Exploring High-Order User Preference on the Knowledge Graph for Recommender Systems

HONGWEI WANG, Shanghai Jiao Tong University, China
FUZHENG ZHANG, Meituan-Dianping Group, China
JIALIN WANG, MIAO ZHAO, and WENJIE LI, The Hong Kong Polytechnic University, China
XING XIE, Microsoft Research Asia, China
MINYI GUO, Shanghai Jiao Tong University, China

To address the sparsity and cold-start problem of collaborative filtering, researchers usually make use of side information, such as social networks or item attributes, to improve the performance of recommendation. In this article, we consider the knowledge graph (KG) as the source of side information. To address the limitations of existing embedding-based and path-based methods for KG-aware recommendation, we propose *RippleNet*, an end-to-end framework that naturally incorporates the KG into recommender systems. RippleNet has two versions: (1) The *outward propagation* version, which is analogous to the actual ripples on water, stimulates the propagation of user preferences over the set of knowledge entities by automatically and iteratively extending a user's potential interests along links in the KG. The multiple "ripples" activated by a user's historically clicked items are thus superposed to form the preference distribution of the user with respect to a candidate item. (2) The *inward aggregation* version aggregates and incorporates the neighborhood information biasedly when computing the representation of a given entity. The neighborhood can be extended to multiple hops away to model high-order proximity and capture users' long-distance interests. In addition, we intuitively demonstrate how a KG assists with recommender systems in RippleNet, and we also find that RippleNet provides a new perspective of explainability for the recommended results in terms of the KG. Through extensive experiments on real-world datasets, we demonstrate that both versions of RippleNet achieve substantial gains in a variety of scenarios, including movie, book, and news recommendations, over several state-of-the-art baselines.

CCS Concepts: • **Information systems** → **Recommender systems**; • **Computing methodologies** → *Knowledge representation and reasoning*;

Additional Key Words and Phrases: Recommender systems, knowledge graph, outward propagation, inward aggregation

Preliminary versions of this article appeared in proceedings of the 2018 ACM International Conference on Information and Knowledge Management [42] and the 2019 Web Conference [44].
This work was partially sponsored by National Basic Research 973 Program of China (2015CB352403) and National Natural Science Foundation of China (61272291).
Authors' addresses: H. Wang, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China; email: wanghongwei55@gmail.com; F. Zhang, Meituan-Dianping Group, Beijing, China; email: zhangfuzheng@meituan.com; J. Wang, M. Zhao, and W. Li, Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China; emails: {csjlwang, csmiaozhao, cswjli}@comp.polyu.edu.hk; X. Xie, Microsoft Research Asia, Beijing, China; email: xingx@microsoft.com; M. Guo, Department of Computing, Shanghai Jiao Tong University, Shanghai, China; email: guo-my@cs.sjtu.edu.cn.

1046-8188/2019/03-ART32 $15.00
https://doi.org/10.1145/3312738

ACM Reference format:
Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. 2019. Exploring High-Order User Preference on the Knowledge Graph for Recommender Systems. *ACM Trans. Inf. Syst.* 37, 3, Article 32 (March 2019), 26 pages.
https://doi.org/10.1145/3312738

1 INTRODUCTION

The explosive growth of online content and services has provided overwhelming choices for users, such as news, movies, music, restaurants, and books. Recommender systems (RSs) intend to address the information explosion by finding a small set of items for users to meet their personalized interests. Among recommendation strategies, *collaborative filtering* (CF), which considers users' historical interactions and makes recommendations based on their potential common preferences, has achieved great success [18]. However, CF-based methods usually suffer from the sparsity of user-item interactions and the cold-start problem. To address these limitations, researchers have proposed incorporating *side information* into CF, such as social networks [14], user/item attributes [41], images [55], and contexts [32].

Among various types of side information, *knowledge graphs* (KGs) usually contain much more fruitful facts and connections about items. A KG is a type of directed heterogeneous graph in which nodes correspond to *entities* and edges correspond to *relations*. Recently, researchers have proposed several academic KGs, such as NELL,[1] DBpedia,[2] and commercial KGs, such as Google Knowledge Graph[3] and Microsoft Satori.[4] These knowledge graphs are successfully applied in many applications such as KG completion [20], question answering [9], word embedding [51], and text classification [45].

Inspired by the success of applying KGs in a wide variety of tasks, researchers also tried to utilize KGs to improve the performance of recommender systems. As shown in Figure 1, KGs can benefit the recommendation from three aspects: (1) KGs introduce semantic relatedness among items, which can help find their latent connections and improve the *precision* of recommended items; (2) KGs consist of relations with various types, which is helpful for extending a user's interests reasonably and increasing the *diversity* of recommended items; and (3) KGs connect a user's historical records and the recommended ones, thereby bringing *explainability* to recommender systems.

In general, existing KG-aware recommendation can be classified into two categories:

The first category is *embedding-based methods* [13, 41, 43, 55], which preprocess a KG with *knowledge graph embedding* (KGE) [46] algorithms and incorporate the learned entity embeddings into a recommendation framework. For example, Collaborative Knowledge-based Embedding (CKE) [55] combines a CF module with knowledge embedding, text embedding, and image embedding of items in a unified Bayesian framework. A Deep Knowledge-aware Network (DKN) [43] treats entity embeddings and word embeddings as different channels, then designs a CNN framework to combine them together for news recommendation. Signed Heterogeneous Information Network Embedding (SHINE) [41] designs deep autoencoders to embed sentiment networks, social networks, and profile (knowledge) networks for celebrity recommendations. The Knowledge-enhanced Sequential Recommender (KSR) [13] integrates RNN with a key-value memory network and further incorporates entity embeddings learned from TransE [3] to enhance the model capacity. Embedding-based methods show high flexibility in utilizing KGs to assist with

[1] http://rtw.ml.cmu.edu/rtw/.
[2] http://wiki.dbpedia.org/.
[3] https://www.google.com/intl/bn/insidesearch/features/search/knowledge.html.
[4] https://searchengineland.com/library/bing/bing-satori.

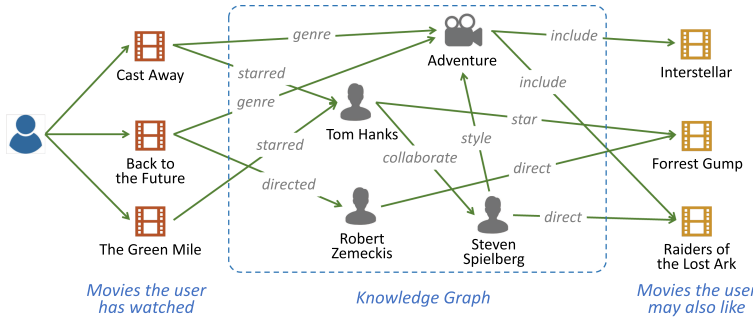


Fig. 1. Illustration of KG-enhanced movie recommender systems. The knowledge graph provides fruitful facts and connections among items, which are useful for improving precision, diversity, and explainability of recommended results.

recommender systems, but the adopted KGE algorithms in these methods focus more on modeling rigorous semantic relatedness (e.g., TransE [3] and TransR [20] assume $head + relation = tail$), which are more suitable for in-graph applications such as KG completion and link prediction rather than recommendation.

The second category is *path-based methods* [54, 57], which explore the various patterns of connections among items in KGs to provide additional guidance for recommendations. For example, Personalized Entity Recommendation (PER) [54] and Factorization Machine with Group lasso (FMG) [57] treat the KG as a heterogeneous information network (HIN) and extract meta-path/meta-graph-based latent features to represent the connectivity between users and items along different types of relation paths/graphs. Path-based methods make use of KGs in a more natural and intuitive way, but they rely heavily on manually designed meta-paths/meta-graphs, which is hard to be optimal in practice. Another concern is that it is impossible to design hand-crafted meta-paths/meta-graphs in certain scenarios (e.g., news recommendation) where entities and relations are not within one domain. A more recent work, Recurrent Knowledge Graph Embedding (RKGE) [33], employs a recurrent network to calculate embeddings of multiple paths linking a user-item pair, then uses a pooling operator to aggregate these path embeddings for recommendation. However, the learned entity embeddings are not regularized by semantics of the KG, which weakens the guidance from the KG and makes RKGE prone to overfitting.

To address the limitations of existing methods, we propose *RippleNet*, an end-to-end framework for KG-aware recommendation. RippleNet is designed for click-through rate (CTR) prediction, which takes a user-item pair as input and outputs the probability of the user engaging (e.g., clicking, browsing) the item. The key idea of RippleNet is to explore users' potential high-order preference on KG entities according to their historical records. We call users' potentially preferred KG entities *ripple sets*, because they share similar properties with the real ripples created by raindrops on the water: (1) analogous to real ripples, a user's potential interest in entities is activated by his or her historical preferences, then spreads along the links in the KG layer by layer, and (2) the strength of a user's potential interest in ripple sets weakens with the increase of distance, which is similar to the gradually attenuated amplitude of real ripples.

Based on the techniques of utilizing users' ripple set, the proposed RippleNet has two versions accordingly: (1) The *outward propagation* version (RippleNet-prop) treats the historical interests of each user as seeds in the KG, then extends the user's interests iteratively in his or her ripple sets to discover his or her hierarchical potential interests with respect to a candidate item. The multiple "ripples" created by seeds superpose to form a resultant preference distribution of the user over the knowledge graph. (2) The *inward aggregation* version (RippleNet-agg) aggregates and incorporates

the information from ripple sets of a given entity biasedly when computing its representation. The ripple sets can be extended to multiple hops away to model high-order proximity and capture users' long-distance interests. The major difference between RippleNet and the existing literature is that RippleNet combines the advantages of embedding-based and path-based methods: on the one hand, RippleNet incorporates KGE methods into recommender systems naturally by outward propagation or inward aggregation of user preferences; on the other hand, RippleNet can automatically discover possible explanatory paths from an item in a user's history to a candidate item, without any sort of hand-crafted design.

Empirically, we apply RippleNet to three real-world scenarios of movie, book, and news recommendations. The experiment results show that RippleNet significantly outperforms state-of-the-art baselines for recommendation. For example, RippleNet-agg achieves average *AUC* gains of 9.1%, 11.5%, and 10.3%, respectively, in movie, book, and news recommendations. In addition, we intuitively demonstrate how the KGs help explore users' high-order preferences for recommender systems in RippleNet. We also find that RippleNet provides a new perspective of explainability for the recommended results in terms of the KG.

In summary, our contributions in this article are as follows:

- To the best of our knowledge, this is the first work to combine embedding-based and path-based methods in KG-aware recommendation.
- We propose RippleNet, an end-to-end framework utilizing KGs to assist recommender systems. In RippleNet, the outward propagation model propagates users' potential preferences and explores their hierarchical interests in KG entities, while the inward aggregation model aggregates the neighborhood information of a given entity biasedly when calculating its representation.
- We conduct experiments on three real-world recommendation scenarios, and results demonstrate the efficacy of RippleNet over several state-of-the-art baselines.
- We release the code and datasets to researchers for validating the reported results and conducting further research. The code and data are available at https://github.com/hwwang55/RippleNet and https://github.com/hwwang55/KGCN.

2 PROBLEM FORMULATION

The KG-aware recommendation problem is formulated as follows. In a typical recommendation scenario, we have a set of M users $\mathcal{U} = \{u_1, u_2, \ldots, u_M\}$ and a set of N items $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$. The user-item interaction matrix $\mathbf{Y} \in \mathbb{R}^{M \times N}$ is defined according to users' implicit feedback, where

$$y_{uv} = \begin{cases} 1, & \text{if interaction } (u, v) \text{ is observed;} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

A value of 1 for y_{uv} indicates that there is an implicit interaction between user u and item v, such as behaviors of clicking, watching, browsing, and so forth. In addition to the interaction matrix $\mathbf{Y}$, we also have a knowledge graph $\mathcal{G}$ available, which consists of entity-relation-entity triples (h, r, t). Here $h \in \mathcal{E}$, $r \in \mathcal{R}$, and $t \in \mathcal{E}$ denote the head, relation, and tail of a knowledge triple, respectively, and $\mathcal{E}$ and $\mathcal{R}$ denote the set of entities and relations in the KG, respectively. For example, the triple (*Jurassic Park*, *film.film.director*, *Steven Spielberg*) states the fact that Steven Spielberg is the director of the film *Jurassic Park*. In many recommendation scenarios, an item $v \in \mathcal{V}$ may associate with one or more entities in $\mathcal{G}$. For example, the movie *Jurassic Park* is linked with its namesake in the KG, while news with title "France's Baby Panda Makes Public Debut" is linked with entities "France" and "panda." In this article, we consider the case where an item corresponds to exactly one entity but note that the proposed model can be easily applied to a multientity case by splitting one

Table 1. Key Symbols Used in This Article

Symbol	Meaning
$\mathcal{U} = \{u_1, u_2, \cdots u_M\}$	Set of users
$\mathcal{V} = \{v_1, v_2, \cdots, v_N\}$	Set of items
$\mathbf{Y} = \{y_{uv}\}, u \in \mathcal{U}, v \in \mathcal{V}$	User-item interaction matrix
$\hat{y}_{uv}$	Predicted engaging probability
$\mathcal{G} = (\mathcal{E}, \mathcal{R})$	Knowledge graph
$\mathcal{E} = \{e_1, e_2, \cdots\}$	Set of entities
$\mathcal{R} = \{r_1, r_2, \cdots\}$	Set of relations
(h, r, t)	A knowledge triple (head, relation, tail)
$\mathcal{E}_u^k$	The k-hop ripple set for user u
$\mathcal{N}(e)$	Neighbors of entity e
$\mathcal{S}(e)$	Sampled neighbors for entity e
π_r^u	Use-relation scoring function

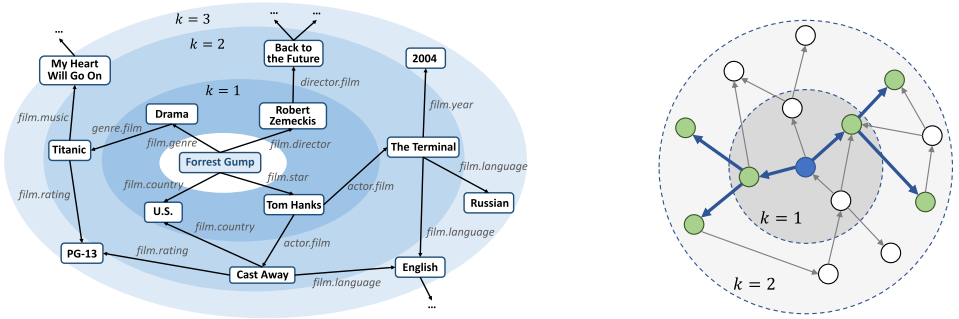


Fig. 2. (a) Illustration of ripple sets of *Forrest Gump* in the KG of movies. The concentric circles denote the ripple sets with different hops. The fading blue indicates decreasing relatedness between the center and surrounding entities. Note that the ripple sets of different hops are not necessarily disjoint in practice. (b) A two-layer ripple set of the given blue entity in a KG where $K = 2$.

user-item interaction into multiple user-entity interactions and averaging the multiple predicted user-entity probabilities as the final user-item score. In the following of this article, we may use notation v and e interchangeably when referring to an item. In addition, we use $\mathcal{N}(h)$ to denote the set of entities that entity h directly connects to, i.e., $\mathcal{N}(h) = \{t \mid (h, r, t) \in \mathcal{G}\}$.

Given interaction matrix $\mathbf{Y}$ as well as knowledge graph $\mathcal{G}$, we aim to predict whether user u has potential interest in item v with which he or she has had no interaction before. Our goal is to learn a prediction function $\hat{y}_{uv} = \mathcal{F}(u, v; \Theta, \mathbf{Y}, \mathcal{G})$, where $\hat{y}_{uv}$ denotes the probability that user u will click item v, and Θ denotes the model parameters of function $\mathcal{F}$.

We list the key symbols used in this article in Table 1.

3 RIPPLE SET

A knowledge graph usually contains fruitful facts and connections among entities. For example, as illustrated in Figure 2(a), the film *Forrest Gump* is linked with "Robert Zemeckis" (director), "Tom Hanks" (star), "U.S." (country), and "Drama" (genre), while "Tom Hanks" is further linked

with films *The Terminal* and *Cast Away*, which he starred in. These complicated connections in KG provide us a deep and latent perspective to explore user preferences. For example, if a user has ever watched *Forrest Gump*, he or she may possibly become a fan of Tom Hanks and be interested in *The Terminal* or *Cast Away*. To characterize users' hierarchically extended preferences in terms of KG, in RippleNet, we recursively define the k-hop ripple set for item v as follows:

Definition 1 (Item Ripple Set). Given knowledge graph $\mathcal{G}$, the k-hop ripple set for item v is defined as

$$\mathcal{E}_v^k = \bigcup_{h \in \mathcal{E}_v^{k-1}} \mathcal{N}(h), \quad k = 1, 2, \ldots, H, \tag{2}$$

where $\mathcal{E}_v^0 = \{v\}$ can be seen as the seed set and H is the predefined maximum hop number.

The item ripple set can be regarded as the set of relevant entities of the item with respect to the KG. Note that the definition of item ripple set is applicable to not only items but also all entities in the KG. Given the definition of item ripple set, we define the k-hop ripple set for user u as follows:

Definition 2 (User Ripple Set). Given interaction matrix $\mathbf{Y}$ and item ripple set $\mathcal{E}_v^k$, the k-hop ripple set for user u is defined as the union of k-hop ripple sets of items that the user engaged:

$$\mathcal{E}_u^k = \bigcup_{v:y_{uv}=1} \mathcal{E}_v^k, \quad k = 0, 1, \ldots, H. \tag{3}$$

The user ripple set can be regarded as natural extensions of a user's historical interests with respect to the KG.

In a real-world knowledge graph, the size of $\mathcal{E}_v^k$ and $\mathcal{E}_u^k$ varies significantly over all users and items, and may be prohibitively large in the worst case (especially when k is big). To keep the computational pattern of each batch fixed and more efficient, we uniformly sample a fixed-size set of neighbors for each entity e instead of using its full neighbors $\mathcal{N}(e)$. Specifically, we define the sampled neighborhood set of entity e as $\mathcal{S}(e)$, where $\mathcal{S}(e) = \{t \mid t \sim \mathcal{N}(e)\}$ and $|\mathcal{S}(e)| = K$ is a configurable constant.[5] We therefore replace $\mathcal{N}(\cdot)$ in Equation (2) with $\mathcal{S}(\cdot)$ to obtain a more compact item ripple set. The definition of user ripple set in Equation (3) is updated accordingly. Figure 2(b) gives an illustrative example of a two-layer ripple set for a given entity, where K is set as 2.

We would like to emphasize that the word "ripple" has two meanings: (1) Analogous to real ripples created by raindrops on the water, a user's potential interest in entities is activated by his or her historical preferences, then propagates along the links in the KG layer by layer, from near to distant. We visualize the analogy by the concentric circles illustrated in Figure 2(a). (2) The strength of a user's potential preferences in ripple sets weakens with the increase of the hop number k, which is similar to the gradually attenuated amplitude of real ripples. The fading blue in Figure 2(a) shows the decreasing relatedness between the center and surrounding entities.

4 OUTWARD PROPAGATION

In this section, we discuss the outward propagation version of RippleNet, i.e., RippleNet-prop, in detail. We also discuss the proposed RippleNet-prop.

4.1 Framework

The framework of RippleNet-prop is illustrated in Figure 3. RippleNet-prop takes a user u and an item v as input and outputs the predicted probability that user u will engage v. For user u, his

[5]Technically, $\mathcal{S}(e)$ may contain duplicates if $\mathcal{N}(e) < K$.

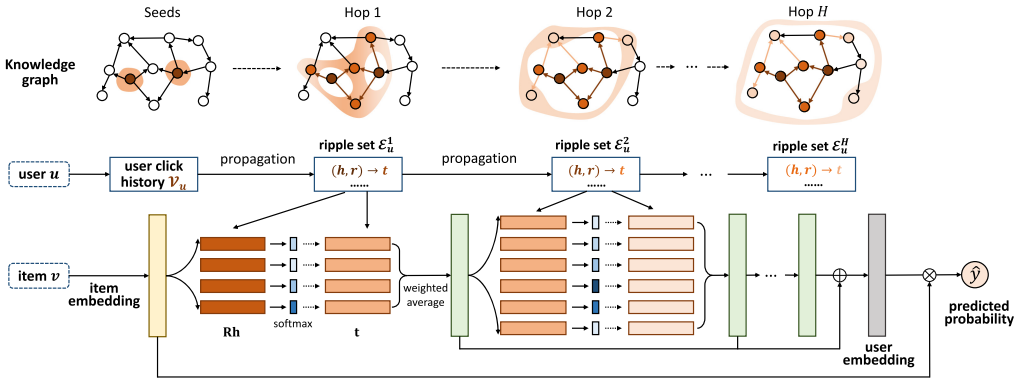


Fig. 3. The overall framework of the RippleNet-prop. The KGs in the upper part illustrate the corresponding ripple sets activated by the user's click history.

or her historical set of interests $\mathcal{V}_u$ is treated as seeds in the KG, then extended along links to form multiple ripple sets $\mathcal{E}_u^k$ $(k = 1, 2, \ldots, H)$. These ripple sets are used to interact with the item embedding (the yellow block) iteratively for obtaining the responses of user u with respect to item v (the green blocks), which are then combined to form the final user embedding (the gray block). Lastly, we use the embeddings of user u and item v together to compute the predicted probability $\hat{y}_{uv}$.

4.2 Preference Propagation

Traditional CF-based methods and their variants [17, 40] learn latent representations of users and items, then predict unknown ratings by directly applying a specific function to their representations such as inner product. In RippleNet, to model the interactions between users and items in a more fine-grained way, we propose a preference propagation technique to explore users' potential interests in his or her ripple sets.

As shown in Figure 3, each item v is associated with an item embedding $\mathbf{v} \in \mathbb{R}^d$, where d is the dimension of embeddings. Item embedding can incorporate one-hot ID [17], attributes [41], bag of words [43], or context information [32] of an item, based on the application scenario. Given the item embedding $\mathbf{v}$ as well as the 0-hop and 1-hop ripple set $\mathcal{E}_u^0$ and $\mathcal{E}_u^1$ of user u, each entity t_i in $\mathcal{E}_u^1$ is assigned a relevance probability by comparing item v to the head $h_i \in \mathcal{E}_u^0$ and the relation r_i in triple (h_i, r_i, t_i):

$$p_i = \text{softmax}\left(\mathbf{v}^{\mathrm{T}}\mathbf{R}_i\mathbf{h}_i\right) = \frac{\exp\left(\mathbf{v}^{\mathrm{T}}\mathbf{R}_i\mathbf{h}_i\right)}{\sum_{(h,r,t)\in\mathcal{G},\ h\in\mathcal{S}_u^0,\ t\in\mathcal{S}_u^1}\exp\left(\mathbf{v}^{\mathrm{T}}\mathbf{R}\mathbf{h}\right)}, \tag{4}$$

where $\mathbf{R}_i \in \mathbb{R}^{d\times d}$ and $\mathbf{h}_i \in \mathbb{R}^d$ are the embeddings of relation r_i and head h_i, respectively. The relevance probability p_i can be regarded as the similarity of item $\mathbf{v}$ and the entity $\mathbf{h}_i$ measured in the space of relation $\mathbf{R}_i$. Note that it is necessary to take the embedding matrix $\mathbf{R}_i$ into consideration when calculating the relevance of item $\mathbf{v}$ and entity $\mathbf{h}_i$, since an item-entity pair may have different similarities when measured by different relations. For example, *Forrest Gump* and *Cast Away* are highly similar when considering their directors or stars but have less in common if measured by genre or writer.

After obtaining the relevance probabilities, we take the sum of entities in $\mathcal{S}_u^1$ weighted by the corresponding relevance probabilities, and the vector $\mathbf{o}_u^1$ is returned:

$$\mathbf{o}_u^1 = \sum_{t_i \in \mathcal{S}_u^1} p_i \mathbf{t}_i, \tag{5}$$

where $\mathbf{t}_i \in \mathbb{R}^d$ is the embedding of tail t_i. Vector $\mathbf{o}_u^1$ can be seen as the one-order response of user u's click history $\mathcal{V}_u$ with respect to item v. This is similar to item-based CF methods [17, 43], in which a user is represented by his or her related items rather than an independent feature vector to reduce the size of parameters. Through the operations in Equation (4) and Equation (5), a user's interests are transferred from his or her history set $\mathcal{V}_u$ to his or her one-hop ripple set $\mathcal{E}_u^1$, which is called *preference propagation* in RippleNet-prop.

Note that through replacing $\mathbf{v}$ with $\mathbf{o}_u^1$ and increasing the hop number by 1 in Equation (4), we can repeat the procedure of preference propagation to obtain user u's two-order response $\mathbf{o}_u^2$, and the procedure can be performed iteratively on user u's ripple sets $\mathcal{S}_u^i$ for $i = 1, \ldots, H$. Therefore, a user's preference is propagated up to H hops away from his or her click history, and we observe multiple responses of user u with different orders: $\mathbf{o}_u^1, \mathbf{o}_u^2, \ldots, \mathbf{o}_u^H$. The embedding of user u with respect to item v is calculated by combining the responses of all orders:

$$\mathbf{u} = \mathbf{o}_u^1 + \mathbf{o}_u^2 + \cdots + \mathbf{o}_u^H. \tag{6}$$

Note that it is necessary to incorporate $\mathbf{o}_u^k$'s for all hops k in calculating user embedding, because a user's preference is theoretically distributed in the set of all entities. Finally, the user embedding and item embedding are combined to output the predicted clicking probability:

$$\hat{y}_{uv} = \sigma(\mathbf{u}^{\mathrm{T}} \mathbf{v}), \tag{7}$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the sigmoid function.

4.3 Learning Algorithm

In RippleNet-prop, we intend to maximize the following posterior probability of model parameters Θ after observing the knowledge graph $\mathcal{G}$ and the matrix of implicit feedback $\mathbf{Y}$:

$$\max p(\Theta | \mathcal{G}, \mathbf{Y}), \tag{8}$$

where Θ includes the embeddings of all entities, relations, and items. This is equivalent to maximizing

$$p(\Theta | \mathcal{G}, \mathbf{Y}) = \frac{p(\Theta, \mathcal{G}, \mathbf{Y})}{p(\mathcal{G}, \mathbf{Y})} \propto p(\Theta) \cdot p(\mathcal{G} | \Theta) \cdot p(\mathbf{Y} | \Theta, \mathcal{G}) \tag{9}$$

according to Bayes' theorem. In Equation (9), the first term $p(\Theta)$ measures the a priori probability of model parameters Θ. Following [55], we set $p(\Theta)$ as Gaussian distribution with zero mean and a diagonal covariance matrix:

$$p(\Theta) = \mathcal{N}(\mathbf{0}, \lambda_1^{-1} \mathbf{I}). \tag{10}$$

The second item in Equation (9) is the likelihood function of the observed knowledge graph $\mathcal{G}$ given Θ. Recently, researchers have proposed a great many knowledge graph embedding methods, including translational distance models [3, 20] and semantic matching models [21, 25] (we will continue the discussion on KGE methods in Section 8.3). In RippleNet-prop, we use a three-way tensor factorization method to define the likelihood function for KGE:

$$p(\mathcal{G} | \Theta) = \prod_{(h,r,t) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}} p\Big((h,r,t) | \Theta\Big) = \prod_{(h,r,t) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}} \mathcal{N}\Big(I_{h,r,t} - \mathbf{h}^{\mathrm{T}} \mathbf{R} \mathbf{t},\ \lambda_2^{-1}\Big), \tag{11}$$

where the indicator $I_{h,r,t}$ equals 1 if $(h,r,t) \in \mathcal{G}$ and is 0 otherwise. Based on the definition in Equation (11), the scoring functions of entity-entity pairs in KGE and item-entity pairs in preference

ALGORITHM 1: Learning algorithm for RippleNet-prop

Input: Interaction matrix $\mathbf{Y}$; knowledge graph $\mathcal{G}(\mathcal{E}, \mathcal{R})$; neighborhood sampling mapping $\mathcal{S} : e \rightarrow 2^{\mathcal{E}}$; trainable parameters: $\{\mathbf{e}\}_{e\in\mathcal{E}}$, $\{\mathbf{R}\}_{r\in\mathcal{R}}$; hyperparameters: d, H, K
Output: Prediction function $\mathcal{F}(u, v|\Theta, \mathbf{Y}, \mathcal{G})$

```
Initialize all parameters;
Calculate ripple sets {S_u^k}_{k=1}^H for each user u;
for number of training iteration do
    Sample minibatch of positive and negative interactions from Y;
    Sample minibatch of true and false triples from G;
    Calculate gradients ∂L/∂E and {∂L/∂R}_{r∈R} on the minibatch by back-propagation according
    to Equations (4) through (13);
    Update E and {R}_{r∈R} by gradient descent with learning rate η;
return F(u, v|Θ);
```

propagation can thus be unified under the same calculation model. The last term in Equation (9) is the likelihood function of the observed implicit feedback given Θ and the KG, which is defined as the product of Bernouli distributions:

$$p(\mathbf{Y}|\Theta, \mathcal{G}) = \prod_{(u,v)\in\mathbf{Y}} \sigma(\mathbf{u}^{\mathrm{T}}\mathbf{v})^{y_{uv}} \cdot \left(1 - \sigma(\mathbf{u}^{\mathrm{T}}\mathbf{v})\right)^{1-y_{uv}} \tag{12}$$

based on Equations (2) through (7).

Taking the negative logarithm of Equation (9), we have the following loss function for RippleNet-prop:

$$\begin{aligned} \min \mathcal{L} &= -\log\left(p(\mathbf{Y}|\Theta, \mathcal{G}) \cdot p(\mathcal{G}|\Theta) \cdot p(\Theta)\right) \\ &= \sum_{(u,v)\in\mathbf{Y}} -\Big(y_{uv}\log\sigma(\mathbf{u}^{\mathrm{T}}\mathbf{v}) + (1 - y_{uv})\log\left(1 - \sigma(\mathbf{u}^{\mathrm{T}}\mathbf{v})\right)\Big) \\ &\quad + \frac{\lambda_2}{2}\sum_{r\in\mathcal{R}} \|\mathbf{I}_r - \mathbf{E}^{\mathrm{T}}\mathbf{R}\mathbf{E}\|_2^2 + \frac{\lambda_1}{2}\left(\|\mathbf{E}\|_2^2 + \sum_{r\in\mathcal{R}}\|\mathbf{R}\|_2^2\right), \end{aligned} \tag{13}$$

where $\mathbf{E}$ is the embedding matrix for all entities, $\mathbf{I}_r$ is the slice of the indicator tensor $\mathbf{I}$ in the KG for relation r, and $\mathbf{R}$ is the embedding matrix of relation r. In Equation (13), the first term measures the cross-entropy loss between ground truth of interactions $\mathbf{Y}$ and predicted value by RippleNet, the second term measures the squared error between the ground truth of the KG $\mathbf{I}_r$ and the reconstructed indicator matrix $\mathbf{E}^{\mathrm{T}}\mathbf{R}\mathbf{E}$, and the third term is the regularizer for preventing overfitting.

It is intractable to solve the above objection directly; therefore, we employ a stochastic gradient descent (SGD) algorithm to iteratively optimize the loss function. The learning algorithm of RippleNet-prop is presented in Algorithm 1. In each training iteration, to make the computation more efficient, we randomly sample a minibatch of positive/negative interactions from $\mathbf{Y}$ and true/false triples from $\mathcal{G}$ following the negative sampling strategy in [22]. Then we calculate the gradients of the loss $\mathcal{L}$ with respect to model parameters Θ and update all parameters by back-propagation based on the sampled minibatch. We will discuss the choice of hyperparameters in the experiments section.

5 INWARD AGGREGATION

In this section, we introduce the inward aggregation version of RippleNet, i.e., RippleNet-agg. We first present the design of a single layer, followed by a complete learning algorithm of RippleNet-agg. We will also discuss how RippleNet-agg works intuitively.

5.1 Representation Aggregation

RippleNet-agg is proposed to capture high-order structural proximity among entities in a knowledge graph. We will start by describing a single RippleNet-agg layer in this subsection. Consider a candidate pair of user u and item (entity) v. We use r_{e_i,e_j} to denote the relation between entity e_i and e_j. We also use a function $g : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ to compute the score between a user and a relation:

$$\pi_r^u = g(\mathbf{u}, \mathbf{r}), \tag{14}$$

where $\mathbf{u} \in \mathbb{R}^d$ and $\mathbf{r} \in \mathbb{R}^d$ are the representations of user u and relation r, respectively, and d is the dimension of representations. In general, π_r^u characterizes the importance of relation r to user u. For example, a user may have more potential interests in the movies that share the same "star" with his or her historically liked ones, while another user may be more concerned about "genre" of movies.

To characterize the topological proximity structure of item v, we compute the linear combination of v's sampled neighborhood (i.e., one-hop sampled ripple set):

$$\mathbf{v}_{\mathcal{S}(v)}^u = \sum_{e \in \mathcal{S}(v)} \tilde{\pi}_{r_{v,e}}^u \mathbf{e}, \tag{15}$$

where $\tilde{\pi}_{r_{v,e}}^u$ is the normalized user-relation score

$$\tilde{\pi}_{r_{v,e}}^u = \frac{\exp(\pi_{r_{v,e}}^u)}{\sum_{e \in \mathcal{S}(v)} \exp(\pi_{r_{v,e}}^u)}, \tag{16}$$

and $\mathbf{e}$ is the representation of entity e. User-relation scores act as *personalized filters* when computing an entity's neighborhood representation, since we aggregate the neighbors biasedly with respect to these user-specific scores.

The final step in a RippleNet-agg layer is to aggregate the entity representation $\mathbf{v}$ and its neighborhood representation $\mathbf{v}_{\mathcal{S}(v)}^u$ into a single vector. We implemented the aggregator $agg : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ in RippleNet-agg as the *sum* operation, which takes the summation of two representation vectors, followed by a nonlinear transformation:

$$agg = \sigma\left(\mathbf{W} \cdot \left(\mathbf{v} + \mathbf{v}_{\mathcal{S}(v)}^u\right) + \mathbf{b}\right), \tag{17}$$

where $\mathbf{W}$ and $\mathbf{b}$ are transformation weight and bias, respectively, and σ is the nonlinear function such *ReLU*. But there are also other implementations, such as the *concat* aggregator [11], which concatenates the two representation vectors first before applying nonlinear transformation: $agg_{concat} = \sigma(\mathbf{W} \cdot concat(\mathbf{v}, \mathbf{v}_{\mathcal{S}(v)}^u) + \mathbf{b})$, or the *neighbor-only* aggregator [37], which directly takes the neighborhood representation of entity v as the output representation: $agg_{neighbor} = \sigma(\mathbf{W} \cdot \mathbf{v}_{\mathcal{S}(v)}^u + \mathbf{b})$.

Aggregation is a key step in RippleNet-agg, since the representation of an item is bound up with its neighbors through the connecting relations (which are analogous to rubber bands) in the KG. We will evaluate the aggregator in experiments.

ALGORITHM 2: Learning algorithm for RippleNet-agg

Input: Interaction matrix $\mathbf{Y}$; knowledge graph $\mathcal{G}(\mathcal{E}, \mathcal{R})$; neighborhood sampling mapping $\mathcal{S} : e \to 2^{\mathcal{E}}$; trainable parameters: $\{\mathbf{u}\}_{u \in \mathcal{U}}$, $\{\mathbf{e}\}_{e \in \mathcal{E}}$, $\{\mathbf{r}\}_{r \in \mathcal{R}}$, $\{\mathbf{W}_i, \mathbf{b}_i\}_{i=1}^{H}$; hyperparameters: H, d, $g(\cdot)$, $f(\cdot)$, $\sigma(\cdot)$, $agg(\cdot)$

Output: Prediction function $\mathcal{F}(u, v|\Theta, \mathbf{Y}, \mathcal{G})$

1 **while** *RippleNet-agg not converge* **do**
2 **for** (u, v) *in* $\mathbf{Y}$ **do**
3 $\{\mathcal{E}[i]\}_{i=0}^{H} \leftarrow$ Get-Ripple-Set (v);
4 $\mathbf{e}^u[0] \leftarrow \mathbf{e}, \forall e \in \mathcal{E}[0]$;
5 **for** $k = 1, \ldots, H$ **do**
6 **for** $e \in \mathcal{E}[k]$ **do**
7 $\mathbf{e}^u_{\mathcal{S}(e)}[k-1] \leftarrow \sum_{d \in \mathcal{S}(e)} \tilde{\pi}^u_{r_{e,d}} \mathbf{d}^u[k-1]$;
8 $\mathbf{e}^u[k] \leftarrow agg\left(\mathbf{e}^u_{\mathcal{S}(e)}[k-1], \mathbf{e}^u[k-1]\right)$;
9 $\mathbf{v}^u \leftarrow \mathbf{e}^u[H]$;
10 Calculate predicted probability $\hat{y}_{uv} = f(\mathbf{u}, \mathbf{v}^u)$;
11 Update parameters by gradient descent;
12 **return** $\mathcal{F}$;
13 **Function** Get-Ripple-Set (v)
 $\mathcal{E}[H] \leftarrow v$;
14 **for** $k = H-1, \ldots, 0$ **do**
15 $\mathcal{E}[k] \leftarrow \mathcal{E}[k+1]$;
16 **for** $e \in \mathcal{E}[k+1]$ **do**
17 $\mathcal{E}[k] \leftarrow \mathcal{E}[k] \cup \mathcal{S}(e)$;
18 **return** $\{\mathcal{E}[i]\}_{i=0}^{H}$;

5.2 Learning Algorithm

Through a single RippleNet-agg layer, the final representation of an entity is dependent on itself as well as its immediate neighbors, which we name one-order entity representation. It is natural to extend RippleNet-agg from one layer to multiple layers to reasonably explore users' potential interests in a broader and deeper way. The technique is intuitive: propagating the initial representation of each entity (zero-order representation) to its neighbors leads to one-order entity representation; then we can repeat this procedure, i.e., further propagating and aggregating one-order representations to obtain two-order ones. Generally speaking, the k-order representation of an entity is a mixture of initial representations of itself and its neighbors up to k hops away. This is an important property for RippleNet-agg, which we will discuss in the next subsection.

The formal description of the above steps is presented in Algorithm 2. H denotes the number of aggregation iterations, and a suffix $[k]$ attached by a representation vector denotes k-order. For a given user-item pair (u, v) (line 2), we first calculate the ripple sets $\mathcal{E}$ of v in an iterative layer-by-layer manner (lines 3, 13–19). Then the aggregation is repeated H times (line 5): in iteration k, we calculate the neighborhood representation of each entity $e \in \mathcal{E}[k]$ (line 7), then aggregate it with its own representation $\mathbf{e}^u[k-1]$ to obtain the one to be used at the next iteration (line 8). The final H-order entity representation is denoted as $\mathbf{v}^u$ (line 9), which is fed into a function $f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ together with user representation $\mathbf{u}$ for predicting the probability:

$$\hat{y}_{uv} = f(\mathbf{u}, \mathbf{v}^u). \tag{18}$$

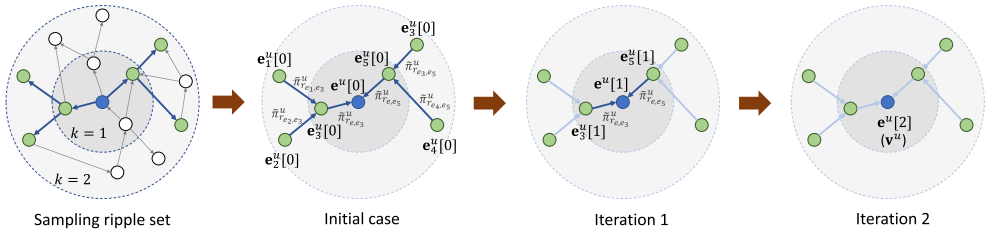


Fig. 4. The overall framework of the RippleNet-agg. The initial entity representations (green nodes in the second graph) are aggregated iteratively to form the final entity (item) embedding (blue node in the rightmost graph).

Figure 4 also illustrates the RippleNet-agg framework with $H = 2$ and $K = 2$, in which each entity representation and its neighborhood representations are mixed to form the representation for the next iteration.

Note that Algorithm 2 traverses all possible user-item pairs (line 2). To make computation more efficient, we use a negative sampling strategy during training. The complete loss function is as follows:

$$\mathcal{L} = -\sum_{u\in\mathcal{U}}\left(\sum_{v:y_{uv}=1}\mathcal{J}(y_{uv},\hat{y}_{uv}) - \sum_{i=1}^{T^u}\mathbb{E}_{v_i\sim P(v_i)}\mathcal{J}(y_{uv_i},\hat{y}_{uv_i})\right) + \lambda\|\mathcal{F}\|_2^2, \tag{19}$$

where $\mathcal{J}$ is cross-entropy loss, P is a negative sampling distribution, and T^u is the number of negative samples for user u. In this article, $T^u = |\{v : y_{uv} = 1\}|$ and P follows a uniform distribution. The last term is the L2-regularizer.

6 DISCUSSION

In this section, we first provide a unified view of the proposed two RippleNet models from the perspective of exploring high-order proximity information of the KG. Then we discuss how RippleNet provides explainability for the recommended results. We also introduce ripple superposition, an interesting phenomenon in RippleNet that enhances the learning of user preferences. Finally, we discuss the computational complexity of the two RippleNet models.

6.1 Unified View of Two RippleNet Models

In previous sections, we present RippleNet-prop and RippleNet-agg, which are named according to the techniques of using high-order neighborhood information (i.e., ripple sets). In this section, we provide a unified view of the two types of RippleNet and show that their major difference lies in whether we characterize the high-order proximity information of the KG at the user end or the item end. Recall that RippleNet is an end-to-end model that predicts the probability of a user engaging an item:

$$\hat{y}_{uv} = f(\mathbf{u},\mathbf{v}), \tag{20}$$

where $\mathbf{u}$ and $\mathbf{v}$ are the representations of user u and item v, respectively. To capture the high-order proximity information of the KG, we have two choices to encode ripple sets in the above prediction function. The first is at the user end:

$$\hat{y}_{uv} = f\left(g_u\left(\{\mathcal{E}_u^k\}_{k=0}^H\right),\mathbf{v}\right), \tag{21}$$

where $\mathcal{E}_u^k$ is the k-hop ripple set of user u. If f and g_u are implemented as in Equations (4) through (7), the model is exactly the RippleNet-prop introduced in Section 4. On the contrary, if we encode

ripple sets at the item end, we have

$$\hat{y}_{uv} = f\left(\mathbf{u}, g_v\left(\{\mathcal{E}_v^k\}_{k=0}^H\right)\right), \tag{22}$$

where $\mathcal{E}_v^k$ is the k-hop ripple set of item v. If f and g_v are implemented as in Algorithm 2, the model is exactly the RippleNet-agg introduced in Section 5. From this point of view, RippleNet-prop and RippleNet-agg are two different implementations of exploring high-order proximity information of the KG under a unified framework.

It is worth mentioning that one may naturally consider combining RippleNet-prop and RippleNet-agg further, i.e., $\hat{y}_{uv} = f(g_u(\{\mathcal{E}_u^k\}_{k=0}^H, g_v(\{\mathcal{E}_v^k\}_{k=0}^H))$. However, introducing ripple sets at both the user and item ends is redundant for characterizing high-order proximity information. More importantly, it will incur higher time complexity and be more prone to overfitting noises from high-order neighbors.

6.2 Explainability

Explainable recommender systems [35] aim to reveal why a user might like a particular item, which helps improve his or her acceptance or satisfaction of recommendations and increase trust in RS. The explanations are usually based on community tags [38], social networks [30], aspect [2], and phrase sentiment [56]. Since RippleNet explores users' interests based on the KG, it provides a new point of view of explainability by tracking the paths from a user's history to an item with high relevance probability (Equation (4)) in the KG. For example, a user's interest in the film *Back to the Future* might be explained by the path "*user* $\xrightarrow{watched}$ *Forrest Gump* $\xrightarrow{directed\ by}$ *Robert Zemeckis* $\xrightarrow{directs}$ *Back to the Future*" if the item *Back to the Future* is of high relevance probability with *Forrest Gump* and "Robert Zemeckis" in the user's one-hop and two-hop ripple set, respectively. Note that different from path-based methods [54, 57] where the patterns of path are manually designed, RippleNet automatically discovers the possible explanation paths according to relevance probability. We will further present a visualized example in the experiments section to intuitively demonstrate the explainability of RippleNet.

6.3 Ripple Superposition

A common phenomenon in RippleNet is that a user's ripple sets may be large in size, which dilutes his or her potential interests inevitably in preference propagation. However, we observe that relevant entities of different items in a user's click history often highly overlap. In other words, an entity could be reached by multiple paths in the KG starting from a user's click history. For example, *Saving Private Ryan* is connected to a user who has watched *The Terminal*, *Jurassic Park*, and *Braveheart* through actor "Tom Hanks," director "Steven Spielberg," and genre "War," respectively. These parallel paths greatly increase a user's interests in overlapped entities. We refer to the case as *ripple superposition*, as it is analogous to the interference phenomenon in physics in which two waves superpose to form a resultant wave of greater amplitude in particular areas. The phenomenon of ripple superposition is illustrated in the second KG in Figure 3, where the darker red around the two lower middle entities indicates higher strength of the user's possible interests. We will also discuss ripple superposition in the experiments section.

6.4 Complexity Analysis

The time complexity analysis for RippleNet-prop is as follows. The complexity of constructing ripple sets for all users is $O(M(K + K^2 + \cdot + K^H)) = O(MHK^H)$, where M is the number of users, K is the neighbor sampling size, and H is the maximum hop number. Suppose the number of

user-item interactions and the number of triples in the KG are Y and G, respectively. In each training iteration of Algorithm 1, the complexity of Equations (4) and (5) are $O(K^k d^2)$ and $O(K^k d)$, respectively, for $k = 1, \ldots, H$, where d is the dimension of embeddings. Therefore, the complexity of calculating the predicted clicking probability for single and all user-item interactions is $O(HK^H d^2)$ and $O(YHK^H d^2)$, respectively. In addition, the complexity of calculating the likelihood of all KG triples is $O(Gd^2)$. In general, the complexity of each training iteration of RippleNet-prop is $O(YHK^H d^2 + Gd^2)$.

The time complexity analysis for RippleNet-agg is as follows. Similar to RippleNet-prop, the complexity of constructing ripple sets for all items is $O(NHK^H)$, where N is the number of items. In each training iteration of Algorithm 2, the complexity of lines 2, 5, 6, 7, and 8 is Y, H, K^H, Kd, and d^2, respectively. Therefore, the overall complexity of each training iteration of RippleNet-agg is $O(YHK^H(Kd + d^2)) = O(YHK^{H+1}d + YHK^H d^2)$.

7 EXPERIMENTS

In this section, we evaluate RippleNet on three real-world scenarios: movie, book, and news recommendations. We first introduce the datasets, baselines, and experiment setup, then present the experiment results. We will also give a case study of visualization and discuss the choice of hyper-parameters in this section.

7.1 Datasets

We utilize the following three datasets in our experiments for movie, book, and news recommendations:

- **MovieLens-20M**[6] is a widely used benchmark dataset in movie recommendations, which consists of approximately 20 million explicit ratings (ranging from 1 to 5) on the MovieLens website.
- **Book-Crossing**[7] contains 1,149,780 explicit ratings (ranging from 0 to 10) of books in the Book-Crossing community.
- **Bing-News** contains 1,025,192 pieces of implicit feedback collected from the server logs of Bing News[8] from October 16, 2016, to August 11, 2017. Each piece of news has a title and a snippet.

Since MovieLens-20M and Book-Crossing are explicit feedback data, we transform them into implicit feedback, where each entry is marked with 1 indicating that the user has rated the item (the threshold of rating is 4 for MovieLens-20M, while no threshold is set for Book-Crossing due to their sparsity), and sample an unwatched set marked as 0 for each user, which is of equal size with the rated ones. For MovieLens-20M and Book-Crossing, we use the ID embeddings of users and items as raw input, while for Bing-News, we concatenate the ID embedding of a piece of news and the averaged word embedding of its title as raw input for the item, since news titles are typically much longer than names of movies or books, hence providing more useful information for recommendation.

We use Microsoft Satori to construct the knowledge graph for each dataset. For MovieLens-20M and Book-Crossing, we first select a subset of triples from the whole KG whose relation name contains "movie" or "book" with the confidence level greater than 0.9. Given the sub-KG, we collect Satori IDs of all valid movies/books by matching their names with the tail of triples *(head,*

[6] https://grouplens.org/datasets/movielens/.
[7] http://www2.informatik.uni-freiburg.de/~cziegler/BX/.
[8] https://www.bing.com/news.

Table 2. Basic Statistics of the Three Datasets

	MovieLens-20M	Book-Crossing	Bing-News
# users	138,159	17,860	141,487
# items	16,954	14,967	535,145
# interactions	13,501,622	139,746	1,025,192
# entities	102,569	77,903	1,565,722
# relations	32	25	2,366
# KG triples	499,474	151,500	8,742,568

film.film.name, tail) or *(head, book.book.title, tail)*. Items with multiple matched or no matched entities are excluded for simplicity. We then match the item IDs with the head of all triples, select all well-matched triples from the sub-KG, and extend the set of entities iteratively up to four hops. The constructing process is similar for Bing-News except that (1) we use entity linking tools to extract entities in news titles and (2) we do not impose restrictions on the names of relations since the entities in news titles are not within one particular domain. The basic statistics of the three datasets are presented in Table 2.

7.2 Baselines

We compare the proposed RippleNet with the following KG-aware baselines. Hyperparameter settings for baselines are introduced in the next subsection.

- **LibFM** [27] is a widely used feature-based factorization model in CTR scenarios. We concatenate user ID, item ID, and the corresponding averaged entity embeddings learned from TransR [20][9] as input for LibFM.
- **Wide&Deep** [7] is a general deep model for recommendation combining a (wide) linear channel with a (deep) nonlinear channel. Similar to LibFM, we use the embeddings of users, items, and entities to feed Wide&Deep.
- **PER** [54] treats the KG as heterogeneous information networks and extracts meta-path-based features to represent the connectivity between users and items.
- **CKE** [55] combines CF with structural, textual, and visual knowledge in a unified framework for recommendation. We implement CKE as a CF plus structural knowledge module in this article.
- **SHINE** [41] designs deep autoencoders to embed a sentiment network, social network, and profile (knowledge) network for celebrity recommendation. Here we use autoencoders for user-item interaction and item profile to predict click probability.
- **DKN** [43] treats entity embedding and word embedding as multiple channels and combines them together in CNN for CTR prediction. In this article, we use movie/book names and news titles as textual input for DKN.

7.3 Experiment Setup

In RippleNet-agg, we set functions g and f as inner product, σ as $ReLU$ for non-last-layer aggregator, and $tanh$ for last-layer aggregator. The complete hyperparameter settings for RippleNet-prop and RippleNet-agg are given in Table 3 and Table 4, respectively, which are determined by optimizing AUC on a validation set.

[9]The implementation of TransR in our experiments is available at https://github.com/thunlp/Fast-TransX.

Table 3. Hyperparameter Settings for RippleNet-prop on the Three Datasets

MovieLens-20M	$d = 16, H = 2, K = 16, \lambda_1 = 2 \times 10^{-7}, \lambda_2 = 1 \times 10^{-2}, \eta = 1 \times 10^{-2}$
Book-Crossing	$d = 4, H = 3, K = 32, \lambda_1 = 1 \times 10^{-5}, \lambda_2 = 1 \times 10^{-2}, \eta = 1 \times 10^{-3}$
Bing-News	$d = 32, H = 3, K = 8, \lambda_1 = 1 \times 10^{-5}, \lambda_2 = 5 \times 10^{-2}, \eta = 5 \times 10^{-3}$

d: dimension of entity embedding, H: depth of ripple sets, K: neighborhood sampling size, λ_1: L2 regularizer weight, λ_2: KGE weight, η: learning rate.

Table 4. Hyperparameter Settings for RippleNet-agg on the Three Datasets

MovieLens-20M	$d = 32, H = 2, K = 4, \lambda = 1 \times 10^{-7}, \eta = 2 \times 10^{-2}$
Book-Crossing	$d = 64, H = 3, K = 8, \lambda = 2 \times 10^{-5}, \eta = 2 \times 10^{-5}$
Bing-News	$d = 32, H = 2, K = 8, \lambda = 1 \times 10^{-5}, \eta = 5 \times 10^{-3}$

d: dimension of entity and user embedding, H: depth of ripple sets, K: neighborhood sampling size, λ: L2 regularizer weight, η: learning rate.

For each dataset, the ratio of training, evaluation, and test set is 6 : 2 : 2. Each experiment is repeated 3 times, and the average performance is reported. We evaluate our method in two experiment scenarios: (1) In click-through rate (CTR) prediction, we apply the trained model to each piece of interactions in the test set and output the predicted click probability. We use *AUC* and *Accuracy* to evaluate the performance of CTR prediction. (2) In top-K recommendation, we use the trained model to select K items with the highest predicted click probability for each user in the test set and choose *Precision@K*, *Recall@K*, *F1@K* to evaluate the recommended sets.

The hyperparameter settings for baselines are as follows. For LibFM, the dimension is $\{1, 1, 8\}$ and the number of training epochs is 50. The dimension of TransR is 32. For Wide&Deep, the dimension of user, item, and entity is 64, and we use a two-layer-deep channel with dimensions of 100 and 50 as well as a wide channel. For PER, we use manually designed user-item-attribute-item paths as features, i.e., "user-movie-director-movie," "user-movie-genre-movie," and "user-movie-star-movie" for MovieLens-20M, and "user-book-author-book" and "user-book-genre-book" for Book-Crossing. For CKE, the dimensions of the three datasets are 64, 128, and 64. For SHINE, the dimension of hidden layers in the autoencoder is 256-64-256, and we use concatenation and inner product as the aggregation and similarity function, respectively. For DKN, the dimension of word embedding and entity embedding is 64, and the number of filters is 128 for each window size 1, 2, 3. Other hyperparameters are the same as reported in their original papers or as default in their codes.

7.4 Empirical Study

We conduct an empirical study to investigate the correlation between the average number of common neighbors of an item pair in the KG and whether they have common rater(s) in RS. For each dataset, we first randomly sample 1 million item pairs, then count the average number of k-hop neighbors that the two items share in the KG under the following two circumstances: (1) the two items have at least one common rater in RS and (2) the two items have no common rater in RS. The results are presented in Figures 5(a), 5(b), and 5(c), respectively, which clearly show that if two items have common rater(s) in RS, they likely share more common k-hop neighbors in the KG for fixed k. The above findings empirically demonstrate that *the similarity of proximity structures of two items in the KG could assist in measuring their relatedness in RS*. In addition, we plot the ratio of the two average numbers with different hops (i.e., dividing the higher bar by its immediate lower bar for each hop number) in Figure 5(d), from which we observe that the proximity structures of

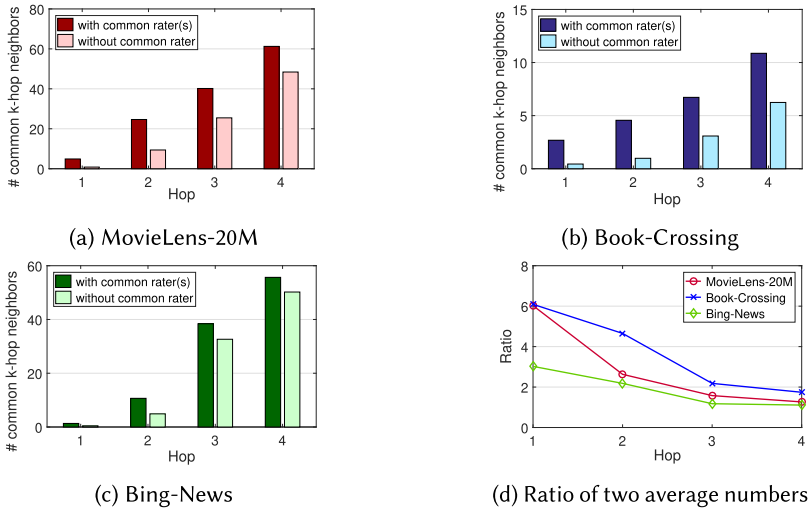


Fig. 5. The average number of k-hop neighbors that two items share in the KG w.r.t. whether they have common raters in (a) MovieLens-20M, (b) Book-Crossing, and (c) Bing-News datasets. (d) The ratio of the two average numbers with different hops.

Table 5. The Results of *AUC* and *Accuracy* in CTR Prediction

Model	MovieLens-20M		Book-Crossing		Bing-News	
	AUC	*Accuracy*	*AUC*	*Accuracy*	*AUC*	*Accuracy*
RippleNet-prop	0.970	0.925	0.729*	0.662	**0.678**	**0.632***
RippleNet-agg**	**0.978***	**0.932***	**0.738***	**0.687***	0.674	0.620
CKE	0.924	0.880	0.674	0.635	0.560	0.517
SHINE	0.911	0.868	0.668	0.631	0.554	0.537
DKN	0.814	0.733	0.621	0.598	0.661	0.604
PER	0.832	0.783	0.623	0.588	-	-
LibFM	0.966	0.914	0.685	0.639	0.644	0.588
Wide&Deep	0.953	0.907	0.711	0.623	0.654	0.595

*Statistically significant improvements over the best baseline by unpaired two-sample t-test with $p = 0.1$.

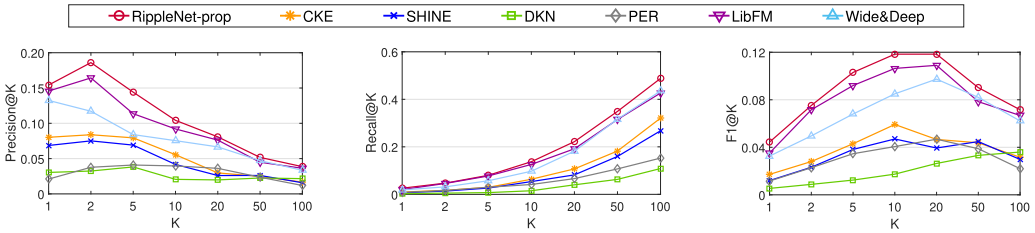
**RippleNet-agg has three variants according to the type of aggregator. We report the best performance among the three variants here. The detailed performance of different versions of RippleNet-agg is shown in Table 6.

two items under the two circumstances become more similar with the increase of the hop number. This is because any two items are probable to share a large amount of k-hop neighbors in the KG for a large k, even if there is no direct similarity between them in reality. The result motivates us to find a moderate hop number in RippleNet to explore users' potential interests as far as possible while avoiding introducing too much noise.

7.5 Results

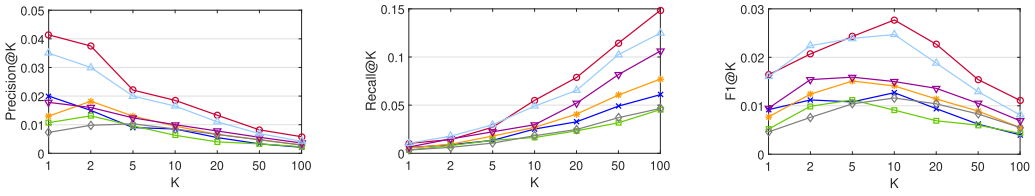
The results of all methods in CTR prediction and top-K recommendation are presented in Table 5 and Figures 6, 7, and 8, respectively.[10] We have the following observations:

[10]We do not show the curves of RippleNet-agg for clarity since they are very close to RippleNet-prop.



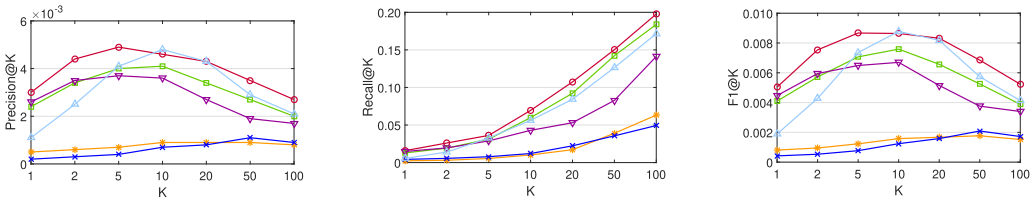
(a) *Precision@K* (b) *Recall@K* (c) *F1@K*

Fig. 6. The results of *Precision@K*, *Recall@K*, and *F1@K* in top-*K* recommendation for MovieLens-20M.



(a) *Precision@K* (b) *Recall@K* (c) *F1@K*

Fig. 7. The results of *Precision@K*, *Recall@K*, and *F1@K* in top-*K* recommendation for Book-Crossing.



(a) *Precision@K* (b) *Recall@K* (c) *F1@K*

Fig. 8. The results of *Precision@K*, *Recall@K*, and *F1@K* in top-*K* recommendation for Bing-News.

- CKE and SHINE perform better in movie and book recommendations than news. This may be because MovieLens-20M and Book-Crossing are denser than Bing-News, which is more favorable for the collaborative filtering part in CKE. Meanwhile, the one-hop triples for news are too complicated when taken as profile input in SHINE.
- DKN performs best in news recommendation compared with other baselines, but performs the worst in movie and book recommendations. This is because movie and book names are too short and ambiguous to provide useful information.
- PER performs unsatisfactorily on movie and book recommendations because the user-defined meta-paths can hardly be optimal. In addition, it cannot be applied in news recommendation since the types of entities and relations involved in news are too complicated to predefine meta-paths.
- As two generic recommendation tools, LibFM and Wide&Deep achieve satisfactory performance, demonstrating that they can make good use of knowledge from KGs in their algorithms.
- RippleNet performs best among all methods in the three datasets. We find that RippleNet-prop performs better in Bing-News, while RippleNet-agg performs better in MovieLens-20M and Book-Crossing. Specifically, RippleNet-agg achieves average *AUC* gains of 9.1%,

Table 6. Comparison among Variants of RippleNet-agg

Aggregator	MovieLens-20M		Book-Crossing		Bing-News	
	AUC	*Accuracy*	*AUC*	*Accuracy*	*AUC*	*Accuracy*
Sum	**0.978**	**0.932**	**0.738**	**0.688**	0.671	0.618
Concat	0.977	0.931	0.734	0.681	**0.674**	**0.620**
Neighbor	0.977	**0.932**	0.728	0.679	0.660	0.598
Avg	0.975	0.929	0.722	0.682	0.642	0.588

Table 7. *AUC* Result of RippleNet-prop with Different Neighbor Sampling Size K

K	2	4	8	16	32	64
MovieLens-20M	0.954	0.956	0.961	0.967	**0.970**	0.969
Book-Crossing	0.694	0.696	0.708	**0.726**	0.706	0.711
Bing-News	0.659	0.672	0.670	0.673	**0.678**	0.671

11.5%, and 10.3% in movie, book, and news recommendations, respectively, compared with baselines. RippleNet also achieves outstanding performance in top-K recommendation as shown in Figures 6, 7, and 8. Note that the performance of top-K recommendation is much lower for Bing-News because the number of news is significantly larger than movies and books.

7.5.1 Comparison among Variants of RippleNet-agg. Table 6 summarizes the performance of RippleNet-agg variants. The first three (sum, concat, neighbor) correspond to different aggregators introduced in the preceding section, while the last variant (avg) is a reduced case of sum where neighborhood representations are directly averaged without user-relation scores (i.e., $\mathbf{v}^u_{\mathcal{N}(v)} = \sum_{e \in \mathcal{N}(v)} \mathbf{e}$ instead of Equation (15)). Therefore, avg is used to examine the efficacy of the "attention mechanism." From the results we find the following:

- Sum aggregator performs best in general, while the performance of neighbor aggregator shows a clear gap on Book-Crossing and Bing-News. This may be because the neighbor aggregator uses the neighborhood representation only, thus losing useful information from the entity itself.
- Avg performs worse than sum, especially in Book-Crossing and Bing-News, where interactions are sparse. This demonstrates that capturing users' personalized preferences and semantic information of the KG does benefit the recommendation.

7.5.2 Impact of Neighborhood Sampling Size K. We vary the neighborhood sampling size K to investigate the robustness of RippleNet. The results of *AUC* of RippleNet-prop and RippleNet-agg are presented in Table 7 and Table 8, respectively. We observe that with the increase of K, the performance of RippleNet is improved at first because a larger K can encode more knowledge from the KG. But notice that the performance drops when the neighborhood sampling size is too large. In general, a size of 32 is enough for RippleNet-prop and a size of 8 is enough for RippleNet-agg according to the experiment results.

7.5.3 Impact of Depth of Ripple Sets H. We also vary the depth of ripple sets H to see how performance changes in RippleNet. The results are shown in Table 9 and Table 10, respectively, which shows that the best performance is achieved when H is 2,3 in RippleNet-prop and

Table 8. *AUC* Result of RippleNet-agg with Different Neighbor Sampling Size K

K	2	4	8	16	32	64
MovieLens-20M	0.978	**0.979**	0.978	0.978	0.977	0.978
Book-Crossing	0.680	0.727	**0.736**	0.725	0.711	0.723
Bing-News	0.669	0.672	**0.674**	0.671	0.662	0.657

Table 9. *AUC* Result of RippleNet-prop with Different Depth of Ripple Sets H

H	1	2	3	4
MovieLens-20M	0.965	**0.969**	0.964	0.965
Book-Crossing	0.727	0.722	**0.730**	0.702
Bing-News	0.662	0.676	**0.679**	0.674

Table 10. *AUC* Result of RippleNet-agg with Different Depth of Ripple Sets H

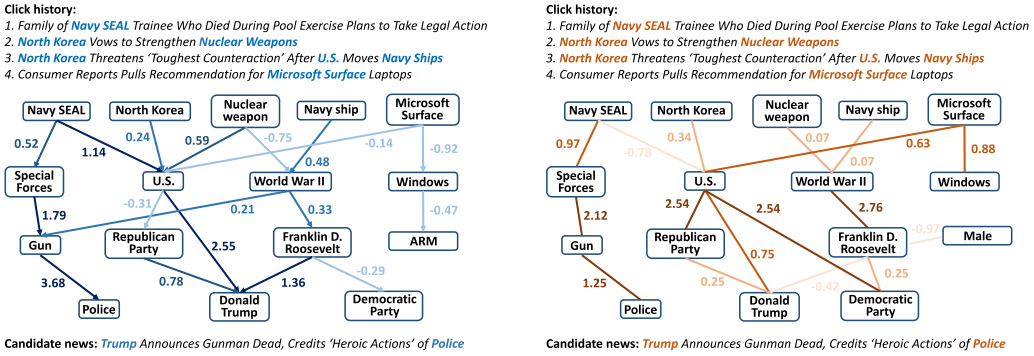
H	1	2	3	4
MovieLens-20M	0.972	**0.976**	0.974	0.514
Book-Crossing	0.724	0.731	**0.738**	0.706
Bing-News	0.667	**0.673**	0.671	0.631

RippleNet-agg. We attribute the phenomenon to the tradeoff between the positive signals from long-distance dependency and negative signals from noises: too small of an H can hardly explore interentity relatedness and dependency of long distance, while too large of an H brings much more noise than useful signals, as stated in Section 7.4. In addition, RippleNet-agg is more sensitive to H, as we observe the occurrence of serious model collapse when H = 3 or 4 in RippleNet-agg. This is also in accordance with our intuition, since a too-long relation chain makes little sense when inferring interitem similarities.

7.6 Case Study

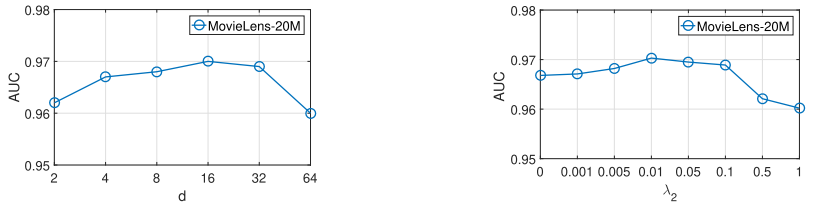
To intuitively demonstrate the preference propagation in RippleNet-prop and representation aggregation in RippleNet-agg, we randomly sample a user with four clicked pieces of news and select one candidate news from his or her test set with label 1. In RippleNet-prop, for each of the user's k-hop relevant entities, we calculate the (unnormalized) relevance probability between the entity and the candidate news or its k-order responses. In RippleNet-agg, we calculate the (unnormalized) user-relation scores between the given user and all types of relations and draw all the triples linking the click history and the candidate news within three hops. The results are presented in Figure 9, in which the darker shade of blue indicates larger values, and we omit names of relations for clearer presentation.

From Figure 9(a), we observe that RippleNet-prop associates the candidate news with the user's relevant entities with different strengths. The candidate news can be reached via several paths in the KG with high weights from the user's click history, such as "Navy SEAL"–"Special Forces"–"Gun"–"Police." These highlighted paths automatically discovered by preference propagation can



(a) RippleNet-prop (b) RippleNet-agg

Fig. 9. Visualization of relevance probabilities (in RippleNet-prop)/user-relation scores (in RippleNet-agg) for a randomly sampled user w.r.t. a piece of candidate news with label 1. Links with value lower than −1.0 are omitted.



(a) Dimension of embedding (b) Training weight of KGE term

Fig. 10. Parameter sensitivity of RippleNet-prop.

thus be used to explain the recommendation result, as discussed in Section 6.2. Additionally, it is also worth noticing that several entities in the KG receive more intensive attention from the user's history, such as "U.S.," "World War II," and "Donald Trump." These central entities result from the ripple superposition discussed in Section 6.3 and can serve as the user's potential interests for future recommendation.

The visualization result of RippleNet-agg is presented in Figure 9(b), from which we can see that the high-score links in RippleNet-agg are more decentralized, and the explainable paths are also less than RippleNet-prop. This is because the user-relation scores are calculated for the whole graph in RippleNet-agg, but the relevance probabilities are calculated exactly from the user's click history in RippleNet-prop. From this point of view, RippleNet-agg characterizes the user's global preferences on different relations in the KG, while RippleNet-prop concentrates more on local connection patterns from the user's click history to the candidate item.

7.7 Parameter Sensitivity

In this section, we investigate the sensitivity of RippleNet on the rest of the key hyperparameters, i.e., d and λ_2 in RippleNet-prop and d in RippleNet-agg. We vary d from 2 to 64 in RippleNet-prop and λ_2 from 0.0 to 1.0, respectively, while keeping other parameters fixed in RippleNet-prop. The results of AUC on MovieLens-20M are presented in Figure 10. We observe from Figure 10(a) that, with the increase of d, the performance is boosted at first since embeddings with a larger dimension can encode more useful information, but drops after $d = 16$ due to possible overfitting. From Figure 10(b), we can see that RippleNet-prop achieves the best performance when $\lambda_2 = 0.01$. This

Table 11. Parameter Sensitivity of RippleNet-agg (Dimension of Embedding d)

d	4	8	16	32	64	128
MovieLens-20M	0.968	0.970	0.975	**0.977**	0.973	0.972
Book-Crossing	0.709	0.732	0.733	0.735	**0.739**	0.736
Bing-News	0.669	0.672	**0.675**	0.673	0.668	0.667

is because the KGE term with a small weight cannot provide enough regularization constraints, while a large weight will mislead the objective function. We also vary d and from 4 to 128 in RippleNet-agg, and the results on the three datasets are shown in Table 11. We find that RippleNet-agg achieves best performance when $d = 16 \sim 64$, which is similar with RippleNet-prop.

8 RELATED WORK

8.1 Attention Mechanism

The attention mechanism was originally proposed in image classification [24] and machine translation [1], which aims to learn where to find the most relevant part of the input automatically as it is performing the task. The idea was soon transplanted to recommender systems [5, 29, 43, 47, 59]. For example, DADM [5] considers factors of specialty and date when assigning attention values to articles for recommendation; D-Attn [29] proposes an interpretable and dual attention-based CNN model that combines review text and ratings for product rating prediction; DKN [43] uses an attention network to calculate the weight between a user's clicked item and a candidate item and dynamically aggregate the user's historical interests. RippleNet-prop can be viewed as a special case of attention where tails are average weighted by similarities between their associated heads, tails, and certain item, while RippleNet-agg also uses an attention mechanism to average the neighborhood representations biasedly for a given entity. The difference between our work and the literature is that RippleNet designs a multilevel attention module to explore a user's high-order preference on the KG.

8.2 Memory Networks

Memory networks [23, 31, 49] are a recurrent attention model that utilize an external memory module for question answering and language modeling. The iterative reading operations on the external memory enable memory networks to extract long-distance dependency in texts. Researchers have also proposed using memory networks in other tasks such as sentiment classification [19, 34] and recommendation [6, 12]. Note that these works usually focus on entry-level or sentence-level memories, while RippleNet-prop addresses entity-level connections in the KG, which is more fine-grained and intuitive when performing multihop iterations. In addition, RippleNet-prop also incorporates a KGE term as a regularizer for more stable and effective learning.

8.3 Knowledge Graph Embedding

RippleNet also connects to a large body of work in KGE methods [3, 15, 20, 25, 36, 39, 48, 52]. KGE intends to embed entities and relations in a KG into continuous vector spaces while preserving its inherent structure. Readers can refer to [46] for a more comprehensive survey. KGE methods are mainly classified into two categories: (1) Translational distance models, such as TransE [3], TransH [48], TransD [15], and TransR [20], exploit distance-based scoring functions when learning representations of entities and relations. For example, TransE [3] wants $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$ when (h, r, t) holds,

where $\mathbf{h}$, $\mathbf{r}$, and $\mathbf{t}$ are the corresponding representation vector of h, r, and t. Therefore, TransE assumes that the score function $f_r(h,t) = \|\mathbf{h}+\mathbf{r}-\mathbf{t}\|_2^2$ is low if (h,r,t) holds and high otherwise. (2) Semantic matching models, such as ANALOGY [25], ComplEx [36], and DisMult [52], measure the plausibility of knowledge triples by matching latent semantics of entities and relations. For example, DisMult [52] introduces a vector embedding $\mathbf{r} \in \mathbb{R}^d$ and requires $\mathbf{M}_r = \text{diag}(\mathbf{r})$. The scoring function is hence defined as $f_r(h,t) = \mathbf{h}^\top \text{diag}(\mathbf{r})\mathbf{t} = \sum_{i=1}^{d} [\mathbf{r}]_i \cdot [\mathbf{h}]_i \cdot [\mathbf{t}]_i$. Researchers also propose incorporating auxiliary information, such as entity types [50], logic rules [28], and textual descriptions [58], to assist KGE. However, these methods are more suitable for in-graph applications such as link prediction, according to their learning objectives. From this point of view, RippleNet can be seen as a specially designed KGE method that serves recommender systems directly.

8.4 Graph Convolutional Network

RippleNet-agg is also conceptually related to the graph convolutional network (GCN). In general, GCN can be categorized as spectral methods and nonspectral methods. Spectral methods represent graphs and perform convolution in the spectral space. For example, [4] defines the convolution in the Fourier domain and calculates the eigendecomposition of the graph Laplacian, [8] approximates the convolutional filters by Chebyshev expansion of the graph Laplacian, and [16] proposes a convolutional architecture via a localized first-order approximation of spectral graph convolutions. In contrast, nonspectral methods operate on the original graph directly and define convolution for groups of nodes. To handle the neighborhoods with varying size and maintain the weight-sharing property of a CNN, researchers propose learning a weight matrix for each node degree [10], extracting locally connected regions from graphs [26], or sampling a fixed-size set of neighbors as the support size [11]. RippleNet-agg can be seen as a nonspectral method for a special type of graphs (i.e., knowledge graph).

RippleNet-agg also connects to PinSage [53] and GAT [37]. But note that both PinSage and GAT are designed for homogeneous graphs. The major difference between RippleNet-agg and the literature is that we offer a new perspective for recommender systems with the assistance of a heterogeneous knowledge graph.

9 CONCLUSION AND FUTURE WORK

In this article, we propose RippleNet, an end-to-end framework that naturally incorporates the knowledge graph into recommender systems. RippleNet overcomes the limitations of existing embedding-based and path-based KG-aware recommendation methods by introducing ripple sets, which automatically explore users' potential high-order preference on the KG. We propose two versions of RippleNet. The first version, RippleNet-prop, simulates the propagation of user preference in ripple sets and unifies the preference propagation with regularization of KGE in a Bayesian framework. The second version, RippleNet-agg, aggregates the information from ripple sets hierarchically and biasedly when calculating the representation of a given entity. We conduct extensive experiments in three recommendation scenarios. The results demonstrate the significant superiority of both versions of RippleNet over strong baselines.

For future work, we plan to design nonuniform samplers when sampling the neighborhood of entities to better explore users' potential interests and improve the performance. We also notice that our method (and all literature) focuses on modeling item-end KG. Therefore, another interesting direction of future work is to investigate whether leveraging user-end KG is useful to improve recommendation. Designing an algorithm to well combine KGs at the two ends is also a promising direction.

ACKNOWLEDGMENTS

The authors acknowledge the anonymous reviewers for the helpful comments.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[2] Konstantin Bauman, Bing Liu, and Alexander Tuzhilin. 2017. Aspect based recommendations: Recommending items with the most valuable aspects based on user reviews. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 717–725.

[3] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*. 2787–2795.

[4] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. 2014. Spectral networks and locally connected networks on graphs. In *The 2nd International Conference on Learning Representations*.

[5] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. 2017. Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 335–344.

[6] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2018. Sequential recommendation with user memory networks. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*.

[7] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 7–10.

[8] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*. 3844–3852.

[9] Li Dong, Furu Wei, Ming Zhou, and Ke Xu. 2015. Question answering over freebase with multi-column convolutional neural networks. In *ACL*. 260–269.

[10] David K. Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*. 2224–2232.

[11] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[12] Haoran Huang, Qi Zhang, and Xuanjing Huang. 2017. Mention recommendation for Twitter with end-to-end memory network. In *IJCAI*.

[13] Jin Huang, Wayne Xin Zhao, Hongjian Dou, Ji-Rong Wen, and Edward Y. Chang. 2018. Improving sequential recommendation with knowledge-enhanced memory networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, 505–514.

[14] Mohsen Jamali and Martin Ester. 2010. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the 4th ACM Conference on Recommender Systems*. ACM, 135–142.

[15] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. 2015. 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)*. 687–696.

[16] Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *The 5th International Conference on Learning Representations*.

[17] Yehuda Koren. 2008. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 426–434.

[18] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009).

[19] Zheng Li, Yu Zhang, Ying Wei, Yuxiang Wu, and Qiang Yang. 2017. End-to-end adversarial memory network for cross-domain sentiment classification. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. 2237–2243.

[20] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*. 2181–2187.

[21] Hanxiao Liu, Yuexin Wu, and Yiming Yang. 2017. Analogical inference for multi-relational embeddings. In *Proceedings of the 34th International Conference on Machine Learning*. 2168–2178.

[22] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*. 3111–3119.

[23] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. *arXiv preprint arXiv:1606.03126* (2016).

[24] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. 2014. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems*. 2204–2212.

[25] Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. 2016. Holographic embeddings of knowledge graphs. In *AAAI*. 1955–1961.

[26] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning*. 2014–2023.

[27] Steffen Rendle. 2012. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)* 3, 3 (2012), 57.

[28] Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. 2015. Injecting logical background knowledge into embeddings for relation extraction. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 1119–1129.

[29] Sungyong Seo, Jing Huang, Hao Yang, and Yan Liu. 2017. Interpretable convolutional neural networks with dual local and global attention for review rating prediction. In *Proceedings of the 11th ACM Conference on Recommender Systems*. ACM, 297–305.

[30] Amit Sharma and Dan Cosley. 2013. Do social explanations work?: Studying and modeling the effects of social explanations in recommender systems. In *Proceedings of the 22nd International Conference on World Wide Web*. ACM, 1133–1144.

[31] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. End-to-end memory networks. In *Advances in Neural Information Processing Systems*. 2440–2448.

[32] Yu Sun, Nicholas Jing Yuan, Xing Xie, Kieran McDonald, and Rui Zhang. 2017. Collaborative intent prediction with real-time contextual data. *ACM Transactions on Information Systems* 35, 4 (2017), 30.

[33] Zhu Sun, Jie Yang, Jie Zhang, Alessandro Bozzon, Long-Kai Huang, and Chi Xu. 2018. Recurrent knowledge graph embedding for effective recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*. ACM, 297–305.

[34] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. 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)*. 1556–1566.

[35] Nava Tintarev and Judith Masthoff. 2007. A survey of explanations in recommender systems. In *IEEE 23rd International Conference on Data Engineering Workshop*. IEEE, 801–810.

[36] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *International Conference on Machine Learning*. 2071–2080.

[37] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *Proceedings of the 6th International Conferences on Learning Representations*.

[38] Jesse Vig, Shilad Sen, and John Riedl. 2009. Tagsplanations: Explaining recommendations using tags. In *Proceedings of the 14th International Conference on Intelligent User Interfaces*. ACM, 47–56.

[39] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. Graphgan: Graph representation learning with generative adversarial nets. In *AAAI*. 2508–2515.

[40] Hongwei Wang, Jia Wang, Miao Zhao, Jiannong Cao, and Minyi Guo. 2017. Joint topic-semantic-aware social recommendation for online voting. In *Proceedings of the 2017 ACM Conference on Information and Knowledge Management*. ACM, 347–356.

[41] Hongwei Wang, Fuzheng Zhang, Min Hou, Xing Xie, Minyi Guo, and Qi Liu. 2018. Shine: Signed heterogeneous information network embedding for sentiment link prediction. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*. ACM, 592–600.

[42] 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 *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. ACM.

[43] Hongwei Wang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. DKN: Deep knowledge-aware network for news recommendation. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1835–1844.

[44] Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. 2019. Knowledge graph convolutional networks for recommender systems. In *Proceedings of the 2019 World Wide Web Conference on World Wide Web*.

[45] Jin Wang, Zhongyuan Wang, Dawei Zhang, and Jun Yan. 2017. Combining knowledge with deep convolutional neural networks for short text classification. In *IJCAI*.

[46] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. 2017. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering* 29, 12 (2017), 2724–2743.

[47] Xuejian Wang, Lantao Yu, Kan Ren, Guanyu Tao, Weinan Zhang, Yong Yu, and Jun Wang. 2017. Dynamic attention deep model for article recommendation by learning human editors' demonstration. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2051–2059.

[48] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *AAAI*. 1112–1119.

[49] Jason Weston, Sumit Chopra, and Antoine Bordes. 2014. Memory networks. *arXiv preprint arXiv:1410.3916* (2014).

[50] Ruobing Xie, Zhiyuan Liu, and Maosong Sun. 2016. Representation learning of knowledge graphs with hierarchical types. In *IJCAI*. 2965–2971.

[51] Chang Xu, Yalong Bai, Jiang Bian, Bin Gao, Gang Wang, Xiaoguang Liu, and Tie-Yan Liu. 2014. Rc-net: A general framework for incorporating knowledge into word representations. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. ACM, 1219–1228.

[52] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding entities and relations for learning and inference in knowledge bases. In *Proceedings of the 3rd International Conference on Learning Representations*.

[53] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[54] 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 *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*. 283–292.

[55] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 353–362.

[56] Yongfeng Zhang, Guokun Lai, Min Zhang, Yi Zhang, Yiqun Liu, and Shaoping Ma. 2014. Explicit factor models for explainable recommendation based on phrase-level sentiment analysis. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, 83–92.

[57] Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lun Lee. 2017. Meta-graph based recommendation fusion over heterogeneous information networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 635–644.

[58] Huaping Zhong, Jianwen Zhang, Zhen Wang, Hai Wan, and Zheng Chen. 2015. Aligning knowledge and text embeddings by entity descriptions. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 267–272.

[59] Guorui Zhou, Chengru Song, Xiaoqiang Zhu, Xiao Ma, Yanghui Yan, Xingya Dai, Han Zhu, Junqi Jin, Han Li, and Kun Gai. 2017. Deep interest network for click-through rate prediction. *arXiv preprint arXiv:1706.06978* (2017).

Received September 2018; revised December 2018; accepted February 2019