Knowledge-aware Graph Neural Networks with Label Smoothness Regularization for Recommender Systems

Hongwei Wang
Stanford University
hongweiw@cs.stanford.edu

Fuzheng Zhang
Meituan-Dianping Group
zhangfuzheng@meituan.com

Mengdi Zhang
Meituan-Dianping Group
zhangmengdi02@meituan.com

Jure Leskovec
Stanford University
jure@cs.stanford.edu

Miao Zhao, Wenjie Li
Hong Kong Polytechnic University
{csmiaozhao,cswjli}@comp.polyu.edu.hk

Zhongyuan Wang
Meituan-Dianping Group
wangzhongyuan02@meituan.com

ABSTRACT

Knowledge graphs capture structured information and relations between a set of entities or items. As such knowledge graphs represent an attractive source of information that could help improve recommender systems. However, existing approaches in this domain rely on manual feature engineering and do not allow for an end-to-end training. Here we propose *Knowledge-aware Graph Neural Networks with Label Smoothness regularization* (KGNN-LS) to provide better recommendations. Conceptually, our approach computes user-specific item embeddings by first applying a trainable function that identifies important knowledge graph relationships for a given user. This way we transform the knowledge graph into a user-specific weighted graph and then apply a graph neural network to compute personalized item embeddings. To provide better inductive bias, we rely on *label smoothness* assumption, which posits that adjacent items in the knowledge graph are likely to have similar user relevance labels/scores. Label smoothness provides regularization over the edge weights and we prove that it is equivalent to a label propagation scheme on a graph. We also develop an efficient implementation that shows strong scalability with respect to the knowledge graph size. Experiments on four datasets show that our method outperforms state of the art baselines. KGNN-LS also achieves strong performance in cold-start scenarios where user-item interactions are sparse.

KEYWORDS

Knowledge-aware recommendation; graph neural networks; label propagation

ACM Reference Format:
Hongwei Wang, Fuzheng Zhang, Mengdi Zhang, Jure Leskovec, Miao Zhao, Wenjie Li, and Zhongyuan Wang. 2019. Knowledge-aware Graph Neural Networks with Label Smoothness Regularization for Recommender Systems. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3292500.3330836

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330836

1 INTRODUCTION

Recommender systems are widely used in Internet applications to meet user's personalized interests and alleviate the information overload [4, 29, 32]. Traditional recommender systems that are based on collaborative filtering [13, 22] usually suffer from the cold-start problem and have trouble recommending brand new items that have not yet been heavily explored by the users. The sparsity issue can be addressed by introducing additional sources of information such as user/item profiles [23] or social networks [22].

Knowledge graphs (KGs) capture structured information and relations between a set of entities [8, 9, 18, 24–28, 33, 34, 36]. KGs are heterogeneous graphs in which nodes correspond to *entities* (e.g., items or products, as well as their properties and characteristics) and edges correspond to *relations*. KGs provide connectivity information between items via different types of relations and thus capture semantic relatedness between the items.

The core challenge in utilizing KGs in recommender systems is to learn how to capture *user-specific* item-item relatedness captured by the KG. Existing KG-aware recommender systems can be classified into path-based methods [8, 33, 36], embedding-based methods [9, 26, 27, 34], and hybrid methods [18, 24, 28]. However, these approaches rely on manual feature engineering, are unable to perform end-to-end training, and have poor scalability. Graph Neural Networks (GNNs), which aggregate node feature information from node's local network neighborhood using neural networks, represent a promising advancement in graph-based representation learning [3, 5–7, 11, 15]. Recently, several works developed GNNs architecture for recommender systems [14, 19, 28, 31, 32], but these approaches are mostly designed for *homogeneous* bipartite user-item interaction graphs or user-/item-similarity graphs. It remains an open question how to extend GNNs architecture to *heterogeneous* knowledge graphs.

In this paper, we develop *Knowledge-aware Graph Neural Networks with Label Smoothness regularization* (KGNN-LS) that extends GNNs architecture to knowledge graphs to simultaneously capture semantic relationships between the items as well as personalized user preferences and interests. To account for the relational heterogeneity in KGs, similar to [28], we use a trainable and personalized relation scoring function that transforms the KG into a user-specific weighted graph, which characterizes both the semantic information of the KG as well as user's personalized interests. For example, in the movie recommendation setting the relation scoring function could learn that a given user really cares about "director" relation between movies and persons, while somebody else may care more

about the "lead actor" relation. Using this personalized weighted graph, we then apply a graph neural network that for every item node computes its embedding by aggregating node feature information over the local network neighborhood of the item node. This way the embedding of each item captures it's local KG structure in a user-personalized way.

A significant difference between our approach and traditional GNNs is that the edge weights in the graph are not given as input. We set them using user-specific relation scoring function that is trained in a supervised fashion. However, the added flexibility of edge weights makes the learning process prone to overfitting, since the only source of supervised signal for the relation scoring function is coming from user-item interactions (which are sparse in general). To remedy this problem, we develop a technique for regularization of edge weights during the learning process, which leads to better generalization. We develop an approach based on *label smoothness* [35, 38], which assumes that adjacent entities in the KG are likely to have similar user relevancy labels/scores. In our context this assumption means that users tend to have similar preferences to items that are nearby in the KG. We prove that label smoothness regularization is equivalent to *label propagation* and we design a *leave-one-out* loss function for label propagation to provide extra supervised signal for learning the edge scoring function. We show that the knowledge-aware graph neural networks and label smoothness regularization can be unified under the same framework, where label smoothness can be seen as a natural choice of regularization on knowledge-aware graph neural networks.

We apply the proposed method to four real-world datasets of movie, book, music, and restaurant recommendations, in which the first three datasets are public datasets and the last is from Meituan-Dianping Group. Experiments show that our method achieves significant gains over state-of-the-art methods in recommendation accuracy. We also show that our method maintains strong recommendation performance in the cold-start scenarios where user-item interactions are sparse.

2 RELATED WORK

2.1 Graph Neural Networks

Graph Neural Networks (or Graph Convolutional Neural Networks, GCNs) aim to generalize convolutional neural networks to non-Euclidean domains (such as graphs) for robust feature learning. Bruna et al. [3] define the convolution in Fourier domain and calculate the eigendecomposition of the graph Laplacian, Defferrard et al. [5] approximate the convolutional filters by Chebyshev expansion of the graph Laplacian, and Kipf et al. [11] propose a convolutional architecture via a first-order approximation. In contrast to these *spectral* GCNs, *non-spectral* GCNs operate on the graph directly and apply "convolution" (i.e., weighted average) to local neighbors of a node [6, 7, 15].

Recently, researchers also deployed GCNs in recommender systems: PinSage [32] applies GCNs to the pin-board bipartite graph in Pinterest. Monti et al. [14] and Berg et al. [19] model recommender systems as matrix completion and design GCNs for representation learning on user-item bipartite graphs. Wu et al. [31] use GCNs on user/item structure graphs to learn user/item representations. The difference between these works and ours is that they are all designed for homogeneous bipartite graphs or user/item-similarity graphs where GCNs can be used directly, while here we investigate GCNs for heterogeneous KGs. Wang et al. [28] use GCNs in KGs for recommendation, but simply applying GCNs to KGs without proper regularization is prone to overfitting and leads to performance degradation as we will show later. Schlichtkrull et al. also propose using GCNs to model KGs [17], but not for the purpose of recommendations.

2.2 Semi-supervised Learning on Graphs

The goal of graph-based semi-supervised learning is to correctly label all nodes in a graph given that only a few nodes are labeled. Prior work often makes assumptions on the distribution of labels over the graph, and one common assumption is smooth variation of labels of nodes across the graph. Based on different settings of edge weights in the input graph, these methods are classified as: (1) Edge weights are assumed to be given as input and therefore fixed [1, 37, 38]; (2) Edge weights are parameterized and therefore learnable [10, 21, 35]. Inspired by these methods, we design a module of label smoothness regularization in our proposed model. The major distinction of our work is that the label smoothness constraint is not used for semi-supervised learning on graphs, but serves as regularization to assist the learning of edge weights and achieves better generalization for recommender systems.

2.3 Recommendations with Knowledge Graphs

In general, existing KG-aware recommender systems can be classified into three categories: (1) *Embedding-based methods* [9, 26, 27, 34] pre-process a KG with *knowledge graph embedding* (KGE) [30] algorithms, then incorporate learned entity embeddings into recommendation. Embedding-based methods are highly flexible in utilizing KGs to assist recommender systems, but the KGE algorithms focus more on modeling rigorous semantic relatedness (e.g., TransE [2] assumes $head + relation = tail$), which are more suitable for graph applications such as link prediction rather than recommendations. In addition, embedding-based methods usually lack an end-to-end way of training. (2) *Path-based methods* [8, 33, 36] explore various patterns of connections among items in a KG (a.k.a meta-path or meta-graph) to provide additional guidance for recommendations. Path-based methods make use of KGs in a more intuitive way, but they rely heavily on manually designed meta-paths/meta-graphs, which are hard to tune in practice. (3) *Hybrid methods* [18, 24, 28] combine the above two categories and learn user/item embeddings by exploiting the structure of KGs. Our proposed model can be seen as an instance of hybrid methods.

3 PROBLEM FORMULATION

We begin by describing the KG-aware recommendations problem and introducing notation. In a typical recommendation scenario, we have a set of users $\mathcal{U}$ and a set of items $\mathcal{V}$. The user-item interaction matrix $\mathbf{Y}$ is defined according to users' implicit feedback, where $y_{uv} = 1$ indicates that user u has engaged with item v, such as clicking, watching, or purchasing. We also have a knowledge graph $\mathcal{G} = \{(h, r, t)\}$ available, in which $h \in \mathcal{E}$, $r \in \mathcal{R}$, and $t \in \mathcal{E}$ denote the head, relation, and tail of a knowledge triple, $\mathcal{E}$ and $\mathcal{R}$ are the set of entities and relations in the knowledge graph, respectively. For

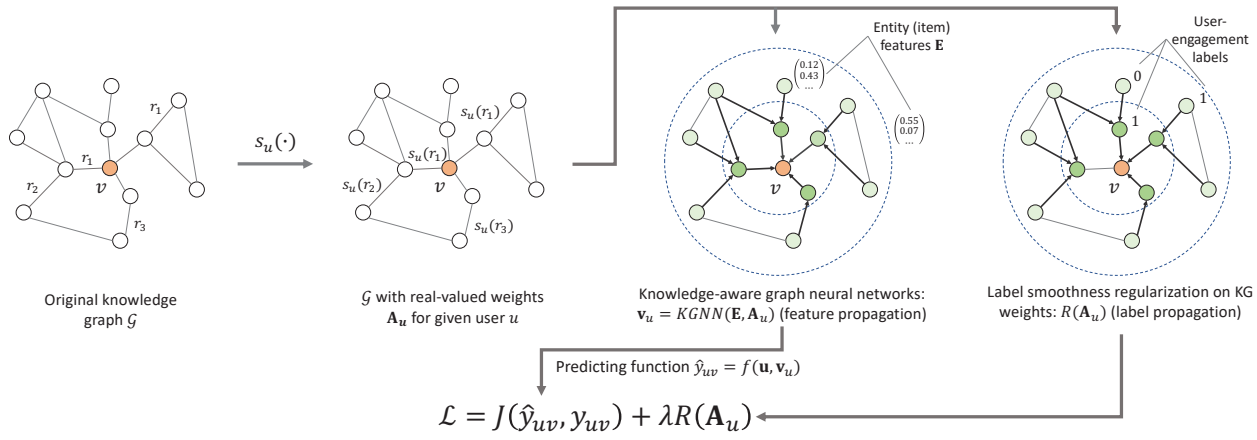


Figure 1: Overview of our proposed KGNN-LS model. The original KG is first transformed into a user-specific weighted graph, on which we then perform feature propagation using a graph neural network with the label smoothness regularization. The two modules constitute the complete loss function $\mathcal{L}$.

example, the triple (*The Silence of the Lambs*, *film.film.star*, *Anthony Hopkins*) states the fact that Anthony Hopkins is the leading actor in film "The Silence of the Lambs". In many recommendation scenarios, an item $v \in \mathcal{V}$ corresponds to an entity $e \in \mathcal{E}$ (e.g., item "The Silence of the Lambs" in MovieLens also appears in the knowledge graph as an entity). The set of entities $\mathcal{E}$ is composed from items $\mathcal{V}$ ($\mathcal{V} \subseteq \mathcal{E}$) as well as non-items $\mathcal{E} \backslash \mathcal{V}$ (e.g. nodes corresponding to item/product properties). Given user-item interaction matrix $\mathbf{Y}$ and knowledge graph $\mathcal{G}$, our task is to predict whether user u has potential interest in item v with which he/she has not engaged before. Specifically, we aim 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 engage with item v, and Θ are model parameters of function $\mathcal{F}$.

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

Symbol	Meaning
$\mathcal{U} = \{u_1, \cdots\}$	Set of users
$\mathcal{V} = \{v_1, \cdots\}$	Set of items
$\mathbf{Y}$	User-item interaction matrix
$\mathcal{G} = (\mathcal{E}, \mathcal{R})$	Knowledge graph
$\mathcal{E} = \{e_1, \cdots\}$	Set of entities
$\mathcal{R} = \{r_1, \cdots\}$	Set of relations
$\mathcal{E} \backslash \mathcal{V}$	Set of non-item entities
$s_u(r)$	User-specific relation scoring function
$\mathbf{A}_u$	Adjacency matrix of $\mathcal{G}$ w.r.t. user u
$\mathbf{D}_u$	Diagonal degree matrix of $\mathbf{A}_u$
$\mathbf{E}$	Raw entity feature
$\mathbf{H}^{(l)},\ l = 0, ..., L-1$	Entity representation in the l-th layer
$\mathbf{W}^{(l)},\ l = 0, ..., L-1$	Transformation matrix in the l-th layer
$l_u(e),\ e \in \mathcal{E}$	Item relevancy labeling function
$l_u^*(e),\ e \in \mathcal{E}$	Minimum-energy labeling function
$\hat{l}_u(v),\ v \in \mathcal{V}$	Predicted relevancy label for item v
$R(\mathbf{A}_u)$	Label smoothness regularization on $\mathbf{A}_u$

Table 1: List of key symbols.

4 OUR APPROACH

In this section, we first introduce knowledge-aware graph neural networks and label smoothness regularization, respectively, then we present the unified model.

4.1 Preliminaries: Knowledge-aware Graph Neural Networks

The first step of our approach is to transform a heterogeneous KG into a user-personalized weighted graph that characterizes user's preferences. To this end, similar to [28], we use a user-specific *relation scoring function* $s_u(r)$ that provides the importance of relation r for user u: $s_u(r) = g(\mathbf{u}, \mathbf{r})$, where $\mathbf{u}$ and $\mathbf{r}$ are feature vectors of user u and relation type r, respectively, and g is a differentiable function such as inner product. Intuitively, $s_u(r)$ characterizes the importance of relation r to user u. For example, a user may be more interested in *directors* of movies, but another user may care more about the *lead actors* of movies.

Given user-specific relation scoring function $s_u(\cdot)$ of user u, knowledge graph $\mathcal{G}$ can therefore be transformed into a user-specific adjacency matrix $\mathbf{A}_u \in \mathbb{R}^{|\mathcal{E}| \times |\mathcal{E}|}$, in which the (i, j)-entry $A_u^{ij} = s_u(r_{e_i, e_j})$, and r_{e_i, e_j} is the relation between entities e_i and e_j in $\mathcal{G}$.[1] $A_u^{ij} = 0$ if there is no relation between e_i and e_j. See the left two subfigures in Figure 1 for illustration. We also denote the raw feature matrix of entities as $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times d_0}$, where d_0 is the dimension of raw entity features. Then we use multiple feed forward layers[2] to update the entity representation matrix by aggregating representations of neighboring entities. Specifically, the layer-wise forward propagation can be expressed as

$$\mathbf{H}_{l+1} = \sigma\left(\mathbf{D}_u^{-1/2}\mathbf{A}_u\mathbf{D}_u^{-1/2}\mathbf{H}_l\mathbf{W}_l\right),\ l = 0, 1, \cdots, L-1. \quad (1)$$

[1] In this work we treat $\mathcal{G}$ an undirected graph, so $\mathbf{A}_u$ is a symmetric matrix. If both triples (h, r_1, t) and (t, r_2, h) exist, we only consider one of r_1 and r_2. This is due to the fact that: (1) r_1 and r_2 are the inverse of each other and semantically related; (2) Treating $\mathbf{A}_u$ symmetric will greatly increase the matrix density.

[2] There are several candidate designs for the architecture of our model, e.g., GCN [11] or GraphSAGE [7]. Here we use GCN [11] as our base model.

In Eq. (1), $\mathbf{H}_l$ is the matrix of hidden representations of entities in layer l, and $\mathbf{H}_0 = \mathbf{E}$. $\mathbf{A}_u$ is to aggregate representation vectors of neighboring entities. In this paper, we set $\mathbf{A}_u \leftarrow \mathbf{A}_u + \mathbf{I}$, i.e., adding self-connection to each entity, to ensure that old representation vector of the entity itself is taken into consideration when updating entity representations. $\mathbf{D}_u$ is a diagonal degree matrix with entries $D_u^{ii} = \sum_j A_u^{ij}$, therefore, $\mathbf{D}_u^{-1/2}$ is used to normalize $\mathbf{A}_u$ and keep the entity representation matrix $\mathbf{H}_l$ stable. $\mathbf{W}_l \in \mathbb{R}^{d_l \times d_{l+1}}$ is the layer-specific trainable weight matrix, σ is a non-linear activation function, and L is the number of layers.

A single GNN layer computes the representation of an entity via a transformed mixture of itself and its immediate neighbors in the KG. We can therefore naturally extend the model to multiple layers to explore users' potential interests in a broader and deeper way. The final output is $\mathbf{H}_L \in \mathbb{R}^{|\mathcal{E}| \times d_L}$, which is the entity representations that mix the initial features of themselves and their neighbors up to L hops away. Finally, the predicted engagement probability of user u with item v is calculated by $\hat{y}_{uv} = f(\mathbf{u}, \mathbf{v}_u)$, where $\mathbf{v}_u$ (i.e., the v-th row of $\mathbf{H}_L$) is the final representation vector of item v, and f is a differentiable prediction function, for example, inner product or a multilayer perceptron. Note that $\mathbf{v}_u$ is user-specific since the adjacency matrix $\mathbf{A}_u$ is user-specific. Furthermore, note that the system is end-to-end trainable where the gradients flow from $f(\cdot)$ via GNN (parameter matrix $\mathbf{W}$) to $g(\cdot)$ and eventually to representations of users u and items v.

4.2 Label Smoothness Regularization

It is worth noticing a significant difference between our model and GNNs: In traditional GNNs, edge weights of the input graph are fixed; but in our model, edge weights $\mathbf{D}_u^{-1/2}\mathbf{A}_u\mathbf{D}_u^{-1/2}$ in Eq. (1) are learnable (including possible parameters of function g and feature vectors of users and relations) and also requires supervised training like $\mathbf{W}$. Though enhancing the fitting ability of the model, this will inevitably make the optimization process prone to overfitting, since the only source of supervised signal is from user-item interactions outside GNN layers. Moreover, edge weights do play an essential role in representation learning on graphs, as highlighted by a large amount of prior works [10, 20, 21, 35, 38]. Therefore, more regularization on edge weights is needed to assist the learning of entity representations and to help generalize to unobserved interactions more efficiently.

Let's see how an ideal set of edge weights should be like. Consider a real-valued label function $l_u : \mathcal{E} \to \mathbb{R}$ on $\mathcal{G}$, which is constrained to take a specific value $l_u(v) = y_{uv}$ at node $v \in \mathcal{V} \subseteq \mathcal{E}$. In our context, $l_u(v) = 1$ if user u finds the item v relevant and has engaged with it, otherwise $l_u(v) = 0$. Intuitively, we hope that adjacent entities in the KG are likely to have similar relevancy labels, which is known as *label smoothness assumption*. This motivates our choice of energy function E:

$$E(l_u, \mathbf{A}_u) = \frac{1}{2} \sum_{e_i \in \mathcal{E}, e_j \in \mathcal{E}} A_u^{ij} \left(l_u(e_i) - l_u(e_j)\right)^2. \tag{2}$$

We show that the minimum-energy label function is *harmonic* by the following theorem:

Theorem 1. *The minimum-energy label function*

$$l_u^* = \underset{l_u : l_u(v) = y_{uv}, \forall v \in \mathcal{V}}{\arg\min} E(l_u, \mathbf{A}_u) \tag{3}$$

w.r.t. Eq. (2) is harmonic, i.e., l_u^* satisfies

$$l_u^*(e_i) = \frac{1}{D_u^{ii}} \sum_{e_j \in \mathcal{E}} A_u^{ij}\, l_u^*(e_j),\ \forall e_i \in \mathcal{E}\backslash\mathcal{V}. \tag{4}$$

Proof. Taking the derivative of the following equation

$$E(l_u, \mathbf{A}_u) = \frac{1}{2} \sum_{i,j} A_u^{ij} \left(l_u(e_i) - l_u(e_j)\right)^2$$

with respect to $l_u(e_i)$ where $e_i \in \mathcal{E}\backslash\mathcal{V}$, we have

$$\frac{\partial E(l_u, \mathbf{A}_u)}{\partial l_u(e_i)} = \sum_j A_u^{ij} \left(l_u(e_i) - l_u(e_j)\right).$$

The minimum-energy label function l_u^* should satisfy that

$$\left.\frac{\partial E(l_u, \mathbf{A}_u)}{\partial l_u(e_i)}\right|_{l_u = l_u^*} = 0.$$

Therefore, we have

$$l_u^*(e_i) = \frac{1}{\sum_j A_u^{ij}} \sum_j A_u^{ij}\, l_u^*(e_j) = \frac{1}{D_u^{ii}} \sum_j A_u^{ij}\, l_u^*(e_j),\ \forall e_i \in \mathcal{E}\backslash\mathcal{V}.$$

□

The harmonic property indicates that the value of l_u^* at each non-item entity $e_i \in \mathcal{E}\backslash\mathcal{V}$ is the average of its neighboring entities, which leads to the following label propagation scheme [39]:

Theorem 2. *Repeating the following two steps:*

(1) Propagate labels for all entities: $l_u(\mathcal{E}) \leftarrow \mathbf{D}_u^{-1}\mathbf{A}_u l_u(\mathcal{E})$, where $l_u(\mathcal{E})$ is the vector of labels for all entities;

(2) Reset labels of all items to initial labels: $l_u(\mathcal{V}) \leftarrow \mathbf{Y}[u, \mathcal{V}]^\top$, where $l_u(\mathcal{V})$ is the vector of labels for all items and $\mathbf{Y}[u, \mathcal{V}] = [y_{uv_1}, y_{uv_2}, \cdots]$ are initial labels;

will lead to $l_u \to l_u^*$.

Proof. Let $l_u(\mathcal{E}) = \begin{bmatrix} l_u(\mathcal{V}) \\ l_u(\mathcal{E}\backslash\mathcal{V}) \end{bmatrix}$. Since $l_u(\mathcal{V})$ is fixed on $\mathbf{Y}[u, \mathcal{V}]$, we are only interested in $l_u(\mathcal{E}\backslash\mathcal{V})$. We denote $\mathbf{P} = \mathbf{D}_u^{-1}\mathbf{A}_u$ (the subscript u is omitted from $\mathbf{P}$ for ease of notation), and partition matrix $\mathbf{P}$ into sub-matrices according to the partition of l_u:

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_{VV} & \mathbf{P}_{VE} \\ \mathbf{P}_{EV} & \mathbf{P}_{EE} \end{bmatrix}.$$

Then the label propagation scheme is equivalent to

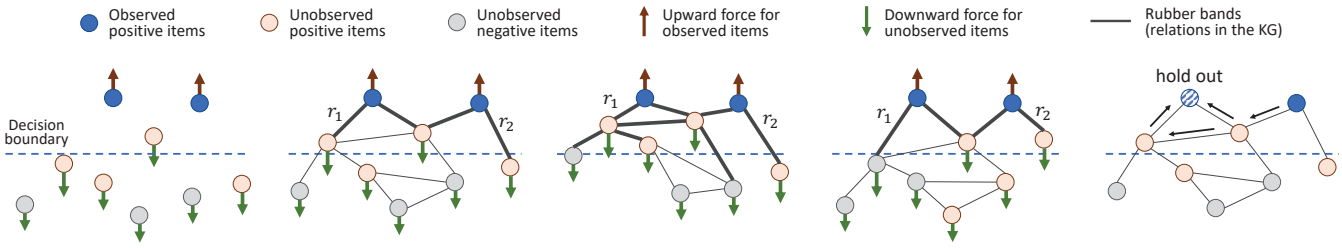
$$l_u(\mathcal{E}\backslash\mathcal{V}) \leftarrow \mathbf{P}_{EV}\mathbf{Y}[u, \mathcal{V}]^\top + \mathbf{P}_{EE} l_u(\mathcal{E}\backslash\mathcal{V}). \tag{5}$$

Repeat the above procedure, we have

$$l_u(\mathcal{E}\backslash\mathcal{V}) = \lim_{n\to\infty} (\mathbf{P}_{EE})^n l_u^{(0)}(\mathcal{E}\backslash\mathcal{V}) + \left(\sum_{i=1}^{n} (\mathbf{P}_{EE})^{i-1}\right) \mathbf{P}_{EV}\mathbf{Y}[u, \mathcal{V}]^\top, \tag{6}$$

where $l_u^{(0)}(\mathcal{E}\backslash\mathcal{V})$ is the initial value for $l_u(\mathcal{E}\backslash\mathcal{V})$. Now we show that $lim_{n\to\infty}(\mathbf{P}_{EE})^n l_u^{(0)}(\mathcal{E}\backslash\mathcal{V}) = \mathbf{0}$. Since $\mathbf{P}$ is row-normalized and $\mathbf{P}_{EE}$ is a sub-matrix of $\mathbf{P}$, we have

$$\exists \epsilon < 1,\ \sum_j \mathbf{P}_{EE}[i, j] \le \epsilon,$$



(a) Without the KG **(b)** $L = 1$ **(c)** $L = 2$ **(d)** $L = 1$ **for another user** **(e) LS regularization**

Figure 2: (a) Analogy of a physical equilibrium model for recommender systems; (b)-(d) Illustration of the effect of the KG; (e) Illustration of the effect of label smoothness regularization.

for all possible row index i. Therefore,

$$\begin{aligned}\sum_j (\mathbf{P}_{EE})^n[i,j] &= \sum_j \left((\mathbf{P}_{EE})^{(n-1)}\mathbf{P}_{EE}\right)[i,j] \\ &= \sum_j \sum_k (\mathbf{P}_{EE})^{(n-1)}[i,k]\,\mathbf{P}_{EE}[k,j] \\ &= \sum_k (\mathbf{P}_{EE})^{(n-1)}[i,k] \sum_j \mathbf{P}_{EE}[k,j] \\ &\le \sum_k (\mathbf{P}_{EE})^{(n-1)}[i,k]\,\epsilon \\ &\le \cdots \le \epsilon^n.\end{aligned}$$

As n goes infinity, the row sum of $(\mathbf{P}_{EE})^n$ converges to zero, which implies that $(\mathbf{P}_{EE})^n l_u^{(0)}(\mathcal{E}\backslash\mathcal{V}) \to \mathbf{0}$. It's clear that the choice of initial value $l_u^{(0)}(\mathcal{E}\backslash\mathcal{V})$ does not affect the convergence.

Since $lim_{n\to\infty}(\mathbf{P}_{EE})^n l_u^{(0)}(\mathcal{E}\backslash\mathcal{V}) = \mathbf{0}$, Eq. (6) becomes

$$l_u(\mathcal{E}\backslash\mathcal{V}) = \lim_{n\to\infty}\left(\sum_{i=1}^{n}(\mathbf{P}_{EE})^{i-1}\right)\mathbf{P}_{EV}\mathbf{Y}[u,\mathcal{V}]^\top.$$

Denote

$$\mathbf{T} = \lim_{n\to\infty}\sum_{i=1}^{n}(\mathbf{P}_{EE})^{i-1} = \sum_{i=1}^{\infty}(\mathbf{P}_{EE})^{i-1},$$

and we have

$$\mathbf{T} - \mathbf{T}\mathbf{P}_{EE} = \sum_{i=1}^{\infty}(\mathbf{P}_{EE})^{i-1} - \sum_{i=1}^{\infty}(\mathbf{P}_{EE})^{i} = \mathbf{I}.$$

Therefore, we derive that

$$\mathbf{T} = (I - \mathbf{P}_{EE})^{-1},$$

and

$$l_u(\mathcal{E}\backslash\mathcal{V}) = (I - \mathbf{P}_{EE})^{-1}\mathbf{P}_{EV}\mathbf{Y}[u,\mathcal{V}]^\top.$$

This is the unique fixed point and therefore the unique solution to Eq. (5). Repeating the steps in Theorem 2 leads to

$$l_u(\mathcal{E}) \to l_u^*(\mathcal{E}) = \begin{bmatrix}\mathbf{Y}[u,\mathcal{V}]^\top \\ (I - \mathbf{P}_{EE})^{-1}\mathbf{P}_{EV}\mathbf{Y}[u,\mathcal{V}]^\top\end{bmatrix}.$$

□

Theorem 2 provides a way for reaching the minimum-energy of relevancy label function E. However, l_u^* does not provide any signal for updating the edge weights matrix $\mathbf{A}_u$, since the labeled part of l_u^*, i.e., $l_u^*(\mathcal{V})$, equals their true relevancy labels $\mathbf{Y}[u,\mathcal{V}]$; Moreover, we do not know true relevancy labels for the unlabeled nodes $l_u^*(\mathcal{E}\backslash\mathcal{V})$.

To solve the issue, we propose minimizing the *leave-one-out* loss [35]. Suppose we hold out a single item v and treat it unlabeled. Then we predict its label by using the rest of (labeled) items and (unlabeled) non-item entities. The prediction process is identical to label propagation in Theorem 2, except that the label of item v is hidden and needs to be calculated. This way, the difference between the true relevancy label of v (i.e., y_{uv}) and the predicted label $\hat{l}_u(v)$ serves as a supervised signal for regularizing edge weights:

$$R(\mathbf{A}) = \sum_u R(\mathbf{A}_u) = \sum_u \sum_v J\big(y_{uv}, \hat{l}_u(v)\big), \tag{7}$$

where J is the cross-entropy loss function. Given the regularization in Eq. (7), an ideal edge weight matrix $\mathbf{A}$ should reproduce the true relevancy label of each held-out item while also satisfying the smoothness of relevancy labels.

4.3 The Unified Loss Function

Combining knowledge-aware graph neural networks and LS regularization, we reach the following complete loss function:

$$\min_{\mathbf{W},\mathbf{A}} \mathcal{L} = \min_{\mathbf{W},\mathbf{A}} \sum_{u,v} J(y_{uv}, \hat{y}_{uv}) + \lambda R(\mathbf{A}) + \gamma\|\mathcal{F}\|_2^2, \tag{8}$$

where $\|\mathcal{F}\|_2^2$ is the L2-regularizer, λ and γ are balancing hyper-parameters. In Eq. (8), the first term corresponds to the part of GNN that learns the transformation matrix $\mathbf{W}$ and edge weights $\mathbf{A}$ simultaneously, while the second term $R(\cdot)$ corresponds to the part of label smoothness that can be seen as adding constraint on edge weights $\mathbf{A}$. Therefore, $R(\cdot)$ serves as regularization on $\mathbf{A}$ to assist GNN in learning edge weights.

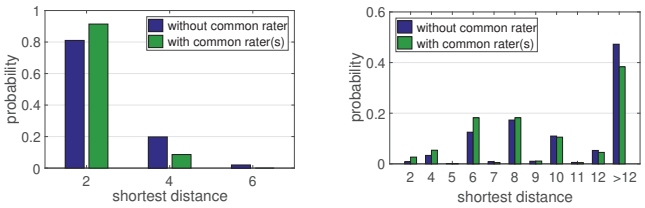
It is also worth noticing that the first term can be seen as *feature propagation* on the KG while the second term $R(\cdot)$ can be seen as *label propagation* on the KG. A recommender for a specific user u is actually a mapping from item features to user-item interaction labels, i.e., $\mathcal{F}_u : \mathbf{E}_v \to y_{uv}$ where $\mathbf{E}_v$ is the feature vector of item v. Therefore, Eq. (8) utilizes the structural information of the KG on both the feature side and the label side of $\mathcal{F}_u$ to capture users' higher-order preferences.

4.4 Discussion

How can the knowledge graph help find users' interests? To intuitively understand the role of the KG, we make an analogy with a

	Movie	Book	Music	Restaurant
# users	138,159	19,676	1,872	2,298,698
# items	16,954	20,003	3,846	1,362
# interactions	13,501,622	172,576	42,346	23,416,418
# entities	102,569	25,787	9,366	28,115
# relations	32	18	60	7
# KG triples	499,474	60,787	15,518	160,519

Table 2: Statistics of the four datasets: MovieLens-20M (movie), Book-Crossing (book), Last.FM (music), and Dianping-Food (restaurant).



(a) MovieLens-20M **(b) Last.FM**

Figure 3: Probability distribution of the shortest path distance between two randomly sampled items in the KG under the circumstance that (1) they have no common user in the dataset; (2) they have common user(s) in the dataset.

physical equilibrium model as shown in Figure 2. Each entity/item is seen as a particle, while the supervised positive user-relevancy signal acts as the force pulling the observed positive items up from the decision boundary and the negative items signal acts as the force pushing the unobserved items down. Without the KG (Figure 2a), these items are only loosely connected with each other through the collaborative filtering effect (which is not drawn here for clarity). In contrast, edges in the KG serve as the rubber bands that impose explicit constraints on connected entities. When number of layers is $L = 1$ (Figure 2b), representation of each entity is a mixture of itself and its immediate neighbors, therefore, optimizing on the positive items will simultaneously pull their immediate neighbors up together. The upward force goes deeper in the KG with the increase of L (Figure 2c), which helps explore users' long-distance interests and pull up more positive items. It is also interesting to note that the proximity constraint exerted by the KG is *personalized* since the strength of the rubber band (i.e., $s_u(r)$) is *user-specific* and *relation-specific*: One user may prefer relation r_1 (Figure 2b) while another user (with same observed items but different unobserved items) may prefer relation r_2 (Figure 2d).

Despite the force exerted by edges in the KG, edge weights may be set inappropriately, for example, too small to pull up the unobserved items (i.e., rubber bands are too weak). Next, we show by Figure 2e that how the label smoothness assumption helps regularizing the learning of edge weights. Suppose we hold out the positive sample in the upper left and we intend to reproduce its label by the rest of items. Since the true relevancy label of the held-out sample is 1 and the upper right sample has the largest label value, the LS regularization term $R(\mathbf{A})$ would enforce the edges with arrows to be large so that the label can "flow" from the blue one to the striped one as much as possible. As a result, this will tighten the rubber bands (denoted by arrows) and encourage the model to pull up the two upper pink items to a greater extent.

5 EXPERIMENTS

In this section, we evaluate the proposed KGNN-LS model, and present its performance on four real-world scenarios: movie, book, music, and restaurant recommendations.

5.1 Datasets

We utilize the following four datasets in our experiments for movie, book, music, and restaurant recommendations, respectively, in which the first three are public datasets and the last one is from Meituan-Dianping Group. We use Satori[3], a commercial KG built by Microsoft, to construct sub-KGs for MovieLens-20M, Book-Crossing, and Last.FM datasets. The KG for Dianping-Food dataset is constructed by the internal toolkit of Meituan-Dianping Group. Further details of datasets are provided in Appendix A.

- **MovieLens-20M**[4] 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. The corresponding KG contains 102,569 entities, 499,474 edges and 32 relation-types.
- **Book-Crossing**[5] contains 1 million ratings (ranging from 0 to 10) of books in the Book-Crossing community. The corresponding KG contains 25,787 entities, 60,787 edges and 18 relation-types.
- **Last.FM**[6] contains musician listening information from a set of 2 thousand users from Last.fm online music system. The corresponding KG contains 9,366 entities, 15,518 edges and 60 relation-types.
- **Dianping-Food** is provided by Dianping.com[7], which contains over 10 million interactions (including clicking, buying, and adding to favorites) between approximately 2 million users and 1 thousand restaurants. The corresponding KG contains 28,115 entities, 160,519 edges and 7 relation-types.

The statistics of the four datasets are shown in Table 2.

5.2 Baselines

We compare the proposed KGNN-LS model with the following baselines for recommender systems, in which the first two baselines are KG-free while the rest are all KG-aware methods. The hyper-parameter setting of KGNN-LS is provided in Appendix B.

- **SVD** [12] is a classic CF-based model using inner product to model user-item interactions. We use the unbiased version (i.e., the predicted engaging probability is modeled as $y_{uv} = \mathbf{u}^\top \mathbf{v}$). The dimension and learning rate for the four datasets are set as: $d = 8$, $\eta = 0.5$ for MovieLens-20M, Book-Crossing; $d = 8$, $\eta = 0.1$ for Last.FM; $d = 32$, $\eta = 0.1$ for Dianping-Food.

[3] https://searchengineland.com/library/bing/bing-satori
[4] https://grouplens.org/datasets/movielens/
[5] http://www2.informatik.uni-freiburg.de/~cziegler/BX/
[6] https://grouplens.org/datasets/hetrec-2011/
[7] https://www.dianping.com/

Model	MovieLens-20M				Book-Crossing				Last.FM				Dianping-Food			
	R@2	*R@10*	*R@50*	*R@100*	*R@2*	*R@10*	*R@50*	*R@100*	*R@2*	*R@10*	*R@50*	*R@100*	*R@2*	*R@10*	*R@50*	*R@100*
SVD	0.036	0.124	0.277	0.401	0.027	0.046	0.077	0.109	0.029	0.098	0.240	0.332	0.039	0.152	0.329	0.451
LibFM	0.039	0.121	0.271	0.388	0.033	0.062	0.092	0.124	0.030	0.103	0.263	0.330	0.043	0.156	0.332	0.448
LibFM + TransE	0.041	0.125	0.280	0.396	0.037	0.064	0.097	0.130	0.032	0.102	0.259	0.326	0.044	0.161	**0.343**	0.455
PER	0.022	0.077	0.160	0.243	0.022	0.041	0.064	0.070	0.014	0.052	0.116	0.176	0.023	0.102	0.256	0.354
CKE	0.034	0.107	0.244	0.322	0.028	0.051	0.079	0.112	0.023	0.070	0.180	0.296	0.034	0.138	0.305	0.437
RippleNet	**0.045**	0.130	0.278	0.447	0.036	0.074	0.107	0.127	0.032	0.101	0.242	0.336	0.040	0.155	0.328	0.440
KGNN-LS	0.043	**0.155**	**0.321**	**0.458**	**0.045**	**0.082**	**0.117**	**0.149**	**0.044**	**0.122**	**0.277**	**0.370**	**0.047**	**0.170**	0.340	**0.487**

Table 3: The results of *Recall@K* in top-K recommendation.

Model	Movie	Book	Music	Restaurant
SVD	0.963	0.672	0.769	0.838
LibFM	0.959	0.691	0.778	0.837
LibFM + TransE	0.966	0.698	0.777	0.839
PER	0.832	0.617	0.633	0.746
CKE	0.924	0.677	0.744	0.802
RippleNet	0.960	0.727	0.770	0.833
KGNN-LS	**0.979**	**0.744**	**0.803**	**0.850**

Table 4: The results of *AUC* in CTR prediction.

- **LibFM** [16] is a widely used feature-based factorization model for CTR prediction. We concatenate user ID and item ID as input for LibFM. The dimension is set as $\{1, 1, 8\}$ and the number of training epochs is 50 for all datasets.
- **LibFM + TransE** extends LibFM by attaching an entity representation learned by TransE [2] to each user-item pair. The dimension of TransE is 32 for all datasets.
- **PER** [33] is a representative of path-based methods, which treats the KG as heterogeneous information networks and extracts meta-path based features to represent the connectivity between users and items. We use manually designed "user-item-attribute-item" as meta-paths, i.e., "user-movie-director-movie", "user-movie-genre-movie", and "user-movie-star-movie" for MovieLens-20M; "user-book-author-book" and "user-book-genre-book" for Book-Crossing, "user-musician-date_of_birth-musician" (date of birth is discretized), "user-musician-country-musician", and "user-musician-genre-musician" for Last.FM; "user-restaurant-dish-restaurant", "user-restaurant-business_area-restaurant", "user-restaurant-tag-restaurant" for Dianping-Food. The settings of dimension and learning rate are the same as SVD.
- **CKE** [34] is a representative of embedding-based methods, which combines CF with structural, textual, and visual knowledge in a unified framework. We implement CKE as CF plus a structural knowledge module in this paper. The dimension of embedding for the four datasets are 64, 128, 64, 64. The training weight for KG part is 0.1 for all datasets. The learning rate are the same as in SVD.
- **RippleNet** [24] is a representative of hybrid methods, which is a memory-network-like approach that propagates users' preferences on the KG for recommendation. For RippleNet, $d = 8, H = 2, \lambda_1 = 10^{-6}, \lambda_2 = 0.01, \eta = 0.01$ for MovieLens-20M; $d = 16, H = 3, \lambda_1 = 10^{-5}, \lambda_2 = 0.02, \eta = 0.005$ for Last.FM; $d = 32, H = 2, \lambda_1 = 10^{-7}, \lambda_2 = 0.02, \eta = 0.01$ for Dianping-Food.

5.3 Validating the Connection between $\mathcal{G}$ and Y

To validate the connection between the knowledge graph $\mathcal{G}$ and user-item interaction Y, we conduct an empirical study where we investigate the correlation between the shortest path distance of two randomly sampled items in the KG and whether they have common user(s) in the dataset, that is there exist user(s) that interacted with both items. For MovieLens-20M and Last.FM, we randomly sample ten thousand item pairs that have no common users and have at least one common user, respectively, then count the distribution of their shortest path distances in the KG. The results are presented in Figure 3, which clearly show that *if two items have common user(s) in the dataset, they are likely to be more close in the KG*. For example, if two movies have common user(s) in MovieLens-20M, there is a probability of 0.92 that they will be within 2 hops in the KG, while the probability is 0.80 if they have no common user. This finding empirically demonstrates that exploiting the proximity structure of the KG can assist making recommendations. This also justifies our motivation to use label smoothness regularization to help learn entity representations.

5.4 Results

5.4.1 Comparison with Baselines. We evaluate our method in two experiment scenarios: (1) In top-K recommendation, we use the trained model to select K items with highest predicted click probability for each user in the test set, and choose *Recall@K* to evaluate the recommended sets. (2) In click-through rate (CTR) prediction, we apply the trained model to predict each piece of user-item pair in the test set (including positive items and randomly selected negative items). We use *AUC* as the evaluation metric in CTR prediction.

The results of top-K recommendation and CTR prediction are presented in Tables 3 and 4, respectively, which show that KGNN-LS outperforms baselines by a significant margin. For example, the *AUC* of KGNN-LS surpasses baselines by 5.1%, 6.9%, 8.3%, and 4.3% on average in MovieLens-20M, Book-Crossing, Last.FM, and Dianping-Food datasets, respectively.

We also show daily performance of KGNN-LS and baselines on Dianping-Food to investigate performance stability. Figure 4 shows their *AUC* score from September 1, 2018 to September 30, 2018. We notice that the curve of KGNN-LS is consistently above baselines over the test period; Moreover, the performance of KGNN-LS is also

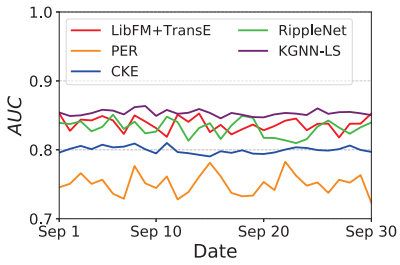


Figure 4: Daily *AUC* of all methods on Dianping-Food in September 2018.

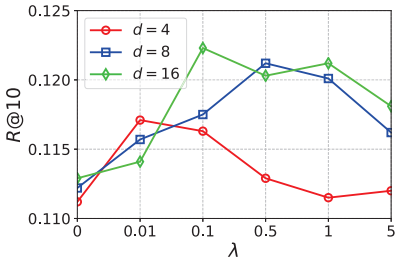


Figure 5: Effectiveness of LS regularization on Last.FM.

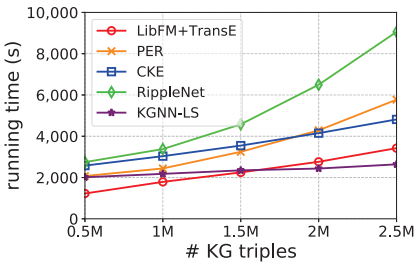


Figure 6: Running time of all methods w.r.t. KG size on MovieLens-20M.

r	20%	40%	60%	80%	100%
SVD	0.882	0.913	0.938	0.955	0.963
LibFM	0.902	0.923	0.938	0.950	0.959
LibFM+TransE	0.914	0.935	0.949	0.960	0.966
PER	0.802	0.814	0.821	0.828	0.832
CKE	0.898	0.910	0.916	0.921	0.924
RippleNet	0.921	0.937	0.947	0.955	0.960
KGNN-LS	**0.961**	**0.970**	**0.974**	**0.977**	**0.979**

Table 5: *AUC* of all methods w.r.t. the ratio of training set *r*.

with low variance, which suggests that KGNN-LS is also robust and stable in practice.

5.4.2 Effectiveness of LS Regularization. Is the proposed LS regularization helpful in improving the performance of GNN? To study the effectiveness of LS regularization, we fix the dimension of hidden layers as 4, 8, and 16, then vary λ from 0 to 5 to see how performance changes. The results of $R@10$ in Last.FM dataset are plotted in Figure 5. It is clear that the performance of KGNN-LS with a non-zero λ is better than $\lambda = 0$ (the case of Wang et al. [28]), which justifies our claim that LS regularization can assist learning the edge weights in a KG and achieve better generalization in recommender systems. But note that a too large λ is less favorable, since it overwhelms the overall loss and misleads the direction of gradients. According to the experiment results, we find that a λ between 0.1 and 1.0 is preferable in most cases.

5.4.3 Results in cold-start scenarios. One major goal of using KGs in recommender systems is to alleviate the sparsity issue. To investigate the performance of KGNN-LS in cold-start scenarios, we vary the size of training set of MovieLens-20M from $r = 100\%$ to $r = 20\%$ (while the validation and test set are kept fixed), and report the results of *AUC* in Table 5. When $r = 20\%$, *AUC* decreases by 8.4%, 5.9%, 5.4%, 3.6%, 2.8%, and 4.1% for the six baselines compared to the model trained on full training data ($r = 100\%$), but the performance decrease of KGNN-LS is only 1.8%. This demonstrates that KGNN-LS still maintains predictive performance even when user-item interactions are sparse.

5.4.4 Hyper-parameters Sensitivity. We first analyze the sensitivity of KGNN-LS to the number of GNN layers L. We vary L from 1 to 4 while keeping other hyper-parameters fixed. The results are shown in Table 6. We find that the model performs poorly when $L = 4$, which is because a larger L will mix too many entity embeddings

L	1	2	3	4
MovieLens-20M	**0.155**	0.146	0.122	0.011
Book-Crossing	0.077	**0.082**	0.043	0.008
Last.FM	**0.122**	0.106	0.105	0.057
Dianping-Food	0.165	**0.170**	0.061	0.036

Table 6: $R@10$ w.r.t. the number of layers L.

d	4	8	16	32	64	128
MovieLens-20M	0.134	0.141	0.143	**0.155**	**0.155**	0.151
Book-Crossing	0.065	0.073	0.077	0.081	**0.082**	0.080
Last.FM	0.111	0.116	**0.122**	0.109	0.102	0.107
Dianping-Food	0.155	**0.170**	0.167	0.166	0.163	0.161

Table 7: $R@10$ w.r.t. the dimension of hidden layers d.

in a given entity, which *over-smoothes* the representation learning on KGs. KGNN-LS achieves the best performance when $L = 1$ or 2 in the four datasets.

We also examine the impact of the dimension of hidden layers d on the performance of KGNN-LS. The result in shown in Table 7. We observe that the performance is boosted with the increase of d at the beginning, because more bits in hidden layers can improve the model capacity. However, the performance drops when d further increases, since a too large dimension may overfit datasets. The best performance is achieved when $d = 8 \sim 64$.

5.5 Running Time Analysis

We also investigate the running time of our method with respect to the size of KG. We run experiments on a Microsoft Azure virtual machine with 1 NVIDIA Tesla M60 GPU, 12 Intel Xeon CPUs (E5-2690 v3 @2.60GHz), and 128GB of RAM. The size of the KG is increased by up to five times the original one by extracting more triples from Satori, and the running times of all methods on MovieLens-20M are reported in Figure 6. Note that the trend of a curve matters more than the real values, since the values are largely dependent on the minibatch size and the number of epochs (yet we did try to align the configurations of all methods). The result show that KGNN-LS exhibits strong scalability even when the KG is large.

6 CONCLUSION AND FUTURE WORK

In this paper, we propose knowledge-aware graph neural networks with label smoothness regularization for recommendation. KGNN-LS applies GNN architecture to KGs by using user-specific relation

scoring functions and aggregating neighborhood information with different weights. In addition, the proposed label smoothness constraint and leave-one-out loss provide strong regularization for learning the edge weights in KGs. We also discuss how KGs benefit recommender systems and how label smoothness can assist learning the edge weights. Experiment results show that KGNN-LS outperforms state-of-the-art baselines in four recommendation scenarios and achieves desirable scalability with respect to KG size.

In this paper, LS regularization is proposed for recommendation task with KGs. It is interesting to examine the LS assumption on other graph tasks such as link prediction and node classification. Investigating the theoretical relationship between feature propagation and label propagation is also a promising direction.

Acknowledgements. This research has been supported in part by NSF OAC-1835598, DARPA MCS, ARO MURI, Boeing, Docomo, Hitachi, Huawei, JD, Siemens, and Stanford Data Science Initiative.

REFERENCES

[1] Shumeet Baluja, Rohan Seth, D Sivakumar, Yushi Jing, Jay Yagnik, Shankar Kumar, Deepak Ravichandran, and Mohamed Aly. 2008. Video suggestion and discovery for youtube: taking random walks through the view graph. In *Proceedings of the 17th international conference on World Wide Web*. ACM, 895–904.

[2] 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.

[3] 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*.

[4] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. ACM, 191–198.

[5] 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.

[6] 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.

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

[8] 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 *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1531–1540.

[9] 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.

[10] Masayuki Karasuyama and Hiroshi Mamitsuka. 2013. Manifold-based similarity adaptation for label propagation. In *Advances in neural information processing systems*. 1547–1555.

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

[12] 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.

[13] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.

[14] Federico Monti, Michael Bronstein, and Xavier Bresson. 2017. Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*. 3697–3707.

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

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

[17] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*. Springer, 593–607.

[18] 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.

[19] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph Convolutional Matrix Completion. *stat* 1050 (2017), 7.

[20] 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*.

[21] Fei Wang and Changshui Zhang. 2008. Label propagation through linear neighborhoods. *IEEE Transactions on Knowledge and Data Engineering* 20, 1 (2008), 55–67.

[22] 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 on Conference on Information and Knowledge Management*. ACM, 347–356.

[23] 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 Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 592–600.

[24] 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 27th ACM International Conference on Information and Knowledge Management*. ACM, 417–426.

[25] 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 Transactions on Information Systems (TOIS)* 37, 3 (2019), 32.

[26] 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*. 1835–1844.

[27] Hongwei Wang, Fuzheng Zhang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. 2019. Multi-Task Feature Learning for Knowledge Graph Enhanced Recommendation. In *Proceedings of the 2019 World Wide Web Conference on World Wide Web*.

[28] 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*.

[29] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 839–848.

[30] 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.

[31] Yuexin Wu, Hanxiao Liu, and Yiming Yang. 2018. Graph Convolutional Matrix Completion for Bipartite Edge Prediction. In *the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*.

[32] 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*. 974–983.

[33] 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*. ACM, 283–292.

[34] 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.

[35] Xinhua Zhang and Wee S Lee. 2007. Hyperparameter learning for graph based semi-supervised learning algorithms. In *Advances in neural information processing systems*. 1585–1592.

[36] 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.

[37] Dengyong Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. 2004. Learning with local and global consistency. In *Advances in neural information processing systems*. 321–328.

[38] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning*. 912–919.

[39] Xiaojin Zhu, John Lafferty, and Ronald Rosenfeld. 2005. *Semi-supervised learning with graphs*. Ph.D. Dissertation. Carnegie Mellon University, language technologies institute, school of computer science.

APPENDIX

A Additional Details on Datasets

MovieLens-20M, Book-Crossing, and Last.FM datasets contain explicit feedbacks data (Last.FM provides the listening count as weight for each user-item interaction). Therefore, we transform them into implicit feedback, where each entry is marked with 1 indicating that the user has rated the item positively. The threshold of positive rating is 4 for MovieLens-20M, while no threshold is set for Book-Crossing and Last.FM due to their data sparsity. Additionally, we randomly sample an unwatched set of items and mark them as 0 for each user, the number of which equals his/her positively-rated ones.

We use Microsoft Satori to construct the KGs for MovieLens-20M, Book-Crossing, and Last.FM datasets. In each triple of Satori KG, the head and the tail are either IDs or textual content, and the relation follows the format "domain.head_category.tail_category" (e.g., "book.book.author"). We first select a subset of triples from the whole Satori KG with a confidence level greater than 0.9. Then we collect Satori IDs of all valid movies/books/musicians by matching their names with the tail of triples *(head, film.film.name, tail)*, *(head, book.book.title, tail)*, or *(head, type.object.name, tail)*, respectively, for the three datasets. Items with multiple matched or no matched entities are excluded for simplicity. After obtaining the set of item IDs, we match these item IDs with the head of all triples in Satori sub-KG, and select all well-matched triples as the final KG for each dataset.

Dianping-Food dataset is collected from Dianping.com, a Chinese group buying website hosting consumer reviews of restaurants similar to Yelp. We select approximately 10 million interactions between users and restaurants in Dianping.com from May 1, 2015 to December 12, 2018. The types of positive interactions include clicking, buying, and adding to favorites, and we sample negative interactions for each user. The KG for Dianping-Food is collected from Meituan Brain, an internal knowledge graph built for dining and entertainment by Meituan-Dianping Group. The types of entities include POI (restaurant), city, first-level and second-level category, star, business area, dish, and tag; The types of relations correspond to the types of entities (e.g., "organization.POI.has_dish").

	Movie	Book	Music	Restaurant
S	16	8	8	4
d	32	64	16	8
L	1	2	1	2
λ	1.0	0.5	0.1	0.5
γ	10^{-7}	2×10^{-5}	10^{-4}	10^{-7}
η	2×10^{-2}	2×10^{-4}	5×10^{-4}	2×10^{-2}

Table 8: Hyper-parameter settings for the four datasets (S: number of sampled neighbors for each entity; d: dimension of hidden layers, L: number of layers, λ: label smoothness regularizer weight, γ: L2 regularizer weight, η: learning rate).

B Additional Details on Hyper-parameter Searching

In KGNN-LS, we set functions g and f as inner product, σ as *ReLU* for non-last-layers and *tanh* for the last-layer. Note that the number of neighbors of an entity in a KG may be significantly different from each other. Therefore, we uniformly sample a fixed-size set of neighbors for each entity instead of using all of its neighbors to keep the computation more efficient. The number of sampled neighbors for each entity is denoted by S. Hyper-parameter settings for the four datasets are given in Table 8, which are determined by optimizing R@10 on a validation set. The search spaces for hyper-parameters are as follows:

- $S = \{2, 4, 8, 16, 32\}$;
- $d = \{4, 8, 16, 32, 64, 128\}$;
- $L = \{1, 2, 3, 4\}$;
- $\lambda = \{0, 0.01, 0.1, 0.5, 1, 5\}$;
- $\gamma = \{10^{-9}, 10^{-8}, 10^{-7}, 2 \times 10^{-7}, 5 \times 10^{-7}, 10^{-6}, 2 \times 10^{-6}, 5 \times 10^{-6}, 10^{-5}, 2 \times 10^{-5}, 5 \times 10^{-5}, 10^{-4}, 2 \times 10^{-4}, 5 \times 10^{-4}, 10^{-3}\}$;
- $\eta = \{10^{-5}, 2 \times 10^{-5}, 5 \times 10^{-5}, 10^{-4}, 2 \times 10^{-4}, 5 \times 10^{-4}, 10^{-3}, 2 \times 10^{-3}, 5 \times 10^{-3}, 10^{-2}, 2 \times 10^{-2}, 5 \times 10^{-2}, 10^{-1}\}$.

For each dataset, the ratio of training, validation, and test set is $6 : 2 : 2$. Each experiment is repeated 5 times, and the average performance is reported. All trainable parameters are optimized by Adam algorithm.