Interactive Song Recommender

Dat Quoc Ngo
Erik Jonsson School of Engineering
Unviersity of Texas at Dallas)
Richardson, USA
dqn170000@utdallas.du

Gokul Ravi Kumar
Erik Jonsson School of Engineering
Unviersity of Texas at Dallas)
Richardson, USA
gxr180026@utdallas.edu

***Abstract*—Music streaming platforms like Spotify have successfully attracted a large number of users not only by their huge collection of licensed songs but also by their song recommendation system (RS). Collaborative filtering (CF) has been a to-go recommendation algorithm that learns the similarity of users based on their user-item interaction history. The drawback of CF is its limit in learning the dynamic user preference that could be solved by introducing Deep Neural networks into RS. A common issue of DNN methods is the requirement of a huge amount of user-interaction data to train an effective recommendation policy due to the rare user responses and the large action spaces consisting of a large number of candidate items. It is infeasible to collect much data which may hurt user experience. To solve the issues, our work proposes to utilize Knowledge Graph (KG) in DNN-based RS that provides rich information of the song correlation for the better candidate retrieval, for the richer representation of song and user states, and for better adapting to user preferences among correlated songs in KG. Our experiments *need experimental results here*.**

***Index Terms*—Interactive Recommender System, Deep Collaborative Filtering, Knowledge Graphs, Graph Neural Networks**

I. Introduction

Spotify's Weekly Discover is well-known for its intelligence in creating a 2-hour playlist of fresh and new songs that fit to users' music taste. A recent publication by Spotify introduces BaRT [3], a Deep Learning (DL) model that aims at exploring new songs in friends' playlists. A key advantage of BaRT over traditional Collaborative Filtering (CF) algorithms is the explanation and interpretation in recommendations that benefits the user experience in recommendation.

Inspired by the rewarding policy in BaRT [3], our work attempts to inject DL into Interactive Recommendation System (IRS) in order to solve limits of Collaborative Filtering (CF) in learning the dynamic user preference. Matrix Factorization (MF), a class of Collaborative Filtering, leverages Singular Value Decomposition to find similar friends based on their user-item interaction history and recommend new songs preferred by friends [4]. However, CF algorithms are treated as a single-step prediction task that is limited in adapting to users' changes in preference. On the other hand, IRS is a multi-step recommendation task that sequentially delivers a candidate item to the user at each time step and may receive feedback from her to derive the next recommendation. This approach based on the recommendation-feedback interaction aims at dynamically exploring users' new interests while providing accurate prediction [1].

One way to implement IRS is MAB methods [3], [6], [7] that balance the exploration and exploitation of users' preference. These methods are commonly modeled by a linear function that learns the interactions with proper exploration-exploitation trade-off. Recently, researchers proposed inject Deep Reinforcement Learning (DRL) into interactive recommender systems [8]–[10] to leverage the DRL's decision-making potential in the dynamic environment for the dynamic user preference [11]. A common drawback of DRL in IRS is the requirement of a large volume of interaction data and action spaces of candidate items. Collecting data may harm the user experience in recommendation. Unlike previous methods, our work formulates interactive recommendation system as a hybrid system of Deep Learning-based CF (e.g. Neural Collaborative Filtering [12]) enhanced with knowledge graph due to the potential of Graph Neural Networks in explicitly capturing the correlation of items and users [13].

The Netflix Prize competition [14] has shown the potential of matrix factorization in modeling the user-item representation to propose neighborhoods for recommendations. Then, due to the rise of Deep Learning, Neural Collaborative Filtering (NCF) was created by encoding user and items as continuous vectors (also known as embeddings) combined with traditional matrix factorization methods [12] to capture non-linear and linear user-item relationships. However, NCF provides no or limited interpretation of user-item relationships and of decision-making in recommendations. Fortunately, knowledge graph provides a rich prior knowledge of correlations between attributes of items and linked items if they share common attributes [15]. Hence, KG can explicitly demonstrate the correlations of users' selection history which is useful in explaining the decision making process in recommendations. Recently, Graph Neural Network models applied over KG of items and attributes have outperformed previous methods and achieved state-of-the-art results on public benchmark datasets [15], [16].

Our work proposes to leverage Graph Neural Network to enhance Neural Collaborative Filtering in music recommendation systems in order to provide better interpretation in decision-making of recommendations and to dynamically new items according to users' preference changes represented by correlations of past selection in knowledge graph. In details, our approach consists of:

- **Knowledge Graph** is an undirected graph composed of

item nodes (i.e. songs) and non-item nodes (i.e. release year, artist name). The KG provides rich information of items' attributes to represent users' selection history that delivers better interpretation in the recommendation decision-making.

- **Neural Collaborative Filtering** at each time step recommends 10 items which is the initial set to look fc relevant neighboring items in KG.
- **Graph Neural Network** consists of 2 sub-module Graph Convolution and Behavior Aggregation. Embec dings of items from the users' past selection from Grap Convolution are processed by Behavior Aggregation i the sequential setting to yield the current user-taste rep resentation. For each item in the users' past selectio its all neighbor item nodes in the KG are extracted t yield corresponding graph embeddings. Then, the use state representation and graph embeddings of all neighbc item nodes are input to Multilayer Perceptron (MLP) fc selecting final candidate recommendations.

II. Related Work (A: 50% B: 50%)

A. KG Enhanced Recommendation

The IEEEtran class file is used to format your paper and style the text. All margins, column widths, line spaces, and text fonts are prescribed; please do not alter them. You may note peculiarities. For example, the head margin measures proportionately more than is customary. This measurement and others are deliberate, using specifications that anticipate your paper as one part of the entire proceedings, and not as an independent document. Please do not revise any of the current designations.

B. Deep Learning in IRS

mplementation of Deep learning for recommender systems isn't new. The first model to perform well was the Neural collaborative filtering method that we have implemented in our project. The implementation of NCF is discussed in the below sections. Apart from the NCF, we have other Deep Learning based networks. One such network is the Wide and [5]. This model combines the generalization and memorization for deep recommender systems.Other methods use Deep reinforcement learning to learn better policy incorporating the user item interactions and learning to predict best item that can give highest possible reward which is the highest rating in our case. Recently, researchers proposed an approach to diversify the recommended items using deep reinforcement learning based methods such that we show items from various categories and also improve the highest possible rewards [18]. Another work proposes to use uses deep reinforcement learning for online advertising recommender system. This paper proposes a framework which continuously changes the advertising strategy such that it can maximize its reward in the long run [19]. With more advancements came the usage of knowledge graphs with deep learning systems to enhance the models better. A recent work uses a knowledge graph to encode the current state for an Reinforcement learning framework [1]. Based on this current state, it predicts the next item such that it can maximize the reward obtained. Past work shows that there are many implementations of Deep recommender systems. However in our case, we use the NCF model combined with knowledge graphs to enhance the prediction ("Fig. 1").

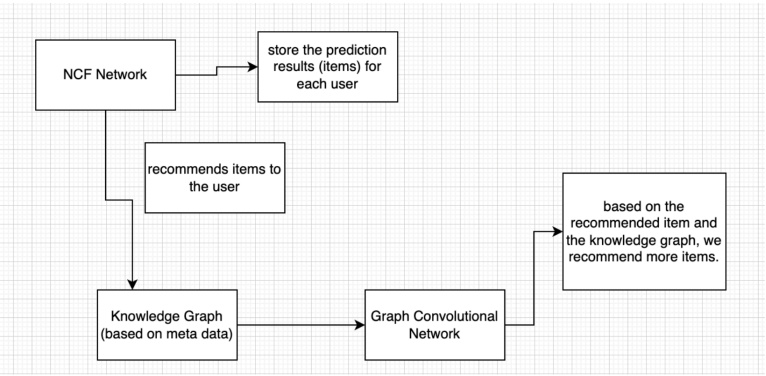

Fig. 1. The final goal of an interactive recommnder system.

III. Problem Formulation (A: 50% B: 50%

In feed streaming recommendation scenario, the interactive nature between the recommender system and the user is a multi-step interaction process that lasts for a period of time. At each timestep t, according to the observations on past interactions, the recommendation agent delivers an item $i_{\mathrm{t}}I$ to the user, and receives feedback (e.g., click, purchase or skip) from her. This process continues until the user leaves the recommender system. Under such circumstances, the interactive recommendation process can be formulated as a collection of users' past selection which is used to explore new interests/items given their attributes' correlation represented in the knowledgee graph. The ultimate goal of the recommender system is to learn a recommendation policy $\pi : S\beta I$, which maximizes the cumulative utility over the whole interactive recommendation process as

Notations	Descriptions
$\mathcal{U}, \mathcal{I}$	Set of users and items in IRS.
$\mathcal{G} = (\mathcal{E}, \mathcal{R})$	Knowledge graph.
$\mathcal{E}, \mathcal{R}$	Set of entities and relations in $\mathcal{G}$.
$o_t = \{i_1, i_2, ...i_n\}$	Recorded user's positively interacted at timestep t.
$\mathbf{s}_t$	Dense representation of user's preference at timestep t.
r_t	The user's reward at timestep t.
T	Episode length.
$\mathbf{e}_h, h \in \mathcal{E}$	Dense representation of an entity.
$\mathcal{I}_t(\mathcal{G})$	Candidate action space at timestep t.
θ_S	Parameters of state representation network.
θ_Q, θ'_Q	Parameters of online Q-network /target Q-network.
$\mathcal{D}$	Replay Buffer.

Fig. 2. Notation and Description. Since our approach is not RL-based, hence, T, r_t, D, θ_Q, θ'_Q are not used in following equations

Here $s_{\mathrm{t}}S$ is a representation abstracted from user's positively interacted items $o_{\mathrm{t}} = i_1, i_2, ...i_{\mathrm{n}}$ that denotes user's preference

at time step t; (s_t, i_t) is the user's immediate feedback to the recommended item it at the state s_t.

Here G is the knowledge graph comprised of subject-property-object triples facts, e.g., triple (Adam Elvin, SingerOf, One More Night), which denotes Nolan is the director of Inception. And it is often present as (head, relation, tail), $headE, relationR, tailEandE$, R denote the set of entities and relationships in G, respectively. Usually, an item iI can be linked to an entity eE in the knowledge graph, e.g., the movie Godfather from MovieLens dataset has a corresponding entity entry in DBpedia. We will introduce how we design the knowledge enhanced DRL framework for IRS in the following sections and the key notations used in this paper are summarized in "Fig. 2"

IV. Approach (A: 50% B: 50%)

Inspired by the Knowledge-Graph-enahanced Interaction Recommendation System [1] framework, our approach modifies the above framework to consists of Neural Collaborative Filtering, Graph Convolutional Network, and Knowledge Graph as in "Fig. 3".

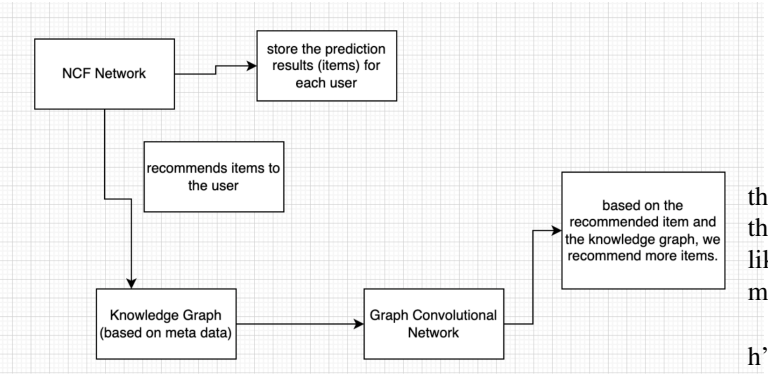

Fig. 3. Our overall framework. The recommended items from the NCF network is input to Knowledge Graph and Graph Convolutionial Network to find most relevant candidates for recommendation.

A. Graph Convolutional Network

In details, our graph network consists of two layers: Graph Convolutional Embedding Layer and Behavior Aggregation Layer in "Fig. 4"

1) Graph Convolutional Embedding Layer: Generally, the state representation in IRS is abstracted from the user's clicked items, since the positive items represent the key information about what the user prefers [23]. Given the user's history, we first convert the clicked item set $\{i_t\}$ into embedding vectors $i_t \in R^d$, where d is the dimension of the embeddings. Since we have already linked items with entities in KG, we can take advantage of the semantic and correlation information among items in KG for better item embedding $i_t(G)$.

The computation of the node's representation in a single graph convolutional embedding layer is a two-step procedure: aggregation and integration. These two procedures can naturally be extended to multiple hops, and we use the notation

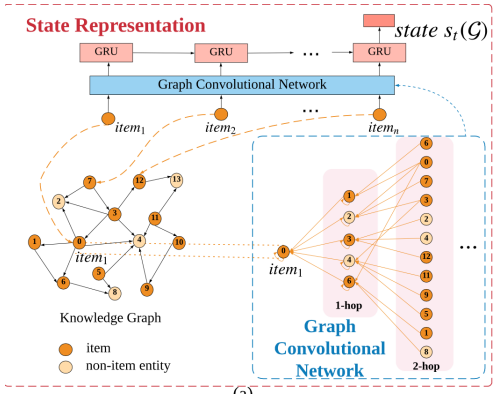

Fig. 4. The overall architecture of the Graph Convolutional Network

k to identify k -th hop. In each layer, first, we aggregate the representations of the neighboring nodes of a given node h as in "Fig. 5".

$$\mathbf{e}_{N(h)}^{k-1} = \frac{1}{|N(h)|} \sum_{t \in N(h)} \mathbf{e}_t^{k-1},$$

Fig. 5. A classic implementation of Graph Convolution.

where $N(h) = N(head) = tail|(head, relation, tail)G$ is the set of neighboring nodes of h. Notice that, here we consider the classic Average aggregator for example, other aggregator like concat aggregator [25], neighbor aggregator or attention mechanism (GAT) [25] can also be implemented.

Second, we integrate the neighborhood representation with h's representation as in "Fig. 6"

$$\mathbf{e}_h^k = \sigma(\mathbf{W}_k \mathbf{e}_{N(h)}^{k-1} + \mathbf{B}_k \mathbf{e}_h^{k-1}),$$

Fig. 6. The Neighborhood Representation.

where $\mathbf{W}_k$ and $\mathbf{B}_k$ are trainable parameters for k-hop neighborhood aggregator and σ is the activation function implemented as $ReLU(x) = max(0, x)$. In "Fig. 6", we assume the neighborhood representation and the target entity representation are integrated via a multi-layer perceptron. After k-hop graph convolutional embedding layer, each clicked item is then converted into $i_t(G) = e^k_{i_t}$.

2) Behavior Aggregation Layer: Since the interactive recommendation is a sequential decision-making process, at each step, the model requires the current observation of the user as input, and provides a recommended item it as output. It is natural to use auto- regressive models such as recurrent neural networks (RNN) to rep- resent the state based on the observation-action sequence [10, 23]. Thus, we use an RNN with a gated recurrent unit (GRU) as the network cell [6] to

aggregate user's historical behaviors and distill user's state st (G). The update function of a GRU cell is defined as in "Fig. 7"

$$\begin{aligned}
\mathbf{z}_t &= \sigma_g\left(\mathbf{W}_z \mathbf{i}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z\right), \\
\mathbf{r}_t &= \sigma_g\left(\mathbf{W}_r \mathbf{i}_t + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{b}_r\right), \\
\hat{\mathbf{h}}_t &= \sigma_h\left(\mathbf{W}_h \mathbf{i}_t + \mathbf{U}_h\left(\mathbf{r}_t \circ \mathbf{h}_{t-1}\right) + \mathbf{b}_h\right), \\
\mathbf{h}_t &= \left(1 - \mathbf{z}_t\right) \circ \mathbf{h}_{t-1} + \mathbf{z}_t \circ \hat{\mathbf{h}}_t,
\end{aligned}$$

Fig. 7. The GRU model in the behavior aggregation layer.

where i_t denotes the input vector, z_t and r_t denote the update gate and reset gate vector respectively, is the element-wise product operator. The update function of hidden state h_t is a linear interpolation of previous hidden state ht 1 and a new candidate hidden state hˆt. The final output is call state representation $s_t(G)$.

B. Neighbor-based Candidate Selection

Generally, the clicked items have some inherent semantic characteristics, e.g., similar genre movies [15]. Since users are usually not likely to be interested in all items, we can focus on selecting the potential candidates for restricted retrieval based on this semantic information in KG. Specifically, we utilize KG to filter some irrelevant items (i.e., actions) and dynamically obtain the potential candidates. The restricted retrieval will focus the data samples on the area that is more useful, as suggested in the structure of item correlation. Thus, these potential candidates would not only reduce the large searching space, but also improve the sample efficiency of policy learning.

More specifically, we perform a sampling strategy based on the k-hop neighborhood in KG. In each time step t, the user's historical interacted items serve as the seed set $E^0{}_t = i_1, i_2, ...i_n$. The k-hop neighborhood set starting from the seed entities is denoted as in "Fig. 8"

$$\mathcal{E}_t^k = \left\{ tail \mid (head, relation, tail) \in \mathcal{G} \text{ and } head \in \mathcal{E}_t^{l-1} \right\}, \quad l = 1, 2, \ldots, k.$$

Fig. 8. All neighbor nodes of the initial item set extracted from knowledge graph.

Then, the candidate action set for the current user state is defined as "Fig. 9"

$$\mathcal{I}_t(\mathcal{G}) = \left\{ item | item \in \bigcup_{l=1}^{k} \mathcal{E}_t^l \text{ and } item \in \mathcal{I} \right\},$$

Fig. 9. The candidate set extracted from neighbor items of the initial set.

with a user-defined cardinality. The shallow part in "candidate selection" in "Fig. 10" denotes the selected actions with the information from KG. Then all candidate items get their embedding through the graph convolutional layers.

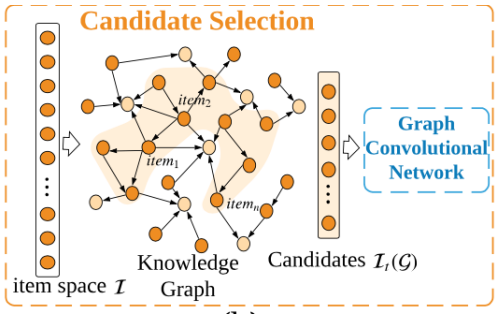

Fig. 10. The candidate set extracted from neighbor items of the initial set.

C. Neural Collaborative Filtering

For any machine learning based work, it's very important that we have data. In a loose form, study about data and its characteristics is what ML mostly comprises of. As for recommendation model, we need data about the users and the items that we will recommend to build our model. In our case, we use the million song dataset to recommend songs. Each user has an user id and songs have an unique song id. We preprocess our song and user dataset and create the rating dataset which includes what songs each user has interacted with. The rating data frame consists of three columns which include, user id, song id and a rating column which is the number of play counts. Initially, when we build the dataset, we have only the interactions where the user has interacted with the item. However, when we train our model we need to know which users haven't interacted with the songs so that we our model can understand the user preferences better. This part is explained in more details in the later section.

1) Implicit Feedback Conversion: In recommender systems, the rating matrix is usually constructed from the explicit ratings that a user gives to the movie. This is called explicit feedback. The problem with explicit feedback is that a lot of users don't provide ratings and therefore it becomes difficult to understand the user preferences and recommend items accordingly. Another way of creating the rating data frame is through implicit feedback where we check how much time or how many number of times the user has interacted with an item and set the rating value to 1 if the interaction time is more than a specific threshold. Since the rating column is the number of play counts (an explicit feedback). The playcounts of 1 and larger than 1 is converted to 0 and 1 respectively.

2) Architecture: In traditional recommendation models, the recommendation was done using user-item similarity and matrix factorization techniques. In case of User- Item similarity, we use similarity techniques to find a cluster of similar users and similar items. Later, we find out how all the similar users have interacted with other items and recommend those items to the users. In case of item similarity, we check for users who have interacted with the items and we recommend other items

obtained from the item clusters. One of the common similarity methods used is Cosine similarity or Jaccard similarity models.

Another famous technique of recommending items is the Matrix Factorization approach. In case of matrix factorization approach, we have a matrix of user item pairs and how they have interacted with each other. If there is an interaction between the user and item, we fill the intersection of user and item by 1 otherwise, it's 0. We factorize the obtained matrix into a lesser dimensional user and item latent vectors which when multiplied will give us back the original matrix. This technique is called Matrix Factorization. One of the main problems with Matrix Factorization method is sparsity and cold start problems. We won't really understand the preferences of a new user straightaway. Also, any change in dynamic behavior of an user can be detected only when we run the model again a couple of times.

As discussed above, in order to overcome the problems of sparsity and cold start problem, we implement the Neural network based approach of recommending items. One such model is the Neural collaborative filtering approach ("Fig. **??**"). The specific reason for going with this algorithm is it's two part network. One part which generalizes the matrix factorization approach and another part which uses a deep learning network to understand the user item interaction.

The first part is called the MF or matrix factorization part and the second part is called the MLP or the multilayer perceptron part. In below few lines we discuss the implementation details about how we process the user item rating dataset and learn the recommendation model.

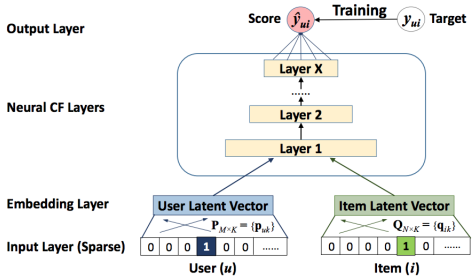

Fig. 11. Neural Collaborative Filtering Architecture. The left part is the embedding layer for the user data. The right part is the embedding layer for the item data.

We have the user item interaction dataset which is called the rating data frame. This data consists of only the user and their respective interactions. Therefore, it's important that we have some negative samples of when the user has not interacted with the songs. This will help to train the model better and understand the user item preferences. In order to obtain this, we get all the songs and also make a list of songs that the user has not interacted with and set the rating value to 0 when there was no interaction. We use leave one out strategy to build our train and test dataset. For all users, we hold out one user item interaction pair and assign it to the test dataset in order to see how well our model does. The model architecture for the MLP part is shown below.

V. Experiment

A. Dataset

Echonet Taste Profile is a subset of Million Song Dataset [2] that provides the user-song interaction data. As explained above, the playcount in the user-song data is binarized into 0 and 1 if the playcount is equal to 1 and larger than 1 respectively. There is 1M unique users, 384K unique songs, and 48M user-song rows of interactions.

B. Experimental Settings

Our NCF model has 4 hidden layers of 64, 32, 16, and 8 units. Since performing classic graph convolution, the graph convolution layer has no hidden unit and activation function. Instead, it averages 256-wide embedding matrices of nodes found within the k-hop=3 traversal. K-hop is set to 3 in order to accelerate the training process. The averageed graph embeddings of users' past seleection is processed through a linear layer of 256 units and the ReLU activation [22]. In the behavior aggregation layer, the GRU [**?**] layer applied over graph embeddings of nodes from the graph convolution layer has 128 hidden units with default activation. Finally, the Adam optimizer [21] is used to optimize weight and biases.

C. Evaluation Metrics

We also define the NCDG and HR metrics below. The NCDG refers to the numerical cumulative discounted gain which means we not only worry about the correct items predicted but also the index at which they are predicted. Consider we prefer song 1 to song 2 and our model predicts song 2 with higher probability, then the NCDG score goes down and vice versa. HR refers to Hit Ratio which means we care only the number of correct items predicted. NCDG is a stricter metric compared to the HR metric.

D. Analysis

	Avg. NDCG	Avg. HR
EchoNest	0.438	0.578

Fig. 12. Evaluation Results on the Echonet Taste Profile dataseet.

The performance of our approach on the Echonet User-Taste Profille dataset is in "Fig. 12". As we train the model on 100k data points, we are able to see a average performance of our model. Using the GNN to predict more items, we let the user interact more with the app and therefore enhancing the performance. We also use negative sampling of data to make sure we learn the negative preferences of the user also. Using GNN, we are also able to provide diversity. The connected nodes with respect to their meta data can give more better and diversified results which will help the user explore more options. Through this, we can also learn a lot of users preferences which can be used for future scenarios.

VI. Conclusion

As we previously discussed the architecture, in future we plan to make more developments. Although, we can use leave one out strategy and improve further on this to make more predictions for more songs, the best result can be obtained only when we let a user directly interact with an app and see how long he interacts with the app. The more time an user interacts with an app and listens to song, the better the predictions and this also acts as a business metric as we can use this interactive nature to show more ads and improve the business. In order for this future work, we have proposed an architecture which is an add on to the NCF enhanced knowledge graph interactive recommender model. We use a streamlit based application to develop an app and for each user, we get the stored predictions. We use these predictions to traverse through the graph to predict n more songs based on the graph learnt through different metadata and at different depth levels. We can also provide feedback to the model based on graph recommended items, NCF model recommended items to learn better ways of representations. The ideas included in this section is a work in progress.

References

[1] S. Zhoou, X. Dai, H. Chen, W. Zhang, K. Ren, R. Tang, X. he, and Y. Yu, "Interactive Recommender System via Knowledge Graph-enhanced Reinforcement Learning," ACM SIGIR Conference on Research and Development in Information Retrieval, 2020.

[2] T. Bertin-Mahieux, D. Ellis, B. Whitman and P. Lamere, "The Million Song Dataset," Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR), 2012.

[3] J. McInerneey, B. Lacker, S. Hansen, K. Higley, H. Bouchard, A. Gruson, and R. Mehrotra, "Explore, Exploit, and Explain: Personalizing Explainable Recommendations with Bandits," 2018.

[4] Y. Koren, R. Bell, and C. Volinsky, "Matrix Factorization Techniques for Recommender System," IEEE Computer Society, 2009.

[5] H. Chng, L. koc. J. Harmsen, and et al., "Wide & Deep Learning for Recommender Systems", 2016.

[6] H. Wang, Q. Wu, and H. Wang, "Factorization bandits for interactive recommendation," AAAI, 2017.

[7] C. Zeng, Q. wang, S. Mokhtari, and Tao Li, "Online Context-Aware Recommendation with Time Varying Multi-Armed Bandit," SIGKDD, pp. 353-362, 2016.

[8] H. Chen, X. Dai, D. Cai, and et al., "Large-Scale Interactive Recommendation with Tree-Structured Pollicy Gradient," AAAI, pp. 3312-3320, 2019.

[9] Y. Hu, Q. Da, A. Zeng, Y. Yu, and Y. Xu, "Reinforcement Learning to Rank in E-Commerce Search Engine: Formalaization, Analysis, and Application," SIGKDD, pp. 368-377, 2018.

[10] X. Zhao, L. Xia, L. Zhang, Z. Ding, D. Yin, and J. Tang, "Deep Reinforcement Learning for page-Wise Recommendations," ACM RecSys, pp.95-103, 2018.

[11] D. Silver, A. Huang, C. Maddison, and et al., "Masteering the Game of Go with Deep Neural Network and Tree Search," Nature, p. 484, 2016.

[12] X. He, L. Liao, H. Zhang, L. Nie. X. Hu, and T. Chua, "Neural Collaborative Filtering," 2017.

[13] S. Wu, F. Sun, W. Zhang, and B. Cui, "Graph Neural Networks in Recommender Systems: A Survey," 2021.

[14] J. Bennet and S. Lanning, "The Netflix Price," Proceedings of the KDD Cup Workshop 2007, ACM, pp. 3-6, 2007.

[15] H. Wang, F. Zhang, J. Wang, M. Zhao, W. Li, X. Lie, and M. Guo, "RippletNet: Propogating Useer Preferencees on the Knowledge Graph for Recommender Systems," CIKM, ACM, pp. 417-426, 2018.

[16] X. he, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, "LightGCN: Simplifying and Powering Graph Convolution Network for Recommndation," Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, pp. 639-648, 2020.

[17] X. Wang, X. he, M. Wng, F. Feng, and T. Chua, "Neural Graph Collaborative Filtering," Proceedings of the 42nd International ACM SIGIR Confeerence on Research and Development in Information Retrieval,", ACM, 2019.

[18] LL. Zou, L. Xia, Z¿ Ding, D. Yin, J. Song, and W. Liu, "Reinforcement Learning to Diversify Top-N Recommendation," Lecture Nodes in Computer Sciience, vol. 11447, 2019.

[19] X. Zhao, C¿ Gu, H. Zhang, X. Yang, X. liu, J. Tang, and H. Liu, "DEAR: Deep Reinforcement Learning for Online Advertising Impression in Recommender Systems," 2021.

[20] K. Cho, B. Merriienboer, and et al., "Learning Phrase Representations using RNN Encoder-Decoder for statistical Machine Translation," 2014.

[21] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization,", ICLR, 2015.

[22] A. Agarap, "Deep Learning using Rectified Linear Units (ReLU)," 2019

[23] X. Zhao, L. Zhang, Z. Ding, L. Xia, J. Tang, and D. Yiin, "Recommendations with Negative Feedback via Pairwise Deep Reinforcement Learning,", SIGKDD, ACM, pp. 1040-1048, 2013.

[24] W. Hamillton, Z. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," NeuIPS, pp. 1024-1034, 2017.

[25] P. Velickovic, G. Cucurull, A. Casanova, A. Rromero, P. Lio, and Y. Bengio, "Graph Attention Networks," 2017.