Deep Hybrid Knowledge Graph Embedding for Top-N Recommendation

Jian Li, Zhuoming Xu(✉), Yan Tang, Bo Zhao, and Haimei Tian

College of Computer and Information, Hohai University, Nanjing 210098, China
{jli, zmxu, tangyan, bzhao, hmtian}@hhu.edu.cn

Abstract. In knowledge graph (KG) based recommender systems, path-based methods make recommendations by building user-item graphs and exploiting connectivity patterns between the entities in the graph. To overcome the limitations of traditional meta-path based methods that rely heavily on handcrafted meta-paths, recent deep neural network based methods, such as the Recurrent Knowledge Graph Embedding (RKGE) approach, can automatically mine the connectivity patterns between entities in the KG, thereby improving recommendation performance. However, these methods usually use only one type of neural network to encode path embeddings, which cannot fully extract path features, limiting performance improvement of the recommender system. In this paper, we propose a Deep Hybrid Knowledge Graph Embedding (DHKGE) method for top-N recommendation. DHKGE encodes embeddings of paths between users and items by combining convolutional neural network (CNN) and the long short-term memory (LSTM) network. Furthermore, it uses an attention mechanism to aggregate the encoded path representations and generate a final hidden state vector, which is used to calculate the proximity between the target user and candidate items, thus generating top-N recommendation. Experiments on the MovieLens 100K and Yelp datasets show that DHKGE overall outperforms RKGE and several typical recommendation methods in terms of Precision@N, MRR@N, and NDCG@N.

Keywords: Top-N recommendation · Knowledge graph · Deep hybrid model · CNN · LSTM · Attention mechanism

1 Introduction

Knowledge graphs (KGs) have proven to be effective in improving recommendation performance [7, 16]. According to [7], there are three categories of KG-based recommendation methods: path-based methods, embedding-based methods, and unified methods. Path-based methods make recommendations by building a KG which contains users, items, and user-item interactions, and then exploiting connectivity patterns between the entities (users or items) in the KG. The traditional meta-path based methods use the semantic similarity of entities in different meta-paths [18] as graph regularization to refine representations of users and items [7]. However, such methods rely heavily on handcrafted meta-paths, which further rely on domain knowledge [14].

G. Wang et al. (Eds.): WISA 2020, LNCS 12432, pp. 59–70, 2020.
https://doi.org/10.1007/978-3-030-60029-7_6

To overcome the limitations of meta-path based methods, deep neural network based methods have recently been devised to automatically mine the connectivity patterns between entities (i.e., path embeddings) in the KG. Path representations are learned by extracting path features from connectivity patterns to characterize user preferences towards items, which are finally used to generate recommendation.

However, existing deep neural network based methods, such as the Recurrent Knowledge Graph Embedding (RKGE) approach [14], usually use only one type of neural network to encode path embeddings. But this cannot fully extract path features, which limits performance improvement of the recommender system. Recently proposed deep hybrid models, such as [12], can combine several neural building blocks to form a more powerful recommendation model. To the best of our knowledge, existing deep hybrid models seldom use KGs for recommendation.

To overcome the weaknesses of existing methods, in this paper we propose a Deep Hybrid Knowledge Graph Embedding (DHKGE) method for top-N recommendation. DHKGE encodes embeddings of paths between users and items that are involved in the recommender system by combining convolutional neural network (CNN) and the long short-term memory (LSTM) network. It further uses an attention mechanism to aggregate the encoded path representations and generate a final hidden state vector. This vector is then used to calculate the proximity between the target user and candidate items, and generate top-N recommendation for the user by ranking the proximity.

In summary, the main contributions of this paper are as follows:

- We propose the Deep Hybrid Knowledge Graph Embedding (DHKGE) method for top-N recommendation, which exploits a deep hybrid model to encode the path between users and items.
- We propose to use the attention mechanism to distinguish the importance of multiple semantic paths between a user-item pair, so that salient paths play a greater role in modeling user preferences.
- We evaluated our method on the MovieLens 100K and Yelp datasets. The experimental results show that our method overall outperforms RKGE and several typical recommendation methods in terms of Precision@N, MRR@N, and NDCG@N.

2 Related Work

2.1 Path-Based Recommendation Methods

Path-based methods make recommendations by building user-item graphs and exploiting connectivity patterns between the entities in the graph [7]. Traditional meta-path based methods rely heavily on handcrafted meta-paths. Deep neural network based methods can automatically mine the connectivity patterns between entities in the graph, thereby improving recommendation performance. For example, Hu *et al.* [9] proposed to leverage meta-path based context for top-N recommendation with a neural co-attention model. Sun *et al.* [14] proposed the RKGE approach that employs RNN to learn high-quality representations of both users and items, which are then used to generate better recommendations. Wang *et al.* [15] proposed the Knowledge-aware Path Recurrent Network (KPRN) which exploits KG to generate better recommendation, where the path embeddings in the KG are encoded with LSTM.

Existing path-based recommendation methods usually use only one type of neural network to encode path embeddings, while our proposed DHKGE exploits a deep hybrid model to encode path embeddings, which can generate a more comprehensive path representation for better recommendation.

2.2 Deep Neural Network-Based Recommendation

Deep neural networks have been widely used in recommender systems. The existing recommendation models can be divided into two categories: recommendation with neural building blocks and recommendation with deep hybrid models [4, 20].

In the first category, the recommendation models are divided into several subcategories [20] that exploit the deep learning models: CNN, recurrent neural network (RNN), and attentional model (AM), etc. For example, Kim *et al.* [10] proposed a context-aware recommendation model named convolutional matrix factorization (ConvMF) that integrates CNN into probabilistic matrix factorization.

Recently, researchers have proposed deep hybrid models, which can combine several neural building blocks to complement one another and form a more powerful recommendation model [20]. For instance, Lee *et al.* [12] proposed a deep learning recommender system that combines RNN and CNN to learn semantic representation of each utterance and build a sequence model for the dialog thread. To the best of our knowledge, existing deep hybrid models seldom use KGs for recommendation.

3 DHKGE: Deep Hybrid KG Embedding Method

In this section, we expatiate on our DHKGE method. After introducing concepts and notations, we first briefly explain its overall framework, then describe its main components, and finally describe model learning and recommendation generation.

Given a user set $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$ and an item set $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ of the recommender system, we construct the users' implicit feedback matrix $\mathbf{R} \in \mathbb{R}^{m \times n}$, where each element is defined as follows: when user u_i interacted with item v_j set $r_{ij} = 1$ indicating that the user prefers the item, otherwise set $r_{ij} = 0$. Based on the matrix $\mathbf{R}$ and an external knowledge source (e.g., the IMDB dataset) that describes the items, we build a KG for recommendation, which contains the users, items, user's preference for the items, and the item descriptions extracted from the knowledge source, such as actors, directors and genres (as entities), as well as rating, categorizing, acting, and directing (as entity relations) in the domain of movie recommendation. We refer to all objects (e.g., users, items, actors, directors, and genres) except for various relations in the KG as *entities*. The definition [14] of the KG is given below.

Definition 1 (Knowledge Graph). KG is defined as a directed graph $\mathcal{G} = (\mathcal{E}, \mathcal{L})$, where $\mathcal{E} = \{e_1, e_2, \ldots, e_{|\mathcal{E}|}\}$ denotes the sets of entities and $\mathcal{L}$ the sets of links. An entity type mapping function $\phi : \mathcal{E} \rightarrow \mathcal{A}$ and a link type mapping function $\varphi : \mathcal{L} \rightarrow \mathcal{R}$

are defined for the graph. Each entity $e \in \mathcal{E}$ belongs to an entity type $\phi(e) \in \mathcal{A}$, and each link $l \in \mathcal{L}$ belongs to a link type (relation) $\varphi(l) \in \mathcal{R}$.

Based on the KG definition, we further define the connected semantic paths between entity pair (e_i, e_j) as $\mathcal{P}(e_i, \mathrm{e}_j) = \{p_1, p_2, \ldots, p_s\}$ with s being the number of paths. A semantic path of length T in $\mathcal{P}$ is denoted as: $p = e_i \xrightarrow{r_1} e_1 \xrightarrow{r_2} \cdots \xrightarrow{r_T} e_j$.

Following the two semantic path mining strategies proposed in [14], DHKGE only considers user-item paths $\mathcal{P}(u_i, v_j)$, $u_i \in \mathcal{U}$, $v_j \in \mathcal{V}$ that connect user u_i with all her rated items v_j, and sets a length constraint for such paths, i.e., path length is T.

3.1 Overview

Our goal is to fully extract the information in the semantic path to model user preferences, which are then used to generate better recommendations. To achieve this goal, we propose the deep hybrid knowledge graph embedding (DHKGE) method.

The core ideas of DHKGE is as follows: Given a user and an item, DHKGE first automatically extracts all semantic paths between the user and the item from the KG according to the semantic path mining strategies. It then uses a deep hybrid model to obtain a final hidden vector for quantifying the relation (proximity) between the user and the item. Finally, it generates a top-N recommendation list for the user by sorting the proximity scores of the candidate items in descending order.

The overall framework of DHKGE is depicted in Fig. 1. As shown in the figure, DHKGE is composed of four key components: the embedding layer, CNN layer, LSTM layer, and attention layer, which are further described as follows:

- The embedding layer: This layer takes the semantic path of length T as input, learns $T+1$ low-dimensional embedding vectors for $T+1$ entities on the semantic path, and outputs these vectors as an embedding of the path.
- The CNN layer: This layer takes the path embedding as input, uses multiple filters to extract the local features of the path to form T local feature vectors, and outputs these vectors.
- The LSTM layer: This layer takes the ordered local feature vectors as input, encodes them to get a representation of the path, and outputs the path representation.
- The attention layer: This layer takes the representations of s paths as input, uses the attention mechanism to aggregate these path representations by weighting them to obtain a final hidden state vector, and outputs the vector.

3.2 Embedding Layer

Given a set of s semantic paths of length T between user u_i and item v_j, $\mathcal{P}(u_i, v_j) = \{p_1, p_2, \ldots, p_s\}$, where the start entity and end entity of each path in $\mathcal{P}$ are u_i and v_j, respectively. As shown in Fig. 1, $e_0 = u_i$ and $e_T = v_j$ in path p_1. The embedding layer maps each entity e_t in such a path into a d-dimensional vector $\mathbf{e}_t \in \mathbb{R}^d$, which captures the semantic meaning of the entity. The vectors of all entities in the path constitute an embedding $\mathbf{p_1} = \{\mathbf{e}_0, \mathbf{e}_1, \ldots, \mathbf{e}_T\}$ of the path.

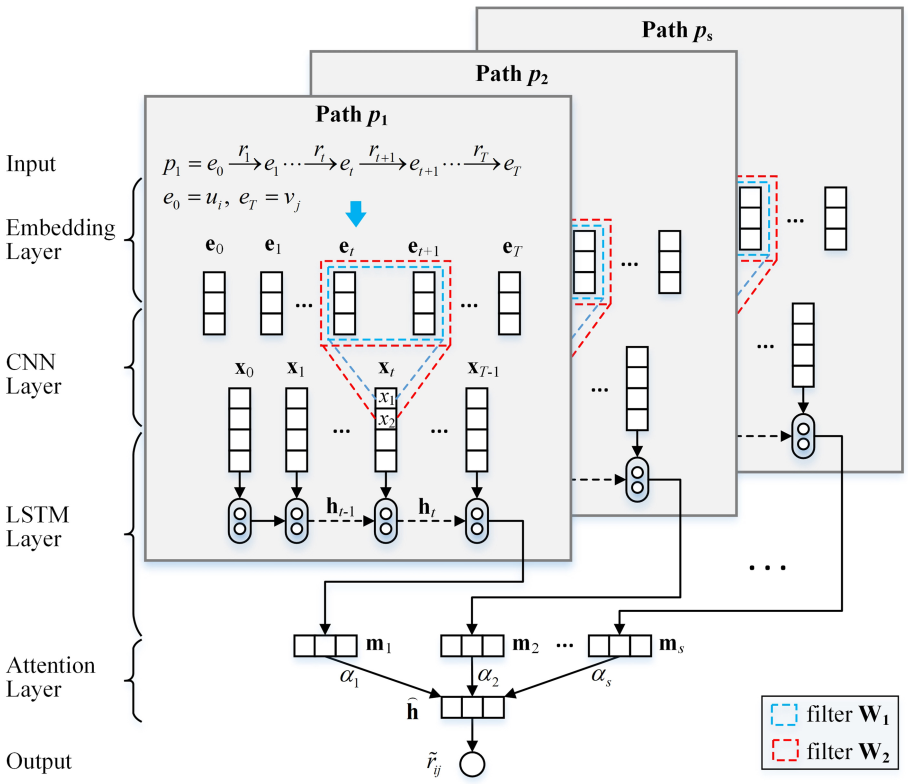
Path p_s
Path p_2
Path p_1
Input
$p_1 = e_0 \xrightarrow{r_1} e_1 \cdots \xrightarrow{r_t} e_t \xrightarrow{r_{t+1}} e_{t+1} \cdots \xrightarrow{r_T} e_T$
$e_0 = u_i,\ e_T = v_j$
Embedding Layer
$\mathbf{e}_0$ $\mathbf{e}_1$ $\mathbf{e}_t$ $\mathbf{e}_{t+1}$ $\mathbf{e}_T$
CNN Layer
$\mathbf{x}_0$ $\mathbf{x}_1$ $\mathbf{x}_t$ $\mathbf{x}_{T\text{-}1}$
x_1 x_2
LSTM Layer
$\mathbf{h}_{t\text{-}1}$ $\mathbf{h}_t$
Attention Layer
$\mathbf{m}_1$ $\mathbf{m}_2$ $\mathbf{m}_s$
α_1 α_2 α_s
$\hat{\mathbf{h}}$
Output
$\tilde{r}_{ij}$
filter $\mathbf{W}_1$
filter $\mathbf{W}_2$

Fig. 1. The overall framework of DHKGE, which describes the case of a user-item pair (u_i, v_j).

3.3 CNN Layer

The CNN layer takes path embedding $\mathbf{p_1}$ as input, and then slides multiple filters with the same window size over the path embedding to extract local features of the path. Let $\mathbf{W}_1 \in \mathbb{R}^{2\times d}$ be a filter with a window size of 2. As shown in Fig. 1, $\mathbf{W}_1$ is applied to two embeddings $\mathbf{e}_t$ and $\mathbf{e}_{t+1}$ of the adjacent entities to generate a local feature x_1, which is defined as Eq. (1) [5, 11].

$$x_1 = f(\mathbf{W}_1 \circ [\mathbf{e}_t, \mathbf{e}_{t+1}] + b_1) \tag{1}$$

where $\circ$ denotes the convolution operation, b_1 is the bias, and $f(\cdot)$ is the nonlinear activation function ReLU.

This way, k filters with the same window size, $\mathbf{W}_1, \mathbf{W}_2, \ldots,$ are applied to the two entity embeddings $\mathbf{e}_t$ and $\mathbf{e}_{t+1}$ to obtain a local feature vector $\mathbf{x}_t = [x_1, x_2, \ldots, x_k]$, where k is a hyperparameter. The CNN layer slides k filters from entity embedding $\mathbf{e}_0$ to entity embedding $\mathbf{e}_{T-1}$ with stride 1, thus forming a sequence of local feature vectors $\{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{T-1}\}$.

3.4 LSTM Layer

Taking T ordered local feature vectors $\{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{T-1}\}$ as input, the LSTM layer uses LSTM to encode the sequence information in the local feature vectors to generate a path representation. At the time step $t-1$, LTSM outputs a hidden state vector $\mathbf{h}_{t-1} \in \mathbb{R}^{d'}$, where hyperparameter d' is the number of LSTM hidden units. As shown in Fig. 1, the hidden state vector $\mathbf{h}_{t-1}$ and the local feature vector $\mathbf{x}_t$ are used to learn the hidden state vector $\mathbf{h}_t$ at time step t, and $\mathbf{h}_t$ is defined as Eq. (2) [6, 15].

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{U}_i\mathbf{x}_t + \mathbf{W}_i\mathbf{h}_{t-1} + \mathbf{b}_i)\\
\mathbf{f}_t &= \sigma(\mathbf{U}_f\mathbf{x}_t + \mathbf{W}_f\mathbf{h}_{t-1} + \mathbf{b}_f)\\
\mathbf{o}_t &= \sigma(\mathbf{U}_o\mathbf{x}_t + \mathbf{W}_o\mathbf{h}_{t-1} + \mathbf{b}_o)\\
\hat{\mathbf{c}}_t &= \tanh(\mathbf{U}_c\mathbf{x}_t + \mathbf{W}_c\mathbf{h}_{t-1} + \mathbf{b}_c)\\
\mathbf{c}_t &= \mathbf{i}_t \odot \hat{\mathbf{c}}_t + \mathbf{f}_t \odot \mathbf{c}_{t-1}\\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned} \tag{2}$$

where, $\mathbf{i}_t, \mathbf{f}_t, \mathbf{o}_t \in \mathbb{R}^{d'}$ represent the input, forget, and output gates at time step t, respectively. $\hat{\mathbf{c}}_t, \mathbf{c}_t, \mathbf{h}_t \in \mathbb{R}^{d'}$ denote the information transform module, cell state vector, and hidden state vector at time step t, respectively. $\mathbf{U}_i, \mathbf{U}_f, \mathbf{U}_o, \mathbf{U}_c \in \mathbb{R}^{d' \times k}$ are input weights, $\mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_o, \mathbf{W}_c \in \mathbb{R}^{d' \times d'}$ are recurrent weights, and $\mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o, \mathbf{b}_c \in \mathbb{R}^{d'}$ are biases. $\sigma(\cdot)$ is the sigmoid activation function and $\odot$ stands for the element-wise product of two vectors.

As shown in Fig. 1, the learning process continues until the LSTM layer obtains the hidden state vector at the final time step $T-1$. This hidden state vector is therefore output as a path representation, denoted $\mathbf{m}_1 \in \mathbb{R}^{d'}$.

3.5 Attention Layer

Once the path representations are obtained, the attention layer takes these path representations as input and uses the attention mechanism to generate a final hidden state vector and output it. The process of generating hidden state vectors is as follows: First, this layer learns an attention score $score(\mathbf{m}_i)$ for each path representation $\mathbf{m}_i$ in the path representation set $\{\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_s\}$. Then these scores are normalized, and finally these path representations are aggregated by weighting them to obtain a final hidden state vector $\overset{\frown}{\mathbf{h}} \in \mathbb{R}^{d'}$, which characterizes the user preferences towards items. The above process is defined as Eq. (3) [17].

$$\begin{aligned}
&score(\mathbf{m}_i) = \mathbf{w}_a^{\mathrm{T}} \tanh(\mathbf{W}_a\mathbf{m}_i)\\
&\alpha_i = \frac{\exp(score(\mathbf{m}_i))}{\sum_{i=1}^{s} \exp(score(\mathbf{m}_i))}\\
&\overset{\frown}{\mathbf{h}} = \sum_{i=1}^{s} \alpha_i \mathbf{m}_i
\end{aligned} \tag{3}$$

where $\mathbf{w}_a \in \mathbb{R}^{d'}$ and $\mathbf{W}_a \in \mathbb{R}^{d' \times d'}$ are weights, and α_i the normalized attention score.

Finally, DHKGE uses a fully connected layer to quantify the proximity $\tilde{r}_{ij}$ of user u_i and item v_j, which is defined as Eq. (4) [14]:

$$\tilde{r}_{ij} = \sigma(\mathbf{W}_r \widehat{\mathbf{h}} + b_r) \tag{4}$$

where $\mathbf{W}_r \in \mathbb{R}^{1 \times d'}$ and b_r are the weights and bias, respectively.

3.6 Method Learning and Recommendation Generation

Like RKGE [14], given the training data $\mathcal{D}_{\text{train}}$, which contains instances in the form of $(u_i,\ v_j,\ r_{ij},\ \mathcal{P}(u_i,\ v_j))$, DHKGE also uses stochastic gradient descent (SGD) to minimize the loss function defined as Eq. (5) to learn all the parameters in DHKGE.

$$\mathcal{J} = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum\nolimits_{r_{ij} \in \mathcal{D}_{\text{train}}} BCELoss(\tilde{r}_{ij},\ r_{ij}) \tag{5}$$

where $BCELoss(\cdot)$ is the binary cross-entropy between the observed ratings and estimated ones.

The recommendation problem can be dealt with as a binary classification problem [8, 14]. When user u_i prefers item v_j, namely $r_{ij} = 1$, we expect the estimated proximity $\tilde{r}_{ij}$ to approach 1, otherwise it approaches 0. Once the learning process is completed, DHKGE can obtain all trained embeddings of the users and items.

Following [14, 18], during the testing process DHKGE can obtain the proximity scores between the target user and candidate items by calculating the inner products of the user embedding and the item embeddings. DHKGE finally generates top-N recommendation lists for the user by sorting the proximity scores in descending order.

4 Experimental Evaluation

4.1 Experimental Setup

Datasets. Our experiment used two datasets MovieLens 100K[1] and Yelp published on GitHub[2] by [14]. The former is a movie dataset containing user interaction with movies. Sun *et al.* [14] combined this dataset with the IMDB dataset[3] to add description information of movies, such as genre, actor, and director. Yelp contains user check-ins to local business, user reviews, and local business information, and no external information needs to be added to this dataset. The two datasets were used to build two KGs following **Definition** 1. The statistics of two datasets are shown in Table 1.

[1] The experiment of [14] used MovieLens 1M, but the pre-training vectors of users and items in MovieLens 1M were not published on GitHub, so we can only use MovieLens 100K.

[2] https://github.com/sunzhuntu/Recurrent-Knowledge-Graph-Embedding/tree/master/data.

[3] https://www.imdb.com/.

Table 1. Dataset and knowledge graph statistics

Datasets		MovieLens 100K	Yelp
User-item interaction	# Users	943	37,940
	# Items	1,675	11,516
	# Ratings	99,975	229,178
Knowledge graph	# Entities	7,744	50,028
	# Entity types	5	4
	# Links	112,321	272,057
	# Links types	7	5

Following [3, 14], we sorted the two datasets according to the feedback timestamp, and used the earlier 80% feedback as training data and the more recent 20% feedback as test data. For each user-item pair in the training set, we extracted all paths with a length of 3 and randomly selected five paths from them to train our model.

Evaluation Metrics. Three popular evaluation metrics [1], Precision at N (Prec@N), Mean Reciprocal Rank at N (MRR@N), and Normalized Discounted Cumulative Gain at N (NDCG@N), are adopted to evaluate the top-N recommendation methods in our experiment. We set N = {1, 5, 10, 20} for Prec@N, and N = {5, 10, 20} for MRR@N and NDCG@N.

Comparison Methods and Their Implementation. We compared our DHKGE with the following four recommendation methods:

- BPRMF [13]: It is a Bayesian personalized ranking method based on Matrix Factorization. We used the Cornac[4] framework to implement BPRMF.
- NCF [8]: It is a classic neural network-based recommendation method. It was also implemented by using the Cornac framework.
- CKE [19]: It is the recently proposed state-of-the-art KG embedding based recommendation method. This method directly used the Python code[5] provided in [2].
- RKGE [14]: It is a state-of-the-art recommendation method based on KG path. This method directly used the Python code published on GitHub[6] by the authors.

We used PyTorch to generate the code of DHKGE by modifying the recurrent network module and performance evaluation module in the RKGE code.

Hyperparameter Settings. For DHKGE, we used grid search to select both the dimension d of the entity embedding and the number k of convolution filters in $\{10, 20, 30, 40, 50, 100\}$, the number d' of LSTM hidden units in $\{16, 32, 64, 128\}$, and the learning rate λ of SGD in $\{0.001, 0.01, 0.1, 0.2\}$. The hyperparameters for

[4] https://github.com/PreferredAI/cornac.

[5] https://github.com/TaoMiner/joint-kg-recommender.

[6] https://github.com/sunzhuntu/Recurrent-Knowledge-Graph-Embedding.

DHKGE were set to $d = 10,\ k = 10,\ d' = 16,\ \lambda = 0.2$ on MovieLens 100K and $d = 20,\ k = 40,\ d' = 32,\ \lambda = 0.01$ on Yelp. For the four comparison methods, the hyper-parameters were set as suggested by the original papers.

4.2 Experimental Results

Tables 2 and 3 show the results of top-N recommendation performed on the two datasets. In the tables, bold numbers indicate the best performance among all the methods; underlined numbers are the best performance among the four comparison methods; the numbers in the "Improve" column indicate the percentage (%) of performance improvement achieved by DHKGE relative to the best performance among the comparison methods. The same way as in [14], we also created two views for each dataset: "All Users" means that all users are considered in the test data, whereas "Cold Start" indicates that the test data only includes users with less than 5 ratings.

Observing these results, we can obtain the following findings:

1. The performance of both DHKGE and RKGE in terms of all metrics except for Prec@1 is significantly better than the other three methods. This indicates DHKGE and RKGE can make full use of the path information to model user's preference for items, thereby improving the recommendation performance.
2. DHKGE's performance is better than RKGE in all metrics (the performance in terms of Prec@1 on MovieLens 100K is the same). This indicates that deep hybrid

Table 2. Results of top-N recommendation on MovieLens 100K

Views	Metrics	BPRMF	NCF	CKE	RKGE	DHKGE	Improve (%)
All Users	Prec@1	0.0382	0.0509	0.1044	0.1469	**0.1548**	5.38
	Prec@5	0.0418	0.0492	0.0826	0.1044	**0.1103**	5.65
	Prec@10	0.0431	0.0467	0.0735	0.0890	**0.0962**	8.09
	Prec@20	0.0467	0.0481	0.0660	0.0777	**0.0822**	5.79
	MRR@5	0.1135	0.1247	0.2272	0.2760	**0.2883**	4.46
	MRR@10	0.1221	0.1388	0.2637	0.3284	**0.3416**	4.02
	MRR@20	0.1278	0.1416	0.3031	0.3760	**0.3876**	3.09
	NDCG@5	0.0412	0.0498	0.1976	0.2552	**0.2623**	2.78
	NDCG@10	0.0488	0.0541	0.2133	0.2853	**0.3006**	5.36
	NDCG@20	0.0705	0.0735	0.2597	0.3256	**0.3316**	1.84
Cold Start	Prec@1	0.0250	0.0500	0.0503	**0.0625**	**0.0625**	0.00
	Prec@5	0.0225	0.0275	0.0287	0.0325	**0.0350**	7.69
	Prec@10	0.0163	0.0213	0.0265	0.0263	**0.0288**	9.51
	Prec@20	0.0175	0.0194	0.0239	0.0206	**0.0213**	3.40
	MRR@5	0.0794	0.0872	0.0922	0.1008	**0.1056**	4.76
	MRR@10	0.0823	0.0953	0.1003	0.1143	**0.1206**	5.51
	MRR@20	0.0846	0.1067	0.1107	0.1237	**0.1289**	4.20
	NDCG@5	0.0278	0.0365	0.0744	0.1029	**0.1103**	7.19
	NDCG@10	0.0345	0.0459	0.0953	0.1354	**0.1404**	3.69
	NDCG@20	0.0530	0.0635	0.1184	0.1604	**0.1658**	3.37

Table 3. Results of top-N recommendation on Yelp

Views	Metrics	BPRMF	NCF	CKE	RKGE	DHKGE	Improve (%)
All Users	Prec@1	0.0038	0.0050	0.0076	0.0093	**0.0102**	9.68
	Prec@5	0.0041	0.0044	0.0053	0.0078	**0.0085**	8.97
	Prec@10	0.0039	0.0043	0.0050	0.0066	**0.0075**	13.64
	Prec@20	0.0036	0.0041	0.0047	0.0058	**0.0063**	8.62
	MRR@5	0.0134	0.0164	0.0171	0.0194	**0.0210**	8.25
	MRR@10	0.0162	0.0181	0.0195	0.0231	**0.0252**	9.09
	MRR@20	0.0185	0.0206	0.0226	0.0270	**0.0286**	5.93
	NDCG@5	0.0072	0.0084	0.0197	0.0243	**0.0253**	4.12
	NDCG@10	0.0103	0.0118	0.0213	0.0320	**0.0339**	5.94
	NDCG@20	0.0148	0.0172	0.0256	0.0418	**0.0432**	3.35
Cold Start	Prec@1	0.0031	0.0035	**0.0064**	0.0061	0.0062	−3.13
	Prec@5	0.0030	0.0034	0.0045	0.0048	**0.0050**	4.17
	Prec@10	0.0028	0.0031	0.0036	0.0040	**0.0043**	7.50
	Prec@20	0.0027	0.0030	0.0031	0.0032	**0.0035**	9.38
	MRR@5	0.0108	0.0114	0.0117	0.0121	**0.0127**	4.96
	MRR@10	0.0134	0.0142	0.0144	0.0143	**0.0150**	4.90
	MRR@20	0.0149	0.0161	0.0162	0.0159	**0.0168**	3.70
	NDCG@5	0.0069	0.0079	0.0134	0.0149	**0.0157**	5.37
	NDCG@10	0.0100	0.0114	0.0187	0.0200	**0.0212**	6.00
	NDCG@20	0.0151	0.0170	0.0212	0.0259	**0.0274**	5.79

models can encode semantic paths more efficiently than one type of neural network, because by extracting the local features of the path and encoding the sequence information in the path, DHKGE can generate a more comprehensive path representation for recommendation.

3. On most metrics, the performance improvement of DHKGE in the "All Users" view is higher than in the "Cold Start" view. This indicates that in the "Cold Start" view, the quality and quantity of the semantic paths extracted from the KG are limited, which affects the recommendation performance of DHKGE since this method relies on the user's historical interaction information to make recommendations.

Based on these findings, we can draw the conclusion that DHKGE's recommendation performance is generally better than RKGE and other comparison methods.

5 Conclusions

To overcome the weaknesses of existing KG path-based recommendation methods, in this paper we propose the DHKGE method for top-N recommendation. DHKGE exploits a deep hybrid model to encode the path between users and items, and uses the attention mechanism to distinguish the importance of multiple semantic paths between a user-item pair. Experiments on the MovieLens 100K and Yelp datasets show that

DHKGE overall outperforms RKGE and several typical recommendation methods in terms of Precision@N, MRR@N, and NDCG@N. In future work, we plan to improve our method by adding entity relations to path embeddings.

References

1. Aggarwal, C.C.: Evaluating recommender systems. Recommender Systems, pp. 225–254. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29659-3_7
2. Cao, Y., Wang, X., He, X., Hu, Z., Chua, T.: Unifying knowledge graph learning and recommendation: towards a better understanding of user preferences. In: Proceedings of the 28th International Conference on World Wide Web, WWW 2019, pp. 151–161. ACM (2019). https://doi.org/10.1145/3308558.3313705
3. Catherine, R., Cohen, W.W.: Personalized recommendations using knowledge graphs: a probabilistic logic programming approach. In: Proceedings of the 10th ACM Conference on Recommender Systems, RecSys 2016, pp. 325–332. ACM (2016). https://doi.org/10.1145/2959100.2959131
4. Da'u, A., Salim, N.: Recommendation system based on deep learning methods: a systematic review and new directions. Artif. Intell. Rev. **53**(4), 2709–2748 (2020). https://doi.org/10.1007/s10462-019-09744-1
5. Goodfellow, I.J., Bengio, Y., Courville, A.C.: Convolutional networks. In: Goodfellow, I.J., Bengio, Y., Courville, A.C. (eds.) Deep Learning, pp. 330–372. MIT Press (2016). http://www.deeplearningbook.org/contents/convnets.html
6. Goodfellow, I.J., Bengio, Y., Courville, A.C.: Sequence modeling: recurrent and recursive nets. In: Goodfellow, I.J., Bengio, Y., Courville, A.C. (eds.) Deep Learning, pp. 373–420. MIT Press (2016). http://www.deeplearningbook.org/contents/rnn.html
7. Guo, Q., Zhuang, F., Qin, C., et al.: A survey on knowledge graph-based recommender systems. CoRR abs/2003.00911 (2020). https://arxiv.org/abs/2003.00911
8. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chua, T.: Neural collaborative filtering. In: Proceedings of the 26th International Conference on World Wide Web, WWW 2017, pp. 173–182. ACM (2017). https://doi.org/10.1145/3038912.3052569
9. Hu, B., Shi, C., Zhao, W.Z., Yu, P.S.: 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, KDD 2018, pp. 1531–1540. ACM (2018). https://doi.org/10.1145/3219819.3219965
10. Kim, D., Park, C., Oh, J., Lee, S., Yu, H.: Convolutional matrix factorization for document context-aware recommendation. In: Proceedings of the 10th ACM Conference on Recommender Systems, RecSys 2016, pp. 233–240. ACM (2016). https://doi.org/10.1145/2959100.2959165
11. Kim, Y.: Convolutional neural networks for sentence classification. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, pp. 1746–1751. ACL (2014). https://doi.org/10.3115/v1/D14-1181
12. Lee, H., Ahn, Y., Lee, H., Ha, S., Lee, S.: Quote recommendation in dialogue using deep neural network. In: Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, pp. 957–960. ACM (2016). https://doi.org/10.1145/2911451.2914734

13. Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-Thieme, L.: BPR: Bayesian personalized ranking from implicit feedback. In: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2009, pp. 452–461. AUAI Press (2009). https://dslpitt.org/uai/papers/09/p452-rendle.pdf
14. Sun, Z., Yang, J., Zhang, J., Bozzon, A., Huang, L., Xu, C.: Recurrent knowledge graph embedding for effective recommendation. In: Proceedings of the 12th ACM Conference on Recommender Systems, RecSys 2018, pp. 297–305. ACM (2018). https://doi.org/10.1145/3240323.3240361
15. Wang, X., Wang, D., Xu, C., He, X., Cao, Y., Chua, T.: Explainable reasoning over knowledge graphs for recommendation. In: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, pp. 5329–5336. AAAI Press (2019). https://doi.org/10.1609/aaai.v33i01.33015329
16. Xu, W., Xu, Z., Ye, L.: Computing user similarity by combining item ratings and background knowledge from linked open data. In: Meng, X., Li, R., Wang, K., Niu, B., Wang, X., Zhao, G. (eds.) WISA 2018. LNCS, vol. 11242, pp. 467–478. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02934-0_43
17. Yang, Z., Yang, D., Dyer, C., He, X., Smola, A.J., Hovy, E.H.: Hierarchical attention networks for document classification. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics, HLT-NAACL 2016, pp. 1480–1489. ACL (2016). https://doi.org/10.18653/v1/N16-1174
18. Yu, X., Ren, X., Sun, Y., Sturt, B., Khandelwal, U., Gu, Q., Norick, B., Han, J.: Recommendation in heterogeneous information networks with implicit user feedback. In: Proceedings of the 7th ACM Conference on Recommender Systems, RecSys 2013, pp. 347–350. ACM (2013). https://doi.org/10.1145/2507157.2507230
19. Zhang, F., Yuan, N.J., Lian, D., Xie, X., Ma, W.: Collaborative knowledge base embedding for recommender systems. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2016, pp. 353–362. ACM (2016). https://doi.org/10.1145/2939672.2939673
20. Zhang, S., Yao, L., Sun, A., Tay, Y.: Deep learning based recommender system: a survey and new perspectives. ACM Comput. Surv. **52**(1), 5:1–5:38 (2019). https://doi.org/10.1145/3285029