Knowledge-guided Deep Reinforcement Learning for Interactive Recommendation

Xiaocong Chen¹, Chaoran Huang¹, Lina Yao¹, Xianzhi Wang², Wei Liu¹, Wenjie Zhang¹
¹School of Computer Science and Engineering, University of New South Wales, Australia
²School of Computer Science, University of Technology Sydney, Australia
¹{xiaocong.chen, chaoran.huang, lina.yao, wei.liu, wenjie.zhang}@unsw.edu.au
²xianzhi.wang@uts.edu.au

Abstract—Interactive recommendation aims to learn from dynamic interactions between items and users to achieve responsiveness and accuracy. Reinforcement learning is inherently advantageous for coping with dynamic environments and thus has attracted increasing attention in interactive recommendation research. Inspired by knowledge-aware recommendation, we proposed Knowledge-Guided deep Reinforcement learning (KGRL) to harness the advantages of both reinforcement learning and knowledge graphs for interactive recommendation. This model is implemented upon the actor-critic network framework. It maintains a local knowledge network to guide decision-making and employs the attention mechanism to capture long-term semantics between items. We have conducted comprehensive experiments in a simulated online environment with six public real-world datasets and demonstrated the superiority of our model over several state-of-the-art methods.

Index Terms—Recommender System, Reinforcement Learning, Deep Neural Network

I. INTRODUCTION

Recommender systems have been widely used by industry giants such as Amazon, YouTube, and Netflix to identify relevant, personalized content from large information spaces. Modern recommendation systems are facing severe pressures for coping with emerging new users, ever-changing pools of recommendation candidates, and context-dependent interests [1]. In contrast, traditional recommendation methods focus on modeling user’s consistent preferences and may not reflect high dynamics in user interest and environments. In such situations, interactive recommendation rises as an effective solution that incorporates dynamic recommendation processes to improve the recommendation performance. An interactive recommendation system would recommend items to an individual user and then receive the feedback to adjust its policies during the iterations [2]. Many studies model interaction recommendation as a Multi-Armed Bandit (MAB) problem [3]–[5]. Such methods generally assume a user’s preference is consistent during the recommendation and focus on the trade-off between immediate and future rewards. Therefore, they face challenges for handling environments with dynamically changing user preference or interest. Reinforcement learning (RL) is a promising approach to interactive recommendation. Considerable efforts have shown the outstanding performance of RL methods in recommendation systems [6]–[8].

We proposed a novel model KGRL where the knowledge graph is introduced into the reinforcement learning pro-
TABLE I: Main notations

| Symbols | Meaning |
|---------|---------|
| \(\mathcal{U}\) | Set of users |
| \(\mathcal{I}\) | Set of items |
| \(\mathcal{R}\) | Set of relations |
| \(\mathcal{E}\) | Set of entities |
| \(|\cdot|\) | Number of unique elements in \(\cdot\) |
| \(\mathcal{S}_{u,t}\) | User \(u\)'s recent actions before timestamp \(t\) |
| \(\mathcal{G} = (\mathcal{E}, \mathcal{R})\) | Constructed Knowledge Graph |
| \(\mathbf{E}\) | Item embedding |
| \(W\) | Parameter matrices |
| \(S_t\) | State space at timestamp \(t\) |
| \(\alpha_t\) | Action space at timestamp \(t\) |
| \(d\) | Dimension of the latent space |

cess to help the agent make decisions.

- To improve the efficiency, we maintain a local knowledge network which is based on the knowledge graph, to fasten the process while keeping the performance;
- Comprehensive experiments in the simulated online environment with six real-world datasets prove the performance of our propose approach.

II. PROBLEM DEFINITION

An interactive recommendation system features incorporating user’s feedback dynamically during the training process. Given a set of users \(\mathcal{U} = \{u, u_1, u_2, u_3, \ldots\}\) and a set of items \(\mathcal{I} = \{i, i_1, i_2, i_3, \ldots\}\), the system first recommends item \(i_1\) to user \(u_1\) and then gets a feedback \(x\). The system aims to incorporate feedback to improve future recommendations. To this end, it needs to figure out an optimal policy \(\pi^*\) regarding which item to recommend to the user to achieve positive feedback. We can formulate the problem as a Markov Decision Process (MDP) by treating the user as the environment and the system as the agent. We define the key components of the MDP as follows (Table I summarizes the main notations used in this paper):

- State: A state \(S_t\) is determined by the recent \(l\) items in which the user was interested before time \(t\).
- Action: Action \(\alpha_t\) represents a user’s dynamic preference at time \(t\) as predicted by the agent.
- Reward: Once the agent chooses a suitable action \(\alpha_t\) based on the current state \(S_t\) at time \(t\), the user will receive the item recommended by the agent. The user’s feedback on the recommended item (i.e., clicking the item, ignoring it) accounts for the reward \(r(S_t, \alpha_t)\), which will be considered to improve the recommendation policy \(\pi\).
- Discount Factor \(\gamma\): The discount factor \(\gamma \in [0, 1]\) is used to balance the future and immediate rewards—the agent will fully focus on the immediate reward when \(\gamma = 0\) and take into account all the (immediate and future) rewards otherwise.

III. METHODOLOGY

Our approach involves two steps: knowledge preparation and deep reinforcement recommendation.

A. Knowledge Preparation

We construct the knowledge graph based on entity-relation-entity tuples \(\{i, r, j\}, i, j \in \mathcal{E}, r \in \mathcal{R}\). For example, the tuple \((\text{The Elements of Style}, \text{book.author}, \text{William Strunk Jr.})\) means that William Strunk Jr authored the book \textit{The Elements of Style}. We consider every item (e.g., \textit{The Elements of Style}) as an entity in the knowledge graph \(\mathcal{G}\) and transform the knowledge graph to represent user’s preference more precisely [15]. Given a user \(u \in \mathcal{U}\) and an item \(q \in \mathcal{I}\), suppose \(\mathcal{D}(i)\) is the set of items that has direct relationship with item \(i\) and \(j\). We calculate the user-specific relation scores as follows:

\[
f_{ui}^{ij} = g(u, r_{ij}) \quad \text{where} \quad g : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}
\]

where \(g\) is a scoring function (e.g., inner product) to compute the score between user and relation; \(d\) is the dimension of user representation and relation representation; \(u \in \mathbb{R}^d, r_{ij} \in \mathbb{R}^d\), \(f_{ui}^{ij}\) measures the importance of \(r_{ij}\) to user \(u\).

Let \(\mathcal{D}(i)\) be the set of candidates to recommend, we normalize the user-specific relation scores as follows:

\[
\overline{f}_{ui}^{ij} = \frac{f_{ui}^{ij}}{\sum_{d \in \mathcal{D}(i)} f_{ui}^{jd}} \in [0, 1]
\]

Inspired by [16], we transform the knowledge graph into a user-specific graph \(A_u\), which is an adjacency matrix of \(\mathbb{R}^{\mathcal{I} \times \mathcal{I}}\). In this matrix, each position \((i, j)\) corresponds to a score \(\overline{f}_{ui}^{ij}\), and a higher score indicates a stronger relation between two items \(i\) and \(j\).

B. Deep Reinforced Recommendation

We develop our recommendation model (Figure 1) based on the Actor-Critic reinforcement learning framework [17], where the actor generates actions, the critic evaluates actions, and the actor network updates the policy based on the suggestion made by the critic.

1) Actor Network \(\phi\): Given a current state \(S_t\), the actor network employs a neural network to infer an optimal policy \(\pi^*\) to work out an action \(\alpha_t\). Given \(S_t\), which consists of user’s recent interests (shown in Figure 1), we first obtain vector representation of user’s recent interest via embedding. Suppose we have a set of user recently interested items before time \(t\), \(S_{u,t} = \{S_{u,1}, S_{u,2}, \ldots, S_{u,l}\}\). The actor network takes an input sequence \(S_{u,t}\) and the corresponding feedback sequence \(\{F_{u,1}, F_{u,2}, \ldots, F_{u,l}\}\) to deliver an output sequence \(\{S_{u,1}^2, S_{u,2}^3, \ldots, S_{u,l+1}^d\}\). Given an original item embedding matrix \(M \in \mathbb{R}^{\mathcal{I} \times \mathbb{R}^d}\), to preserve the order of user’s previously interested items, which updates the item embedding into the following:

\[
\mathcal{E} = \begin{bmatrix}
M_1 + P_1 \\
M_2 + P_2 \\
\vdots \\
M_l + P_l
\end{bmatrix}
\]

We then fed this embedding into a self-attention layer to reduce impurity in the embedding [19]. The layer uses the
scaled dot-product attention \[^{[20]}\]
which is originally defined as follows:

\[
\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right)V
\]

where \(Q, K, V\) denotes queries, keys, and values, respectively; \(\sqrt{d_k}\) is the scaling factor to regulate the value range of \(QK^T\). After applying the embedding \(\mathbb{E}\) as the input, the attention turns into the following:

\[
\text{Attention}(\mathbb{E}W^Q, \mathbb{E}W^K, \mathbb{E}W^V)
\]

where \(\mathbb{E}W^Q, \mathbb{E}W^K, \mathbb{E}W^V \in \mathbb{R}^{d \times d}\). We fed this embedding into two fully connected layers, which use ReLU and tanh as the activation functions, respectively as described in \[^{[18]}\]. The output of the attention layer is the state \(S_t\) at time \(t\).

2) Critic Network \(\psi\): We design the critic network to estimate the Q-value function \(Q(S_t, a_t)\) to evaluate actor’s policy. The critic network takes state representation \(S_t\) and action representation \(a_t\) as the input (shown in Figure 1). We design a local knowledge network within the critic network to capture the high-order structural proximity among the items in the knowledge graph using graph convolutional network (GCN). Specifically, given a user-specific graph \(g_u^k\) generated from the current state \(S_t\), we feed it into a two-layer GCN that applies the following layer-wide propagation rule:

\[
H^{l+1} = \sigma(D^{-\frac{1}{2}}\hat{A}_u D^{-\frac{1}{2}}H^lW^l)
\]  \((1)\)

where \(H^{l+1}\) is the representation of entities at layer \(l + 1\); \(A_u\) is the input matrix that aggregates the neighbour’s entities; \(\hat{A}_u\) is set to \(A_u + I\), where the \(I\) is an identity matrix used to avoid negligence of the old representation via self-connection; \(D_u\) is the diagonal degree matrix for \(\hat{A}_u\) where \(D_u^{ii} = \sum_j \hat{A}_{uj}\) (the symmetric normalization was applied to keep the representation \(H^1\) stable, as denoted by \(D^{-\frac{1}{2}}\hat{A}_u D^{-\frac{1}{2}}\)); \(W^l\) is the weight matrix for layer \(l\); and \(\sigma(\cdot)\) denotes the non-linear activation function.

Recent research shows the feasibility of searching in graphs processed by GCN \[^{[21]}\]. Since GCN capture’s all the structural information in the knowledge graph, it will not affect the search results. In this study, we assume an unweighted graph where a user is equally interested in every item. Then, we start searching with the actor predicted action \(a_t\) (i.e., predicted item \(i_p\)) to the real target \(i_s\), based on the user’s personalized interest knowledge (i.e., trained graph with all parameters \(\theta_{kg}\)). Finally, we calculate the Q value by estimating the reward \(r\) based on the distance between the predicted item and the target item:

\[
r = \frac{100}{\sqrt{\text{Distance}(i_p, i_s) + \epsilon}} \times W_{pt}
\]

where \(W_{pt}\) is the sum of weight of the shortest path from \(i_p\) to \(i_s\); \(\epsilon\) is the parameter to avoid the denominator becoming 0. We calculate the distance using the Dijkstra’s algorithm with MinHeap.
C. Complexity Analysis

We analyze the time and space complexity of the critic network, especially the search part, in this section. We consider a vector composition (i.e., the combination of the state vector and action vector) and assume the transmission time as a constant \( c \). Given a user interested in \( I_u \) items, we consider the worst case—a complete graph and each item \( i \) having \( M \) nearest non-duplicate neighbours. Thus, we get a graph with \( I_u + I_u M \) nodes (exclude the centralised user node) and \((I_u + I_u M)(I_u + I_u M - 1)/2\) edges. We then calculate the time and space complexity as \( O((I_u + I_u M)^2 + |I_u + I_u M| \log |I_u + I_u M|) \sim O((I_u + I_u M)^2) \) and \( O(2(I_u + I_u M)) \sim O(|I_u + I_u M|) \). In comparison, if we feed the environment knowledge graph to the critic network directly, the time and space complexity would be \( O(|I + IM|^2) \) and \( O(|I + IM|) \). Apparently, the local knowledge network significantly improves the performance and saves the memory space in our model \( (I_u \ll I) \). Moreover, the local knowledge network is easier to converge as it has fewer nodes than the environment knowledge graph.

D. Training Strategy

Training the actor-critic network requires train two parts of the neural network simultaneously. We apply the Deep Deterministic Policy Gradient (DDPG) (Algorithm 1) to train our model \([22]\), where we train the critic by minimising a loss function:

\[
l(\theta_\psi) = \frac{1}{N} \sum_{j=1}^{N} ((r + \gamma \xi) - \psi_{\theta_\psi}(S_t, a_t))^2
\]

where \( \xi = \psi_{\theta_\psi}'(S_{t+1}, \phi_{\theta_\phi}(S_{t+1})) \)

where \( \theta_\psi \) is the parameter in critic; \( \theta_\phi \) is the parameter in actor; \( N \) is the size of mini-batch from the replay buffer; \( \psi_{\theta_\psi}' \) and \( \phi_{\theta_\phi}' \) are the target critic and target actor network, respectively.

Algorithm 2 describes the training of the local knowledge network, where we define the same loss function for all users for the local knowledge network:

\[
l_k = \sum_{u \in U} \left( \sum_{v \in y_{ui}} J(y_{ui}, \bar{y}_{ui}) \right)
\]

where \( J \) is the cross-entropy; \( y_{ui} \) is a piece-wise function to reflect the interest/action (defined below):

\[
y_{ui} = \begin{cases} 1 & \text{if } u \text{ interested in } i \\ 0 & \text{otherwise} \end{cases}
\]

IV. Experiments

In this section, we report our experimental evaluation of our model in comparison with several state-of-the-art models using real-world datasets.

Algorithm 1: DDPG algorithm for our model

1. Initialize actor network \( \phi \) with parameter \( \theta_\phi \) and critic network \( \psi \) with parameter \( \theta_\psi \) randomly;
2. Initialize target network \( \phi' \) and \( \psi' \) with weight \( \theta_\phi \), \( \theta_\psi \) respectively;
3. Initialize the local knowledge network;
4. Initialize Replay Buffer \( B \);
5. for \( i = 0 \) to \( n \) do
   6.   Receive the initial state \( S_i \);
   7.   for \( t = 1 \) to \( T \) do
      8.     Infer a action \( a_t \) according to the \( \phi(\cdot) \);
      9.     Execute the action \( a_t \) to receive a reward \( r_t \) and observe a new state \( S_{t+1} \);
      10.    \( B \).append(\( S_t, a_t, r_t, S_{t+1} \));
      11.    Sample a random mini-batch of \( N \) transitions \((S_k, a_k, r_k, S_{k+1})\) from \( B \);
      12.    Set \( y_i = r_i + \gamma \xi \);
      13.    Update Critic by minimise the loss \( l(\theta_\phi) \);
      14.    Update local knowledge net by Algorithm 2;
      15.    Update the Actor policy by using the sampled policy gradient:
      16.    \( \nabla_{\theta_\phi} = \frac{1}{N} \sum_{j=1}^{N} \nabla_\phi \psi(S_i, a_t) | a = \phi(S_k) \nabla_\phi \phi(S_k) \)
      17.    Update target network:
      18.    \( \theta_\phi' \leftarrow \tau \theta_\phi + (1 - \tau) \theta_\phi' \);
      19.    \( \theta_\psi' \leftarrow \tau \theta_\psi + (1 - \tau) \theta_\psi' \);
   20. end

Algorithm 2: Training the local knowledge network

input: The user specific graph \( g_i^n \), environment KG \( G_e \)
1. Initialize the parameters for GCN \( \theta \) ;
2. Initialize the depth of graph \( d_g \) ;
3. Initialize the reward storage \( P \) ;
4. for \( i \) in \( g_i^n \) do
   5.   Receive the reward \( r \) from \( G_e \) ;
   6.   \( P \).append(\( r \));
   7. end
8. \( r = \min(P) \);
9. while GCN is not converge do
   10. if \( d_g < r \) then
       11.   aggregate next level’s neighbours into \( g_i^n \)
       12. end
   13. Update the GCN and its corresponding \( \theta \);
14. end

A. Datasets

We conducted experiments on six public real-world datasets (Table II shows the statistics). All these datasets provide the necessary information for building the respective knowledge graphs.
## TABLE II: Statistics of our experimental datasets

| Dataset          | # of users | # of items | # of interactions |
|------------------|------------|------------|-------------------|
| Amazon CD        | 75,255     | 64,443     | 3,749,004         |
| Librarything     | 73,882     | 337,561    | 979,053           |
| Book-Crossing    | 278,858    | 271,379    | 1,149,780         |
| GoodReads        | 808,749    | 1,561,465  | 225,394,930       |
| MovieLens-20M    | 138,493    | 27,278     | 20,000,263        |
| Netflix          | 480,189    | 17,770     | 100,498,277       |

**Book-Crossing**: This dataset contains user’s demographic information and book information from the Book-Crossing community. It is extremely sparse with a density of 0.0041%.

**MovieLens-20M**: This is a well-known benchmark dataset that contains 20 million ratings from around 140 thousand users on the MovieLens website. It also provides movie tags, which can be used to build relations in the knowledge graph.

**Librarything**: This dataset contains book review information collected from the librarything website.

**Amazon CDs and Vinyl**: This is a highly sparse dataset that contains the product metadata, user reviews, ratings, and item relations, as part of the Amazon e-commerce dataset.

**Netflix Prize**: This dataset contains 100 million ratings from 480 thousand users and item information for yearly open competition to improve Netflix’s recommendation performance.

**Goodreads**: This dataset contains user’s ratings and reviews to books on the Goodreads book review website.

### B. Evaluation Metrics

We evaluate the performance of recommendation using three metrics: precision, recall, and normalized Discounted Cumulative Gain (nDCG). All the metrics were calculated based on the top-10 recommendations to each user for each test case. To ease processing, we removed users who have fewer than ten interactions and scaled the ratings from all datasets to the range of $[0, 5]$. Only the items with a rating score higher than three were considered a relevant item.

### C. Experimental Setup

We evaluated our model in a simulated online environment built upon offline public datasets, using the algorithm proposed in [9] and the aforementioned reward function. This way, we avoided collecting private user information and expensive online training [23]. Specifically, the simulator generated feedback based on logistic matrix factorization (LMF) [24]. We randomly split each dataset into a training set (70%), a validation set (10%), and a testing set (20%) to conduct 10-fold cross-validation. The discount factor $\gamma$ was initialized to 0.99.

### D. Compared Methods

We compared our model with several competitive baselines:

- **Policy-Guided Path Reasoning (PGPR)** [13]: A state-of-the-art knowledge-aware model that employs reinforcement learning for explainable recommendation.

- **Tree-structured Policy Gradient Recommendation (TPGR)** [25]: A state-of-the-art model that uses reinforcement learning and binary tree for large-scale interactive recommendation.

- **HLinearUCB** [3]: A contextual-bandit approach that learns extra hidden features for each arm to model the reward for interactive recommendation.

- **Wolpertinger** [6]: A deep reinforcement learning framework that uses DDPG and KNN for recommendations in large discrete action spaces.

- **DeepPage** [7]: A DDPG-based reinforcement learning model that learns a ranking vector for page-wise recommendation.

- **DRN** [8]: A DQN-based recommendation method that employs deep Q learning to estimate Q-value for news recommendation.

- **FactorUCB** [26]: A matrix factorization-based bandit algorithm for interactive recommendation.

- **ICTRUCB** [5]: A MAB approach that uses a depend arm for online interactive collaborative filtering.

### E. Results

Table III shows our evaluation results of recommendation models. We observed that our model outperformed all the baselines in all metrics almost on all the datasets—it performed only slightly worse than TPGR on the Book-Crossing dataset. This may be attributed to the specific design of TPGR to deal with large-scale datasets. None of the MAB-based methods (HLinearUCB, FactorUCB and ICTRUCB) performed well on those datasets because they all assume static user interest and may not give up-to-date recommendations. We also observed that PGPR performed worse than DRN on the Amazon CD and Book-Crossing datasets—these sparse datasets might not provide sufficient relation for PGPR to infer the recommendation path. Finally, all the models achieved their best results on the MovieLens-20M dataset, given the rich information and dense relation in the dataset.

### F. Ablation and Complexity Studies

We conducted ablation studies to explore the impact of the attention mechanism and local knowledge network on the performance of our model on the above six datasets. We selectively choose MovieLens-20M and the Book-Crossing as the example because the Book-Crossing dataset is the most sparse one and the MovieLens-20M is the most dense one; they can show the capability of our model in the normal case and extreme case. Due to the exponential increase in time usage, we only show the first five level of neighbours. The results (Figure 2a,d) show that our model’s performance dropped slightly (by 1% in precision, 2% in recall, and 1% in nDCG) without the attention mechanism while elevated...
slightly without the local knowledge network because the model already contains all the information, including abundant relation between items to support the decision making.

We also used valgrind\footnote{http://www.valgrind.org/} to monitor the memory usage, which, on the other hand, reveals the huge advantages of using a local knowledge network in reducing both the time and space complexity (also see Figure 2(b)). We mentioned that in figure 2(c,f), the model $M-K$ have an incredible increase in time consumption when the level goes over 2. One possible reason is that as the level goes higher, the graph comes more and more complex, which will affect the search critically.

V. RELATED WORK

Most existing work models interactive recommendation as a Multi-Armed Bandit (MAB) problem. And the primary solution lies in finding an Upper Confidence Bound (UCB). Li et al.\cite{li2015learning} employ the first linear model to calculate the UCB for each arm. Since then, many researchers combine other techniques such as matrix factorization, to find the UCB\cite{wang2015personalizing}. For example, Wang et al.\cite{wang2015personalizing} proposed a new approach by choosing a dependent arm to calculate the UCB; Shen et al.\cite{shen2017deep}, instead, use deep learning-based methods to solve MAB.

Recent studies have shown the effectiveness of reinforcement learning in modeling interactions-related recommendation processes, where the recommendation problems are usually formulated as Markov Decision Processes. One approach is based on Deep Q-learning (DQN)\cite{dqn}, which maximizes the Q-value from the predicted item and the target item. Zheng et al.\cite{zheng2016learning} combine the DQN with the Dueling Bandit Gradient Decent (DBGD)\cite{dueling} policy to recommend news. Another thread of methods is DDPG-based\cite{ddpg}. Such methods aim to let the agent learn a proper policy instead of using the Q-value. For example, Liu et al.\cite{liu2017diversification} adopt DDPG to promote the diversity in interactive recommendation; Zhao et al.\cite{zhao2018deep} use DDPG for page-wise recommendation. It is also worth mentioning that knowledge graphs can be useful for providing guidance in explainable recommendation\cite{chen2018knowledge}. Knowledge-aware recommendation systems heavily rely on the use of relation inference to generate paths for recommendations\cite{yu2018learning}. Wang et al.\cite{wang2019explainable} show graph convolutional network can help learn neighbour representations and thus boost the recommen-
TABLE III: The overall results of our model comparison with several state-of-the-arts models in different datasets. The result was reported by using the percentage and based on top-10 recommendation as mentioned before. The highlighted result in bold is the best result.

| Dataset | Recall | Precision | nDCG |
|---------|--------|-----------|------|
| Amazon CD | | | |
| Wolpertinger | 1.542 ± 0.192 | 1.527 ± 0.145 | 3.331 ± 0.201 |
| HLinearUCB | 3.112 ± 0.331 | 2.647 ± 0.171 | 4.005 ± 0.341 |
| FactorUCB | 3.531 ± 0.232 | 4.512 ± 0.242 | 6.012 ± 0.251 |
| ICTRU/UCB | 4.124 ± 0.293 | 3.110 ± 0.395 | 5.982 ± 0.662 |
| DeepPage | 7.124 ± 0.181 | 4.127 ± 0.134 | 7.245 ± 0.154 |
| DRN | 8.006 ± 0.232 | 4.234 ± 0.241 | 6.112 ± 0.241 |
| TPGR | 7.294 ± 0.312 | 2.872 ± 0.531 | 6.128 ± 0.541 |
| PGPR | 6.619 ± 0.123 | 1.892 ± 0.143 | 5.970 ± 0.131 |
| Ours | **8.208 ± 0.241** | **4.782 ± 0.341** | **7.876 ± 0.511** |

| Librarything | Recall | Precision | nDCG |
|------------|--------|-----------|------|
| Wolpertinger | 0.782 ± 0.121 | 1.235 ± 0.131 | 0.976 ± 0.242 |
| HLinearUCB | 2.421 ± 0.131 | 1.724 ± 0.141 | 2.865 ± 0.322 |
| FactorUCB | 3.123 ± 0.141 | 2.976 ± 0.223 | 3.536 ± 0.241 |
| ICTRU/UCB | 3.441 ± 0.121 | 3.421 ± 0.333 | 4.001 ± 0.321 |
| DeepPage | 5.124 ± 0.323 | 3.245 ± 0.142 | 6.976 ± 0.142 |
| DRN | 7.124 ± 0.122 | 4.123 ± 0.112 | 7.433 ± 0.142 |
| TPGR | 7.246 ± 0.321 | **4.523 ± 0.442** | **7.870 ± 0.412** |
| PGPR | 6.998 ± 0.112 | 3.932 ± 0.121 | 7.333 ± 0.133 |
| Ours | **8.004 ± 0.223** | **4.521 ± 0.332** | **7.459 ± 0.401** |

VI. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a knowledge-guided deep reinforcement learning framework (KGRL) for interactive recommendation. KGRL uses the critic-actor learning framework to harness the interaction between users and the recommendation system and employs a local knowledge network to improve the stability and quality of the critic network for better decision-making. Extensive experiments over an online simulator with six public real-world datasets demonstrate its superior performance over state-of-the-art models. To verify the effectiveness for each component, we conduct the ablation study for the local knowledge network and attention mechanism and selectively present the performance both in normal case and extreme case. We are planning to introduce various types of user information (e.g., user’s thought when browsing items) to enrich the interaction and deploy our model in online business platforms to further test the performance in the future.

In addition, the cold-start problem is another big challenge to be focused on. Besides, the algorithm used to train the model still lacks the knowledge about how to update step size will affect the training time and the convergence which can be solved in the future work.

REFERENCES

[1] S. Zhang, L. Yao, A. Sun, and Y. Tay, “Deep learning based recommender system: A survey and new perspectives,” ACM Computing Surveys (CSUR), vol. 52, no. 1, pp. 1–38, 2019.
[2] X. Zhao, W. Zhang, and J. Wang, “Interactive collaborative filtering,” in Proceedings of the 22nd ACM international conference on Information & Knowledge Management. ACM, 2013.
[3] H. Wang, Q. Wu, and H. Wang, “Learning hidden features for contextual bandits,” in Proceedings of the 25th ACM International Conference on Information and Knowledge Management. ACM, 2016, pp. 1633–1642.
[4] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, “Community preserving network embedding,” in Thirty-First AAAI Conference on Artificial Intelligence, 2017.
[5] Q. Wang, C. Zeng, W. Zhou, T. Li, S. S. Iyengar, L. Shwartz, and G. Grabarnik, “Online interactive collaborative filtering using multi-armed bandit with dependent arms,” IEEE Transactions on Knowledge and Data Engineering, 2018.
