (d): The user and item neighborhoods are collected and fed to Neighborhood Interactions (NI). The framework is trained end-to-end with the loss term presented above.

3 EXPERIMENTS

3.1 Datasets

We combine 4 recommendation datasets with 2 public knowledge graphs in our experiments. The datasets and experiment code are publicly available[3] for reproducibility and further study.

The first two smaller datasets are released by [26].

- **C-Book** combines Book Crossing[4] and Microsoft Satori[5].
- **Movie-1M** combines MovieLens[6]-1M and and Microsoft Satori.

We follow the procedures of [26] to process the other two larger datasets, which are then linked to Freebase [1]. The linkages are studied and provided by KB4Rec [32]. Note that another dataset LFM in KB4Rec is not included in our experiments, because it follows a quite different scheme from the others and does not contain any rating or click information.

- **A-Book** combines Amazon Book[7] and Freebase. Amazon Book [8] contains over 22.5 million ratings (ranging from 1 to 5) collected from 8 million users and 2.3 million items.
- **Movie-20M** combines MovieLens-20M and Freebase. MovieLens-20M contains ratings (ranging from 1 to 5) collected from the MovieLens website.

A-Book and Movie-20M are processed as follows. Since A-Book and Movie-20M are originally in rating format, we convert ratings to binary feedbacks: 4 and 5 stars are converted to positive feedbacks (denoted by "1") and the other ratings to negative feedbacks. For each user, we sample the same amount of negative samples (denoted by "0") as their positive samples from unseen items. We also drop low-frequency users and items. The threshold is 5 for A-Book and 20 for Movie-20M.

After the datasets are processed, we split each dataset into training/validation/test sets at 6:2:2. Then we map the items of training set to corresponding entities in Freebase, with the help of KB4Rec [32]. For each dataset, we use the linked items as initial queries to find related non-item entities. These entities are added to KIG and used for further expansion. We repeat this process 4 times to ensure sufficient knowledge is included in the final dataset. We also remove entities appearing less than 5 times on A-Book (the threshold is 20 for Movie-20M), and relations appearing less than 5000 times (same for Movie-20M). The basic statistics of the 4 datasets are presented in Table 1.

3.2 Compared Models

We compare NI (without knowledge graph) and KNI (with knowledge graph) with 2 feature-based, 2 meta path-based, and 4 graph-based models. For fair comparison, we pre-train TransR models and extract structural features for feature-based models. Note that the main difference between NI models and baseline models is, NI models make prediction from graph structures directly, while others compress the structural information into two node embeddings.

[3] https://github.com/Atomu2014/KNI
[4] http://www2.informatik.uni-freiburg.de/ cziegler/BX/
[5] https://searchengineland.com/library/bing/bing-satori
[6] https://grouplens.org/datasets/movielens/
[7] http://jmcauley.ucsd.edu/data/amazon/

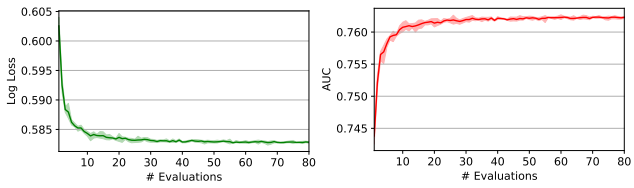

Figure 2: Evaluation stabilizes after sufficient evaluations.

From another perspective, NI models further take the interactions between neighbors into consideration.

libFM [21] is a widely used feature-based model, and is well known for modeling feature interactions. In our experiments, we concatenate the user ID, item ID, and the average embedding of related entities learned from TransR [17] as the input to libFM.

Wide&Deep [3] is another feature-based model, which takes advantages of both shallow models and deep models and achieves state-of-the-art recommendation results. We provide the same input as libFM to Wide&Deep.

PER [29] is a meta path-based model, which builds heterogeneous information network (HIN) on side information, and extracts meta path-based features from HIN. In our experiments, we use all item-attribute-item relations as meta paths.

MCRec [10] is a co-attentive model built on HIN. MCRec learns context representations from meta-paths, and is a state-of-the-art recommendation model. Besides, their code is released[8].

CKE [30] proposes a general framework to jointly learn structural/textual/visual embeddings from knowledge graph, texts and images for collaborative recommendation. We adopt the structural embedding and recommendation components of CKE.

DKN [27] is another knowledge graph-based recommendation model. In our experiments, we use pre-trained TransR embeddings as the input for DKN, with code[9].

PinSage [28] uses GCN for web-scale recommendation. In our experiments, we use PinSage as a representative GCN approach and explore different network structures and sampling methods on PinSage.

RippleNet [26] is a state-of-the-art knowledge graph-based recommendation model. RippleNet uses attention networks to simulate user preferences on KG. In our experiments, we use RippleNet as a representative GAT approach, with code[10].

It is worth noting that, Wide&Deep, MCRec, PinSage and RippleNet are recently proposed state-of-the-art models.

3.3 Experiment Setup and Evaluation

We evaluate these models on 2 tasks, click-through rate (CTR) prediction and top-N recommendation. For CTR prediction, we use the metrics Area Under Curve (AUC) and Accuracy (ACC), which are widely used in binary classification problems. For top-N recommendation, we use the best models obtained in CTR prediction to generate top-N items, which are compared with the test set to compute Precision@K, Recall@K, and F1@K. We repeat each experiment 5 times and report the average scores.

General hyper-parameters include learning rate, embedding size, regularization, etc. Graph-based models, including PinSage, RippleNet and our models, are trained with graph network modules. For these models, 2 hyper-parameters are critical, *i.e.*, the hop number and the sampling method.

A larger hop number indicates a larger neighborhood. In the graph construction stage, we expand the items 4 times on Freebase, thus an item needs 4 steps to visit certain neighbors. For graph-based models, we tune the hop number from 1 to 4. Sampling methods are mainly introduced to speed up training on large graphs, and sometimes influence model convergence and performance. We tune neighbor sampling (NS) and random walk-based sampling in experiments.

We then apply grid search on embedding dimension, learning rate, l2 regularization, etc., for all the compared models. The hyper-parameters are chosen according to the AUC scores on validation sets, and the parameter settings are explained in Section 3.5.

We repeat evaluation several times and use the average scores to compute the metrics. We perform an empirical experiments to determine the number of repetitions, shown in Fig. 2. According to the figure, we conclude the prediction becomes stable after sufficient evaluations. In the following experiments, we fix this number to 40.

3.4 Experiment Results

In this section, we present and analyze the evaluation results of CTR prediction (Table 2) and top-N recommendation (Fig. 3, 4, 5, 6). From Table 2 we can observe:

(i) Meta path-based and graph-based models outperform feature-based models. MCRec, PinSage and RippleNet outperform the other baseline models.

(ii) Meta-path design requires much human expertise, and is not end-to-end. Even though MCRec achieves competitive results with RippleNet, it requires more efforts to manually design and pre-process meta-paths. This restricts the application of meta path-based models on large graphs and scenarios with complex schema.

(iii) High-order neighborhood information contains much more noise. We increase the hop numbers of different models from 1 to 4, and find performance usually decreases with 3- or 4-hops. We attribute this problem to the noise brought by the huge amounts of high-order neighbors (Table 3).

(iv) NI shows significant improvements over baseline models. To our surprise, NI outperforms PinSage and RippleNet even without knowledge graphs. This means the local neighborhood structures are more valuable than high-order neighbors. We also observe that high-order neighborhoods increase dramatically.

(v) Integrating knowledge graphs, KNI obtains even better results than NI. Compared with Wide&Deep, MCRerc, PinSage, and RippleNet, KNI achieves 1.1%-8.4% AUC improvements on 4 datasets.

(vi) From the data perspective, book datasets are more sparse (> 99.9%) than movie datasets, according to Table 3. However, KNI achieves better improvements on the 2 book datasets (4%-5% AUC improvements over best baselines) than the 2 movie datasets (1%-2% AUC improvements). This means KNI can better solve data sparsity.

[8]https://github.com/librahu/MCRec
[9]https://github.com/hwwang55/DKN
[10]https://github.com/hwwang55/RippleNet

Table 2: The results of CTR prediction. *Note*: "*" indicates the statistically significant improvements over the best baseline, with p-value smaller than 10^{-6} in two-sided t-test.

Model	C-Book		Movie-1M		A-Book		Movie-20M	
	AUC	ACC	AUC	ACC	AUC	ACC	AUC	ACC
libFM	0.6850	0.6390	0.8920	0.8120	0.8300	0.7597	0.9481	0.8805
Wide&Deep	0.7110	0.6230	0.9030	0.8220	0.8401	0.7684	0.9507	0.8831
PER	0.6230	0.5880	0.7120	0.6670	0.7392	0.6939	0.8161	0.7327
MCRec	0.7250	0.6707	0.9127	0.8331	0.8708	0.7930	0.9558	0.8872
CKE	0.6760	0.6422	0.8974	0.8171	0.8572	0.7839	0.9574	0.8940
DKN	0.6488	0.6333	0.8835	0.8070	0.8455	0.7679	0.9473	0.8787
PinSage	0.7102	0.6477	0.9213	0.8443	0.8634	0.7804	0.9597	0.8960
RippleNet	0.7290	0.6630	0.9210	0.8440	0.8736	0.7975	0.9579	0.8942
NI	0.7468	0.6796	0.9401	0.8679	0.9160	0.8362	0.9693	0.9110
KNI	**0.7723***	**0.7063***	**0.9449***	**0.8721***	**0.9238***	**0.8472***	**0.9704***	**0.9120***

Table 3: Data sparsity statistics and AUC improvements. *Note*: The n-hop columns represent the number of n-hop neighbors. The sparsity is calculated as # missing edges / # node pairs. The improvements are absolute AUC gains of KNI compared with best baselines.

Datasets	1-hop	2-hop	3-hop	Sparsity	Improvement
C-Book	1	58	40	99.97%	4.33%
Movie-1M	14	42,227	35,534	97.45%	2.36%
A-Book	5	17,027	49,419	99.98%	5.02%
Movie-20M	17	40,547	14,966	99.35%	1.07%

Table 4: Training time of RippleNet and KNI.

Models	C-Book	Movie-1M	A-Book	Movie-20M
RippleNet	17.75s	66.85s	120.38s	937.92s
KNI	2.05s	11.58s	21.52s	166.72s

For the top-N recommendation task, we compare KNI with baseline models. From Fig. 3, 4, 5, and 6 we can observe:

(i) The top-N recommendation results are consistent with CTR prediction. Meta path-based and graph-based models perform better than feature-based models. KNI performs the best.

(ii) On the two book datasets, KNI performs much better than baselines when K is small, especially in top-1 recommendation. This indicates that KNI captures user preference very well. On the 2 movie datasets, KNI outperforms state-of-the-art baseline models by a wide margin.

3.5 Parameter Settings

For hop number, we tune RippleNet following [26], and find RippleNet performs best with hop=3 (C-Book), hop=2 (Movie-1M), hop=1 (A-Book and Movie-20M). For PinSage and our models, we find 1-hop is good enough. After an analysis of the datasets, we found that the main reason for this problem is the explosive increase of high-order neighbors. From Table 3 we can see that the average neighborhood size increases dramatically when it goes from 1-hop to 2-hop, especially on the 2 movie datasets. This may be caused by some high degree nodes in the knowledge graph. The noise brought by the high-order neighbors increases training difficulties. Similar results can be found in many other studies. For example, [26] shows that larger hop numbers may decrease model performance. In [23], the author claims that 1 layer GCN performs the best.

As for the sampling method, we tune NS and random walk-based sampling on PinSage. We find that random walk-based sampling does not always produce better results than NS, besides, random walk-based sampling requires more time. Thus we only apply NS on the other models. The number of neighbors to be sampled is tuned from {4, 8, 16, 32, 64, 128} (128 is not applicable on A-Book and Movie-20M due to memory constraints), and we find 4 (C-Book), 32 (Movie-1M), 8 (A-Book), and 32 (Movie-20M) perform slightly better. We also test the training speed of RippleNet and KNI. When fixing the maximum neighbor size to be 32, KNI with NS could be 5.6-8.6 times faster than RippleNet to train one iteration in the same GPU environment, shown in Table 4. This result confirms that the model complexity of KNI (Section 2.4) could be well controlled through sampling and parallelization.

We perform grid search on the embedding dimension, learning rate and l2 regularization for each model, and we find that the embedding dimension 128 is the best of {4, 8, 16, 32, 64, 128} (we do not try higher dimensions considering the memory size), and the learning rate 10^{-3} is generally better than $\{10^{-4}, 2*10^{-4}, 5*10^{-4}, 2*10^{-3}, 5*10^{-3}, 10^{-2}\}$ (different models vary slightly), and we set the L2 regularization differently on different datasets: 10^{-5} (C-Book), 10^{-7} (Movie-1M), 10^{-7} (A-Book), 10^{-8} (Movie-20M). For other hyper-parameters provided by open-source softwares, we tune them carefully in the grid search.

3.6 Case Study

To show the early summarization problem discussed in Section 2.1, as well as to understand how NI model improves other models, we conduct a case study on in this section. We randomly choose 10k users, 6k items, and 250k responses from the MovieLens-20M dataset, and randomly split training/validation/test sets at 6:2:2. We compare attention aggregation model (AAM) (Eq.(10)) and NI (Eq. (14)) solely on the user-item interaction graph. Recall the general form of graph-based recommendation models in Eq. (11), *i.e.*, $\hat{y} = \sigma(\mathbf{A} \odot \mathbf{Z})$. AAM learns the weight matrix $\mathbf{A}$

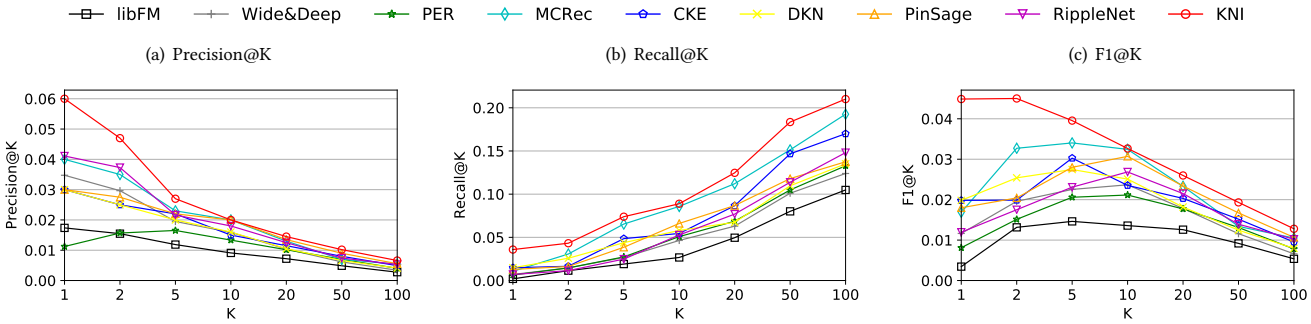

Figure 3: Top-N recommendation results for C-Book.

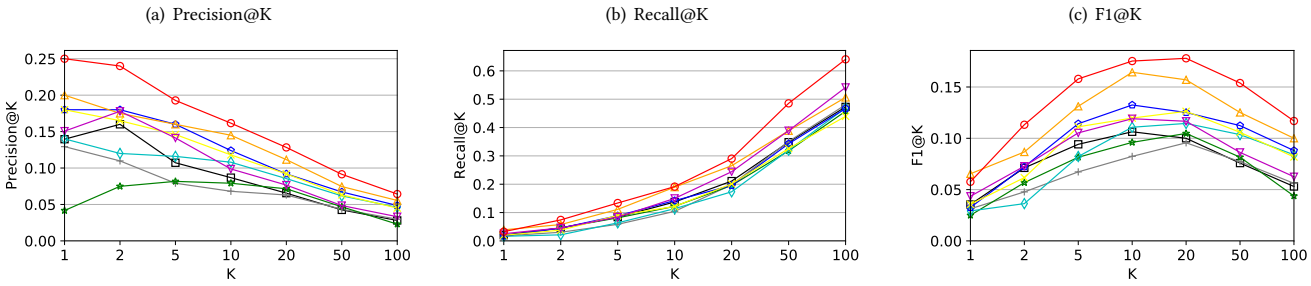

Figure 4: Top-N recommendation results for Movie-1M.

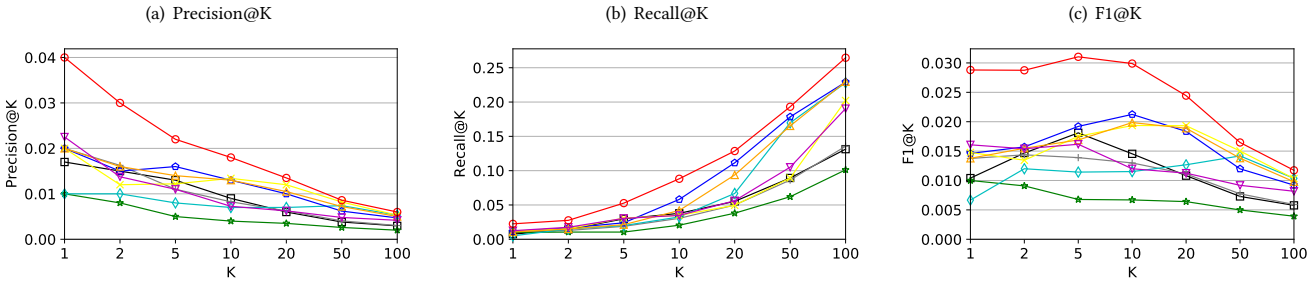

Figure 5: Top-N recommendation results for A-Book.

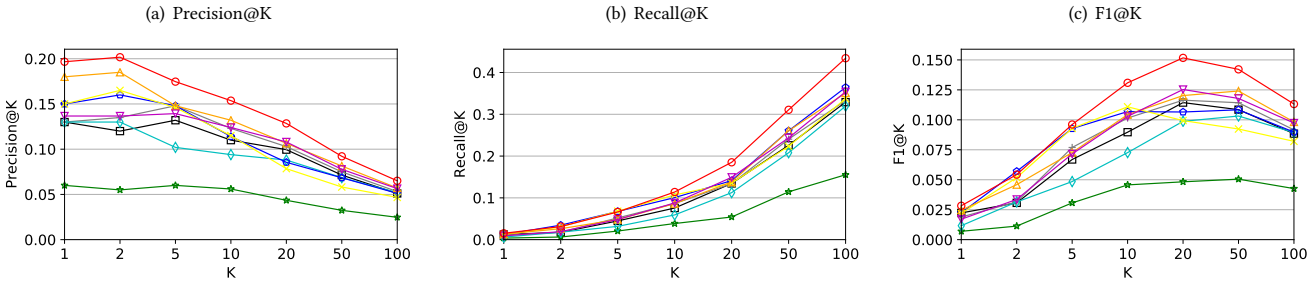

Figure 6: Top-N recommendation results for Movie-20M.

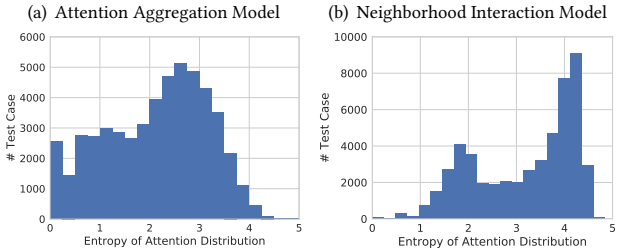

Figure 7: Entropy histogram. *Note:* The x-axis represents the entropy of attention distribution.

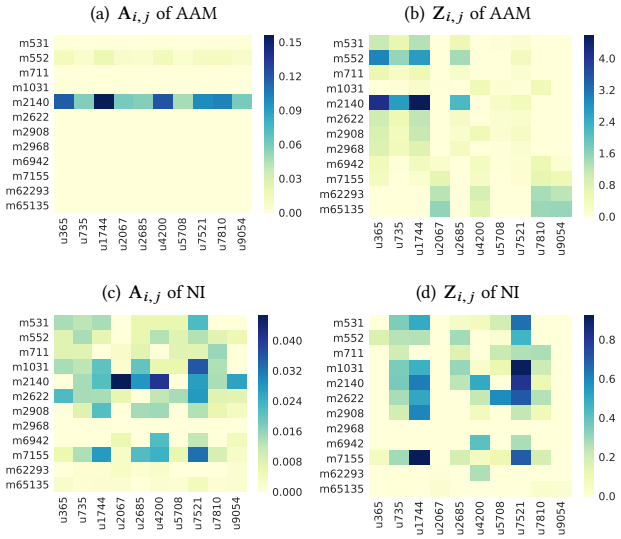

Figure 8: Case study of test case (u46, m3993). *Note*: In (a)-(d), the y-axis represents neighbors of user 46, and the x-axis represents neighbors of item 3993. AAM: attention aggregation model (Eq. (10)).

through user-side and item-side attention networks separately, *i.e.*, $\mathbf{A}_{i,j} = \alpha_{u,i}\alpha_{v,j}$, yet NI learns from both sides, *i.e.*, $\mathbf{A}_{i,j} = \alpha_{i,j}$.

Since the elements in $\mathbf{A}$ sum to 1, a weight matrix can be regarded as a distribution. Thus we can calculate its entropy to quantitatively measure the information it contains. We calculate the entropy of the weight matrix $\mathbf{A}$ of each test sample and plot the histograms of entropy in Fig. 7. The x-axis represents the entropy value, the larger value it has, the more information it contains. We can see that the weight matrices in NI model have higher entropy, *i.e.*, more informative. Besides, the average entropy of GAT is 2.12, and 3.18 for NI. Considering the significant improvements of NI over RippleNet (a special case of AAM) in Table 2, these results confirm the early summarization problem, and our NI model has the capability to learn more informative neighborhood interactions.

We also randomly select a user-item pair ("u46", "m3993") from the test set and plots the weight matrix $\mathbf{A}$ and interaction matrix $\mathbf{Z}$. We compare AAM and NI in Fig. 8. The x-axis represents the neighbors of item "m3993", and y-axis for the neighbors of user "u46". In user-item interaction graph, users are linked to items with positive feedbacks. Thus user neighbors are items, and item neighbors are users. Grids with darker colors have larger values.

We can observe that: (i) Comparing (a) and (c), we find AAM mainly focuses on a single neighbor "m2140" of the user, while NI focuses on many more other neighbor pairs. (ii) Comparing (a) and (b), we find AAM disregards those neighbor pairs with high interactions, *e.g.*, , ("m552", "u1744"). While in (c) and (d), we find NI preserves more neighbor pairs with high interactions. (iii) Checking in training set, we find the pairs with high interactions in our NI model, such as ("m7155", "u1744"), ("m1031", "u7521") and ("m2140", "u1744") are positive samples, which should be fully considered in prediction. Based on the above observations, we conclude AAM may lose useful information after compressing neighborhood information into single representation, while NI can preserve more useful information.

4 RELATED WORK

Our work is highly related with knowledge-enhanced recommendation, and graph representation models.

4.1 Knowledge-enhanced Recommendation

Traditional recommender systems mostly suffer from several inherent issues such as data sparsity and cold start problems. To address the above problems, researchers usually incorporate side information. The utilization of side information mainly categorizes into 3 groups.

The first is feature-based, which regards side information as plain features and concatenates those features with user/item IDs as model input, including Matrix factorization models [13, 16], DNN models [6, 19, 20], etc. Feature-based models highly rely on manual feature engineering to extract structural information, which is not end-to-end and less efficient.

The second way is meta path-based, which builds heterogeneous information network (HIN) on the side information. For example, PER [29] and FMG [31] extract meta path/meta graph-based features to represent the connection between users and items along different types of relation paths. MCRec [10] instead learns context representations from meta paths to facilitate recommendation. DeepCoevolve [4] further leverages user-item ineteraction network in sequential recommendation. Though these models are more intuitive, they usually require much expertise in meta-path design, making them less applicable in scenarios with complex schema.

Compared with the previous 2 ways, external knowledge graph contains much more fruitful facts and connections about items [1]. For example, CKE [30] proposes a general framework to jointly learn from the auxiliary knowledge graph, textual and visual information. DKN [27] is later proposed to incorporate knowledge embedding and text embedding for news recommendation. More recently, RippleNet [26] is proposed to simulate user preferences over the set of knowledge entities. It automatically extends user preference along links in the knowledge graph, and achieves state-of-the-art performance in knowledge graph-based recommendation. The major difference between prior work and ours is that NI focuses more on the interactions between neighbor nodes, and predict from graph structures directly.

4.2 Graph Representation

Graph representation learning aims to learn latent, low-dimensional representations of graph vertices, while preserving graph topology structure, node content, and other information. In general, there are two main types of graph representation methods: unsupervised and semi-supervised methods.

Most of the unsupervised graph representation algorithms focus on preserving graph structure for learning node representations [5, 18, 22]. For example, DeepWalk [18] uses random walks to generate node sequences and learn node representations. Node2vec [5] further exploits a biased random walk strategy to capture more flexible contextual structures. LINE [22] uses first-order and second-order proximity to model a joint probability distribution and a conditional probability distribution on connected vertices.

Another type is semi-supervised models [12, 15, 24]. In this type, there exist some labeled vertices for representation learning. For example, LANE [12] incorporates label information into the attributed network embedding while preserving their correlations. GCN [15] utilizes a localized graph convolutions for a classification task. GAT [24] uses self-attention network for information propagation, which utilizes a multi-head attention mechanism to increase model capacity. GCN and GAT are popular architectures of the general graph networks, and can be naturally regarded as plug-in graph representation modules in other supervised tasks. In this work, we mainly utilize graph networks to generate structural node embeddings for KIG.

5 CONCLUSION

In this paper, we review previous graph-based recommender systems and find an early summarization problem of the existing methods. We extend user-item interactions to neighbor-neighbor interactions, and propose Neighborhood Interaction (NI) to further explore the neighborhood structures of users and items. Integrating high-order neighborhood information with Graph Neural Networks and Knowledge Graphs into NI, we obtain an end-to-end model, namely Knowledge-enhanced Neighborhood Interaction (KNI). We compare KNI with state-of-the-art models on 4 real-world datasets, and the superior results of KNI on CTR prediction and top-N recommendation demonstrate its effectiveness. We also provide a case study to quantitatively measure the early summarization problem. In the future, a promising direction is extending neighborhood interactions to higher-orders. Another direction is integrating user-side information in KIG to adapt to more general scenarios.

ACKNOWLEDGMENTS

We would like to thank the support of National Natural Science Foundation of China (61632017, 61702327, 61772333), Shanghai Sailing Program (17YF1428200). Jian Tang is supported by the Natural Sciences and Engineering Research Council of Canada, as well as the Canada CIFAR AI Chair Program.

REFERENCES

[1] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*. AcM.

[2] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247* (2018).

[3] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM.

[4] Hanjun Dai, Yichen Wang, Rakshit Trivedi, and Le Song. 2016. Deep coevolutionary network: Embedding user and item features for recommendation. *arXiv preprint arXiv:1609.03675* (2016).

[5] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*. ACM.

[6] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247* (2017).

[7] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*.

[8] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*. International World Wide Web Conferences Steering Committee.

[9] Xiangnan He, Zhenkui He, Jingkuan Song, Zhenguang Liu, Yu-Gang Jiang, and Tat-Seng Chua. 2018. NAIS: Neural Attentive Item Similarity Model for Recommendation. *TKDE* (2018).

[10] Binbin Hu, Chuan Shi, Wayne Xin Zhao, and Philip S Yu. 2018. Leveraging meta-path based context for top-n recommendation with a neural co-attention model. In *SIGKDD*. ACM.

[11] Jin Huang, Wayne Xin Zhao, Hong-Jian Dou, Ji-Rong Wen, and Edward Y. Chang. 2018. Improving Sequential Recommendation with Knowledge-Enhanced Memory Networks. In *SIGIR*.

[12] Xiao Huang, Jundong Li, and Xia Hu. 2017. Label informed attributed network embedding. In *WSDM*. ACM.

[13] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. 2016. Field-aware factorization machines for CTR prediction. In *RecSys*. ACM.

[14] Santosh Kabbur, Xia Ning, and George Karypis. 2013. Fism: factored item similarity models for top-n recommender systems. In *SIGKDD*. ACM.

[15] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[16] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *SIGKDD*. ACM.

[17] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion.. In *AAAI*.

[18] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*. ACM.

[19] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *ICDM*.

[20] Yanru Qu, Bohui Fang, Weinan Zhang, and *et al.* 2018. Product-based Neural Networks for User Response Prediction over Multi-field Categorical Data. *arXiv preprint arXiv:1807.00311* (2018).

[21] Steffen Rendle. 2010. Factorization machines. In *ICDM*. IEEE.

[22] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. International World Wide Web Conferences Steering Committee.

[23] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph Convolutional Matrix Completion. *stat* 1050 (2017).

[24] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[25] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2017. Graphgan: Graph representation learning with generative adversarial nets. *arXiv preprint arXiv:1711.08267* (2017).

[26] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. 2018. RippleNet: Propagating User Preferences on the Knowledge Graph for Recommender Systems. In *CIKM*. ACM.

[27] Hongwei Wang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. DKN: Deep Knowledge-Aware Network for News Recommendation. *arXiv preprint arXiv:1801.08284* (2018).

[28] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. *arXiv preprint arXiv:1806.01973* (2018).

[29] Xiao Yu, Xiang Ren, Yizhou Sun, Quanquan Gu, Bradley Sturt, Urvashi Khandelwal, Brandon Norick, and Jiawei Han. 2014. Personalized entity recommendation: A heterogeneous information network approach. In *WSDM*. ACM.

[30] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *SIGKDD*.

[31] Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lun Lee. 2017. Meta-graph based recommendation fusion over heterogeneous information networks. In *SIGKDD*. ACM.

[32] Wayne Xin Zhao, Gaole He, Hongjian Dou, Jin Huang, Siqi Ouyang, and Ji-Rong Wen. 2018. KB4Rec: A Dataset for Linking Knowledge Bases with Recommender Systems. *arXiv preprint arXiv:1807.11141* (2018).