Reviewing paper is on the left, original paper is on the right

ATRank

https://arxiv.org/pdf/1711.06632.pdf

the procedure can be performed iteratively on S_i^j for $j = 1, \ldots, M$. Therefore, a user's preference on the item i is propagated up to M hops, and we observe multiple responses according to the item i with different orders: $\mathbf{KG}_i^1, \mathbf{KG}_i^2, \ldots, \mathbf{KG}_i^M$. The item i with respect to item v is calculated by concatenating the responses of all orders:

$$\mathbf{o}_i = \mathbf{P} \cdot \text{concat}(\mathbf{i}, \mathbf{KG}_i^1, \mathbf{KG}_i^2, \ldots, \mathbf{KG}_i^M),$$

where $\mathbf{P}$ is the projection matrix. Multi-level vector $\mathbf{o}_i$ can be seen as the response of the user's bought item i with respect to the target item v. And all of the user's history behavior is denoted as $\mathbf{O} = [\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_{N_s}]$, where N_s is the length of the session sequence.

As we have mentioned above, all the KGR methods can be incorporated into our method. The only difference is the attention score defined in (2). For LFM, the attention score is defined as

$$a_j = \text{softmax}(\mathbf{h}_j^T \mathbf{R}_j \mathbf{v}) = \frac{\mathbf{h}_j^T \mathbf{R}_j \mathbf{v}}{\sum_{(h,r,t) \in S_i^1} \exp(\mathbf{h}^T \mathbf{R} \mathbf{v})},$$

and neutral network is

$$a_j = \mathbf{t}^T \mathcal{M}^L(\text{concat}(\mathbf{h}_j, \mathbf{r}_j)),$$

where $\mathcal{M}^L(\cdot) = \mathcal{M}(\mathcal{M}(\cdots \mathcal{M}(\cdot)))$, and $\mathcal{M}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$ is a fully-connected neural network layer with weight $\mathbf{W}$, bias $\mathbf{b}$, and nonlinear activation function $\sigma(\cdot)$.

In the following section, we will use the self-attention mechanism to exploit the interest evolution making use of $\mathbf{O}$.

4.2 Interest Evolving Layer

As the joint influence from external environment and internal cognition, users' interests are evolving over time. With the help of the interest extracting layer, we have obtained the users' interest sequence. Many state of the art approaches (e.g. CNN/RNN) can be employed to catch the evolution of the user's interests. In this paper, we use self-attention mechanism to catch the evolution of the users' interests. The advantages are as follows:

- The intense of each behavior could be affected by others, such that attentions for some behaviors could be distracted and the others could be strengthened. This can be catched by the self-attention mechanism.
- By positional encoding, we can make use of the order of the sequence.

The outputs of this layer could be regarded as the interest representation sequence taking considerations of the impacts of the others. In the following subsection, we will introduce the interest evolving layer in detail.

insert a multi-head attention layer over the output of the encoder.

The queries, keys and values are packed into matrices $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$, respectively. The attention matrix is calculated as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}})\mathbf{V}.$$

The multi-head attention is to linearly project the $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ h times with different, learned linear projection. And then perform the attention function on each of the projected $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ in parallel.

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\mathbf{head}_1, \mathbf{head}_2 \ldots, \mathbf{head}_h)\mathbf{W},$$

where $\mathbf{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V)$.

4.2.2 Feed-forward networks. Each layer in the encoder and decoder contains a fully connected feed-forward network, which consists of two linear transformations with a ReLU activation in between.

$$\text{FFN}(\mathbf{x}) = \mathbf{W}_2 \cdot \max(\mathbf{0}, \mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2.$$

We note here that the parameters are different from layer to layer.

4.2.3 Positional Encoding. Since the self-attention mechanism contains no recurrence and no convolution, in order to make use of the order of the sequences, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, we add "positional encodings" to the input embeddings. There are many choices of positional encodings, learned and fixed. In this work, Each item's bought time is represented as a vector of one-hot encoding, and is concated to the interest vector $\mathbf{o}_i$.

4.2.4 Applications of attention in our model. In the encoder layer, the queries, keys, and values come from the interest extract layer, i.e. $\mathbf{Q} = \mathbf{K} = \mathbf{V} = \mathbf{O}^T$. Then take the transposition of the outputs as the inputs of the feed-forward networks.

In the decoder layer, there are two attention layers. The first one is masked multi-head attention layer, where $\mathbf{Q} = \mathbf{K} = \mathbf{V} = \mathbf{v}^T$. The other one is multi-head attention layer, whose queries and keys come form the encoder, and keys come form the masked multi-head attention layer. At last, a feed-forward network $\mathcal{F}$ is connected, whose inputs are from the attention layer. The output of $\mathcal{F}$ keeps the same shape as the input.

And we set the downstream application network to be a point-wise or a pair-wise fully connected neural network to produce the final user embedding $\mathbf{u}$.

https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf?source=techstories.org

Apps Bookmarks stack Other Bookmarks

In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (2)$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1. The dimensionality of input and output is $d_{\text{model}} = 512$, and the inner-layer has dimensionality $d_{ff} = 2048$.

3.4 Embeddings and Softmax

Similarly to other sequence transduction models, we use learned embeddings to convert the input tokens and output tokens to vectors of dimension d_{model}. We also use the usual learned linear transformation and softmax function to convert the decoder output to predicted next-token probabilities. In our model, we share the same weight matrix between the two embedding layers and the pre-softmax linear transformation, similar to [24]. In the embedding layers, we multiply those weights by $\sqrt{d_{\text{model}}}$.

3.5 Positional Encoding

Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, we add "positional encodings" to the input embeddings at the

5

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed [8].

In this work, we use sine and cosine functions of different frequencies:

RippleNet

https://arxiv.org/pdf/1803.03467.pdf

An Attention and Knowledge Graph Network for Session-Based Recommendation System

.
.
.

ABSTRACT

The session-based recommendation system (SBRS) aims to predict users' next actions based on anonymous sessions. For SBRS, capturing the latent user interests behind the behaviors and the evolution of user interests are of great importance. To achieve these goals, some works make use of side information, such as knowledge graph and social networks, to catch user interests. Others exploit the evolution of the interests by RNN, CNN or attention mechanism. In this paper, we propose AKGN, an Attention and Knowledge Graph based Network for recommendation. AKGN is a deep end-to-end framework that combines knowledge graph embedding and attention mechanism to assist SBRS task. Specifically, we employ two main layers to capture the latent user interests and their evolution. For each item in session sequences, we design interest extracting layer to capture the user's latent interest. At this layer, we propose a multi-level attention module based on the propagation of the knowledge triples. As user interests are changing, we present interest evolving layer utilizing the self-attention mechanism to capture the evolution of user interests. In the extensive experiments on the public datasets, AKGN significantly outperforms the state-of-the-art solutions.

KEYWORDS

recommendation system, knowledge graph, attention mechanism, session-based

ACM Reference Format:
., ., and .. 2019. An Attention and Knowledge Graph Network for Session-Based Recommendation System. In *Proceedings of ACM Conference (Conference'19).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Information overload makes it difficult for users to find appropriate items. Recommendation system (RS) overcomes the problem and suggests a small set of items for users to meet their personalized interests. There are two main approaches for computing recommendations. The first one is the general recommenders, which focus on mining the static relevancy between users and items from interactions. For example, collaborative filtering (CF) [7, 27] considers users' historical interactions and makes recommendations based on their potential common preferences, while content-based filtering recommends items sharing similar features to those a user has preferred in the past. In order to improve the effectiveness, researchers have demonstrated a hybrid approach, combining collaborative filtering and content-based filtering together [8]. And the hybrid method can also be used to overcome cold start and the sparsity problem. The other one is recommending next items based on sequential pattern mining [26, 37] or transition modeling [24, 41]. Along this line, researchers recently show more interest in session-based models [5, 11, 12, 22].

When available, side information about users and items is also merged into CF to improve the performance of RS. Knowledge graph (KG) contains a vast of information, and has been successfully applied into recommendation systems. KG is a type of directed heterogeneous graph in which nodes correspond to entities and edges correspond to relations. It can introduce semantic relations among items, including the relations between user's historical items and the recommended ones, besides the relations are of various types. As a result, KG-based methods are superior than many state-of-the-art baselines. In general, existing KG-aware recommendation can be classified into three categories: embedding-based methods [31], path-based methods [42], and the combination of the two method [32].

Actually, users' interactive behaviors naturally form a behavioral sequence, and their interests evolves over time dynamically, RNN/CNN structures are usually exploited to encode the evolution for downstream applications as in the encoder-decoder framework. However, both the basic RNN and CNN based methods suffer from time-consuming and the problem that the fixed-size encoding vector may not support both short and long sequences well. The attention mechanism [30], which is fast in both training and prediction phase, is then introduced. And it has already achieved great successes in recommendation system [44].

Based on all these observations, we propose AKGN, which employs two main modules to capture the evolution of users' interest. One is to extract users' temporal interests based on the bought item, and the other is to model the evolution of users' interests over time.

User behavior is the carrier of latent interest, many applications provide multiple types of related items, which can be seen as a KG whose edges link items via different relationship types. KG comprises various types of relations, and connectes users' historical items and the recommended ones, which is helpful to extend a user's

arXiv:1803.03467v4 [cs.IR] 25 Aug 2018

RippleNet: Propagating User Preferences on the Knowledge Graph for Recommender Systems

Hongwei Wang[1,2], Fuzheng Zhang[3], Jialin Wang[4], Miao Zhao[4], Wenjie Li[4], Xing Xie[2], Minyi Guo[1*]
[1]Shanghai Jiao Tong University, wanghongwei55@gmail.com, guo-my@cs.sjtu.edu.cn
[2]Microsoft Research Asia, xingx@microsoft.com, [3]Meituan AI Lab, zhangfuzheng@meituan.com
[4]The Hong Kong Polytechnic University, {csjlwang, csmiaozhao, cswjli}@comp.polyu.edu.hk

ABSTRACT

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 recommendation performance. This paper considers the knowledge graph as the source of side information. To address the limitations of existing embedding-based and path-based methods for knowledge-graph-aware recommendation, we propose *RippleNet*, an end-to-end framework that naturally incorporates the knowledge graph into recommender systems. Similar to actual ripples propagating on the water, RippleNet 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 knowledge graph. 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, which could be used for predicting the final clicking probability. Through extensive experiments on real-world datasets, we demonstrate that RippleNet achieves substantial gains in a variety of scenarios, including movie, book and news recommendation, over several state-of-the-art baselines.

KEYWORDS

Recommender systems; knowledge graph; preference propagation

ACM Reference Format:
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 *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3269206.3271739

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 (RS) intend to address the information explosion by finding a small set of items for

*M. Guo is the corresponding author. This work was partially sponsored by the National Basic Research 973 Program of China under Grant 2015CB352403.

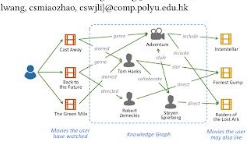

Figure 1: Illustration of knowledge graph enhanced movie recommender systems. The knowledge graph provides fruitful facts and connections among items, which are useful for improving precision, diversity, and explainability of recommended results.

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 [12]. 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 [9], user/item attributes [32], images [43] and contexts [25].

Among various types of side information, *knowledge graph* (KG) usually contains 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 [14], question answering [7], word embedding [40], and text classification [34].

Inspired by the success of applying KG in a wide variety of tasks, researchers also tried to utilize KG to improve the performance of recommender systems. As shown in Figure 1, KG can benefit the recommendation from three aspects: (1) KG introduces semantic relatedness among items, which can help find their latent connections and improve the *precision* of recommended items; (2) KG

[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

bines knowledge graph embedding and attention mechanism to assist SBRS task.

- Given that the users' attention varies on different types of relations and the users' preferences evolve over time, we employ interest extracting layer and interest evolving layer to capture the latent user interests and their evolution.
- We conduct experiments on real-world recommendation scenarios, and the results prove the efficacy of AKGN over several state-of-the-art baselines.

2 RELATED WORKS

2.1 Knowledge Graph Representation

Our work connects to a large body of work in Knowledge Graph Representation (KGR) methods. KGR intends to embed entities and relations in a KG into low-dimensional continuous vector spaces while preserving the KG's structural information. Researchers have proposed many models to embed entities and relations in KGs [28], among which the most common way is translational distance models, such as TransE [1], TransH [36], TransR [17] and TransD [14] and so on. This kind of models exploit distance-based scoring functions when learning representations of entities and relations. For example, TransE [1] wants $\mathbf{h}+\mathbf{r}\approx\mathbf{t}$ when (h,r,t) holds, where h,r,t are the corresponding representation vectors of head, relation and tail. Therefore, TransE assumes 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. In TransD [14], each object is represented by two vectors. Given a triplet (h,r,t), its vectors are $\mathbf{h}$, $\mathbf{h}_p$, $\mathbf{r}$, $\mathbf{r}_p$, $\mathbf{t}$, $\mathbf{t}_p$. And the score function is $f_r(h,t)=\|\mathbf{h}_\perp+\mathbf{r}-\mathbf{t}_\perp\|_2^2$. Another commonly used method is Latent Factor Model (LFM) [13, 29]. LMF employ a relation-specific bilinear form to consider the relatedness between entities and relations, and the score function is defined as $f_r(h,t)=\mathbf{h}^T\mathbf{M}_r\mathbf{t}$, where $\mathbf{M}_r$ is the relation matrix. Neutral network [6] is also employed to learn knowledge graph. For example, for a given knowledge triple (h,r,t), the score function is $f_r(h,t)=\mathbf{t}^T\mathcal{M}^L(\text{concat}(\mathbf{h},\mathbf{r}))$, where $\mathcal{M}^L(\cdot)=\mathcal{M}(\mathcal{M}(\cdots\mathcal{M}(\cdot)))$, and $\mathcal{M}(x)=\sigma(Wx+b)$ is a fully-connected neural network layer with weight W, bias b, and nonlinear activation function $\sigma(\cdot)$. We note here that all of the above KGR methods can be incorporated into our method.

2.2 Attention Mechanism

Attention mechanism was proposed in image classification [21] and machine translation [2], which aims to find the most relevant

ization performance of all the tasks [18, 40]. All of the learning tasks or at least a subset of them are assumed to be related to each other, and learning these tasks jointly can lead to performance improvement compared with learning them individually. In general, MTL algorithms can be classified into several categories, including feature learning approach [35, 38], low-rank approach [10, 20], task clustering approach [46], task relation learning approach [16], and decomposition approach [9]. MTL can also be combined with other learning paradigms to improve the performance of learning tasks further, including semi-supervised learning, active learning, unsupervised learning,and reinforcement learning. In this paper, we aim to utilize the connection of RS and KG to help improve their performance.

3 PROBLEM FORMULATION

Let $\mathcal{U}=\{u_1,u_2,\ldots\}$ and $\mathcal{I}=\{i_1,i_2,\ldots\}$ denote the sets of users and items, respectively. The user-item interaction matrix $\mathbf{Y}=\{y_{ui}|u\in\mathcal{U},i\in\mathcal{I}\}$ is defined according to users' implicit feedback, where

$$y_{ui}=\begin{cases}1 & \text{if there is an interaction between } u \text{ and } i,\\ 0 & \text{otherwise.}\end{cases}$$

In addition to the interaction matrix $\mathbf{Y}$, we also have the category id of each item and several sources of heterogeneous item-to-item relationships $\mathcal{R}=\{r_1,r_2,\ldots,r_{|\mathcal{R}|}\}$, which can be represented by a KG, denoted by $\mathcal{G}$.

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

4 AKGN MODELING FRAMEWORK

In this section, we discuss the proposed method in detail. The framework of our method is illustrated in Figure 1. We take a user u and an item v as inputs, and output the predicted probability that user u will click item v.

For each item in session sequences, we design interest extracting layer to capture the user's latent interest. At this layer, we propose a multi-level attention module based on the propagation of the knowledge triples, and return a vector to denote the response of

Network (DKN) [33] treats entity embeddings and word embeddings as different channels, then designs a CNN framework to combine them together for news recommendation. Collaborative Knowledge base Embedding (CKE) [43] combines a CF module with knowledge embedding, text embedding, and image embedding of items in a unified Bayesian framework. Signed Heterogeneous Information Network Embedding (SHINE) [32] designs deep autoencoders to embed sentiment networks, social networks and profile (knowledge) networks for celebrity recommendations. Embedding-based methods show high flexibility in utilizing KG to assist recommender systems, but the adopted KGE algorithms in these methods are usually more suitable for in-graph applications such as link prediction than for recommendation [35], thus the learned entity embeddings are less intuitive and effective to characterize inter-item relations.

The second category is *path-based methods* [42, 45], which explore the various patterns of connections among items in KG to provide additional guidance for recommendations. For example, Personalized Entity Recommendation (PER) [42] and Meta-Graph Based Recommendation [45] treat 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 KG in a more natural and intuitive way, but they rely heavily on manually designed meta-paths, which is hard to optimize in practice. Another concern is that it is impossible to design hand-crafted meta-paths in certain scenarios (e.g., news recommendation) where entities and relations are not within one domain.

To address the limitations of existing methods, we propose *RippleNet*, an end-to-end framework for knowledge-graph-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 behind RippleNet is *preference propagation*: For each user, RippleNet treats his historical interests as a seed set in the KG, then extends the user's interests iteratively along KG links to discover his hierarchical potential interests with respect to a candidate item. We analogize preference propagation with actual ripples created by raindrops propagating on the water, in which multiple "ripples" superpose to form a resultant preference distribution of the user over the knowledge graph. The major difference between RippleNet and existing literature is that RippleNet combines the advantages of the above mentioned two types of methods: (1) RippleNet incorporates the KGE methods into recommendation

the knowledge graph.

In summary, our contributions in this paper 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 KG to assist recommender systems. RippleNet automatically discovers users' hierarchical potential interests by iteratively propagating users' preferences in the KG.
- We conduct experiments on three real-world recommendation scenarios, and the results prove the efficacy of RippleNet over several state-of-the-art baselines.

2 PROBLEM FORMULATION

The knowledge-graph-aware recommendation problem is formulated as follows. In a typical recommender system, let $\mathcal{U}=\{u_1,u_2,\ldots\}$ and $\mathcal{V}=\{v_1,v_2,\ldots\}$ denote the sets of users and items, respectively. The user-item interaction matrix $\mathrm{Y}=\{y_{uv}|u\in\mathcal{U},v\in\mathcal{V}\}$ is defined according to users' implicit feedback, where

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

A value of 1 for y_{uv} indicates there is an implicit interaction between user u and item v, such as behaviors of clicking, watching, browsing, etc. In addition to the interaction matrix Y, we also have a knowledge graph $\mathcal{G}$ available, which consists of massive 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, $\mathcal{E}$ and $\mathcal{R}$ denote the set of entities and relations in the KG. 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 KG, while news with title "France's Baby Panda Makes Public Debut" is linked with entities "France" and "panda".

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

Finally, the user embedding $\mathbf{u}$ and the target item embedding $\mathbf{v}$ are combined to predict the clicking probability:

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

where $\sigma(\cdot)$ is the sigmoid function, and f is a ranking function which can be either a dot-product function or a more complexed deep neural network.

The loss function L is the sigmoid cross entropy loss, which is definde as:

$$\sum_{(u,v)\in Y}-(y_{uv}\log\sigma(f(\mathbf{u},\mathbf{v}))+(1-y_{uv})\log(1-\sigma(f(\mathbf{u},\mathbf{v}))))\quad(3)$$

4.3 Multi-Task Learning

We use a multi-task learning framework [3] to jointly optimize all two tasks using shared variables.

$$\begin{aligned}&\min\mathcal{L}=\min\ (L+L_{\mathrm{KGE}})\\ &=\sum_{(u,v)\in Y}-(y_{uv}\log\sigma(f(\mathbf{u},\mathbf{v}))+(1-y_{uv})\log(1-\sigma(f(\mathbf{u},\mathbf{v}))))\\ &+\lambda_1\left(\sum_{(h,r,t)\in\mathcal{G}}f_r(\mathbf{h},\mathbf{t})+\sum_{(h',r',t')\notin\mathcal{G}}f_{r'}(\mathbf{h}',\mathbf{t}')\right)\\ &+\lambda_2\left(\|\mathbf{E}\|_F^2\right),\end{aligned}\qquad(4)$$

In eq.(4), the first term measures loss in the recommendation module, where $\mathbf{u}$ and $\mathbf{v}$ traverse the set of users and the items, respectively. The second term measures the squared error between the ground truth of the KG and the reconstructed one, $f_r(\mathbf{h},\mathbf{t})$ is the score function of TransD. And the third term is the regularizer for preventing over-fitting.

We employ a stochastic gradient descent (SGD) algorithm to iteratively optimize the loss function. 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 $\mathcal{G}$. Then we calculate the gradients of the loss $\mathcal{L}$ with respect to model parameters, and update all parameters by back-propagation based on the sampled minibatch. We will discuss the choice of hyper-parameters in the experiments section.

5 EXPERIMENT

In this section, we perform experiments on the real-world datasets to demonstrate the efficiency and quality of our proposed methods.

5.1 Experimental Settings

5.1.1 Datasets and Preprocessing. Amazon Dataset is composed of large corpora of users' reviews on products and multiple types of related items [19]. These data were crawled from Amazon.com and span May 1996 to July 2014. We use three subsets of Amazon dataset: Books, Electronics and Automotive, which have already been reduced to satisfy the 5-core property, such that each of the remaining users and items have 5 reviews each. The feature we use contains user id, item id, cate id, related items and timestamps. This set of data is not able for its high sparsity and variability. Amazon contains four types of item relationships that we use in our model: 'also viewed', 'also bought', 'bought together', and 'buy after viewing'. To collect the related facts from Amazon, we only consider those triples that are directly related to the entities with mapped items. We then preprocess the four datasets by: filtering out low frequency and high frequency users and items (i.e., lower than 5 and longer than 20 in both datasets), filtering out infrequent entities (i.e., lower than 5 in both datasets), cutting off unrelated relations and merging similar relations manually. In these datasets, we regard reviews as behaviors, and sort the reviews from one user by time. Assuming there are T behaviors of user u, our purpose is to use the $T-1$ behaviors to predict whether user u will write reviews that shown in T-th review. The statistics of the four datasets are shown in Table 1.

5.1.2 Baseline Algorithms. To evaluate the performance of the proposed method, we compare it with the following representative baselines:

- BPR-MF [23]: It is one of widely used matrix factorization methods, which optimizes the latent factor model with implicit feedback using a pairwise ranking loss in a Bayesian means.
- Bi-LSTM: We implement a Bi-LSTM method to encode the user behavior history, whose difference with [39] is that we use a bidirectional LSTM as the encoder because of better performance in our experiments. Then we concatenate the two final hidden states to the user embedding. The stacked LSTM depth is set to be 1.
- CNN+Pooling: We use a CNN structure with max pooling to encode the user history as in [15, 43].We use ten kinds of window sizes from one to ten for extracting different features, and all of the feature map have the same kernel size with 32. Then we apply a max-over-time pooling operation over the feature map, and pass all pooled features from different maps to a fully connected layer to produce final user embedding.

Based on the definition in Eq. (11), the scoring functions of entity-entity pairs in KGE and item-entity pairs in preference propagation can thus be unified under the same calculation model. The last term in Eq. (9) is the likelihood function of the observed implicit feedback given Θ and the KG, which is defined as the product of Bernoulli distributions:

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

based on Eq. (2)–(7).

Taking the negative logarithm of Eq. (9), we have the following loss function for RippleNet:

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

where V and E are the embedding matrices for all items and entities, respectively, I_r is the slice of the indicator tensor I in KG for relation r, and R is the embedding matrix of relation r. In Eq. (13), The first term measures the cross-entropy loss between ground truth of interactions Y and predicted value by RippleNet, the second term measures the squared error between the ground truth of the KG I_r and the reconstructed indicator matrix $\mathrm{E}^{\mathrm{T}}\mathrm{RE}$, and the third term is the regularizer for preventing over-fitting.

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 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 Y and true/false triples from $\mathcal{G}$ following the negative sampling strategy in [16]. 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 hyper-parameters in the experiments section.

present a visualized example in the experiments sectior
itively demonstrate the explainability of RippleNet.

3.5.2 Ripple Superposition. A common phenomenon in R
is that a user's ripple sets may be large in size, which di
potential interests inevitably in preference propagation. I
we observe that relevant entities of different items in a us
history often highly overlap. In other words, an entity
reached by multiple paths in the KG starting from a use
history. For example, "Saving Private Ryan" is connected
who has watched "The Terminal", "Jurassic Park" and "Bra
through actor "Tom Hanks", director "Steven Spielberg" a
"War", respectively. These parallel paths greatly increase
interests in overlapped entities. We refer to the case as
perposition, as it is analogous to the interference phenon
physics in which two waves superpose to form a resultant
greater amplitude in particular areas. The phenomenon
superposition is illustrated in the second KG in Figure 2, w
darker red around the two lower middle entities indicate
strength of the user's possible interests. We will also discu
superposition in the experiments section.

3.6 Links to Existing Work

Here we continue our discussion on related work and m
parisons with existing techniques in a greater scope.

3.6.1 Attention Mechanism. The attention mechanism w
nally proposed in image classification [18] and machine tr
[1], which aims to learn where to find the most relevan
the input automatically as it is performing the task. The
soon transplanted to recommender systems [4, 22, 33, 36,
example, DADM [4] considers factors of specialty and da
assigning attention values to articles for recommendation
[22] proposes an interpretable and dual attention-based CN
that combines review text and ratings for product ratin
tion; DKN [33] uses an attention network to calculate th
between a user's clicked item and a candidate item to dyn
aggregate the user's historical interests. RippleNet can b

Attention

https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf?source=techstories.org

An Attention and Knowledge Graph Network for Session-Based Recommendation System.pdf (page 5 of 9) — Edited

Search

$$\begin{aligned}&+\lambda_1\left(\sum_{(h,r,t)\in\mathcal{G}}f_r(\mathbf{h},\mathbf{t})+\sum_{(h',r',t')\notin\mathcal{G}}f_{r'}(\mathbf{h}',\mathbf{t}')\right)\\ &+\lambda_2\left(\|\mathbf{E}\|_F^2\right),\end{aligned}\qquad(4)$$

In eq.(4), the first term measures loss in the recommendation module, where $\mathbf{u}$ and $\mathbf{v}$ traverse the set of users and the items, respectively. The second term measures the squared error between the ground truth of the KG and the reconstructed one, $f_r(\mathbf{h},\mathbf{t})$ is the score function of TransD. And the third term is the regularizer for preventing over-fitting.

We employ a stochastic gradient descent (SGD) algorithm to iteratively optimize the loss function. 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 $\mathcal{G}$. Then we calculate the gradients of the loss $\mathcal{L}$ with respect to model parameters, and update all parameters by back-propagation based on the sampled minibatch. We will discuss the choice of hyper-parameters in the experiments section.

5 EXPERIMENT

In this section, we perform experiments on the real-world datasets to demonstrate the efficiency and quality of our proposed methods.

by time. Assuming there are T behaviors of user u, our purpose is to use the $T-1$ behaviors to predict whether user u will write reviews that shown in T-th review. The statistics of the four datasets are shown in Table 1.

5.1.2 Baseline Algorithms. To evaluate the performance of the proposed method, we compare it with the following representative baselines:

- BPR-MF [23]: It is one of widely used matrix factorization methods, which optimizes the latent factor model with implicit feedback using a pairwise ranking loss in a Bayesian means.
- Bi-LSTM: We implement a Bi-LSTM method to encode the user behavior history, whose difference with [39] is that we use a bidirectional LSTM as the encoder because of better performance in our experiments. Then we concatenate the two final hidden states to the user embedding. The stacked LSTM depth is set to be 1.
- CNN+Pooling: We use a CNN structure with max pooling to encode the user history as in [15, 43].We use ten kinds of window sizes from one to ten for extracting different features, and all of the feature map have the same kernel size with 32. Then we apply a max-over-time pooling operation over the feature map, and pass all pooled features from different maps to a fully connected layer to produce final user embedding.

1711.06632.pdf (page 5 of 9) — Edited

Search

feature that represent the timestamp when that behavior happens. The statics of each categorial is shown in table 2. This dataset is guaranteed that each user has at least 3 behaviors in each behavior group.

Competitors

- BPR-MF: Bayesian Personalized Ranking (Rendle et al. 2009) is a pairwise ranking framework. It trains on pairs of a user's positive and negative examples, maximizing the posterior probability of the difference given the same user. The item vector is a concatenation of item embedding and category embedding, each of which has a dimension of 64, the user vector has a dimension of 128.
- Bi-LSTM: We implement a Bi-LSTM method to encode the user behavior history, whose difference with (Zhang et al. 2014) is that we use a bidirectional LSTM as the encoder because of better performance in our experiments. Then we concatenate the two final hidden states to the user embedding. The stacked LSTM depth is set to be 1.
- Bi-LSTM+Attention: We add a vanilla attention on top of the Bi-LSTM method mentioned above.
- CNN+Pooling: We use a CNN structure with max pooling to encode the user history as in (Zheng, Noroozi, and Yu 2017; Kim 2014). We use ten kinds of window sizes from one to ten for extracting different features, and all of the feature map have the same kernel size with 32. Then we apply a max-over-time pooling operation over the feature map, and pass all pooled features from different maps to a fully connected layer to produce final user embedding.

[1] http://jmcauley.ucsd.edu/data/amazon/

[2] http://www.taobao.com

to be 128. The ranking function f is simply set to be the dot product in these tasks. As we observe better performance with point-wise ranking model, we omit the results of pair-wise models here for simplicity. For ATRank, we set the number of the latent semantic spaces to be 8, whose dimension sizes sum to be the same as the size of the hidden layer.

- Batch Size. The batch size is set to be 32 for all methods.
- Regularization. The l2-loss weight is set to be 5e-5.
- Optimizer. We use SGD as the optimizer and apply exponential decay which learning rate starts at 1.0 and decay rate is set to 0.1.

Evaluation Metrics

We evaluate the average user AUC as following:

$$AUC=\frac{1}{|U^{Test}|}\sum_{u\in U^{Test}}\frac{1}{|I_u^+||I_u^-|}\sum_{i\in I_u^+}\sum_{j\in I_u^-}\delta(\hat{p_{u,i}}>\hat{p_{u,j}})$$

where $\hat{p_{u,i}}$ is the predicted probability that a user $u\in U^{Test}$ may act on i in the test set and $\delta(\cdot)$ is the indicator function.

Results on Single-type Behavior Dataset

We first illustrate the average user AUC of all the methods for the amazon dataset in table 3. We can see that ATRank performs better than the competitors especially when the user behavior is dense, which reveals the benefits of the self-attention based user behavior modeling.

Table 4 illustrates the average vanilla-attention score for different time buckets over the whole amaon dataset, which can be inferred that time encoding via self-attention can