# Deep Learning-Based Implicit Feedback Recommendation


**Xin Xin**

School of Computing Science

College of Science and Engineering
University of Glasgow

This dissertation is submitted for the degree of
*Doctor of Philosophy (PhD)*

January 2021

# Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University.

This dissertation is the result of my own work, under the supervision of Professor Joemon M. Jose and Dr. Ke Yuan.

Xin Xin
January, 2021

# Acknowledgements

Maybe there are a lot of people claiming that doing a Ph.D. is not a so pleasant journey. But I really appreciate the days I live in Glasgow and pursuing the degree. It's the most memorable, thinkable, insightful, and free days in my ever life. I'm writing this short statement to thank all the guys that I encountered during this period. Thank you to let me know that pursuing knowledge is full of touching, achievement, excitement, and at last, happiness.

First of all, I want to give my sincere appreciation to my thesis supervisor Prof. Joemon M. Jose. Joemon is a super nice advisor with an open mind, professional expertise, and kindly heart. He provides me the freedom to conduct research based on my interests. He provides me the opportunities to connect with the best researchers in my research field. He also provides me with careful instructions on academia and kindly cares about daily life. Best wishes to you and your family. Besides, I'd like to express my gratitude to my second supervisor Dr. Yuan Ke. Thank you for your helpful effort and contribution to my progression review and check.

Secondly, I want to show my grateful respect to Dr. Alexandros Karatzoglou and Ioannis Arapakis who are my internship advisor. Alexandros is an amazingly brilliant, kindly heart and easy-going advisor (friend). He has the ability to mine the most insightful truth from complex observations. Exchanging ideas with Alexandros is full of excitement and highlights. Ioannis is also a great supervisor, researcher, and friend. He helps me a lot with his amazing ideas, ingenious visualization skills, and professional writing. Thank you so much. It's my great pleasure to be one of your students (friends).

Thirdly, I want to thank Dr. Fajie Yuan and Dr. Xiangnan He. Fajie is my abecedarian in my Ph.D. academic career. He helps me to establish confidence, provides me with positive feedback and the most helpful expertise on how to do

a Ph.D. and do research. Fajie is one of the most important advisors in the early years of my Ph.D. Xiangnan is an amazing smart friend (teacher) who provides me the most insightful advice, most original ideas, and most promising directions. Xiangnan's wisdom and advice give me the most help in my later Ph.D. career.

Forth, I would like to appreciate my parents, my grandparents, my litter brother, and other family members. Your support provides me the biggest courage to do research, to develop methods, and to pursue the truth. You provide me the warmest home and the most tolerant haven in my life.

Besides, I want to acknowledge my colleague Yanni Ji, Siwei Liu, Xi Wang, and Ting Su. Exchanging ideas and having fun with you guys bring me plenty of happiness and enjoyment.

Finally, I want to thank my dear friend Debao Guan, Yuheng Wang, Mingzhe Wei, Ruocheng Dong, Yongjie Wang, and Shuting Zhang. You guys are the best friends, best community members, and family members for my Ph.D. career. Best wishes to you guys.

Anyway, there are too many people to thank and acknowledge, too many things to appreciate, too many staffs to remember. I would like to dedicate this thesis to all lovely lives. The past has gone, let's focus on the future. Start sailing, conquer the sea!

Best wishes to you all!

# Abstract

Recommender systems are of vital importance, in the era of the Web, to address the problem of information overload. It can benefit both users by recommending personalized interesting items and service providers by increasing their site traffic. Plenty of use cases have emerged as applied recommender systems, including but not limited to multimedia recommendation (e.g., news, movies, music, and videos) and e-commerce recommendation. A recommendation agent can be trained from user-item interaction data which can be categorized as explicit feedback and implicit feedback. Compared with explicit ratings which depict the user preference explicitly, implicit feedback data like clicks, purchases, and dwell time is more prevalent in the real-world scenario. On the other hand, deep learning has achieved great success recently due to the high model expressiveness and fidelity. In this thesis, we investigate deep learning techniques for recommendation from implicit feedback data. We focus on two learning perspectives: deep supervised learning and deep reinforcement learning.

Supervised learning tries to infer knowledge from implicit historical interactions. From this perspective, two models namely Convolutional Factorization Machines (CFM) and Relational Collaborative Filtering (RCF) are proposed. CFM tackles the implicit user-item interactions with side information as feature vectors and utilizes convolutional neural networks to learn high-order interaction signals. RCF considers multiple item relations into the recommendation model and tackles the implicit feedback as relation-enriched data. The two models investigate deep learning techniques for recommendation by tackling the data as two different structures: feature vectors and relations. Experimental results demonstrate that the proposed deep learning models are effective to improve the recommendation accuracy. Besides, RCF also helps to provide explainable recommendation and get a better comprehension of user behaviors.

Reinforcement learning is reward-driven and focuses on long-term optimization in a whole interaction session, which conforms more with the objective of recommender systems. From this perspective, we first formulate the next-item recommendation task from implicit feedback data as a Markov Decision Process (MDP). Then we analyzed that directly utilizing reinforcement learning algorithms for recommendation is infeasible due to the challenge of pure off-policy setting and the lack of negative reward signals. To address the problems, we proposed Self-Supervised Q-learning (SQN) and Self-Supervised Actor-Critic (SAC). The key insight is to combine reinforcement learning with supervised learning and perform knowledge transfer between the two components. Based on SQN and SAC, we further proposed Self-Supervised Negative Q-learning (SNQN) and Self-Supervised Advantage Actor-Critic (SA2C) to introduce the negative sampling strategy to enhance the learning of the reinforcement component. Experimental results demonstrate that the proposed learning frameworks are effective when integrated with different existing base models. Moreover, we show that combining supervised learning and reinforcement learning is a promising direction for future recommender systems. In that case, reinforcement learning introduces reward-driven objectives and long-term optimization perspectives into supervised learning, while supervised learning helps to improve the data efficiency of reinforcement learning.

# Table of Contents

# List of Figures

# List of Tables

# Part I
# Introduction and Preliminary

This thesis focuses on developing deep learning-based recommendation methods for implicit feedback data, including deep supervised learning and deep reinforcement learning. This part includes Chapter 1 for introduction and Chapter 2 for introducing preliminaries and reviewing relevant literature. In Chapter 1, we described the background of item recommendation from implicit feedback data and then provided the motivation, thesis statement and contributions. In Chapter 2, we provided the preliminary and literature review of existing work, including the training objective and procedure for supervised learning methods and the preliminary of reinforcement learning for recommendation. We also described the evaluation metrics for implicit feedback recommendation.

# Chapter 1

# Introduction

## 1.1 Background on Recommendation

Recently, plenty of online services with the growth of Web technology dramatically introduce a huge amount of information, including texts, images, videos as well as physical items. For example, music platforms such as Spotify[1] provide millions of songs available online. Youtube[2] and Netflix[3] deliver videos and movies as services to people. E-commerce platforms such as Amazon[4] and Taobao[5] provide all kinds of items for online shopping. This huge volume of online information could easily overwhelm users and causes difficulty to find useful information, which is known as the information overload problem [6].

To help users get access to his/her interested information more effectively, retrieval and recommendation play a very important role in both academia and industry (Belkin and Croft, 1992). From the perspective of an information retrieval (IR) system, users are expected to provide a query to a search engine, then a list of matched documents are returned to the user according to certain ranking algorithms (Belkin and Croft, 1992). However, recommender systems (RS) aim to automatically generate interesting items according to user preferences and push the recommendation results to users (Costa and Roda, 2011). The main difference is that in an IR system, users are expected to explicitly present their needs as queries, while users in RS are more likely to passively receive the pushed

[1] https://www.spotify.com/us/
[2] https://www.youtube.com
[3] https://www.netflix.com/
[4] https://www.amazon.co.uk/
[5] https://www.taobao.com/
[6] https://en.wikipedia.org/wiki/Information_overload

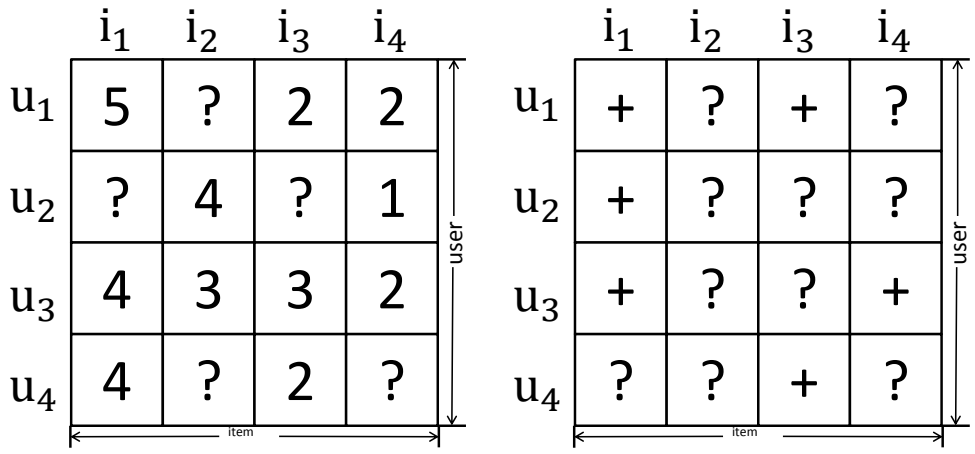
information from the recommendation agents (Shi, 2013). Recently, RS have become the keystone for plenty of online service providers in different kinds of domains (Sheth et al., 2010), such as music recommendation of Spotify, movie recommendation in Netflix, item recommendation in Amazon, and so on. In this thesis, our focus is RS and the research is developing new recommendation methods for RS based on advances in deep learning, including deep supervised learning and deep reinforcement learning.

## 1.2 Implicit Feedback for Recommendation

The definition of a RS can be formally given as "*a subclass of information filtering system that seeks to predict the 'rating' or 'preference' a user would give to an item*" (Ricci et al., 2011). The task of a recommendation agent can thus be divided into rating prediction and item ranking.

For the rating prediction task, the input data for the recommendation agent is usually user-item ratings which are also referred to as explicit feedback, such as 1-to-5 scores. The user-item ratings can explicitly represent the user preference on this item. The training of the recommendation agent using explicit feedback can be seen as a regression task that aims to predict the missing ratings, as shown in Figure 1.1(a). Then the recommendation list can be generated by selecting unvisited items which have the highest estimated ratings.

However, in the real-world scenario, explicit feedback is extremely sparse because users commonly do not provide ratings (Rendle et al., 2009b; Rendle and Freudenthaler, 2014). On the contrary, implicit feedback such as user clicks/purchases, watching history, and dwell time are much easier to track automatically and collect. Figure 1.1(b) shows an example of a simple implicit user-item interaction matrix. Compared with explicit feedback, implicit feedback is much more common and inexpensive. Besides, when characterizing implicit user-item interactions, plenty of side information such as item attributes, contexts of location, daytime can also be collected. This side information-enriched implicit feedback composes the major data source to train a recommendation agent. When input data is implicit feedback, the task of the recommendation agent is usually formulated as item ranking (Rendle et al., 2009b). The agent is




(a) Explicit rating matrix (b) Implicit feedback

Figure 1.1: Examples for explicit ratings (a) and implicit feedback (b) from Yuan (2018). $u$ and $i$ denote users and items, respectively. The numerical scores in (a) represent explicit ratings that users assigned to items, while for implicit feedback, explicit ratings are not provided. "+" and "?" in (b) denote positive (e.g., a click) and unobserved feedback, respectively.

expected to return a ranking list of unvisited items according to the predicted user preference without rating regression.

This thesis focuses on developing recommendation methods based on implicit feedback data without explicit ratings, which is more close to the real-world use cases.

## 1.3 Deep Learning for Recommendation

Implicit feedback data is much more prevalent to train a recommender system. However, implicit feedback also poses new challenges to learn user preference. Unlike explicit ratings, we can not directly infer the real user preference from implicit feedback (Rendle et al., 2009b). Besides, rich side-information can also be collected to depict the detail of user-item interactions (Rendle, 2010). Under this situation, conventional shadow recommendation models may encounter difficulties to mine the real user preference and thus lead to sub-optimal solutions (He et al., 2017b). This motivates us to develop recommendation methods that

have more expressiveness and model fidelity.

Deep learning has achieved great success in fields like computer vision (He et al., 2016b,a) and natural language processing (Vaswani et al., 2017; Brown et al., 2020; Devlin et al., 2018). For example, He et al. (2016a) proposed ResNet which has hundreds of convolutional layers for image classification and achieved promising results. Brown et al. (2020) builds powerful language pre-training models with billions of parameters that can be used for plenty of downstream tasks. The non-linearity and deep structure of neural networks make deep learning have great expressiveness to fit the training data. This also motivates the use of deep learning for the recommendation scenario. He et al. (2017b) proposed neural collaborative filtering to extend the inner product of matrix factorization (Koren et al., 2009) to multi-layer perceptrons (MLP). He and Chua (2017) proposed neural factorization machines to use MLP for click-through ratio (CTR) prediction. A more detailed literature review can be found in Section 2. Based on these attempts, this thesis focuses on developing deep learning-based recommendation methods for implicit feedback recommendation.

The work contains two aspects: deep supervised learning and deep reinforcement learning for recommendation. For deep supervised learning, this thesis developed two deep models for implicit feedback with side information. The deep supervised learning models are optimized by minimizing a loss function which is defined as the discrepancy between the model prediction and the ground truth. However, such a loss function may not completely fit the demand of recommender systems (e.g., long-term user satisfaction in a whole session). On the other hand, deep reinforcement learning (e.g., Deep Q-learning (DQN) (Mnih et al., 2015)) has shown an advantage to optimize the long-term cumulative reward. In this thesis, the second point of our work focuses on utilizing deep reinforcement learning for recommendation.

The main research challenges of the thesis are summarized as follows:

- Given the implicit feedback data, how do we tackle the accompanied side information? How to design deep supervised learning models to better learn knowledge from implicit feedback? How to use deep models to better learn high-order interaction signals from the data?

- Given the implicit feedback data with side information, how to design deep learning models to generate more reasonable and convincing recommendation? Can we utilize deep learning models to better understand user behavior patterns?

- Recommender systems should focus on long-term cumulative gains in a whole interaction session. In that situation, how to utilize deep reinforcement learning to introduce cumulative reward-based training objectives? How to tackle the specific difficulties when utilizing reinforcement learning for recommendation, such as off-line training and lack of reward signals.

The thesis targets at above three research challenges and proposes a series of solutions to address them.

## 1.4 Thesis Statements

The overall statement of the thesis is that applying deep learning methods, both deep supervised learning and deep reinforcement learning, will lead to the improvement of recommendation quality. Firstly, deep supervised models can increase the expressiveness and model fidelity to learn more complex interaction signals from implicit feedback data with side information, leading to more accurate context-aware recommendation. Secondly, applying deep supervised models can also help to integrate multiple item relations into recommendation. This can be used to generate explainable recommendation and get a better comprehension of user behavior. Then deep reinforcement learning is expected to improve recommendation towards long-term cumulative rewards. However, deep reinforcement learning cannot be directly used under the recommendation setting due to some specific conditions. The final important statement is that combining supervised learning and reinforcement learning provides us a solution to integrate deep reinforcement learning for recommendation.

More precisely, the statements of this thesis are shown as follows:

- **Statement (1)**: Deep supervised learning will help to capture more complex signals from implicit user-item interactions which cannot be learned

by conventional shadow models. The implicit feedback with various side information needs deep models with higher model expressiveness to generate more accurate recommendation.

- **Statement (2)**: Deep supervised learning can be used to model semantic meaningful item relations for recommendation. This helps to get a better comprehension of user behavior patterns and generate more explainable and convincing recommendation results.

- **Statement (3)**: Deep reinforcement learning will introduce long-term objectives into recommendation but directly utilizing it is problematic. The reason is that the deep reinforcement learning agent is trained through an "error-and-correction" fashion with huge training examples but we can not afford the price to make many errors under recommendation settings. Combining deep supervised learning and deep reinforcement learning provides a solution to successfully integrate reinforcement learning for recommendation. In that case, deep supervised learning can help deep reinforcement learning to learn better representations with higher sample efficiency while deep reinforcement learning can introduce the long-term reward properties into deep supervised learning.

## 1.5 Thesis Structures and Contributions

The contribution of this thesis lies in two folds. Firstly, for deep supervised learning, we propose two deep recommendation models for implicit feedback data, showing improved recommendation accuracy or more explainable recommendation. For deep reinforcement, we proposed to combine supervised learning and reinforcement learning to introduce long-term rewards into the recommendation. The thesis contains the following four parts:

- **Part I Introduction:** This part contains Chapters 1 and 2. It provides the background, preliminary and related work of this thesis.

- **Part II Deep Supervised Learning for Recommendation:** This part contains Chapter 3 and Chapter 4. Chapter 3 proposed a deep model for

implicit feedback data with side information. The data is handled as feature vectors. Then, semantic meaningful item relations can be inferred from the side information and Chapter 4 proposed a deep model to handle this relational data for more accurate and explainable recommendation. Detailed contribution and extensive experiments are provided in corresponding chapters.

- **Part III Deep Reinforcement Learning for Recommendation:** This part comprises Chapter 5 and Chapter 6. The technical contribution of Chapter 5 is that we combine supervised learning and reinforcement learning for recommendation. The reinforcement learning component serves as a regularizer to introduce the expected rewards into the supervised learning part. Based on Chapter 5, Chapter 6 further improves reinforcement learning with the negative sampling strategy. Then the reinforcement learning component is not only a regularizer but also a good ranking player to directly generate recommendation. This part sheds light on providing a promising direction for off-policy sample efficient reinforcement learning.

- **Part V Conclusion:** This part includes Chapter 7 with conclusion and future work.

## 1.6 Supporting Publications

The thesis generalizes and builds on the following publications (* denotes equal contribution):

1. **Xin Xin***, Fajie Yuan*, Xiangnan He, Joemon Jose. Batch IS NOT Heavy: Learning Word Representations From All Samples. 56th Annual Meeting of the Association for Computational Linguistics. (**ACL'18**)(**Part 1**) (Fajie provided the ideas and I implemented the methods and conduct all experiments. I wrote the paper.)

2. Fajie Yuan, **Xin Xin**, Xiangnan He, Guibing Guo, Weinan Zhang, Chua Tat-Seng, Joemon Jose. fBGD: Learning embeddings from positive unlabeled data with BGD. 2018 Conference on Uncertainty in Artificial Intelligence. (**UAI'18**)(**Part 1**)

3. **Xin Xin***, Bo Chen*, Xiangnan He, Dong Wang, Yue Ding, and Joemon Jose. CFM: Convolutional Factorization Machines for Context-Aware Recommendation. 28th International Joint Conference on Artificial Intelligence. (**IJCAI'19**)(**Part 2**) (I came up with the idea and wrote almost part of the paper. Bo Chen implemented the model.)

4. **Xin Xin**, Xiangnan He, Yongfeng Zhang, Yongdong Zhang, Joemon Jose. Relational Collaborative Filtering: Modeling Multiple Item Relations for Recommendation. 42th International ACM SIGIR conference on Research and Development in Information Retrieval. (**SIGIR'19**)(**Part 2**)

5. **Xin Xin**, Alexandros Karatzoglou, Ioannis Arapakis, Joemon Jose. Self-Supervised Reinforcement Learning for Recommender Systems. 43th International ACM SIGIR conference on Research and Development in Information Retrieval. (**SIGIR'20**)(**Part 3**)

6. **Xin Xin**, Alexandros Karatzoglou, Ioannis Arapakis and Joemon Jose. Self-Supervised Advantage Actor-Critic for Recommender Systems. 30th International World Wide Web Conference. (**SIGIR'21**) (**under-review**) (**Part 3**)

7. Gaoming Zhang*, **Xin Xin***, Li Shen, Xiangnan He and Guibing Guo. Reinforced Teacher Agent for Recommendation. 30th International World Wide Web Conference. (**SIGIR'21**)(**under-review**)(**Part 3 and Part 4**)

8. Bo Chen, Wei Guo, Ruiming Tang, **Xin Xin**, Yue Ding, Xiuqiang He, and Dong Wang. TGCN: Tag Graph Convolutional Network for Tag-Aware Recommendation. 29th ACM International Conference on Information and Knowledge Management. (**CIKM'20**)(**Part 4**)

9. Hao Chen, **Xin Xin**, Yue Ding, Dong Wang. Decomposed Collaborative Filtering: Modeling Explicit and Implicit Factors For Recommender Systems. 14th ACM Conference on Web Search and Data Mining (**WSDM'21**) (**Part 4**)

# Chapter 2

# Preliminaries and Related Work

In this chapter, we provide preliminaries and related work on implicit feedback recommendation. We first introduce the general overview of implicit feedback recommendation, including the type of recommender, the training objective and procedure, and the evaluation method. We then provide a literature review of notable recommendation models. Finally, we describe the motivation to include reinforcement learning for recommendation and introduce the preliminaries of deep reinforcement learning algorithms.

## 2.1 Review on Implicit Feedback Recommenders

As discussed in section 1.2, when learning a recommendation agent from implicit feedback data, the task is usually defined as item ranking. In this case, a user is recommended with items that are at the top-$N$ (e.g., top-10) positions of the final ranking list.

### 2.1.1 Types of Recommendation Algorithms

The recommender algorithms can be generally classified into three categories: collaborative filtering methods, content-based methods, and hybrid methods (Adomavicius and Tuzhilin, 2005).

#### 2.1.1.1 Collaborative Filtering Methods

Collaborative filtering (CF) plays the most important role in the field of recommender systems. The basic idea of CF is that the unknown user preference over

items can be inferred from similar users (items). It can be further classified as user-based CF and item-based CF.

User-based CF serves the most early-stage of online recommender systems (Linden et al., 2003). It is a method of "*making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating)*[1]". One typical example of user-based CF can be provided as "*users who are similar to you also liked/viewed/bought...*". The first-step of user-based CF is to calculate the similarity between users. Then the preference of a user over an item is aggregated from the preference of the most similar users with him/her. Although user-based CF is widely adopted in the early-stage of recommendation, the user similarity is highly dynamic (Linden et al., 2003). Besides, it's also hard to calculate preference for cold-start users with too few interactions to identify the user similarity.

Compared with user-based CF, item-based CF is built on the similarity between items. Item similarity is much more static than user similarity. The meta-information of items also provides solutions for cold-start items. One typical example of item-based CF can be given as "*users who liked/viewed/bought this also liked/viewed/bought...*". Due to the effectiveness and scalability of item-based CF, it has achieved great success in industry recommender systems, such as Amazon's online recommender system (Linden et al., 2003). The idea of ICF is that the user preference on a target item $i$ can be inferred from the similarity of $i$ to all items the user has interacted with in the past (Sarwar et al., 2001; Linden et al., 2003; He et al., 2018c; Kabbur et al., 2013). Under this case, the relation between items is referred to as the collaborative similarity, which measures the co-occurrence in the user interaction history.

From the perspective of similarity calculation, CF methods can be classified as memory-based methods and model-based methods.

Memory-based CF methods, also known as neighborhood-based CF and heuristic-based CF, are mainly used in the early literature (Sarwar et al., 2001; Linden et al., 2003). More precisely, a pre-defined similarity metric is used to calculate the user (item) similarity, such as cosine similarity (Linden et al., 2003), Pearson

[1] https://en.wikipedia.org/wiki/Collaborative_filtering

correlation (Sheugh and Alizadeh, 2015) and Jaccard coefficient[1] from user-item interaction history. Then, the preference prediction can be calculated from the aggregation of most similar users or items (e.g., nearest neighborhood). This is the basic idea of the well-known userKNN (itemKNN) algorithms. Memory-based CF methods are simple to implement but the pre-defined similarity metrics are not date-driven and may not reflect the complex knowledge of the training data.

On the contrary to memory-based CF, model-based CF methods are data-driven and usually utilize machine learning models to automatically optimize parameters in the learning space according to certain objective loss functions. Among model-based CF, one of the most successful methods is matrix factorization (Koren et al., 2009) which first factorizes users and items to a common latent space (e.g., embeddings or representations) and then the preference is defined by the inner product between user and item embeddings. Based on matrix factorization, factorization machines (Rendle, 2010) extend the user-item matrix to side-information enriched interaction data. Recently, deep learning has shown a great advantage in representation learning. Using deep learning models to extract feature representations such as images, reviews (texts) and then incorporate them into the recommendation model has become a common use case. Such kind usage of deep learning is more like feature engineering. While in this thesis, we mainly focus on developing deep learning-based methods for better interaction modeling, other than feature engineering.

#### 2.1.1.2 Content-Based Methods and Hybrid Methods

CF-based methods calculate the user (item) similarity according to historical user-item interaction data. However, in practical cases, there is also plenty of side information. For example, in many practical recommender systems, items can be characterized by a set of features (e.g., actors, directors, and genres of a movie) while users also have their profile (e.g., gender, age, and occupation) (Wit, 2008). Such features can also be utilized to calculate similarity, especially for cold-start users and items in which CF-based methods may not have sufficient interaction data to learn good collaborative signals. Content-based approaches have their roots in the field of information retrieval (Adomavicius and Tuzhilin, 2005). The

[1] http://ase.tufts.edu/chemistry/walt/sepa/Activities/jaccardPractice.pdf

early advancements from text-based retrieval (e.g. document retrieval) encourage the application of text-content for text-oriented recommendation (e.g., news recommendation). This further motivates the use of other content features for better recommendation. Although there may be a variety of features in the training data, we can generally divide them into three classes: user profile, item features, and other context features (e.g., time and location of the interaction). This is useful to perform negative sampling for implicit feedback data. We will describe this in more detail in related chapters.

In a real-world recommender system, the agent usually combines both the idea of collaborative filtering and content information (De Campos et al., 2010). This kind of method can be referred to as the hybrid method. For example, the well-known factorization machines use inner products between the embedding of input (content) features to capture the collaborative signal, which is a hybrid approach. In this thesis, we proposed three deep supervised learning models for recommendation. All of the three models use both content features and collaborative filtering and thus can be regarded as hybrid recommendation methods.

### 2.1.2 Training for Implicit Feedback Recommendation

This thesis focuses on developing recommendation methods for implicit feedback. One of the most important characteristics of implicit feedback is that negative feedback is not provided. From implicit feedback, we can only know which items the user has interacted with before but have no knowledge of negative signals. However, if we perform learning only on the observed positive feedback without negative comparison, the predicted values will have a positive bias and the ranking performance would be very poor. Because the task is item ranking, so we need to tell the model in our training data that one item is better than the other(s) for a user. To provide negative signals for implicit feedback data, there are two kinds of methods: sampling-based approaches (i.e., negative sampling) and non-sampling methods. In this section, we provide an overview of these two kinds of approaches.

#### 2.1.2.1 Sampling-Based Training

Sampling-based approaches use a sampler to sample negative examples from the unobserved interactions for negative comparison. One of the most successful sampling-based approaches for implicit feedback is Bayesian personalized ranking (BPR) (Rendle et al., 2009b) which is a pairwise learning to rank training objective function.

The basic idea of BPR is that the observed positive items for a user should be ranked higher than the sampled negative ones, under the rule of a Bayesian maximum posterior probability (MAP) estimation (Rendle et al., 2009b; Yuan, 2018). The loss function of BPR can be formulated as

$$L(\Theta) = -\sum_{(u,i,j)\in D_s} \ln \sigma(\hat{y}_{uij}) + \lambda_\Theta ||\Theta||^2, \tag{2.1}$$

where $\Theta$ is the total learning space, $\sigma$ is the sigmoid function which is defined as $\sigma(x) = 1/(1+e^{-x})$, $\lambda_\Theta$ is the regularization weight and $D_s$ is the obtained from the training data. If we use $I_u^+$ to denote the set of interacted (positive) items for user $u$, then $D_s = \{(u,i,j)|i \in I_u^+ \wedge j \in I \backslash I_u^+\}$. It represents the set of all pair-wise comparison between two items. $\hat{y}_{uij}$ denotes the logits difference between positive examples and the sampled negative ones (i.e., $\hat{y}_{uij} = \hat{y}_{ui} - \hat{y}_{uj}$ where $\hat{y}_{ui}$ denotes the predicted scores of user $u$ on item $i$).

When training the BPR objective function, we randomly sample mini-batches from $D_s$ and update the learnable parameters according to the loss function shown in Eq.(2.1) until the model converges. We can see that by minimizing Eq.(2.1), the model will try to push the positive $\hat{y}_{ui}$ to high values while decrease the negative $\hat{y}_{uj}$. When generating recommendations, we select top-$N$ unvisited items according to $\hat{y}_{ui}$. The original BPR utilizes a uniform sampler to sample negative ones. Future work (Chen et al., 2018; Rendle and Freudenthaler, 2014) claims that a carefully designed sampler (e.g., an adaptive sampler) can help BPR to achieve faster convergence and better results. In this thesis, we use the simple uniform negative sampler for a comparison of our proposed methods. Designing a better negative sampler doesn't fall into the scope of this thesis.

While BPR is a pair-wise loss function, we can also use point-wise loss functions for implicit feedback recommendation. For implicit feedback data, we can assign the label "1" for positive interactions and label "0" for unobserved ones.

The point-wise square loss under this setting can be formulated as

$$L(\Theta) = \sum_{u \in U} (\sum_{i \in I_u^+} (\hat{y}_{ui} - 1)^2 + \sum_{j \in I_u^-} \hat{y}_{uj}^2) + \lambda_\Theta ||\Theta||^2, \tag{2.2}$$

where $U$ is the whole user set and $I_u^-$ denotes the set of sampled negative examples for user $u$. This point-wise square loss function has also been used in the literature (Xin et al., 2016; Kabbur et al., 2013). Besides the square loss, binary cross-entropy can also be used as the point-wise loss function for implicit feedback data. In this case, the recommendation problem can be considered as a binary classification task. The positive examples and sampled negative examples are regarded as the two binary classes. The binary cross-entropy loss function is formulated as

$$L(\Theta) = -\sum_{u \in U} (\sum_{i \in I_u^+} \log(\sigma(\hat{y}_{ui})) + \sum_{j \in I_u^-} \log(1 - \sigma(\hat{y}_{uj}))) + \lambda_\Theta ||\Theta||^2. \tag{2.3}$$

We can see that the binary cross-entropy loss will try to push the positive $\hat{y}_{ui}$ to high values while decreasing the sampled negative $\hat{y}_{uj}$. As a result, when we perform item ranking according to $\hat{y}_{ui}$, the (potential) positive examples will be ranked higher in top positions. Besides, Shi et al. (2012) proposed a list-wise training approach to train recommenders from implicit feedback, which aims to directly optimize the mean reciprocal rank of the top-$k$ recommended list.

Note that we do not specify the recommendation model for the described loss functions, which means that these loss functions are generic solutions and can be used for various recommendation models. For example, when we use these sampling-based loss functions to factorization machines, we can consider the observed interactions as positive examples. Based on positive examples, we can keep the user profile and the other context features unchanged and replace positive item features with the sampled negative ones. The fixed user profile, context features, and negative item features compose negative examples, which are further fed to the described training loss functions for pair-wise comparison or point-wise learning. All the described sampling-based loss functions are differentiable and can be trained through gradient descent and back-propagation.

#### 2.1.2.2 Non-Sampling Methods

Sampling-based methods have been widely used in both academia and industry due to their effectiveness and scalability. However, these kinds of methods still have shortcomings. For example, the performance of these methods is highly influenced by the sampling distribution and sample size (Yuan et al., 2018; Xin et al., 2018). Besides, sampling-based methods can't provide all kinds of negative signals.

To address these issues, the other solution is to consider all unobserved interactions as negative examples (Pan et al., 2008; Hu et al., 2008; He et al., 2016d; Bayer et al., 2017). We can refer this kind of approached as non-sampling methods. Obviously, treating all unobserved interactions as negative examples is also problematic because the unobserved items could also be positive. The reason for the missed interactions may just because they are not exposed to the user. Therefore, when using non-sampling methods, we usually utilize a flexible re-weighting schema for negative examples. The square loss function for non-sampling approaches can be formulated as

$$L(\Theta) = \sum_{u \in U} (\sum_{i \in I_u^+} (\hat{y}_{ui} - y^+)^2 + \sum_{j \in I \setminus I_u^+} \alpha_{uj} (\hat{y}_{uj} - y^-)^2) + \lambda_\Theta ||\Theta||^2, \tag{2.4}$$

where $y^+$ and $y^-$ denote the positive and negative labels respectively (e.g., $y^+ = 1$ and $y^- = 0$), $I$ is the whole item set and $\alpha_{uj}$ is the weighting for negative example $(u, j)$.

However, we can see that directly optimizing Eq.(2.4) has a time complexity of $O(|U \times I|)$, which is extremely large for real-world recommender systems. To achieve efficient computation of non-sampling methods, plenty of research has been conducted (Yuan et al., 2017; He et al., 2016d; Xin et al., 2018; Bayer et al., 2017). For example, Xin et al. (2018); Yuan et al. (2018) have shown that if the weight $\alpha_{uj}$ is defined item-oriented (i.e., $\alpha_{uj} = \alpha_j$) or user-oriented (i.e., $\alpha_{uj} = \alpha_u$) and the prediction model of $\hat{y}_{ui}$ is a linear model, the time complexity to compute loss function Eq.(2.4) can be reduced to a complexity which is only related to the number of positive examples without any approximation. The key idea is to use commutative laws which enable us to perform pre-computation and further improve the time complexity. We have conducted related research in the supporting publication Xin et al. (2018) and Yuan et al. (2018). While because

the focus of this thesis is deep learning-based methods and the proposed methods in Xin et al. (2018) and Yuan et al. (2018) are limited to the linear model, so we don't elaborate on these approaches in detail here.

As described above, methods to optimize the non-sampling square loss (i.e., Eq(2.4)) are usually limited to the linear model. For deep learning-based recommendation models, another solution of the non-sampling loss function is the full softmax cross-entropy loss function (Yuan et al., 2019; Xin et al., 2020). In that case, the recommendation problem is considered as a multi-class classification task. Each candidate item corresponds to a class. Through a deep recommendation model, we can get the classification logits for each class (item), and then a softmax cross-entropy loss function can be applied upon these logits, which will push the logit of the correct class (interacted item) toward a higher value while decreasing the logits of the other classes (items). We will describe these methods in respective chapters.

### 2.1.3 Evaluation of Implicit Feedback Recommendation

As stated in section 1.2, the task on explicit feedback is usually rating prediction while on implicit feedback data, the task is item ranking. The evaluation of explicit ratings is relatively simple and usually defined as the difference between the predicted ratings and the ground-truth ones. The evaluation metrics for explicit recommendation data include but are not limited to mean absolute error (MAE) and root mean squared error (RMSE). Because in this thesis we focus on implicit feedback recommendation, so in the following sections we will describe the evaluation procedure for implicit feedback more precisely. Please note that in this thesis, we only consider off-line evaluation. Online A/B test is not in our research scope because it often requires access to a working commercial recommender system.

#### 2.1.3.1 Evaluation Protocols

For the evaluation purpose, the whole data is usually divided into training sets, validation sets, and test sets. The training sets are used to train the model. The validation sets are used to tune the model (e.g., selecting hyper-parameters and performing early-stop). The final performance is obtained in test sets. In some

cases, there may also be just training sets and test sets. Generally speaking, the data split strategy includes the following two folds:

- $k$-fold cross-validation: $k$-fold cross-validation first splits the whole datasets into $k$ groups. Each group will be taken as the test sets iteratively and the model is trained on the remaining groups. As a result, the model will be trained $k$ times and the average scores are considered as the final results.

- Leave-one-out evaluation: Leave-one-out evaluation only leaves one sample per user for test purpose (two samples (i.e., one for validation set and one for test set) if there is a validation set). This strategy is often used if the data contains time information. In that case, the latest one or two samples per user are held out for validation and test. The remaining data is considered as the training set. This naturally fits the recommendation scenario, since what we care about is which item will be interacted with by the user in the next timestamp.

The detailed evaluation procedure for each proposed method in this thesis is given in the corresponding chapter.

#### 2.1.3.2 Evaluation Metrics

The target of a recommender system can be elaborated from two perspectives. Firstly, from the user's perspective, a recommender system aims to help the user find the most interesting items from a large volume of information. Secondly, from the perspective of service providers, they expect the recommender system can help to increase traffic and further improve the profit. Generally speaking, a recommender system usually considers several factors to optimize, such as accuracy, novelty, and diversity. In the following, we provide a brief description of these factors.

- Accuracy. Accuracy may be the most basic and important factor for a recommender system (Castells et al., 2011; Vargas and Castells, 2011). It measures whether the recommended items match the user's interests. It is usually defined according to whether there are interactions between the user and the recommended items. We will provide more details later.

- Novelty. Novelty is usually defined as whether the agent can provide unusual or novel items for recommendation (Hurley and Zhang, 2011). It is also an important factor for a real-world recommender system. From the user perspective, novelty can help to prevent the user from being bored and also explore new user interests. There is a trade-off between exploration and exploitation in recommendation. Accuracy can be regarded as exploitation while novelty serves as exploration. From the perspective of service providers, novel recommendation can also help to promote new items and increase product sales.

- Diversity. The accuracy factor could result in a case that similar items or one kind of item account for the most positions in the recommendation list. However, a recommender system should generate recommendation that contains more diverse items so the user would have more choices and thus have a larger probability to interact with the recommendation list (Aggarwal et al., 2016). The novelty factor focuses on the comparison between the current recommendation and previous recommendations while the diversity is usually defined as the difference within one recommendation list (Vargas and Castells, 2011).

Although the recommendation community has recognized that novelty and diversity are important for a real-world recommender system, the research on this topic is still in the early-stage. Benchmark evaluation metrics about novelty and diversity are also not well-established. In this thesis, we focus on the most important evaluation factor: accuracy, which is also the main research line in the recommendation community.

Since the task of implicit recommendation is item ranking, so the ranking-based metrics such as Hit Ratio (HR), Mean Reciprocal Rank (MRR) (Shi et al., 2012) and Normalized Discounted Cumulative Gain (NDCG) (Järvelin and Kekäläinen, 2002) are commonly used. Besides, as classification-based training objectives can also be used for implicit feedback data, Precision, Recall, and Area Under ROC Curve (AUC) can also be used as evaluation metrics. In the following sections, we will introduce the detail of these evaluation metrics.

- Hit ratio. HR@$N$ measures whether the user interact with items which are

in the top-$N$ positions of the recommendation list. It is usually adopted in the leave-one-out evaluation protocol. In that case, it can be formulated as

$$HR@N = \frac{\#hits}{|U|}. \tag{2.5}$$

A hit means the ground-truth item is in the top-$N$ positions of the recommendation list for a user.

- Reciprocal Rank (Shi et al., 2012) is defined on the rank of the first relevant item in the recommendation list of a user, which can be formulated as

$$RR = 1/\text{rank} \tag{2.6}$$

The Mean Reciprocal Rank (MRR) is the average of the reciprocal ranks of all users.

$$MRR = \frac{1}{|U|}\sum_{u=1}^{|U|} 1/\text{rank} \tag{2.7}$$

Under leave-one-out evaluation settings, MRR can be seen as a weighted version of HR, by assigning the weight 1/rank to the hit records.

- NDCG is a popular ranking-based evaluation metric. It measures the accuracy of a recommendation agent based on the graded relevance of the recommended items (Järvelin and Kekäläinen, 2002). NDCG is formulated as

$$NDCG@N = \frac{DCG@N}{IDCG@N}, \text{where } DCG@N = \sum_{i=1}^{N} \frac{2^{\text{rel}_i} - 1}{\log_2(i+1)} \tag{2.8}$$

where $\text{rel}_i$ indicates whether the $i$-th item in the recommendation list is relevant to the user (i.e., interacted by the user). Usually a binary indicator function is used for $\text{rel}_i$ as $\text{rel}_i = 1$ if the $i$-th item is relevant, otherwise $\text{rel}_i = 0$. $IDCG@N$ is similar with $DCG@N$ but it is calculated from the ground-truth data. For example, assuming for a ranked item list, the ground-truth data is $\{1, 0, 1, 0, 0\}$ while the model prediction is $\{1, 1, 0, 0, 0\}$, where '1' and '0' denote positive and negative labels. $IDCG$ will be computed as $\frac{1}{\log_2(1+1)} + \frac{1}{\log_2(3+1)} = 1.5$ while $DCG$ will be computed as $\frac{1}{\log_2(1+1)} = 1$. So NDCG in this example is calculated as $1/1.5 = 0.67$.

- Precision and recall are usually used for the binary classification task. Because in implicit feedback there are positive examples and (sampled) neg-

ative examples and binary classification loss function can also be used to train the recommendation model, so precision and recall can also be used to measure the recommendation accuracy.

Precision and Recall are defined as:

$$\text{Precision@N} = \frac{\#TP}{\#TP + \#FP}, \quad \text{Recall@N} = \frac{\#TP}{\#TP + \#FN} \tag{2.9}$$

where $\#TP$ is the number of items that are positive examples and occurs in the top-$N$ recommendation list, $\#FP$ is the number of items that occur in the list but are actually not interacted with by the user. $\#FN$ is the number of items that don't occur in the top-$N$ list but are interacted by the user according to the ground-truth data. The final reported precision and recall are usually the average across all users in the test set. Note that in the setting of leave-one-out evaluation, the average recall is equivalent to HR.

- AUC (Rendle et al., 2009a) is also a classification-based metric. It measures whether the model has a good ability to rank positive examples higher than negative examples. The average AUC can be formulated as

$$\frac{1}{|U|} \sum_{u} \frac{1}{|I_u^+||I_u^-|} \sum_{i \in I_u^+} \sum_{j \in I_u^-} I(\hat{y}_{ui} > \hat{y}_{uj}) \tag{2.10}$$

where $I(\cdot)$ is an indicator function as

$$I(\hat{y}_{ui} > \hat{y}_{uj}) = \begin{cases} 1, & \hat{y}_{ui} > \hat{y}_{uj} \\ 0, & \text{else} \end{cases} \tag{2.11}$$

## 2.2 Recommendation Model Overview

In this section, we provide a brief literature review of several notable recommendation models, including (shadow) factorization models and deep learning models. The detailed review of related work is provided in each contribution chapter.

### 2.2.1 Factorization Models

Factorization models, also known as latent factor models, are the most successful methods in the recommendation field, due to their effectiveness and efficiency. In this subsection, we briefly introduce several most notable factorization models.

#### 2.2.1.1 Matrix Factorization

Matrix factorization (MF) (Koren et al., 2009) achieved great success in both academia and industry. It serves as the most widely adopted model in the early-stage of the model-based recommender system. Even nowadays, MF still plays an important role in most serving recommendation agents. MF is also a hot research topic in other machine learning fields like natural language processing (Pennington et al., 2014).

The basic idea of MF is to map users and items to latent representations (i.e., embeddings) in a common latent space. Then the user preference is defined as the inner product between user embeddings and item embeddings.

Let $\mathbf{q}_i \in \mathbb{R}^k$ denotes the latent representation of item $i$ and $\mathbf{p}_u \in \mathbb{R}^k$ denotes the latent presentation of user $u$. The latent representation indicates the item (user) feature with respect to certain factors. While we couldn't tell the specific semantic meaning of these factors, so they are called "latent" factors. Then the predicted user preference is defined as how much relevance are the user and item latent representations, which is formulated as the inner product:

$$\hat{y}_{ui} = \mathbf{q}_i^T \mathbf{p}_u = \sum_{f=1}^{k} p_{uf} q_{if} \tag{2.12}$$

where higher value of $\hat{y}_{ui}$ denotes a higher user preference.

Besides the inner product between user and item embeddings, Gogna and Majumdar (2015) proposed that the prediction should also contain certain biases. For example, a tedious user may tend to assign a smaller preference to all items while a good item may receive a large preference from all kinds of users. When taking these biases into consideration, BiasedMF (Gogna and Majumdar, 2015) defined the prediction logits as

$$\hat{y}_{ui} = b + b_u + b_i + \mathbf{q}_i^T \mathbf{p}_u \tag{2.13}$$

where $b_u$, $b_i$ and $b$ are user bias, item bias, and the average bias on the training data. BiasedMF is usually used to perform rating prediction on explicit feedback data. For implicit feedback, $b_u$ and $b$ can be removed because the two biases will not affect the ranking of items for the same user.

FISM (Kabbur et al., 2013) is an item-based factorization model. Unlike MF and BiasedMF which factorize the user-item interaction matrix, FISM using

factorization techniques on item-item similarity matrix. The prediction rule of FISM can be formulated as

$$\hat{y}_{ui} = b + b_u + b_i + (|I_u^+| - 1)^{-\alpha} \sum_{j \in I_u^+ \setminus \{i\}} \mathbf{q}_j^T \mathbf{p}_i. \tag{2.14}$$

The inner product $\mathbf{q}_j^T \mathbf{p}_j$ can be regarded as the similarity between the target item $i$ and historical item $j$. It conforms with the basic idea of collaborative filtering: user preference can be inferred from interacted (similar) items. In the FISM model, there is no user embedding, the user is directly modeled as his/her interacted items. FISM is a widely adopted item-based collaborative filtering method. Plenty of work has been done following the research line of FISM, such as incorporating user information (Elbadrawy and Karypis, 2015; Xin et al., 2016), neural network-enhanced approaches (Wu et al., 2016; He et al., 2018c,b) and involving local latent space (Christakopoulou and Karypis, 2018; Lee et al., 2013).

Although these methods have improved the performance of FISM, all of them are based solely on the collaborative similarity between items. This item relation is coarse-grained and lacks semantic meaning, introducing the bottleneck of the model and the difficulty of generating convincing results.

SVD++ (Koren, 2008) is a more advanced matrix factorization models. It extends the naive matrix factorization by considering both explicit and implicit feedback. For the modeling of users, besides the user embedding, SVD++ also considers the interacted items from the users' interaction history. According to the results of Koren and Bell (2015), SVD++ achieves a better accuracy compared with naive MF and BiasedMF. The prediction rule of SVD++ is formulated as

$$\hat{y}_{ui} = b_u + b_i + b + \mathbf{q}_i^T (\mathbf{p}_u + |I_u^+|^{-\frac{1}{2}} \sum_{j \in I_u^+} \mathbf{z}_j), \tag{2.15}$$

where $\mathbf{z}_j \in \mathbb{R}^k$ is another item embedding, representing the implicit collaborative signal. We can see that for the user modeling of SVD++, despite the user embedding $\mathbf{p}_u$, a second term $\sum_{j \in I_u^+} \mathbf{z}_j$ is considered. The second term is the aggregation of the user's historical interacted items, which is actually the inner product part of FISM. So SVD++ can be seen as the combination of BiasedMF and FISM.

The naive MF and SVD++ are designed to tackle data which has no side

information. However, despite the essential user ID and item ID, plenty of side information is also available, which may further help to improve the recommendation accuracy. SVDFeature (Chen et al., 2012) is a recommendation model which are designed for feature-enhanced recommendation. The prediction rule of SVDFeature is formulated as

$$\hat{y}_{ui} = \sum_{d=1}^{s} \gamma_d b_d^g + \sum_{d=1}^{n_1} \alpha_d b_d^u + \sum_{d=1}^{n_2} \beta_d b_d^i + (\sum_{d=1}^{n_1} \mathbf{p}_d \alpha_d)^T (\sum_{d=1}^{n_2} \mathbf{q}_d \beta_d), \tag{2.16}$$

where $\alpha_d$, $\beta_d$ and $\gamma_d$ denote user features, item features and global features, respectively. $b^g$,$b^u$,$b^i$ are the bias vectors for global features, user features and item features, correspondingly. $\mathbf{p}$ and $\mathbf{q}$ are embeddings of user features and item features. The total learning space is $\Theta = \{b^g, b^u, b^i, \mathbf{p}, \mathbf{q}\}$. Despite the bias terms, we can see that SVDFeature models the users as the sum of user feature embeddings (i.e., $\sum_{d=1}^{n_1} \mathbf{p}_d \alpha_d$) while items are modeled similarity as the sum of item feature embeddings (i.e., $\sum_{d=1}^{n_2} \mathbf{q}_d \beta_d$). The interaction between users and items is still modeled as the inner product between users and items. As a result, SVDFeature can recover the BiasedMF model if the feature vector only includes user ID and item ID.

We can see that SVDFeature models the interaction between user-field features and item-field features while the feature interaction within each field is not considered. Another successful feature-based recommendation model is factorization machines, which model all pair-wise feature interactions (Rendle, 2010).

#### 2.2.1.2 Factorization Machines

Factorization Machine (Rendle, 2010, 2012) is a generic framework that integrates the advantages of flexible feature engineering and high-accuracy prediction of latent factor models. In FM, every transaction (interaction) is represented by a multi-field categorical feature vector $\mathbf{x} \in \mathbb{R}^m$ which utilizes one-hot/multi-hot encoding to depict contextual information. An example is illustrated as follows with three feature fields.

$$\underbrace{[0,0,0,1,0,0,0]}_{\text{weekday=Thursday}} \quad \underbrace{[0,1,...,0]}_{\text{location=London}} \quad \underbrace{[1,1,0,...,0]}_{\text{historical items (multi-hot)}}$$

The scoring function of FM is defined as

$$\hat{y}_{FM}(\mathbf{x}) = w_0 + \sum_{i=1}^{m} w_i x_i + \sum_{i=1}^{m} \sum_{j=i+1}^{m} x_i x_j \cdot \langle \mathbf{v}_i, \mathbf{v}_j \rangle, \tag{2.17}$$

where $w_0$ represents the global bias, $w_i$ represents the bias factor for the $i$-th variable. The pairwise interaction of feature $x_i$ and $x_j$ is captured by a factorized parametrization $\langle \mathbf{v}_i, \mathbf{v}_j \rangle = \sum_{f=1}^{d} v_{if} v_{jf}$, where $\langle \cdot, \cdot \rangle$ denotes the inner product of two vectors. $\mathbf{v}_i \in \mathbb{R}^d$ can be seen as the embedding vector for feature $x_i$. Directly calculating Eq.(2.17) has a time complexity of $O(m^2)$, which is inefficient when $m$ is large. To reduce the time complexity, Rendle (2010) reformulate Eq.(2.17) as:

$$\hat{y}_{FM}(\mathbf{x}) = w_0 + \sum_{i=1}^{m} w_i x_i + \frac{1}{2}(\langle \sum_{i=1}^{m} x_i \mathbf{v}_i, \sum_{i=1}^{m} x_i \mathbf{v}_i \rangle - \sum_{i=1}^{m} \langle x_i \mathbf{v}_i, x_i \mathbf{v}_i \rangle). \tag{2.18}$$

Eq.(2.18) can be calculated with the time complexity of $O(m)$, which is a linear time complexity with respect to $m$.

The original FM is designed for rating prediction tasks (Rendle, 2010), while the task of implicit recommendation is item ranking. Previous research (Cremonesi et al., 2010) has demonstrated that rating prediction-based recommendation algorithms are not well suited to perform the ranking for item recommendations. To tackle this problem, Guo et al. (2016) proposed pair-wise ranking factorization machines (PRFM) which combine pair-wise learning to ranking (LTR) techniques with original FM. Besides, Yuan et al. (2016) proposed LambdaFM to sample more informative negative examples for pair-wise ranking.

Some other research has also been done to improve FM, including accelerating the training speed (Freudenthaler et al., 2011; Rendle, 2013), selecting better features (Cheng et al., 2014) and distributed factorization machines (Li et al., 2016; Rendle et al., 2016).

#### 2.2.1.3 Tensor Decomposition

Matrix factorization represents the input user-item interaction as a two-dimensional matrix while tensor decomposition is usually used to tackle multi-dimensional tensors. For example, a three-dimensional tensor can be used to represent sequential user-item interactions with time as the third dimension. Tag-based recommendation (Rendle et al., 2009a) may also utilize tags as an additional input dimension.

Context-aware recommendation can use tensors to represent the interactions between users, items, and contexts (Karatzoglou et al., 2010).

Tucker decomposition (Tucker, 1966) is one of the most adopted tensor decomposition methods. It decomposes a tensor into a set of matrices and one small core tensor. For example, for a three-dimensional tensor, the prediction rule is formulated by multiplying three matrices with the core tensor:

$$Y = \hat{C} \times_u \hat{U} \times_i \hat{I} \times_t \hat{T} \tag{2.19}$$

where $\hat{C}$ is the core tensor. $\hat{U}$, $\hat{I}$, and $\hat{T}$ are the three matrices for users, items, and the third dimension, correspondingly. $\times x$ denotes the tensor product that multiples a matrix on dimension $x$ of a tensor. The parameters in the learning space are shown as:

$$\Theta = \{\hat{C} \in \mathbb{R}^{k_U \times k_I \times k_T}, \hat{U} \in \mathbb{R}^{|U| \times k_U}, \hat{I} \in \mathbb{R}^{|I| \times k_I}, \hat{T} \in \mathbb{R}^{|T| \times k_T}\}$$

where $k_U$, $k_I$ and $k_T$ are the dimension of the core tensor with respect to the user dimension, item dimension and the third dimension. $T$ is the size of the third dimension. The predicted result $Y$ is also a third-dimension tensor as $Y \in \mathbb{R}^{|U| \times |I| \times |T|}$. For a specific element $\hat{y}_{uit}$ in tensor $Y$, the predicted value is formulated as:

$$\hat{y}_{uit} = \sum_{f_1=1}^{k_U} \sum_{f_2=1}^{k_I} \sum_{f_3=1}^{k_T} c_{f_1,f_2,f_3} \hat{u}_{u,f_1} \hat{i}_{i,f_2} \hat{t}_{t,f_3} \tag{2.20}$$

Although tensor decomposition can be used for multi-dimensional tensors, the factorization of three-dimensional tensors is the most common case. More dimensions would lead to the overfitting problem and much more time complexity.

### 2.2.2 Deep Learning Models

Although factorization models have achieved great success in both industry and academia, they are still limited to the shadow structure and model linearity, which encounters difficulties to model complex user-item interactions (He et al., 2017b). The implicit feedback motivates researchers to develop recommendation methods with better model expressiveness and fidelity to capture more complex interaction signals. On the other hand, deep learning methods have shown a great advantage to learn knowledge from complex data, such as images (He et al., 2016b) and languages (Vaswani et al., 2017). The community of recommender systems has also paid attention to utilizing deep learning models for recommendation. In this sub-

section, we provide a brief review of related deep learning-based recommendation models, including neural collaborative filtering (NCF), deep learning-enhanced factorization machines, convolutional neural network-based recommendation, attention mechanism for recommendation, graph convolution matrix completion (GCMC), and next-item recommendation.

#### 2.2.2.1 Neural Collaborative Filtering

NCF (He et al., 2017b) is one of the earliest attempts to use deep learning methods for recommendation. The key point of NCF is that the linear inner product cannot model complex user-item interactions. NCF proposed to use a neural network to replace the inner product to introduce non-linearity and improve model expressiveness. The NCF model contains two modules: a generalized matrix factorization (GMF) model and a multi-layer perceptron (MLP). More precisely, GMF extends MF by adding another layer and an activation function. The prediction rule of GMF is formulated as

$$\hat{y}_{ui} = \delta(\mathbf{h}^T(\mathbf{p}_u \odot \mathbf{q}_i)) \tag{2.21}$$

where $\mathbf{h} \in \mathbb{R}^k$ is the parameters of the added layer and $\delta$ is the activation function (e,g., sigmoid function in the original paper). $\odot$ is the element-wise product. We can see that if we use an identity function as the activation function and define $\mathbf{h}$ as a constant vector of 1, GMF will recover the basic matrix factorization. From this perspective, MF is a special case of GMF. Generally speaking, GMF can be seen as a model which reweights the latent factors of MF and then adds an activation function.

The MLP component is built upon the concatenation of user embeddings and item embeddings. To improve the model expressiveness, the authors of NCF stacked more hidden layers on the concatenation vector to learn high-order interaction signals. According to He et al. (2017b), the MLP model of NCF is

formulated as

$$\begin{aligned} \mathbf{z}_1 &= [\mathbf{p}_u, \mathbf{q}_i] \\ \psi_2(\mathbf{z}_1) &= \delta_2(\mathbf{W}_2^T \mathbf{z}_1 + \mathbf{b}_2) \\ &... \\ \psi_L(\mathbf{z}_{L-1}) &= \delta_L(\mathbf{W}_L^T \mathbf{z}_{L-1} + \mathbf{b}_L), \\ \hat{y}_{ui} &= \delta(\mathbf{h}^T \psi_L(\mathbf{z}_{L-1})) \end{aligned} \tag{2.22}$$

where $[\cdot]$ denotes the concatenation operation, $\mathbf{W}_x$, $\mathbf{b}_x$ and $\delta_x$ represent the weight matrix, bias vector and activation function for $x$-th layer respectively. Finally, NCF combines GMF and MLP through the final output layer. The prediction rule of NCF is formulated as

$$\hat{y}_{ui} = \delta(\mathbf{h}^T[(\mathbf{p}_u \odot \mathbf{q}_i), \psi_L(\mathbf{z}_{L-1})]) \tag{2.23}$$

In the original NCF paper, GMF and MLP use separate embedding tables. The author argues that two separate embedding tables will help the NCF model to achieve best performance on both GMF component and MLP component.

#### 2.2.2.2 Neural Network-based Factorization Machines

The NCF model is designed for recommendation with only user IDs and item IDs, while the FM model (Rendle, 2010) are tailed for recommendation with side information as feature vectors. The limitation of FM is that it only models the second-order feature interactions in a linear way[1], while the advantage of deep neural network (DNN) is that it can learn non-linear inherent structures from input data (LeCun et al., 2015). As a result, plenty of neural network-based factorization machines are proposed to overcome the limitations of FM.

NFM extend the idea of NCF (i.e., using deep neural networks to capture interaction signals) to factorization machines. The most important contribution of NFM is the proposed Bilinear Interaction (Bi-Interaction) pooling operation which is used to model pair-wise feature interactions. The Bi-Interaction pooling function is formulated as

$$\mathbf{f}_{BI}(\mathbf{x}) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\mathbf{v}_i \odot \mathbf{v}_j) x_i x_j \tag{2.24}$$

where $\mathbf{x}$ is the input sparse feature vector, $\mathbf{v}_i$ and $\mathbf{v}_j$ are the embedding vectors

[1] Although FM has high-order formulations (Rendle, 2010), it still belongs to linear models and is proved to be difficult to estimate.

of feature $x_i$ and $x_j$, respectively and $n$ is the number of features. Computing such a function requires a time complexity of $O(n^2)$. The authors of NFM utilize the similar techniques of FM to reduce the time complexity. The mathematical transformation is formulated as

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n}(\mathbf{v}_i \odot \mathbf{v}_j)x_i x_j = \frac{1}{2}((\sum_{i=1}^{n} x_i\mathbf{v}_i) \odot (\sum_{i=1}^{n} x_i\mathbf{v}_i) - \sum_{i=1}^{n}(x_i\mathbf{v}_i \odot x_i\mathbf{v}_i)). \quad (2.25)$$

This reformulation enables a time complexity of $O(n)$. The output of the Bi-Interaction layer is still a vector. Then NFM stacks a MLP upon the Bi-Interaction layer to learn complex and high-order interaction signals. The prediction rule of NFM is shown as:

$$\hat{y}(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \mathbf{h}^T \delta_L(\mathbf{W}_L(...\sigma_1\mathbf{W}_1\mathbf{f}_{BI}(\mathbf{x}) + \mathbf{b}_1)...) + \mathbf{b}_L \quad (2.26)$$

where $\mathbf{W}_l$,$\mathbf{b}_l$ and $\delta_l$ are the weight matrix, bias vector and activation function for the $l$-th layer of the MLP, correspondingly. $\mathbf{h}$ is the weight vector of the final output layer.

Besides, Zhang et al. (2016b) proposed factorization machine supported neural network (FNN) which uses FM as the bottom layer of a DNN. Guo et al. (2017) borrowed the idea from Wide&Deep (Cheng et al., 2016) and formulated the scoring function as the sum of FM and a DNN. Although these proposed models have a more expressive capability, the way they modeling feature interactions (e.g., inner product, element-wise product, and concatenation) assumes that embedding dimensions are independent of each other. However, it has been proven to be an impractical assumption (Zhang et al., 2014). Besides, they all use an implicit manner (e.g, simple hidden layers) to learn high-order interaction signals. Finally, the architecture they used (i.e., MLP) also makes them harder to train and go deeper (He et al., 2018a).

#### 2.2.2.3 Convolutional Neural Networks for Recommendation

Unlike MLP which suffers from a large number of parameters and low generalization ability, CNN is much easier to train and go deeper because of the shared parameters. The power of CNN in feature extraction makes it widely used to learn signals from images and text (Zhang et al., 2017). Plenty of research has been done in that fashion. For example, He and McAuley (2016b) proposed to

use CNN to learn item visual latent factors and improve the pair-wise ranking (Rendle et al., 2009b). Zheng et al. (2017) adopt two parallel CNN to model user behaviors and item properties by abstracting review comments, then FM is applied in the final layer to perform scoring. Yu et al. (2018) proposed to use CNN to extract aesthetic features when generating clothing recommendations.

Although CNN is usually used to model local patterns from images, it can actually serve as a general signal learner. Compared with MLP, CNN is much easier to train with fewer parameters and potential deeper structures (He et al., 2018a). Plenty of research about using CNN on embedding maps also emerged recently. Liu et al. (2015) proposed to directly apply CNN on feature map to perform click prediction. Tang and Wang (2018) utilized CNN to capture sequential signals from the item embedding map. He et al. (2018a); Du et al. (2019) proposed to use CNN to learn the user-item interaction patterns from an outer product-based interaction map. This research motivates us to further explore CNN to learn high-order feature interaction signals for implicit feedback recommendation in Chapter 3.

#### 2.2.2.4 Attention Mechanism for Recommendation

The attention mechanism has become very popular in the fields of computer vision (Mnih et al., 2014; Xu et al., 2015) and natural language processing (Vaswani et al., 2017; Bahdanau et al., 2014) because of its enhanced performance and the resulting interpretability for deep learning models. The key insight of attention is that human tends to pay different weights to different parts of the whole perception space. Based on this motivation, He et al. (2018c) improved FISM by replacing the mean aggregation with the attention-based summation, also known as the NAIS model. Chen et al. (2017) proposed to utilize the attention mechanism to generate multimedia recommendation. Kang and McAuley (2018) exploited self-attention for sequential recommendation. There are many other works focusing on involving attention mechanism for better recommendation (Xiao et al., 2017; Tay et al., 2018).

#### 2.2.2.5 Next Item Recommendation

Next-item recommendation aims to provide sequential recommendation for next timestamps given the user-item interaction sequence, which is one of the most common use cases in the real-world scenario. Early work focusing on next item recommendation mainly rely on Markov Chain (MC) models (He and McAuley, 2016a; Cheng et al., 2013; Rendle et al., 2010) and factorization-based methods (Rendle, 2010; Hidasi and Tikk, 2016). Rendle et. al (Rendle et al., 2010) introduced to use first-order MC to capture short-term user preferences and combined the MC with matrix factorization (MF) (Koren et al., 2009) to model long-term preferences. Methods with high-order MCs that consider more longer interaction sequences were also proposed in (He and McAuley, 2016a; He et al., 2016c). Factorization-based methods such as factorization machines (FM) (Rendle, 2010) can utilize the previous items a user has interacted with as context features. The general factorization framework (GFF) (Hidasi and Tikk, 2016) models a session as the average of the items that the user interacted within that session.

MC-based methods face challenges in modeling complex sequential signals such as skip behaviors in the user-item sequences (Tang and Wang, 2018; Yuan et al., 2019) while factorization-based methods do not model the order of user-item interactions. As a result, plenty of deep learning-based approaches have been proposed to model the interaction sequences more effectively. Hidasi et al. (2015) proposed to utilize gated recurrent units (GRU) (Cho et al., 2014) to model the session. Tang and Wang (2018) and Yuan et al. (2019) utilized convolutional neural networks (CNN) to capture sequential signals. Kang and McAuley (2018) exploited the well-known Transformer (Vaswani et al., 2017) in the field of next item recommendation with promising results. Generally speaking, all of these models can serve as models which aim to map a sequence of user-item interactions to a latent representation that describes the corresponding user state in that timestamp.

#### 2.2.2.6 Graph Convolution Matrix Completion

GCMC (Berg et al., 2017) is one of the earliest attempts to utilize graph convolution for recommendation. The user-item interactions can be naturally represented

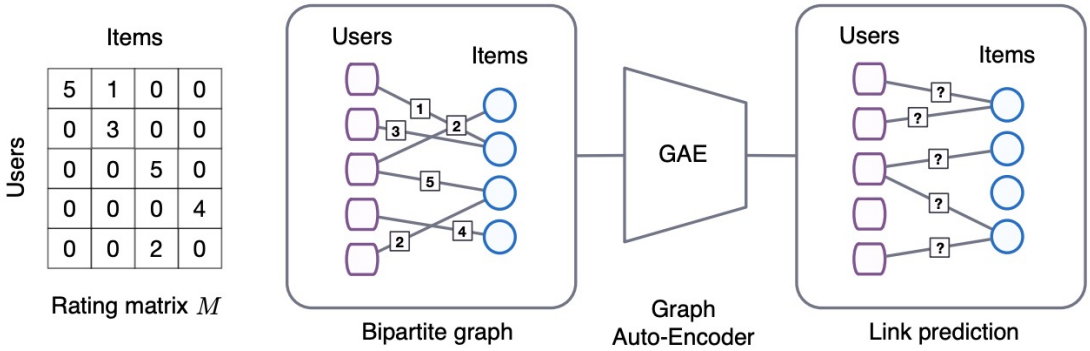



Figure 2.1: The model structure of GCMC from Berg et al. (2017). The rating matrix of user-item interactions can be considered as a bipartite graph. Users and items are nodes, edges correspond to interactions. Numbers on edges denote the rating a user has given to a particular item. The prediction for unobserved interactions can be seen as a link prediction problem.

as an interaction graph. The procedure of GCMC is shown in Figure 2.1. Each rating corresponds to one kind of edge and then the rating prediction task can be regarded as a link prediction task. The model is trained using an end-to-end trainable graph auto-encoder.

The graph auto-encoder is based on the message passing schema of graph convolution. Taking a user node $i$ as an example, $\mu_{j\to i,r}$ denotes the message from item $j$ to user $u$ through link type $r$ and is formulated as

$$\mu_{j\to i,r} = \frac{1}{c_{ij}}\mathbf{W}_r\mathbf{x}_j. \tag{2.27}$$

where $\mathbf{x}_j \in \mathbb{R}^k$ is the initial embedding of item $j$. $\mathbf{W}_r \in \mathbb{R}^{k\times k'}$ is the weight matrix with respect to the link type $r$, mapping the initial node embedding to the latent space. $c_{ij}$ is the normalization constant. Let $d_i$ denote the degree of node $i$, $c_{ij}$ is usually formulated as $1/d_i$ or $1/\sqrt{d_i d_j}$. Based on this message passing rule, the new user representation $\mathbf{u}_i$ can be formulated as

$$\mathbf{u}_i = \delta_1(\mathbf{W}\delta_2(\text{accum}(\sum_{j\in N_{i,1}}\mu_{j\to i,1}, ..., \sum_{j\in N_{i,R}}\mu_{j\to i,R}))), \tag{2.28}$$

where $N_{i,r}$ denotes the neighbourhood of user $u$, under the specific link type $r$. accum$(\cdot)$ is the accumulation operation, such as concatenation of vectors or summation of all messages. $\mathbf{W} \in \mathbb{R}^{k'\times o}$ is the trainable weight matrix. $\delta_1$ and $\delta_2$ are activation functions. Figure 2.2 demonstrates an illustration of this message passing schema.

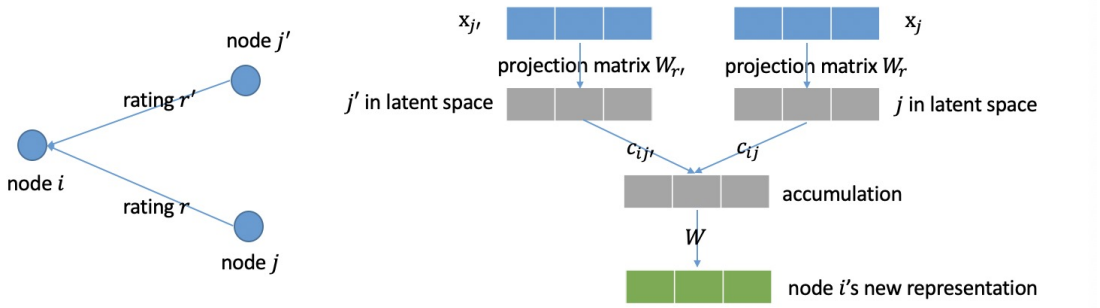



Figure 2.2: The message passing schema of GCMC.

The new item representation $\mathbf{v}_j$ can be calculated analogously. The final rating prediction is a link prediction task, which aims to predict the edge between user $i$ and item $j$ belongs to which link type:

$$p(\hat{y}_{ij} = r) = \frac{e^{\mathbf{u_i^T} \mathbf{Q}_r \mathbf{v}_j}}{\sum_{s \in R} e^{\mathbf{u_i^T} \mathbf{Q}_s \mathbf{v}_j}}, \tag{2.29}$$

where $\mathbf{Q}_r \in \mathbb{R}^{o \times o}$ is a trainable parameter matrix with respect to the link type $r$.

Although GCMC achieves promising performance compared with MF (Berg et al., 2017), we can see that the GCMC model is proposed for the rating prediction task on explicit data. Also, GCMC can not incorporate the context information.

## 2.3 Preliminaries of Reinforcement Learning

The described training procedure in section 2.1.2 belongs to the scope of supervised learning. Actually, most state-of-the-art recommendation methods are based on supervised learning. The agent is trained to optimize a loss function, which is defined as the discrepancy between model predictions and ground-truth values. However, such a loss function may not reflect certain expectations of service providers, for example, to gain profits from long-term user interactions.

On the other hand, reinforcement learning has achieved great success in the field of game control. A RL agent is trained to interact with the environment to obtain the maximum long-term cumulative reward, as shown in Figure 2.3. It naturally fits the demand of recommender systems (i.e., maximizing cumulative gains in a whole interaction session). As a result, exploiting RL for recommenda-

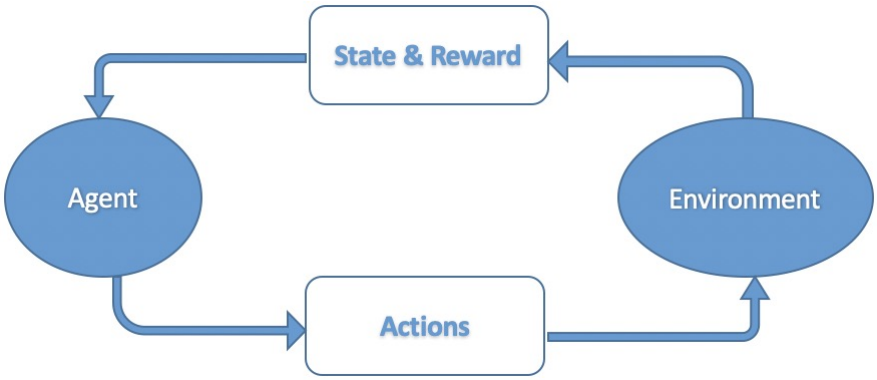



Figure 2.3: A RL example. The RL agent is trained to interact with the environment by taking actions. The environment will update the state and return a reward according to the taken action. The target of the RL agent is to get the maximum cumulative reward in a whole interaction session.

tion has become a promising research direction. In this section, we provide the basic preliminaries of RL.

### 2.3.1 Markov Decision Process

The interaction procedure between an agent and the environment can be referred to as a Markov Decision Process (MDP). RL algorithms are actually the methods to solve a MDP to find the optimal solution in which the agent can obtain the maximum cumulative reward. The Markov property of MDP means that only the current state of the environment will affect the decision making of the agent other than all historical states. A MDP is formulated as tuples of $(\mathcal{S}, \mathcal{A}, \mathbf{P}, R, \rho_0, \gamma)$ where

- $\mathcal{S}$: the space to describe the environment state.
- $\mathcal{A}$: the action space of the agent.
- $\mathbf{P}$: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the state transition probability, describing the state update by taking actions.
- $R$: $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, where $r(s, a)$ denotes the immediate reward by taking action $a$ at state $s$.
- $\rho_0$ is the initial state distribution.

- $\gamma$ is the discount factor for future rewards.

The solution for a given MDP is a policy $\pi(a|s)$, which maps the given state $s$ to (the distribution of) the action $a \in \mathcal{A}$. When the agent takes actions according to $\pi(a|s)$, we want to get the maximum expectation of discounted cumulative rewards, as shown in Eq.(2.30).

$$\max_{\pi} \mathbb{E}_{\tau\sim\pi}[R(\tau)], \text{ where } R(\tau) = \sum_{t=0}^{|\tau|} \gamma^t r(s_t, a_t). \tag{2.30}$$

To find the optimal solution of a MDP, we first introduce two value-based concepts: state-value function and action-value function.

The state-value function $V_\pi(s)$ of an MDP is the expected return starting from state $s$ and then following policy $\pi$. It can be formulated as:

$$V_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s], \text{where } G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \tag{2.31}$$

We can see that the value-function $V_\pi(s)$ can be decomposed into the sum of immediate reward and discounted value of successor state, as shown in Eq.(2.32).

$$V_\pi(s) = \mathbb{E}_\pi[r_{t+1} + \gamma V_\pi(s_{t+1})] \tag{2.32}$$

The action-value function $Q_\pi(s, a)$, aka Q-value, is the expected return starting from state $s$, taking action $a$, and then following policy $\pi$. It can be formulated as

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a]. \tag{2.33}$$

Similarly, the Q-value can also be decomposed as

$$Q_\pi(s, a) = \mathbb{E}_\pi[r_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1})]. \tag{2.34}$$

We further define the optimal $V_*(s, a)$ and $Q_*(s, a)$ as

$$V_*(s) = \max_{\pi} V_\pi(s), Q_*(s, a) = \max_{\pi} Q_\pi(s, a). \tag{2.35}$$

### 2.3.2 Value-Based Reinforcement Learning

According to the above definition, to get the maximum cumulative reward, an optimal policy should take actions that will lead to the highest optimal Q-values given the state $s$. If we know $Q_*(s, a)$, we can immediately get the optimal policy by

$$\pi_*(a|s) = \begin{cases} 1 & if a = argmax_a Q_*(s, a) \\ 0 & otherwise \end{cases} \tag{2.36}$$

Value-based RL aims to learn the optimal Q-values and then takes actions with highest optimal Q-values. The Bellman Equation (Bellman, 1966) provides the solution to learn the optimal Q-values. The Bellman Equation for optimal Q-values is defined as

$$Q_*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p^a_{ss'} \max_{a'} Q_*(s',a') \tag{2.37}$$

where $p^a_{ss'}$ is the probability of $s'$ if action $a$ is taken under state $s$.

Deep Q-learning (DQN) (Mnih et al., 2015) is one of the most successful value-based RL algorithms. DQN utilizes a replay buffer to store the interaction experience between the agent and the environment. Q-values are approximated through a deep learning model (i.e., deep neural network), aka Q-network. Based on the replay buffer and Eq.(2.37), DQN approximates the optimal Q-values as

$$Q_\theta(s_t,a) = r(s_t,a) + \gamma \max Q_\theta(s_{t+1},a') \tag{2.38}$$

where $s_t$ and $s_{t+1}$ are sampled from the replay buffer, $\theta$ denotes the parameter of the Q-network. Then DQN trains the Q-network through time-difference (TD) learning. The loss function of one-step TD learning is formulated as

$$L_\theta = Q_\theta(s_t,a) - (r(s_t,a) + \gamma \max Q_\theta(s_{t+1},a'))^2. \tag{2.39}$$

This is a recursive learning procedure. In practical usage, to enhance stability, we usually use a fixed network (target network) to calculate $Q_\theta(s_{t+1},a')$ and synchronize the target network and the main network after certain training steps. Plenty of work has been done based on DQN. For example, Double Q-learning (Hasselt, 2010) proposed to use two Q-networks and iteratively train one of them to further enhance the training stability.

### 2.3.3 Policy-Gradient Approaches

Value-based RL algorithms need to calculate the value functions firstly and then choose the actions by selecting the highest Q-values while policy gradient-based methods aim to directly optimize the parameters of the policy network. This is especially helpful when the action space is continuous, in which there are infinite Q-values to be computed for value-based methods.

For policy gradient-based methods, the policy is directly parameterized through a neural network, aka policy network. We use $\pi_\theta(a|s)$ to denote the policy network parameterized by $\theta$. The expected cumulative reward then will also be

parameterized by $\theta$ as

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)]. \tag{2.40}$$

Then the parameter $\theta$ can be updated directly by

$$\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta). \tag{2.41}$$

By using the "log-trick", the gradient of $J(\theta)$ can be derived as

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] = \nabla_\theta \int \pi_\theta(\tau)R(\tau)d\tau = \int \nabla_\theta\pi_\theta(\tau)R(\tau)d\tau \\ &= \int \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau)R(\tau)d\tau = \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(\tau)R(\tau)] \\ &= \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)(\sum_{t=1}^{T} \log \pi_\theta(a_t|s_t))]\end{aligned} \tag{2.42}$$

We can see that the $R(\tau)$ term in $\nabla_\theta J(\theta)$ contains $T$ sources of variance. This may cause the instability of the learning. Instead, we can use the return $G_t$ to replace $R(\tau)$ since rewards of the past interactions should not contribute to the future gradient. It can also help to reduce the high variance. Hence, $\nabla_\theta J(\theta)$ can be further reformulated as

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\tau\sim\pi_\theta}[(\sum_{t=1}^{T} G_t \log \pi_\theta(a_t|s_t))]. \tag{2.43}$$

This is the key point of the notable REINFORCE algorithm (Williams, 1992). We can see from Eq.(2.43) that if $G_t$ is high, it means that on average we have taken actions that lead to high cumulative rewards. We then want to push the probabilities of these actions. On the other hand, if $G_t$ is low, we want to push down the probabilities of these actions.

Policy gradient-based approaches have drawn much attention recently. Plenty of following work has been conducted to improve the basic REINFORCE algorithm. For example, Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) improves the REINFORCE algorithm by defining a trust-region and making sure that the policy will not move too far away from the last timestamp. This is achieved through a constrained training using the KL-divergence. Proximal Policy Optimisation (PPO) (Schulman et al., 2017) converts the constrained training of TRPO to a penalty term, which enables much simpler computation and training.

Based on value-based methods and policy gradient, the more advanced "actor-

critic" architecture is proposed (Konda and Tsitsiklis, 2000). Actor-critic methods combine value-based approaches and policy gradient. There are two components in actor-critic architecture. The actor is usually defined as a policy network, which is actually used to generate the action. While the critic is used to evaluate the goodness of the action and is usually trained through value-based methods (e.g., DQN). Successful actor-citric methods include but not limited to Advantage actor-critic (A2C) (Mnih et al., 2016), Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) and Soft Actor-Critic (Haarnoja et al., 2018).

### 2.3.4 On-Policy and Off-Policy Training

The RL algorithms can also be divided into on-policy methods and off-policy methods. We can see from Eq.(2.43) that for policy gradient methods, the expectation is taken over $\tau \sim \pi_\theta$. It indicates that for the training purpose, we need to collect interaction experience (i.e., take actions and observe the reward) according to $\pi_\theta$. This kind of method is referred to as the on-policy method. Obviously, the REINFORCE algorithm is on-policy.

On the other hand, we can see from Eq.(2.38) that for value-based Q-learning, we need to calculate $\max Q(s_{t+1}, a)$. It indicates that the action $a$ is taken according to a fixed greedy policy other than the target policy. This kind of method is referred to as the off-policy method. Q-learning is a typical off-policy algorithm.

Generally speaking, for policy gradient methods, the expectation is calculated through Monte Carlo (MC) methods. It usually needs a large volume of on-policy examples to converge, resulting in large sample complexity.

The critic of actor-critic methods is usually defined with Q-learning which is off-policy and uses a replay buffer to store historical experience. This may help to reduce the sample complexity of actor-critic methods. As a result, plenty of actor-critic methods are claimed to be off-policy such as DDPG and Soft Actor-Critic. However, the actor component of these algorithms is still on-policy and needs to collect on-policy examples for training (Fujimoto et al., 2018). In a nutshell, for actor-critic methods, the critic component can be off-policy but the actor component which is finally used to generate actions is usually on-policy.

For the off-policy Q-learning, although the action is taken according to a fixed greedy policy, we can see from Eq.(2.37) that there is a term of $\sum_{s' \in \mathcal{S}} p^a_{ss'}$.

While from Eq.(2.38), we can see that Q-learning removes this distribution of $p_{ss'}$ and directly sample $s'$ from the replay buffer. This indicates that Q-learning also needs plenty of training examples to achieve a good performance. Besides, Fujimoto et al. (2018) have shown that the state distribution in the replay buffer of Q-learning can not be decorrelated with the target policy. Otherwise, the "extrapolation error" (Fujimoto et al., 2018) may occur.

RL has achieved great success in the field of game control (Mnih et al., 2015). The reason is that the RL agent is actually trained through an "error-and-correction" manner. The game control field can offer an environment to let the agent make "errors". However, in plenty of practical scenarios, making errors in the environment is risky. As a result, perform off-policy training is especially common. But designing efficient and effective off-policy RL algorithms is still an open research question.

### 2.3.5 Reinforcement Learning for Recommendation

Attempts to utilize RL for recommendation have also been made. To address the problem of distribution discrepancy under the off-policy settings, Chen et al. (2019a) proposed to utilize propensity scores to perform the off-policy correction. However, the estimation of propensity scores has high variances and there is a trade-off between bias and variance, which introduces additional training difficulties. Zhao et al. (2018) proposed to utilize negative sampling along with Q-learning. But their method doesn't address the off-policy problem. Model-based RL approaches (Chen et al., 2019b; Shang et al., 2019; Zou et al., 2019) firstly build a model to simulate the environment to avoid any issues with off-policy training. However, these two-stage approaches heavily depend on the accuracy of the simulator. Moreover, recent work has also been done on providing slate-based recommendations (Ie et al., 2019; Gong et al., 2019; Chen et al., 2019a,b) in which actions are considered to be sets (slates) of items to be recommended. This assumption creates an even larger action space as a slate of items is regarded as one single action.

Bandit algorithms that share the same reward schema and long-term expectation with RL have also been investigated for recommendation (Li et al., 2011, 2010). Bandit algorithms assume that taking actions does not affect the state

(Li et al., 2010) while in full RL the assumption is that the state is affected by the actions. Generally speaking, recommendations actually have an effect on user behavior (Rohde et al., 2018) and hence RL is more suitable for modeling the recommendation task. Another related field is imitation learning where the policy is learned from expert demonstrations (Ho and Ermon, 2016; Ho et al., 2016; Torabi et al., 2018). In the recommendation field, the interacted items in the recommendation list can be seen as expert behaviors to perform imitation learning.

## 2.4 Review Findings

Through the literature review, the following findings can be obtained, which motivates the method development of this thesis:

- Deep learning has shown improved performance on the recommendation task. This finding motivates us to continue researching to further develop deep learning-based approaches for recommendation.
- Although some works have been proposed to use deep learning for recommendation, they still suffer from certain limitations. For example, NCF can not tackle side information; NFM models high-order feature interactions in a rather implicit manner. This finding motivates us to develop more effective deep learning-based approaches to learn complex and high-order interaction signals from implicit feedback data.
- Although the attention mechanism has been used in some literature, all of them do not aim to model the multiple item relations in recommendation. In fact, users tend to pay different weights on different item relations and it's a promising direction to utilize attention mechanism under such circumstances for more explainable results.
- Using reinforcement learning is a promising direction to encourage long-term cumulative gains in recommendation. However, how to addressing the specific challenges when using RL for recommendation (e.g., off-policy training) is still an open research problem.

# Part II

# Deep Supervised Learning for Recommendation

In this part, two models based on deep supervised learning are proposed for implicit feedback recommendation. More precisely, in Chapter 3, we proposed Convolutional Factorization Machines (CFM) for feature-based input data. In Chapter 4, Relational Collaborative Filtering (RCF) is proposed for relational input data. Extensive experiments are conducted to evaluate the performance of each model. The results demonstrate that deep supervised learning can not only improve recommendation accuracy but also generate more reasonable recommendation.

# Chapter 3

# Convolutional Factorization Machines

In this chapter, we introduce Convolutional Factorization Machines, which extends the notable Factorization Machines with deep convolutional networks for complex and high-order feature interaction modeling. This chapter is mainly based on my work "CFM: Convolutional Factorization Machines for Context-Aware Recommendation" published in International Joint Conferences on Artificial Intelligence (IJCAI) 2019 with DOI: https://doi.org/10.24963/ijcai.2019/545.

Factorization Machine (FM) is an effective solution for feature-based context-aware recommender systems (CARS) which models second-order feature interactions by the inner product. However, it is insufficient to capture high-order and nonlinear interaction signals for complex real-world implicit feedback data. While several recent efforts have enhanced FM with neural networks (He and Chua, 2017; Guo et al., 2017), they assume the embedding dimensions are independent of each other and model high-order interactions in a rather implicit manner. In this chapter, we propose Convolutional Factorization Machine (CFM) to address the above limitations. Specifically, CFM models second-order interactions with the outer product, resulting in "images" which capture correlations between embedding dimensions. Then all generated "images" are stacked, forming an interaction cube. 3D convolution is applied above it to learn high-order interaction signals in an explicit approach. Besides, we also leverage a self-attention mechanism to perform the pooling of features to reduce time complexity. We conduct extensive experiments on three real-world datasets, demonstrating sig-

nificant improvement of CFM over competing methods for context-aware top-$N$ recommendation from implicit feedback data. The results conform with our thesis statement (1).

## 3.1 Introduction

Generating recommendation from implicit feedback such as click/purchase on products is extremely common in practical applications. When characterizing the implicit user-item interactions, besides the most essential information of user ID and item ID, rich side (context) information is also available. Examples of contexts include but are not limited to user demographics, item attributes, time/location of the current transaction, historical records, and the information of last transactions (Bayer et al., 2017; Wu et al., 2019). To learn from such context-rich data, a typical solution is to first convert it into high-dimensional generic feature vectors (e.g., by using one-hot/multi-hot encoding on univalent/multivalent categorical variables) (Zhou et al., 2018), and then build predictive models on the featured inputs (He and Chua, 2017). Distinct from continuous real-valued features that are naturally found in images and audios, the featured inputs of CARS are mostly categorical, resulting in high-dimensional yet sparse feature vectors. This poses difficulties to build predictive models, such that traditional supervised learning solutions such as SVMs and deep neural networks are sub-optimal and less efficient since they are not tailored for learning from sparse data.

To design effective models for sparse feature inputs, the key ingredient is to account for the interactions among features (Wang et al., 2017). Existing solutions to feature interaction modeling can be categorized into two types:

- Manually constructing cross features: This type of method manually constructs combinatorial features, which explicitly encode feature interactions. Then the cross features are fed into predictive models such as logistic regression (Cheng et al., 2016) and deep neural networks (Wang et al., 2018c). Apparently, the cross feature construction process requires heavy engineering efforts and domain knowledge, making the solution less adaptable to other domains. In addition, another drawback is that it cannot generalize to cold-start feature interactions that are unseen in training data.

- Automatically learning feature interactions: This type of method learns feature interactions and their effects in a unified model. A typical paradigm is to associate each feature with an embedding, expressing the feature interaction as a function over feature embeddings. For example, FM (Rendle, 2010) models the interaction of two features as the inner product of their embeddings, and Deep Crossing (Shan et al., 2016) concatenates feature embeddings and feeds them into a multi-layer perceptron (MLP) to learn high-order interactions. However, these methods implicitly assume embedding dimensions are independent of each other, which goes against the semantics of latent dimensions (Zhang et al., 2014) and limits the model expressiveness. Moreover, MLP upon embedding concatenation captures feature interactions in a rather implicit manner, which has been manifested inefficient to model multiplicative relations (Beutel et al., 2018).

In this chapter, we focus on developing methods for implicit feedback data as sparse feature vectors with the aim of addressing the above-mentioned drawbacks of existing solutions. To reduce engineering efforts in constructing cross features, we explore embedding-based methods and propose Convolutional Factorization Machine (CFM) which automatically learns feature interactions. More precisely, feature embeddings are firstly fed to a self-attention pooling layer, which can dramatically reduce the computational cost and debilitate the influence of noisy features. Then, we model second-order interactions with the outer product, resulting in a list of 2D matrices, which is more effective to capture correlations between embedding dimensions compared with the inner product. After that, we stack the generated matrices and obtain a 3D interactions cube that encodes all second-order interactions. To explicitly learn high-order signals, we propose to employ 3D convolution on the interaction cube, stacking a multi-layer 3D convolution neural network (CNN) above it. As a result, CFM addresses the major limitations of state-of-the-art feature-based methods — independent embedding dimensions and implicit high-order interaction modeling. The main contributions of this chapter are as follows:

- We propose to utilize an interaction cube to represent feature interactions, which encodes both interaction signals and embedding dimension correla-

tions.

- We propose to employ 3D CNN above the interaction cube, which can effectively capture high-order interactions in an explicit way. To the best of our knowledge, this is the first attempt to explore 3D CNN in feature interaction modeling.
- We leverage a self-attention mechanism to perform pooling operations for features, reducing computational complexity.
- We conduct comprehensive experiments on publicly accessible datasets to comparatively evaluate and demonstrate the effectiveness of the proposed method.

## 3.2 The Proposed Method

In this section, we present the detail of the model and the training procedure of the proposed CFM. We also analyze the relationship between CFM and similar research approaches. Before diving into the technical details, we first introduce some basic notations (work only within this chapter).

Throughout this chapter, we use bold uppercase letter (e.g., $\mathbf{M}$) to denote a matrix, bold lowercase letter to denote a vector (e.g., $\mathbf{x}$), and calligraphic uppercase letter to denote a 3D tensor (e.g., $\mathcal{C}$). Scalar is represented by lowercase letters (e.g., $y$). The target of CFM is to generate a ranked item list for given user profiles and context features.

### 3.2.1 The CFM Model

It can been seen from Eq.(2.17) that the original FM only accounts for second-order feature interactions in a linear way by inner product, which fails to learn complex signals. To address this problem, the prediction rule of CFM is formulated as Eq.(3.1):

$$\hat{y}_{CFM}(\mathbf{x}) = w_0 + \sum_{i=1}^{m} w_i x_i + g_\theta(\mathbf{x}), \tag{3.1}$$

where $g_\theta(\mathbf{x})$ denotes the core component to model feature interactions. In the following parts, we will elaborate how to learn $g_\theta(\mathbf{x})$ by outer product and 3D CNN,

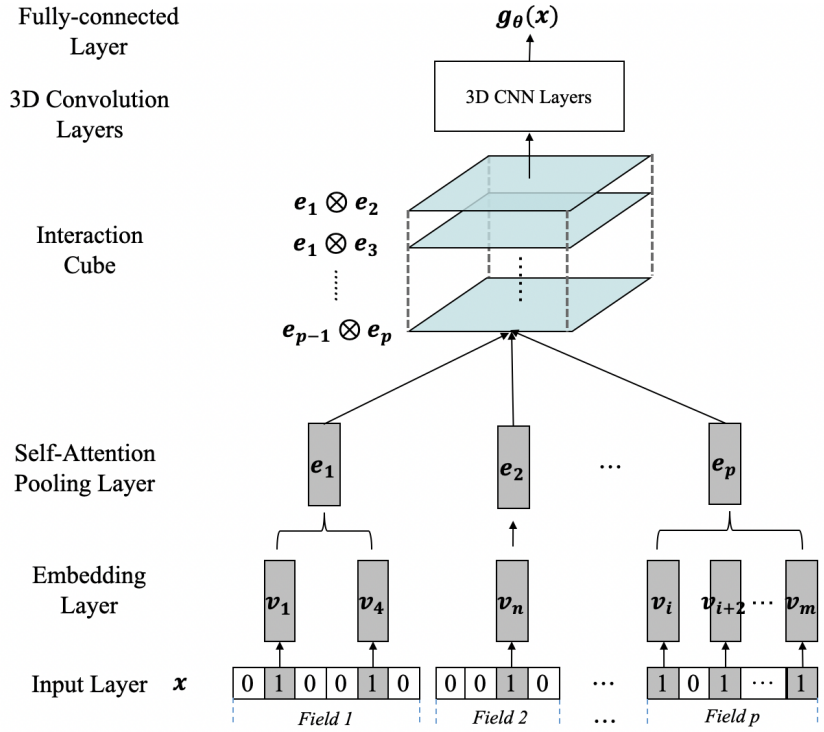



Figure 3.1: CFM model structure. Field 1 and Field p contain multi-hot features. Field 2 is one-hot in which $\mathbf{e}_2$=$\mathbf{v}_n$.

which explicitly captures high-order interaction signals. Figure 3.1 illustrates the overall structure of the proposed CFM model.

#### 3.2.1.1 Input and Embedding Layer

The input layer is fed with a sparse contextual vector $\mathbf{x}$, which may contain both one-hot features (e.g., userID) and multi-hot features (e.g., historical items) to describe a specific user context and item attributes. Then each feature $x_i$ is projected into a $d$-dimensional dense vector representation $\mathbf{v}_i \in \mathbb{R}^d$ by the embedding layer. Due to the sparsity of $\mathbf{x}$, we only need to consider the non-zero features (i.e., $x_i \neq 0$). This can be easily achieved through an embedding table lookup.

#### 3.2.1.2 Self-Attention Pooling Layer

In the real-world scenario, the number of non-zero features in $\mathbf{x}$ may be very large, especially with various multi-hot features (e.g., historical items). The original time complexity to account for all pairwise feature interactions is $O(m^2)$. Then, FM utilizes a reformulation to rewrite Eq.(2.17) and makes the time complexity to be linear with $m$. However, the involved reformulation can only suit the inner product.

Compared with the large number of features, it's obvious that the number of fields is much smaller. As a result, another solution to reduce computational cost is to perform the pooling operation on features and learn a single embedding for each field. Intuitive approaches include max-pooling and average-pooling. However, we argue that this kind of method is sub-optimal due to the lack of a learning process. To capture the intuition that different features have varying importance, we propose to use an attention mechanism to compute the importance of each feature and perform the pooling operation.

Suppose the set of non-zero features in field $j$ is $\mathcal{X}_j$, we parameterize the attention score of feature $x_i \in \mathcal{X}_j$ with a MLP, which is defined as

$$a_i = \mathbf{h}_j^T tanh(\mathbf{W}_j \mathbf{v}_i + \mathbf{b}_j), \tag{3.2}$$

where $\mathbf{W}_j$ and $\mathbf{b}_j$ are corresponding weight matrix and bias vector that project the input embedding into a hidden state, and $\mathbf{h}_j^T$ is the vector which projects the hidden state into the attention score. The size of hidden state is termed as "attention factor". Then, the importance of $x_i$ is calculated by normalizing the attention score through the softmax function:

$$\alpha_i = softmax(a_i) = \frac{exp(a_i)}{\sum_{x_{i'} \in \mathcal{X}_j} exp(a_{i'})}. \tag{3.3}$$

Finally, the after-pooling embedding $\mathbf{e}_j$ for field $j$ is formulated as

$$\mathbf{e}_j = \sum_{x_i \in \mathcal{X}_j} \alpha_i \mathbf{v}_i. \tag{3.4}$$

In fact, the self-attention pooling layer not only reduces the computational cost but also debilitates the influence of noisy features and redundant feature interactions.

#### 3.2.1.3 Interaction Cube

After the pooling layer, we propose to use a 3D interaction cube to represent feature interactions. Specifically, the interaction between $\mathbf{e}_i$ and $\mathbf{e}_j$ is modeled by an outer product operation between them, as shown in Eq.(3.5).

$$\mathbf{M}_{i,j} = \mathbf{e}_i \otimes \mathbf{e}_j = \begin{bmatrix} e_{i1}e_{j1} & e_{i1}e_{j2} & \cdots & e_{i1}e_{jd} \\ e_{i2}e_{j1} & e_{i2}e_{j2} & \cdots & e_{i2}e_{jd} \\ \vdots & \vdots & \vdots & \vdots \\ e_{id}e_{j1} & e_{id}e_{j2} & \cdots & e_{id}e_{jd} \end{bmatrix} \tag{3.5}$$

The above $d \times d$ matrix can be seen as a two-dimensional "image" which contains both interaction signals and embedding dimension correlations. Suppose the contextual vector $\mathbf{x}$ contains $p$ feature fields, the total number of generated "images" is $p(p-1)/2$. All these "images" are stacked to form a 3D tensor $\mathcal{C}$ which is the input of the following 3D CNN.

$$\mathcal{C} = [\mathbf{M}_{1,2}, \mathbf{M}_{1,3}, \cdots, \mathbf{M}_{i,j}, \cdots, \mathbf{M}_{p-1,p}] \tag{3.6}$$

The major advantage of using this cube to represent feature interactions lies in the following points:

- The outer product matrix is more effective to capture dimension correlations compared with conventional inner and element-wise product, which assume that embedding dimensions are independent with each other.
- The 3D structure provides an explicit solution to model high-order interactions, which can be seen as interactions between different "floors" of this cube. For example, in Figure 3.1, the interaction between the first floor ($\mathbf{e}_1 \otimes \mathbf{e}_2$) and the second floor ($\mathbf{e}_1 \otimes \mathbf{e}_3$) can be considered as a third-order interaction among $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$.
- The 3D format also provides a good input for the well-developed 3D CNN. The 3D architecture can benefit the modeling of high-order interactions, which can be regarded as convolutions in the depth direction.

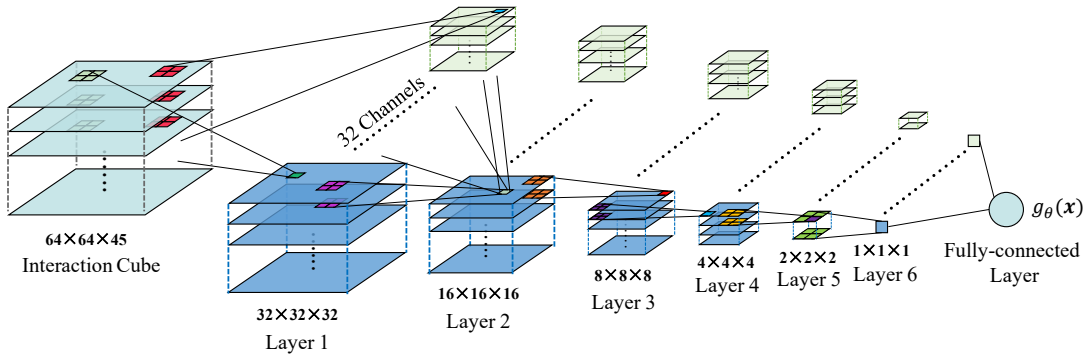



Figure 3.2: The architecture of 3D convolution layers for CFM with the embedding size $d = 64$ and $p = 10$.

#### 3.2.1.4 3D Convolution Layers

In order to tackle the 3D interaction cube and extract signals more effectively, a multi-layer 3D CNN is applied to learn feature interaction patterns[1]. The motivation of using CNN to learn feature interaction signals can be summarized as follows:

- CNN has been widely adopted in the field of computer vision to learn from images. The CNN is trained with a hierarchy fashion (i.e., convolution is performed to aggregate low-level features to high-level representations). In the field of recommendation, high-order feature interaction signals should also be modeled with similar hierarchy approaches. For example, the third-order interaction is actually the interaction between a single feature and another second-order cross feature.
- Utilizing CNN to learn from feature map has been widely adopted in the literature (He et al., 2018a; Yuan et al., 2019; Du et al., 2019; Tang and Wang, 2018; Liu et al., 2015) and has shown promising performance. In that case, CNN is regarded as a general signal extractor which is much easier to train with deep architectures compared with naive MLP (He et al., 2018a).

The convolution process can be abstracted as

$$\mathbf{g} = \text{3DCNN}(\mathcal{C}). \tag{3.7}$$

[1]Although multi-channel 2D CNN can also be used to process the interaction cube, the 3D CNN is a more effective approach. Experimental results are shown in the following part.

Suppose the embedding size $d = 64$ and the number of feature fields $p = 10$, the size of the interaction cube is $64 \times 64 \times 45$. Figure 3.2 illustrates the structure of the stacked 3D CNN with 6 hidden layers, where each hidden layer has 32 channels and convolution operations are performed in all three directions (i.e., width, height, and depth).

In each layer, we first perform 3D convolutions between a 3D kernel and the input cube, after which we add a bias and perform a nonlinear transformation by using ReLU (Hahnloser et al., 2000) as the activation function to obtain a new output cube. It's obvious that the convolution in the depth direction captures high-order feature interactions in a rather explicit manner. According to Figure 3.2, the filter shape in the first layer is [2,2,14] and the stride is [2,2,1], which is corresponding to width, height, and depth. In the subsequent layers, the filter shape and stride are both [2,2,2].

The output of the 3D convolution layers is a vector $\mathbf{g}$. After that, we adopt a fully-connected layer to re-weight each dimension of $\mathbf{g}$ and calculate a real-valued scalar as $g_\theta(\mathbf{x})$:

$$g_\theta(\mathbf{x}) = \mathbf{w}^T \mathbf{g} + b. \tag{3.8}$$

### 3.2.2 Training Detail

The focus of CFM is generating top-$N$ recommendation for implicit feedback data other than rating prediction. Therefore, we optimize the proposed CFM model with the BPR framework (Rendle et al., 2009b):

$$L = \sum - \ln \sigma(\hat{y}_{CFM}(\mathbf{x}^+) - \hat{y}_{CFM}(\mathbf{x}^-)), \tag{3.9}$$

where $\mathbf{x}^-$ is the sampled negative transaction corresponding to the positive one $\mathbf{x}^+$, and $\sigma$ is the sigmoid function. More detailly, we first sample a mini-batch of positive user-item transactions (i.e., $\mathbf{x}^+$), which contain feature vectors to describe specific user contexts and item attributes. Thereafter, for every specific user context, negative items are randomly sampled from a uniform distribution. Then we combine the features of the sampled negative items and the corresponding user context features to form negative user-item transactions (i.e., $\mathbf{x}^-$). Finally, both positive and negative transactions are fed to train the loss function defined in Eq.(3.9).

The embedding layer is pre-trained with FM using BPR loss. To avoid over-

fitting, we involve $L_2$ regularization on the embedding layer, convolution layers, and the fully-connected layer. Besides, before the final fully-connected layer, a drop-out layer is also inserted.

### 3.2.3 Discussion

#### 3.2.3.1 Time Complexity

As described above, the depth of the interaction cube is $p(p-1)/2$. Given the embedding size $d$, the time complexity to perform 3D convolution is $O(p^2d^2)$. Given the situation that $p \ll m$, we can see that the major complexity comes from $d^2$ which is introduced by the convolution. However, this burden can be largely reduced through GPU acceleration.

The other part of complexity comes from feature pooling. Assuming the attention factor is $n$, the time complexity to calculate attention is $O(mnd)$. Therefore, the total time complexity of CFM is $O(p^2d^2 + mnd)$.

#### 3.2.3.2 Relationship with Other Models

In this subsection, we provide an analysis to demonstrate the relationship between the proposed CFM with some other related deep learning-based models.

Neural factorization machine (NFM) (He and Chua, 2017) is also proposed to address the linearity problem of FM. The difference between CFM and NFM mainly lies in the modeling of feature interactions. NFM uses an inner product-based Bi-interaction pooling vector to represent feature interactions. After that, a MLP is utilized to learn signals from the pooling vector. Although the Bi-interaction pooling enables the involvement of MLP, we argue that this pooling operation makes no difference with the original FM. The point is that NFM uses a MLP to learn non-linear combinations of the pooling vector's dimensions, while FM just sums all dimensions of the vector. On the other hand, even if NFM is claimed to support the learning of high-order interactions, we argue that the way NFM used (simple hidden layers) is too implicit to capture the complex signals effectively. However, our CFM uses an outer product-based interaction cube and 3D CNN to model the feature interactions. Firstly, the outer product-based cube contains much more signals than the one-dimensional pooling vector. Secondly, the 3D structure also provides a more explicit manner to model high-order in-

teractions. Besides, 3D CNN is also easier to train compared with the naive MLP which needs more parameters and even suffers from detrimental performance when the network goes deeper (He et al., 2018a). Another similar research is outer product-based neural collaborative filtering (ONCF) (He et al., 2018a), which also utilizes outer product to model user-item interactions. The main difference between ONCF and CFM lies in the following three points:

- ONCF uses outer-product to model interactions between users and items while CFM models the interactions between features. In this view, the relationship between CFM and ONCF is something like the relationship between SVDfeature (Chen et al., 2012) and FM.
- ONCF uses a two-dimensional interaction map while CFM involves a three-dimensional cube when modeling feature interactions. Although ONCF can also integrate contextual information, the user (item) embedding it learns is just a linear sum of the corresponding user (item) feature embeddings. However, there is no concept of user (item) embeddings in CFM but only the concept of feature embeddings. At this point, ONCF can be seen as a special case of CFM in which all user (item) feature embeddings are aggregated linearly to formulate a user (item) embedding.
- ONCF uses conventional 2D CNN while CFM utilizes 3D CNN when extracting the signal of interactions. The latter is more effective to capture high-order interactions, which can be seen as the convolution between different "floors" of the interaction cube.

## 3.3 Experiments

In this section, we conduct experiments to verify the effectiveness of the proposed CFM. We need to observe the model performance, efficiency, and ablation study. As a result, the experiments are conducted based on the following research questions.

**RQ1**: Does the CFM model outperform state-of-the-art methods for feature-based top-$N$ recommendation?

Table 3.1: Datasets statistics for Chapter 3.

| Dataset | #users | #items | #transactions | #fields |
|---|---|---|---|---|
| Frappe | 957 | 4,082 | 96,203 | 10 |
| Last.fm | 1,000 | 20,301 | 214,574 | 4 |
| MovieLens | 6,040 | 3,665 | 939,809 | 4 |

**RQ2**: How do the special designs of CFM (i.e., interaction cube and 3DCNN) affect the model performance?

**RQ3**: What's the effect of the attention-based feature pooling?

## 3.3.1 Experimental Settings

### 3.3.1.1 Data Description

To evaluate the performance of the proposed CFM model, we conduct comprehensive experiments on three real-world implicit feedback datasets: Frappe[1], Last.fm[2] and MovieLens[3]. Table 3.1 summarizes the statistics of these datasets.

Frappe: This dataset is conducted by Baltrunas et al. (2015) to generate right app recommendation for right moments. Frappe contains 96, 203 app usage logs of different user contexts. Each log contains 10 contextual feature fields (i.e., $p = 10$) including user ID, item ID, daytime and some other information.

Last.fm: The Last.fm dataset is for music recommendation. We extract the latest one-day listening history of 1,000 users. The user context is described by user ID and the last music ID that the user has listened to within 90 minutes. The item attributes include music ID and artist ID.

MovieLens: The original MovieLens dataset is designed for explicit rating prediction. Here we binarize it into implicit feedback. The user context is described by user ID and historical items (multi-hot). The item feature is composed of movie ID and movie genres (multi-hot).

[1]http://baltrunas.info/research-menu/frappe
[2]http://www.dtic.upf.edu/ ocelma/MusicRecommendationDataset
[3]https://grouplens.org/datasets/movielens/latest/

#### 3.3.1.2 Evaluation Protocols

We adopt the leave-one-out evaluation to test the performance of models, which has been widely used in literature (He et al., 2017b; Yuan et al., 2016; He et al., 2018c). More specifically, for Last.fm and MovieLens, the latest transaction of each user is held out for testing and the remaining data is treated as the training set. For the Frappe dataset, because there is no timestamp information so we randomly select one transaction for each specific user context as the test example.

The recommendation quality is evaluated by Hit Ratio (HR) and Normalized Discounted Cumulative Gain (NDCG). HR@$N$ is a recall-based metric, measuring whether the test item is in the top-$N$ positions of the recommendation list (1 for yes and 0 otherwise). NDCG@$N$ are weighted versions that assign higher scores to the top-ranked items (Järvelin and Kekäläinen, 2002). These measures are described precisely in section 2.1.3.2. All experiments are run 5 times with different random seeds and the average performance is reported.

#### 3.3.1.3 Baselines

We implemented CFM[1] using TensorFlow. We compare the performance of CFM with the following baselines:

- PopRank: This method returns top-$N$ most popular items. It acts as a basic benchmark.
- FM: The original factorization machine (Rendle, 2010) trained by BPR loss (Rendle et al., 2009b).
- NFM: Neural factorization machine (He and Chua, 2017) is a strong baseline that uses a MLP to learn nonlinear and high-order interaction signals.
- DeepFM: This method (Guo et al., 2017) ensembles the original FM and a MLP to generate recommendation.
- ONCF: This method (He et al., 2018a) is a newly proposed algorithm that improves MF with the outer product.

[1]Codes are available at https://github.com/chenboability/CFM

#### 3.3.1.4 Parameter Settings

We implemented Poprank, FM, and DeepFM by Tensorflow[1]. NFM and ONCF are based on the author's implementation. To fairly compare the performance of models, we train all of them by optimizing the BPR loss with mini-batch Adagrad (Duchi et al., 2011). The learning rate is searched between [0.01,0.02,0.05] for all models. The batch size is set as 256. For all models except PopRank and FM, we pre-train them using the original FM with 500 iterations. The dropout ratio for NFM, DeepFM, ONCF, and CFM is tuned in [0.1,0.2,$\cdots$,0.9]. The embedding size and attention factor are set as 64 and 32, respectively. The output channels of CNN-based models (i.e., ONCF and CFM) are set as 32. Regarding NFM, the number of MLP layers is set as 1 with 64 neurons, which is the recommended setting of their original paper (He and Chua, 2017). For the deep component of DeepFM, we set the MLP according to their original paper (Guo et al., 2017), which has 3 layers and 200 neurons in each layer.

Because the number of feature fields $p$ differs between different datasets, the sizes of interaction cubes are also different, resulting in different kernel sizes and strides of 3D convolution layers[2]. More specifically, we use the structure illustrated in Figure 3.2 on the Frappe dataset. For Last.fm and MovieLens, the filter shape is [2,2,2] and the stride is [2,2,1] for all six layers.

### 3.3.2 Performance Comparison (RQ1)

Table 3.2 shows the top-$N$ recommendation performance on all three datasets. It's obvious that CFM achieves the best performance on all datasets regarding both HR and NDCG, with a significant difference. This observation provides strong support for the thesis statement (1). We argue that this significant improvement lies in the following two points:

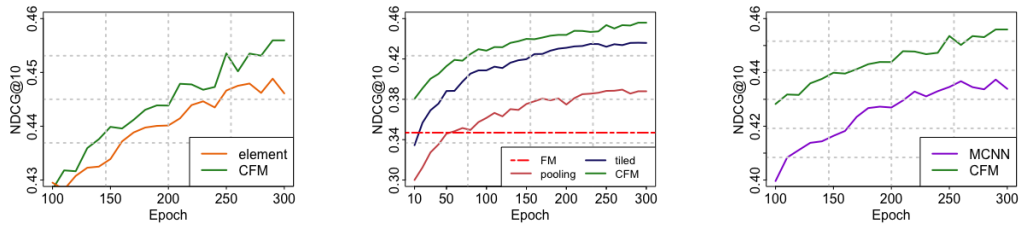
- The outer product-based interaction cube is a fairly good approach to represent feature interactions. This can be seen from the comparison between CFM and NFM, which uses inner product-based pooling vectors to present feature interactions.

[1] https://www.tensorflow.org/

[2] Another approach is to use padding so that the settings of CNN layers can be fixed.

Table 3.2: Comparison between different models of Chapter 3 when generating top-$N$ recommendation. $N \in \{5, 10, 20\}$. Boldface denotes the highest score. $*$ denotes the statistical significance for $p < 0.05$ compared with the best baseline.

| Top-5 | Frappe | | Last.fm | | Movielens | |
|---|---|---|---|---|---|---|
| | HR@5 | NDCG@5 | HR@5 | NDCG@5 | HR@5 | NDCG@5 |
| PopRank | 0.2539 | 0.1595 | 0.0013 | 0.0007 | 0.0121 | 0.0071 |
| FM | 0.4204 | 0.3054 | 0.1658 | 0.1142 | 0.0512 | 0.0295 |
| DeepFM | 0.4632 | 0.3308 | 0.1773 | 0.1204 | 0.0563 | 0.0355 |
| NFM | 0.4798 | 0.3469 | 0.1827 | 0.1235 | 0.0634 | 0.0374 |
| ONCF | 0.5359 | 0.3940 | 0.2183 | 0.1493 | 0.0579 | 0.0343 |
| CFM$^*$ | **0.5462** | **0.4153** | **0.2375** | **0.1573** | **0.0697** | **0.0426** |
| Top-10 | Frappe | | Last.fm | | Movielens | |
| | HR@10 | NDCG@10 | HR@10 | NDCG@10 | HR@10 | NDCG@10 |
| PopRank | 0.3493 | 0.1898 | 0.0023 | 0.0011 | 0.0235 | 0.0107 |
| FM | 0.5486 | 0.3469 | 0.2382 | 0.1374 | 0.0998 | 0.0452 |
| DeepFM | 0.6035 | 0.3765 | 0.2612 | 0.1473 | 0.1170 | 0.0526 |
| NFM | 0.6197 | 0.3924 | 0.2676 | 0.1488 | 0.1192 | 0.0553 |
| ONCF | 0.6531 | 0.4320 | 0.3208 | 0.1823 | 0.1110 | 0.0514 |
| CFM$^*$ | **0.6720** | **0.4560** | **0.3538** | **0.1948** | **0.1323** | **0.0627** |
| Top-20 | Frappe | | Last.fm | | Movielens | |
| | HR@20 | NDCG@20 | HR@20 | NDCG@20 | HR@20 | NDCG@20 |
| PopRank | 0.4136 | 0.2060 | 0.0032 | 0.0013 | 0.0429 | 0.0156 |
| FM | 0.6590 | 0.3750 | 0.3537 | 0.1665 | 0.1762 | 0.0644 |
| DeepFM | 0.7322 | 0.4092 | 0.3799 | 0.1772 | 0.2033 | 0.0723 |
| NFM | 0.7382 | 0.4225 | 0.3783 | 0.1765 | 0.2029 | 0.0748 |
| ONCF | 0.7691 | 0.4614 | 0.4611 | 0.2176 | 0.2002 | 0.0738 |
| CFM$^*$ | **0.7774** | **0.4859** | **0.4841** | **0.2277** | **0.2248** | **0.0858** |




(a) Effect of outer product (b) 3D architecture study (c) Effect of 3D CNN

Figure 3.3: Study of the interaction cube and 3D CNN of CFM.

- The involved 3D CNN is more effective to extract signals compared with 2D CNN (ONCF) and MLP (NFM, DeepFM), especially for high-order interactions.

Among the baselines, we can see that ONCF achieves the best performance while the improvement of NFM and DeepFM over original FM is relatively marginal. The reason is that ONCF uses the outer product to model user-item interactions while NFM and DeepFM are still inner product-based, which cannot capture the correlations between embedding dimensions. Besides, ONCF utilizes 2D CNN to extract signals, which is much more effective compared with the MLP of NFM and DeepFM.

### 3.3.3 Model Investigation (RQ2)

#### 3.3.3.1 Study of the interaction cube.

In this part, we conduct experiments to further demonstrate the effectiveness of special designs of the introduced interaction cube, including the outer product and the stacked 3D architecture.

- Outer product: To show the effectiveness of the outer product, we replace it with an alternative solution — element-wise product (element). Each pair-wise feature interaction is represented by an element-wise product between feature embeddings, resulting in a vector. All these vectors are aggregated as a matrix and we use a six-layers 2D CNN to learn signals from it. Figure

3.3(a) shows the results on Frappe dataset. It's obvious that CFM overperforms the element-wise product-based method when both of them use CNN to extract signals. This is because the element-wise product assumes that embedding dimensions are independent of each other while the outer product eliminates this assumption and captures the dimension correlations.

- 3D architecture: To show the effect of the 3D architecture, we convert the interaction cube to a 2D feature map through two different operations. The first is to tile each feature interaction image to form a bigger feature map (tiled)[1]. The second is to use a max-pooling operation on the depth direction of the interaction cube (pooling). Then we use a six-layers 2D CNN to learn signals from the map. Figure 3.3(b) shows the results on Frappe. We can see that CFM achieves better performance than the tiled method. We argue that the reason is the stacked 3D architecture provides a rather explicit manner to model high-order interactions. However, the tiled feature map can only support implicit high-order interaction modeling. Besides, we can also see that the max pooling operation will downgrade the performance because it only considers the most important second-order interactions. However, the performance is still better than the original FM, which demonstrates the effectiveness of the outer product and the strong learning capability of CNN.

#### 3.3.3.2 Study of 3D CNN

To tackle the 3D interaction cube, another solution is to use multi-channel 2D CNN (MCNN). Here, we also conduct experiments to make a comparison between them. Figure 3.3(c) illustrates the results on the Frappe dataset. Results on other datasets show the same trend. We can see that compared with MCNN, our CFM achieves better performance. The reason is that the convolution in the depth direction of 3DCNN explicitly models high-order feature interactions while MCNN cannot achieve this in such an explicit manner.

[1]For Frappe, the size of the feature map is $320 \times 576 = 64 \times 64 \times 45$.

Table 3.3: Effect of self-attention on CFM. Max and Mean denote replacing the self-attention with max-pooling and mean-pooling, respectively.

| Model | HR@10 | HR@20 | NDCG@10 | NDCG@20 |
|---|---|---|---|---|
| Max | 0.1257 | 0.2142 | 0.0591 | 0.0813 |
| Mean | 0.1291 | 0.2212 | 0.0603 | 0.0833 |
| CFM | 0.1323 | 0.2248 | 0.0627 | 0.0858 |

Table 3.4: Effect of feature pooling on CFM. CFM-wfp denotes the CFM model without feature pooling. Time denotes the running time for one single iteration.

| Indicator | FM | CFM | CFM-wfp |
|---|---|---|---|
| Time | 0.53m | 4.63m | 50.52m |
| HR@10 | 0.0998 | 0.1323 | 0.1297 |
| NGCG@10 | 0.0452 | 0.0627 | 0.0605 |

### 3.3.4 Study of Feature Pooling (RQ3)

The proposed CFM leverages a self-attention mechanism to perform feature pooling. To demonstrate the effectiveness of the involved attention mechanism, we replace it with max-pooling and mean-pooling. Table 3.3 shows the results on MovieLens[1]. We can see that mean-pooling achieves better performance than max-pooling because max-pooling only considers the most important feature and lots of information is discarded. However, the attention-based CFM achieves the best performance. In fact, the attention-based feature pooling automatically assigns different importance to different features. It can not only retain the rich feature information but also debilitate the noise influence.

Table 3.4 shows the comparison between CFM with and without feature pooling on MovieLens dataset. We can see that the attention-based feature pooling can dramatically reduce the training time without affecting the model performance. It can even result in better recommendation quality.

[1]The other two datasets only contain one-hot features and we don't use feature pooling on them.

## 3.4 Chapter Summary

In this chapter, we propose a deep supervised recommendation model CFM for implicit feedback data, which seamlessly combines automatic feature interaction modeling of FM and the strong learning capability of CNN. The key design of CFM is to use an outer product-based interaction cube to represent feature interactions and then utilize deep CNN to extract signals from it. As a result, correlations among embedding dimensions can be effectively captured and higher-order interaction signals can also be learned in a rather explicit approach. Besides, we also utilize a self-attention mechanism to perform feature pooling and reduce computational cost. Extensive experiments on three datasets demonstrate that CFM has superior performance compared with state-of-the-art models when generating top-$N$ recommendation from implicit feedback data. This observation provides support for the thesis statement (1). Generally speaking, all the features can be divided into the user-related field, item-related field, and the (other) context field. If we perform feature pooling among the three fields, the depth of the interaction cube would be three, corresponding to user-item interactions, user-context interactions, and item-context interactions, respectively. It shows a similar property with the three RGB colors of images. As a result, we think CFM will provide some hints to further utilize the advances in the computer vision field for recommendation.

# Chapter 4

# Relational Collaborative Filtering

In Chapter 3, we developed CFM for feature-based implicit feedback data. CFM has shown improved performance compared with the state-of-the-art baselines. However, the CFM model can be seen as a black box, which has limitations to generate explainable recommendation. In this chapter, we introduce Relational Collaborative filtering, which targets utilizing multiple item relations to generate more reasonable recommendation results. This chapter is mainly based on my previous work "Relational Collaborative Filtering: Modeling Multiple Item Relations for Recommendation" published in International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR) 2019 with DOI: https://doi.org/10.1145/3331184.3331188.

Item-based collaborative filtering (ICF) methods leverage only the relation of collaborative similarity — i.e., the item similarity evidenced by user interactions. Nevertheless, there exist multiple relations between items in real-world scenarios, e.g., two movies share the same director, two products complement each other, etc. Distinct from the collaborative similarity that implies co-interact patterns from the user's perspective, these relations reveal fine-grained knowledge on items from different perspectives of meta-data, functionality, etc. However, how to incorporate multiple item relations is less explored in recommendation research.

In this chapter, we propose Relational Collaborative Filtering (RCF) to exploit multiple item relations for implicit feedback recommendation. We find that both the relation type (e.g., shared director) and the relation value (e.g., Steven Spielberg) are crucial in inferring user preference. To this end, we develop a two-level hierarchical attention mechanism to model user preference — the first-

level attention discriminates which types of relations are more important, and the second-level attention considers the specific relation values to estimate the contribution of a historical item. To make the item embeddings be reflective of the relational structure between items, we further formulate a task to preserve the item relations and jointly train it with user preference modeling.

Empirical results on two real datasets demonstrate the strong performance of RCF. Furthermore, we also conduct qualitative analyses to show the benefits of explanations brought by RCF's modeling of multiple item relations. The results provide strong support for the thesis statement (2).

## 4.1 Introduction

Item-based collaborative filtering (ICF) is one of the most successful recommendation methods owing to its interpretability and effectiveness (Kabbur et al., 2013; He et al., 2018c), being highly preferred in industrial applications (Smith and Linden, 2017; Covington et al., 2016; Eksombatchai et al., 2018). The key assumption of ICF is that a user shall prefer the items that are similar to her historically interacted items (Linden et al., 2003; Xue et al., 2018; Wang et al., 2019a). The similarity is typically judged from user interactions — how likely two items are co-interacted by users in the past.

Despite prevalence and effectiveness, we argue that existing ICF methods are insufficient since they only consider the collaborative similarity relation, which is macro-level, coarse-grained, and lacks concrete semantics. In real-world applications, there typically exist multiple relations between items that have concrete semantics, and they are particularly helpful to understand user behaviors. For example, in the movie domain, some movies may share the same director, actors, or other attributes; in E-commerce, some products may have the same functionality, similar image, etc. These relations reflect the similarity of items from different perspectives, and more importantly, they could affect the decisions of different users differently. For example, after two users ($u1$ and $u2$) watch the same movie "E.T. the Extra-Terrestrial", $u1$ likes the director and chooses "Schindler's List" to watch next, while $u2$ likes the fiction theme and watches "The Avenger" in the next. Without explicitly modeling such micro-level and fine-grained relations be-

tween items, it is conceptually difficult to reveal the true reasons behind a user's decision, not to mention to recommend desired items with persuasive explanations like *"The Avenger" is recommended to you because it is a fiction movie like "E.T. the Extra-Terrestrial" you watched before* for the user $u2$.

In this chapter, we develop Relational Collaborative Filtering (RCF) to integrate multiple item relations for implicit feedback recommendation. To retain the fine-grained semantics of relations and facilitate the reasoning on user preference, we represent a relation as a concept with a two-level hierarchy:

1. Relation type, which can be shared director or genre in the above movie example, or functionality or visual similarity for E-commerce products. It describes how items are related to each other in an abstract way. The collaborative similarity is also a relation type from the macro view of user behaviors.

2. Relation value, which gives details on the shared relation of two items. For example, the value of relation shared director for "E.T. the Extra-Terrestrial" and "Schindler's List" is Steven Spielberg, and the values for relation shared genre include fiction, action, romantic, etc. The relation values provide important clues for scrutinizing a user's preference since a user could weigh different values of a relation type differently when making decisions.

Figure 4.1 gives an illustrative example on the item relations. Note that multiple relations may exist between two items; for example, badminton birdies balls and badminton rackets have two relations of complementary functionality and shared category. Moreover, a relation value may occur in multiple relations of different types; for example, a director can also be the leading actor of other movies, thus it is likely that two types of relations have the same value which refers to the same stuff. When designing a method to handle multiple item relations, these factors should be taken into account, making the problem more complicated than the standard ICF.

To integrate such relational data into ICF, we devise a two-level neural attention mechanism (Bahdanau et al., 2014) to model the historically interacted items. Specifically, to predict a user's preference on a target item, the first-level

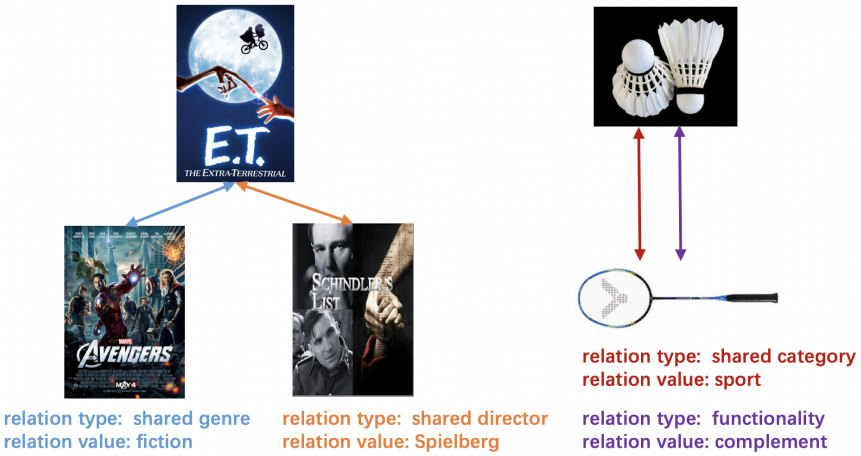



Figure 4.1: An example of multiple item relations. Each relation is described with a two-level hierarchy of type and value. Multiple relations may exist between two items and the same value may occur in relations of different types.

attention examines the types of relations that connect the interacted items with the target item and discriminates which types affect more on the user. The second-level attention is operated on the interacted items under each relation type, so as to estimate the contribution of an interacted item in recommending the target item. The two-level attention outputs a weight for each interacted item, which is used to aggregate the embeddings of all interacted items to obtain the user's representation. Furthermore, to enhance the item embeddings with the multi-relational data, we formulate another learning task that preserves the item relations with embedding operations. Finally, we jointly optimize the two tasks to make maximum usage of multiple relations between items.

To summarize, this chapter makes the key contributions as follows:

- We propose a new and general item-based implicit feedback recommendation task, that is incorporating the multiple relations between items to better predict user preference.
- We devise a new method RCF, which leverages the relations in two ways: constructing user embeddings by improved modeling of historically interacted items and enhancing item embeddings by preserving the relational structure.

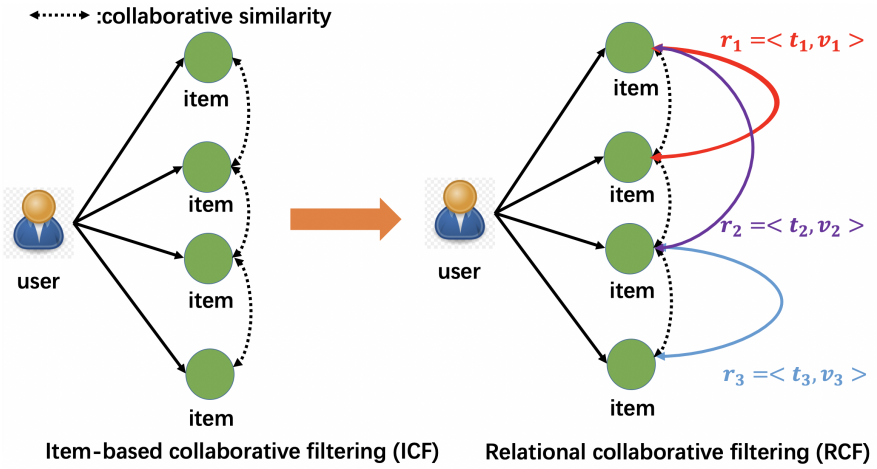



Figure 4.2: Comparison between ICF and RCF. The links between items of ICF are implicit and single, which denote the collaborative similarity. However, the links between items of RCF are explicit and multiple.

- We conduct experiments on two datasets to validate our proposal. Quantitative results show RCF outperforms several recently proposed methods, and qualitative analyses demonstrate the recommendation explanations of RCF with multiple item relations.

## 4.2 Methodology

We first introduce the problem of using multiple item relations for implicit feedback recommendation and then elaborate on our proposed RCF method.

### 4.2.1 Problem Formulation

Given a user and his interaction history, conventional ICF methods aim at generating recommendations based on the collaborative similarity which encode the co-interact patterns of items. Its interaction graph can be shown as the left part of Figure 4.2, where the links between items are just the implicit collaborative similarity. However, there are multiple item relations in the real world that have meaningful semantics. In this chapter, we define the item relations as:

**Definition 1.** *Given an item pair $(i, j)$, the relations between them are defined as a set of $r =< t, v >$ where $t$ denotes the relation type and $v$ is the relation value.*

Table 4.1: Notations for Chapter 4

| Notation | Description |
|---|---|
| $\mathcal{U}, \mathcal{I}$ | the set of users and items |
| $\mathcal{T}$ | the set of relation types |
| $\mathcal{V}$ | the set of relation values |
| $\mathcal{I}_u^+$ | the item set which user $u$ has interacted with |
| $\mathcal{I}_{u,i}^t$ | the items in $\mathcal{I}_u^+$ that have the relation of type $t$ with the target item $i$ |
| $I_r(i,j)$ | an indicator function where $I_r(i,j) = 1$ if relation $r$ holds for item $i$ and $j$, otherwise 0 |
| $\mathbf{p}_u \in \mathbb{R}^d$ | the ID embedding for user $u \in \mathcal{U}$, which represents the user's inherent interests |
| $\mathbf{q}_i \in \mathbb{R}^d$ | the embedding for item $i \in \mathcal{I}$ |
| $\mathbf{x}_t \in \mathbb{R}^d$ | the embedding for relation type $t \in \mathcal{T}$ |
| $\mathbf{z}_v \in \mathbb{R}^d$ | the embedding for relation value $v \in \mathcal{V}$ |

The target of RCF is to generate recommendations based on both the implicit user-item interaction history and item relational data. Generally speaking, the links between items in the interaction graph of RCF contain not only the implicit collaborative similarity but also the explicit multiple item relations, which are represented by the heterogeneous edges in the right part of Figure 4.2. The notations for this chapter are summarized in Table 4.1.

In the remainder of this section, we first present the attention-based model to infer user-item preference. We then illustrate how to model the item relational data to introduce the relational structure between item embeddings. Based on that, we propose to integrate the two parts in an end-to-end fashion through a multi-task learning framework. Finally, we provide a discussion on the relationship between RCF and some other models.

### 4.2.2 User-Item Preference Modeling

An intuitive motivation when modeling user preference is that users tend to pay different weights to relations of different types (e.g., some users may prefer movies that share the same actors, some users may prefer movies that fall into the same genres). Given multiple item relations which consist of relation types and relation

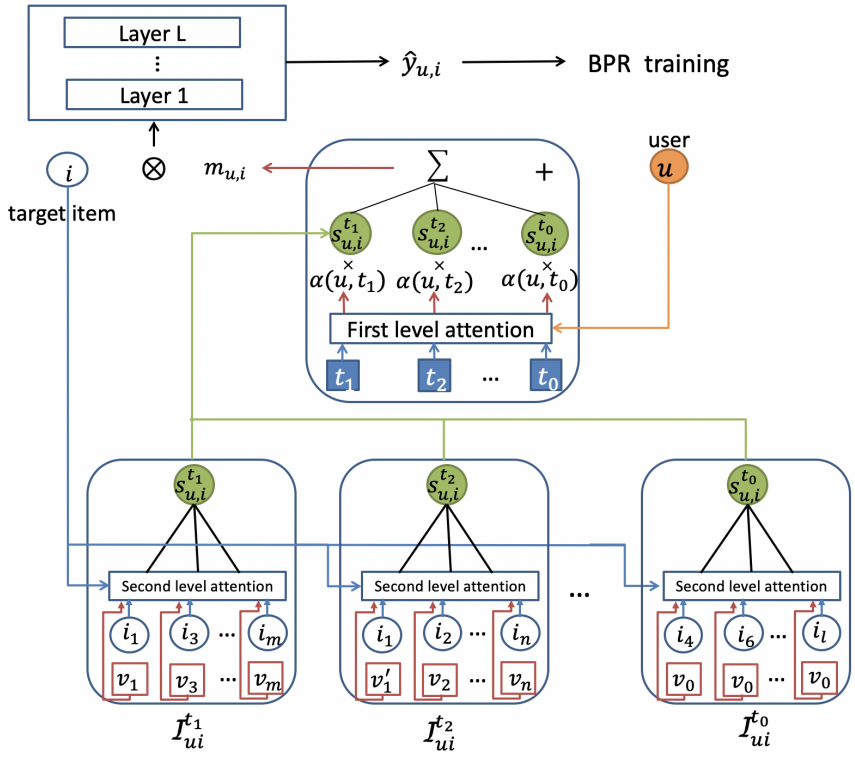



Figure 4.3: Illustration of the proposed RCF model. The target-aware user embedding ($\mathbf{m}_{u,i}$) is modeled with a two-level hierarchy attention mechanism. The input of the first level attention contains the user ID embedding and relation types. The second level attention is used to calculate the weights of specific historical items. There are three inputs during this state, including the target item, the historical item, and the relation value. Note that one historical item (e.g., $i_1$) can occur in different $\mathcal{I}_{u,i}^{t}$ when there are multiple relations between it and the target item.

values, we propose to use a hierarchy attention mechanism to model the user preference. Figure 4.3 demonstrates the overall structure of our model.

Given the item relational data, we first divide the interacted items of user $u$ (i.e., $\mathcal{I}_u^+$) into different sets (i.e., $\mathcal{I}_{u,i}^t$) according to the relation types between these items and the target item. Note that a single item may occur in different $\mathcal{I}_{u,i}^t$ when there are multiple relations between this item and $i$. Besides, there may be some items which have no explicit relation with the target item. To tackle with these items, we introduce a latent relation $r_0 =< t_0, v_0 >$ and put these items into $\mathcal{I}_{u,i}^{t_0}$, as shown in Figure 4.3. Here $r_0$ can be regarded as the

collaborative similarity which just indicates the item co-interact patterns. Then the target-aware user embedding can be formulated as

$$\mathbf{m}_{u,i} = \mathbf{p}_u + \sum_{t \in \mathcal{T}} \alpha(u,t) \cdot \mathbf{s}_{u,i}^t \,, \tag{4.1}$$

where $\alpha(u,t)$ is the first-level attention which aims to calculate the importance of different relation types for this user and $\mathbf{s}_{u,i}^t$ describes the user's profile based on the items in $\mathcal{I}_{u,i}^t$. More precisely, we define $\alpha(u,t)$ with the standard softmax function:

$$\alpha(u,t) = \frac{exp(a(\mathbf{p}_u, \mathbf{x}_t))}{\sum_{t' \in \mathcal{T}} exp(a(\mathbf{p}_u, \mathbf{x}_{t'}))} \,, \tag{4.2}$$

where $a(\mathbf{p}_u, \mathbf{x}_t)$ is the attention score between user $u$ and relation type $t$. We define it with a feedforward neural network, as shown in Eq.(4.3)

$$a(\mathbf{p}_u, \mathbf{x}_t) = \mathbf{h}_1^T(ReLU(\mathbf{W}_1(\mathbf{p}_u \otimes \mathbf{x}_t) + \mathbf{b}_1)). \tag{4.3}$$

$\mathbf{W}_1$ and $\mathbf{b_1}$ are corresponding weight matrix and bias vector that project the input into a hidden state, and $\mathbf{h}_1^T$ is the vector which projects the hidden state into the attention score. We term the size of hidden state as "attention factor", for which a larger value brings a stronger representation power for the attention network. $\otimes$ denotes the element-wise product.

The next step is to model $\mathbf{s}_{u,i}^t$. It's obvious that the relation value accounts for an important part of this process. For example, a user may pay attention to genres when watching a movie. However, among all the genres, he is most interested in fiction other than romantic. As a result, we should consider both the items and the corresponding relation values when modeling the second-level attentions. From that view, we define $\mathbf{s}_{u,i}^t$ as

$$\mathbf{s}_{u,i}^t = \sum_{j \in \mathcal{I}_{u,i}^t} \beta_t(i,j,v) \cdot \mathbf{q}_j \,, \tag{4.4}$$

where $\beta_t(i,j,v)$ represents the specific weight of item $j$. Similar to Eq.(4.2), a straight-forward solution to calculate $\beta_t(i,j,v)$ is to use the softmax function. However we found that such a simple solution would lead to bad performance. The reason is that the number of items between different $\mathcal{I}_{u,i}^t$ vary greatly. For those items in large $\mathcal{I}_{u,i}^t$, the standard softmax function will have very big denominator, causing the gradient vanishing problem of corresponding $\mathbf{q}_j$. Same observations can also be found in He et al. (2018c) under similar circumstances.

To tackle with this problem, we utilize a *smoothed* softmax function to replace

the standard solution. As a result, the weight $\beta_t(i,j,v)$ is formulated as

$$\beta_t(i,j,v) = \frac{exp(b_t(\mathbf{q}_i, \mathbf{q}_j, \mathbf{z}_v))}{[\sum_{j' \in \mathcal{I}^t_{u,i}} exp(b_t(\mathbf{q}_i, \mathbf{q}_{j'}, \mathbf{z}_{v'}]^\rho}, \tag{4.5}$$

where $\rho$ is a smoothing factor between (0,1] and is commonly set as 0.5 (He et al., 2018c). $b_t(\mathbf{q}_i, \mathbf{q}_j, \mathbf{z}_v)$ is the second-level attention score which is defined as

$$b_t(\mathbf{q}_i, \mathbf{q}_j, \mathbf{z}_v) = \mathbf{h}^T_{2,t}(ReLU(\mathbf{W}_{2,t}[\mathbf{q}_i, \mathbf{q}_j, \mathbf{z}_v] + \mathbf{b}_{2,t})), \tag{4.6}$$

where $[\cdot]$ denotes the vector concatenation. $\mathbf{W}_{2,t}$, $\mathbf{b}_{2,t}$ and $\mathbf{h}_{2,t}$ are corresponding attention parameters. Different from Eq.(4.3) which utilizes element-wise product to learn signals from inputs, here we concatenate the input embeddings and send them to a feedforward neural network. The reason is that there are three inputs when modeling the second-level attention. Utilizing the element-wise product under such a situation would have a high risk of suffering from vanishing or exploding gradients.

Now we have completed the modeling of the target-aware user embedding $\mathbf{m}_{u,i}$. Based on that, we utilize a multilayer perceptron (MLP) to calculate the final predicted score of user $u$ on item $i$, which is shown as:[1]

$$\hat{y}_{ui} = MLP(\mathbf{m}_{u,i} \otimes \mathbf{q}_i), \tag{4.7}$$

Given the final predicted score $\hat{y}_{ui}$, we want the positive items to have a higher rank than negative ones. We utilize the BPR pairwise learning framework (Rendle et al., 2009b) to define the objective function, which is shown as

$$L_{rec} = - \sum_{(u,i,k) \in \mathcal{D}_I} \ln \sigma(\hat{y}_{ui} - \hat{y}_{uk}), \tag{4.8}$$

where $\sigma$ denotes the sigmoid function and $\mathcal{D}_I$ is the set of training triplets:

$$\mathcal{D}_I = \left\{(u,i,k) | u \in \mathcal{U} \wedge i \in \mathcal{I}^+_u \wedge k \in \mathcal{I} \backslash \mathcal{I}^+_u \right\}. \tag{4.9}$$

### 4.2.3 Item-Item Relational Data Modeling

The second task of RCF is to model the item relational data. Typically, the relational data is organized as knowledge graphs (KG). A knowledge graph is a directed heterogeneous graph in which nodes correspond to entities and edges correspond to relations. It can be represented by a set of triplets $(e_1, r, e_2)$ where $e_1$ denotes the head entity, $r$ is the relation and $e_2$ represents the tail entity.

[1] We introduce a dropout layer (Srivastava et al., 2014) before each layer of the MLP to prevent overfitting.

Knowledge graph embedding (KGE) is a popular approach to learn signals from relational data which aims at embedding a knowledge graph into a continuous vector space.

However, directly using techniques from KGE (Bordes et al., 2013; Lin et al., 2015; Yang et al., 2014) to model the item relations of RCF is infeasible due to the following challenges in our specific domain:

1. The item relation is defined with a two-level hierarchy: relation type and relation value. As shown in Figure 4.1, the relation between "E.T. the Extra-Terrestrial" and "The Avenger" is described as <shared genre, fiction>. To represent this relation properly, we must consider both the first-level (i.e., shared genre) for type constraints and the second-level (i.e., fiction) for model fidelity. As a result, we can not assign a single embedding for an item relation $r =< t, v >$, which is a common case in the field of KGE (Bordes et al., 2013; Lin et al., 2015; Yang et al., 2014).

2. Different from the conventional KG which is represented as a directed graph, the item relations are reversible (i.e., the relation $r$ holds for both $(e_1, e_2)$ and $(e_2, e_1)$), resulting in an undirected graph structure. Traditional KGE methods (Bordes et al., 2013; Lin et al., 2015) may encounter difficulties under such situations. For example, the most popular TransE (Bordes et al., 2013) models the relation between two entities as a translation operation between their embeddings, that is, $\mathbf{e}_1 + \mathbf{r} \approx \mathbf{e}_2$ when $(e_1, r, e_2)$ holds, where $\mathbf{e}_1, \mathbf{r}, \mathbf{e_2}$ are corresponding embeddings for head entity, relation and tail entity. Based on that, TransE defines the scoring function for this triplet as $f(e_1, r, e_2) = \|\mathbf{e}_1 + \mathbf{r} - \mathbf{e}_2\|_2$ where $\|\cdot\|_2$ denotes the $L_2$ norm of a vector. However, because of the undirected structure, we will get both $\mathbf{e}_1 + \mathbf{r} \approx \mathbf{e}_2$ and $\mathbf{e}_2 + \mathbf{r} \approx \mathbf{e}_1$ on our item relational data. Optimizing objective functions based on such equations may lead to a trivial solution that $\mathbf{r} \approx \mathbf{0}$ and $\mathbf{e}_1 \approx \mathbf{e}_2$.

To tackle the first challenge, we use the summation of the two-level hierarchy components as relation embeddings. More precisely, the representation of a specific relation $r =< t, v >$ is formulated as the following equation:

$$\mathbf{r} = \mathbf{x}_t + \mathbf{z}_v. \tag{4.10}$$

By doing so, we can make sure that relations with the same type keep similar with each other to some degree. Meanwhile, the model fidelity is also guaranteed because of the value embedding. It also empowers the model with the ability to tackle the situation that the same values occur in relations of different types.

To address the second challenge, we find that the source of the trivial solution is the minus operation in TransE, which only suits for directed structures. To model undirected graphs, we need the model which satisfies the commutative law (i.e., $f(e_1, r, e_2) = f(e_2, r, e_1)$). Another state-of-the-art methods of KGE is DistMult (Yang et al., 2014). It defines the scoring function as $f(e_1, r, e_2) = \mathbf{e}_1^T \mathbf{M}_r \mathbf{e}_2$, where $\mathbf{M}_r$ is a matrix representation of $r$. It's obvious that DistMult is based on the multiply operation and satisfies the desired commutative property. Based on that, given a triplet $(i, r, j)$ which means item $i$ and $j$ has relation $r$, we define the scoring function for this triplet as

$$f(i, r, j) = \mathbf{q}_i^T \cdot diag(\mathbf{r}) \cdot \mathbf{q}_j. \tag{4.11}$$

Here $diag(\mathbf{r})$ denotes a diagonal matrix whose diagonal elements equal to $\mathbf{r}$ correspondingly.

Similar to the BPR loss used in the recommendation part, we want to maximize $f(i, r, j)$ for positive examples and minimize it for negative ones. Based on that, the objective function is defined by contrasting the scores of observed triplets $(i, r, j)$ versus unobserved ones $(i, r, j^-)$:

$$L_{rel} = - \sum_{(i,r,j,j^-) \in \mathcal{D}_R} \ln \sigma(f(i, r, j) - f(i, r, j^-)), \tag{4.12}$$

where $\mathcal{D}_R$ is defined as

$$\mathcal{D}_R = \left\{ (i, r, j, j^-) | i, j, j^- \in \mathcal{I} \wedge I_r(i, j) = 1 \wedge I_r(i, j^-) = 0 \wedge r \neq r_0 \right\}. \tag{4.13}$$

The above objective function encourages the positive item $j$ to be ranked higher than negative items $j^-$ given the context of the head item $i$ and relation $r$. Because $r_0$ is defined as a latent relation so we don't include it during this process.

### 4.2.4 Multi-Task Learning

To effectively learn parameters for recommendation, as well as preserve the relational structure between item embeddings, we integrate the recommendation part (i.e., $L_{rec}$) and the relation modeling part (i.e., $L_{rel}$) in an end-to-end fash-

**Algorithm 1** Learning algorithm for RCF

**Input:** user-item interaction data $\mathcal{D}_I$, item relationl data $\mathcal{D}_R$, learning rate $\eta$, smoothing factor $\rho$, $\gamma$

**Output:** all parameters in the learning space $\Theta$

Initialize all parameters in $\Theta$
**repeat**
  Draw a mini-batch of $(u, i, k)$ from $\mathcal{D}_I$
  Draw a mini-batch of $(i, r, j, j^-)$ from $\mathcal{D}_R$
  Compute $L_{rec}$ according to Eq.(4.1)-(4.9)
  Compute $L_{rel}$ according to Eq.(4.10)-(4.13)
  $L \leftarrow L_{rec} + \gamma L_{rel}$
  **for** each parameter $\vartheta \in \Theta$ **do**
    Compute $\partial L / \partial \vartheta$ on the mini-batch by back-propagation
    Update $\vartheta \leftarrow \vartheta - \eta \cdot \partial L / \partial \vartheta$
  **end for**
  **for** $\boldsymbol{\theta} \in \{\mathbf{p}_u, \mathbf{q}_i, \mathbf{x}_t, \mathbf{z}_v\}$ **do**
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} / \max(1, \|\boldsymbol{\theta}\|_2)$
  **end for**
**until** converge
return all parameters in $\Theta$

ion through a multi-task learning framework. The total objective function of RCF is defined as

$$\begin{aligned} \min_{\Theta} \quad & L = L_{rec} + \gamma L_{rel} \ , \\ s.t. \quad & \|\mathbf{p}_u\|_2 \leq 1, \|\mathbf{q}_i\|_2 \leq 1, \|\mathbf{x}_t\|_2 \leq 1, \|\mathbf{z}_v\|_2 \leq 1 \\ & \forall u \in \mathcal{U}, i \in \mathcal{I}, t \in \mathcal{T}, v \in \mathcal{V} \end{aligned} \tag{4.14}$$

where $\Theta$ is the total parameter space, including all embeddings and variables of attention networks. It's obvious that both $L_{rec}$ and $L_{rel}$ can be decreased by simply scaling up the norm of corresponding embeddings. To avoid this problem during the training process, we explicitly constrain the embeddings to fall into a unit vector space. This constraint differs from traditional $L_2$ regularization which pushes parameters to the origin. It has been shown to be effective in both fields of KGE (Bordes et al., 2013; Lin et al., 2015) and recommendation (He et al., 2017a; Kang et al., 2018; Tay et al., 2018). The training procedure of RCF is illustrated in Algorithm 1.

### 4.2.5 Discussion

Here we examine three types of related recommendation models and discuss the relationship between RCF and them.

#### 4.2.5.1 Conventional collaborative filtering

RCF extends the item relations from the collaborative similarity to multiple and semantically meaningful relations. It can easily generalize the conventional CF methods. If we downgrade the MLP in Eq.(4.7) to inner product and only consider one item relation (i.e., the collaborative similarity), we can get the following predicted score:

$$\hat{y}_{ui} = \underbrace{\mathbf{p}_u^T \mathbf{q}_i}_{\text{MF}} + \underbrace{\mathbf{q}_i^T \left( \sum_{j \in \mathcal{I}_u^+ \setminus \{i\}} \beta(i,j) \cdot \mathbf{q}_j \right)}_{\text{NAIS}}, \tag{4.15}$$

which can be regarded as an ensemble of matrix factorization (Koren et al., 2009) and the item-based NAIS model (He et al., 2018c). In fact, compared with conventional ICF methods, RCF captures item relations in an explicit and fine-grained level, and thus enjoys much more expressiveness to model user preference.

#### 4.2.5.2 Knowledge graph enhanced recommendation

Recently, incorporating KG as an additional data source to enhance recommendation has become a hot research topic. These works can be categorized into embedding-based methods and path-based methods. Embedding-based methods (Zhang et al., 2016a; Huang et al., 2018; Wang et al., 2018b; Cao et al., 2019) utilize KG to guide the representation learning. However, the central part of ICF is the item similarity and none of these methods is designed to explicitly model it. On the contrary, RCF aims at directly modeling the item similarity from both the collaborative perspective and the multiple concrete relations. Path-based methods (Wang et al., 2019b; Sun et al., 2018; Hu et al., 2018a; Wang et al., 2018a; Ai et al., 2018) first construct paths to connect users and items, then the recommendation is generated by reasoning over these paths. However, constructing paths between users and items isn't a scalable approach when the number of users and items is very large. Under such a situation, sampling (Wang et al., 2018a; Ai et al., 2018) and pruning (Wang et al., 2019b; Sun et al., 2018) must be involved. However, RCF is free from this problem. Besides, the recommendation model of RCF is totally different from the path-based methods.

#### 4.2.5.3 Relation-aware recommendation

MCF (Park et al., 2017) proposed to utilize the "also-viewed" relation to enhance rating prediction. However, the "also-viewed" relation is just a special case of the item co-interact patterns and thus still belongs to the collaborative similarity. Another work that considers heterogeneous item relations is MoHR (Kang et al., 2018). But it only suits the sequential recommendation. The idea of MoHR is to predict both the next item and the next relation. The major drawback of MoHR is that it can only consider the relation between the last item of $\mathcal{I}_u^+$ and the target item. As a result, it fails to capture the long-term dependencies. On the contrary, RCF models the user preference based on all items in $\mathcal{I}_u^+$. The attention mechanism empowers RCF to be effective when capturing both long-term and short-term dependencies.

## 4.3 Experiments

In this section, we conduct experiments[1] on two real-world datasets to evaluate the proposed RCF model. We need to examine the performance of RCF and see how the item relations affect the recommendation accuracy and interpretability. Based on such motivation, we aim to answer the following research questions in this section:

**RQ1:** Compared with state-of-the-art recommendation models, how does RCF perform?

**RQ2:** How do the multiple item relations affect the model performance?

**RQ3:** How does RCF help to comprehend the user behavior? Can it generate more explainable recommendation?

In the following parts, we will first present the experimental settings and then answer the above research questions.

### 4.3.1 Experimental Settings

#### 4.3.1.1 Datasets

We perform experiments with two publicly accessible datasets: MovieLens[2] and KKBox[3], corresponding to movie and music recommendation, respectively. Table 4.2 summarizes the statistics of the two datasets.

**1. MovieLens.** This is the stable benchmark published by GroupLens (Harper and Konstan, 2016), which contains 943 users and 1,682 movies. We binarize the original user ratings to convert the dataset into implicit feedback. To introduce item relations, we combine it with the IMBD dataset[4]. The two datasets are linked by the titles and release dates of movies. The relation types of this data contain genres[5], directors, actors, and $t_0$, which is the relation type of the latent relation.

**2. KKBox.** This dataset is adopted from the WSDM Cup 2018 Challenge[6]

[1] Code can be found at https://github.com/XinGla/RCF

[2] https://grouplens.org/datasets/movielens/

[3] https://www.kaggle.com/c/kkbox-music-recommendation-challenge/data

[4] https://www.imdb.com/interfaces/

[5] Here, genres mean that two movies share at least one same genre, as shown in Figure 4.1. The same definition also suits the following relation types.

[6] https://wsdm-cup-2018.kkbox.events/

Table 4.2: Dataset statistics for Chapter 4.

| | Dataset | MovieLens | KKBox |
|---|---|---|---|
| User-Item Interactions | #users | 943 | 24,613 |
| | #items | 1,682 | 61,877 |
| | #interactions | 100,000 | 2,170,690 |
| Item-Item Relations | #types | 4 | 5 |
| | #values | 5,126 | 42,532 |
| | #triplets | 924,759 | 70,237,773 |

and is provided by the music streaming service KKBox. Besides the user-item interaction data, this dataset also contains the description of music, which can help us to introduce the item relations. We process this dataset by removing the songs that have a missing description. The final version contains 24,613 users, 61,877 items and 2,170,690 interactions. The relation types of this dataset contain genre, artist, composer, lyricist, and $t_0$.

#### 4.3.1.2 Evaluation protocols

To evaluate the performance of item recommendation, we adopt the leave-one-out evaluation, which has been widely used in literature (Kabbur et al., 2013; Chen et al., 2017; He et al., 2018c). More precisely, for each user in MovieLens, we leave his latest two interactions for validation and test and utilize the remaining data for training. For the KKBox dataset, because of the lack of timestamps, we randomly hold out two interactions for each user as the test example and the validation example and keep the remaining for training. Because the number of items is large in this dataset, it's too time-consuming to rank all items for every user. To evaluate the results more efficiently, we randomly sample 999 items that have no interaction with the target user and rank the validation and test items with respect to these 999 items. This has been widely used in many other works (Chen et al., 2017; Tay et al., 2018; Wang et al., 2019b; He et al., 2018c).

The recommendation quality is measured by three metrics: hit ratio (HR), mean reciprocal rank (MRR), and normalized discounted cumulative gain (NDCG). HR@$N$ is a recall-based metric, measuring whether the test item is in the top-$N$ positions of the recommendation list (1 for yes and 0 otherwise). MRR@$N$ and

NDCG@$N$ are weighted versions that assign higher scores to the top-ranked items in the recommendation list. Detailed definition of these metrics can be found in section 2.1.3.2.

#### 4.3.1.3 Compared methods

We compare the performance of the proposed RCF with the following baselines:

- MF (Koren et al., 2009): This is the standard matrix factorization which models the user preference with the inner product between user and item embeddings.
- FISM (Kabbur et al., 2013): This is a state-of-the-art ICF model which characterizes the user with the mean aggregation of the embeddings of his interacted items.
- NAIS (He et al., 2018c): This method enhances FISM through a neural attention network. It replaces the mean aggregation of FISM with an attention-based summation.
- FM (Rendle, 2010): Factorization machine is a feature-based baseline that models the user preference with feature interactions. Here we treat the auxiliary information of both datasets as additional input features.
- NFM (He and Chua, 2017): Neural factorization machine improves FM by utilizing a MLP to model the high-order feature interactions.
- CKE (Zhang et al., 2016a): This is an embedding-based KG-enhanced recommendation method, which integrates the item embeddings from MF and TransR (Lin et al., 2015).
- MoHR (Kang et al., 2018): This method is a state-of-the-art relation-aware CF method. We only report its results on the MovieLens dataset because it's designed for sequential recommendation and the KKBox dataset contains no timestamp information.

#### 4.3.1.4 Parameter settings

To fairly compare the performance of models, we train all of them by optimizing the BPR loss with mini-batch Ada-grad (Duchi et al., 2011). The learning rate is set as 0.05 and the batch size is set as 512. The embedding size is set as 64 for all models. For all the baselines, the $L_2$ regularization coefficients are tuned between $[1e^{-5}, 1e^{-4}, 1e^{-3}, 0]$. For FISM, NAIS, and RCF, the smoothing factor $\rho$ is set as 0.5. We pre-train NAIS with 100 iterations of FISM. For the attention-based RCF and NAIS, the attention factor is set as 32. Regarding NFM, we use FM embeddings with 100 iterations as pre-training vectors. The number of MLP layers is set as 1 with 64 neurons, which is the recommended setting of their original paper (He and Chua, 2017). The dropout ratio is tuned between $[0, 0.1, \cdots, 0.9]$. For the MLP of RCF, we adopt the same settings with NFM to guarantee a fair comparison. For MoHR, we set the multi-task learning weights as 1 and 0.1 according to their original paper (Kang et al., 2018). For RCF, we find that it achieves satisfactory performance when $\gamma = 0.01$. We report the results under this setting if there is no special mention.

### 4.3.2 Model Comparison (RQ1)

Table 4.3 demonstrates the comparison between all related methods when generating top-$N$ recommendation. It's obvious that the proposed RCF achieves the best performance among all methods on both datasets regarding all different top-$N$ values. This observation also provides strong support for the thesis statement (1). Both RCF and CFM show that deep supervised learning can help to improve recommendation accuracy. However, RCF and CFM are designed for different use cases. CFM is mainly tailored for feature-based context-aware recommendation, which can be regarded as a black-box deep model. While RCF aims to incorporate item-item relations for more convincing recommendation.

Compared with the conventional item-based FISM and NAIS which only consider the collaborative similarity, our RCF is based on the multiple and concrete item relations. We argue that this is the major source of improvement. From this perspective, the results demonstrate the importance of multiple item relations when modeling user preference.

Table 4.3: Top-$N$ recommendation performance comparison of different models ($N = 5, 10, 20$) in Chapter 4. NG is short for NDCG. $*$ denotes the significance $p$-value $< 0.05$ compared with the best baseline on the corresponding metric (indicated by boldface).

| Model | MovieLens | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | HR@5 | MRR@5 | NG@5 | HR@10 | MRR@10 | NG@10 | HR@20 | MRR@20 | NG@20 |
| MF | 0.0774 | 0.0356 | 0.0458 | 0.1273 | 0.0430 | 0.0642 | 0.2110 | 0.0482 | 0.0833 |
| FISM | 0.0795 | 0.0404 | 0.0500 | 0.1325 | 0.0474 | 0.0671 | 0.2099 | 0.0526 | 0.0865 |
| NAIS | 0.0827 | 0.0405 | 0.0508 | 0.1367 | 0.0477 | 0.0683 | 0.2142 | 0.0528 | 0.0876 |
| FM | 0.0827 | 0.0421 | 0.0521 | 0.1410 | **0.0496** | 0.0707 | 0.1994 | 0.0535 | 0.0852 |
| NFM | **0.0880** | 0.0427 | **0.0529** | **0.1495** | 0.0495 | 0.0725 | 0.2153 | 0.0540 | **0.0889** |
| CKE | 0.0827 | 0.0414 | 0.0515 | 0.1404 | 0.0476 | 0.0688 | 0.2089 | 0.0528 | 0.0884 |
| MoHR | 0.0832 | **0.0490** | 0.0499 | 0.1463 | 0.0485 | **0.0733** | **0.2249** | **0.0554** | 0.0882 |
| RCF | **0.1039**$^*$ | **0.0517**$^*$ | **0.0646**$^*$ | **0.1591**$^*$ | **0.0598**$^*$ | **0.0821**$^*$ | **0.2354**$^*$ | **0.0642**$^*$ | **0.1015**$^*$ |
| Model | KKBox | | | | | | | | |
| | HR@5 | MRR@5 | NG@5 | HR@10 | MRR@10 | NG@10 | HR@20 | MRR@20 | NG@20 |
| MF | 0.5575 | 0.3916 | 0.4329 | 0.6691 | 0.4065 | 0.4690 | 0.7686 | 0.4135 | 0.4942 |
| FISM | 0.5676 | 0.4084 | 0.4356 | 0.6866 | 0.4103 | 0.4844 | 0.7654 | 0.4258 | 0.5244 |
| NAIS | 0.5862 | 0.4156 | 0.4409 | 0.6932 | 0.4153 | 0.4966 | 0.7810 | 0.4333 | 0.5315 |
| FM | 0.5793 | 0.4064 | 0.4495 | 0.6949 | 0.4219 | 0.4869 | **0.7941** | 0.4288 | 0.5121 |
| NFM | **0.5973** | 0.4183 | **0.4630** | **0.7178** | **0.4432** | **0.5088** | 0.7768 | **0.4476** | 0.5244 |
| CKE | 0.5883 | **0.4191** | 0.4613 | 0.6930 | 0.4332 | 0.4952 | 0.7865 | 0.4397 | **0.5389** |
| RCF | **0.7158**$^*$ | **0.5612**$^*$ | **0.5999**$^*$ | **0.7940**$^*$ | **0.5718**$^*$ | **0.6253**$^*$ | **0.8563**$^*$ | **0.5762**$^*$ | **0.6412**$^*$ |

Table 4.4: Performance of RCF when replacing the attention with an average summation. Avg-1 denotes the first-level attention (i.e.,$a(u,t)$) is replaced. Avg-2 means the second-level attention (i.e., $\beta_t(i,j,v)$) is replaced. Avg-both denotes replacing both attentions. Dec is the average decrease of performance. $*$ denotes the statistical significance for $p < 0.05$.

| Models | MovieLens | | | |
|---|---|---|---|---|
| | HR@10 | MRR@10 | NDCG@10 | Dec |
| Avg-1 | 0.1478 | 0.0556 | 0.0746 | -7.6% |
| Avg-2 | 0.1346 | 0.0501 | 0.0694 | -15.6% |
| Avg-both | 0.1294 | 0.0495 | 0.0684 | -17.8% |
| RCF | **0.1591**$^*$ | **0.0598**$^*$ | **0.0821**$^*$ | |
| Models | KKBox | | | |
| | HR@10 | MRR@10 | NDCG@10 | Dec |
| Avg-1 | 0.7657 | 0.5484 | 0.5773 | -5.0% |
| Avg-2 | 0.6983 | 0.4331 | 0.5249 | -16.8% |
| Avg-both | 0.6792 | 0.4103 | 0.4946 | -20.4% |
| RCF | **0.7940**$^*$ | **0.5718**$^*$ | **0.6253**$^*$ | |

Compared with the feature-based FM and NFM, RCF still achieves significant improvement. The reason is that although FM and NFM also incorporate the auxiliary information, they fail to explicitly model the item relations based on that data. Besides, we can also see that NFM achieves better overall performance than FM because it introduces a MLP to learn high-order interaction signals. However, RCF achieves higher performance under the same MLP settings, which confirms the effectiveness of modeling item relations.

Compared with CKE, we can see that although CKE utilizes KG to guide the learning of item embeddings, it fails to directly model user preference based on multiple item relations, resulting in lower performance than RCF. Besides, we can see that although MoHR is also relation-aware, RCF still achieves better results than it. The reason is that MoHR only considers the relation between the last historical item and the target item, and thus fails to capture the long-term dependencies among the user interaction history.

Table 4.5: Modification of RCF. Single denotes only considering one relation (i.e., collaborative similarity). RCF-type only considers relation types for the attention. RCF-value only considers relation values. $g$ denotes the attention function.

| | Modification |
|---|---|
| Single | Eq.(4.1)$\Rightarrow \mathbf{m}_{u,i} = \mathbf{p}_u + \sum_{j \in \mathcal{I}_u^+} g(i,j) \cdot \mathbf{q}_j$ |
| RCF-type | Eq.(4.4)$\Rightarrow \mathbf{s}_{u,i}^t = \sum_{j \in \mathcal{I}_{u,i}^t} g(i,j) \cdot \mathbf{q}_j$ |
| RCF-value | Eq.(4.1)$\Rightarrow \mathbf{m}_{u,i} = \mathbf{p}_u + \sum_{j \in \mathcal{I}_u^+} g(i,j,v) \cdot \mathbf{q}_j$ |

### 4.3.3 Studies of Item Relations (RQ2)

#### 4.3.3.1 Effect of the hierarchy attention

RCF utilizes a hierarchy attention mechanism to model user preference. In this part, we conduct experiments to demonstrate the effect of the two-level attentions. Table 4.4 shows the results of top-10 recommendation when replacing the corresponding attention with average summation. It's obvious that both the first-level and the second-level attentions are necessary to capture user preference, especially the second-level attention, which aims at calculating a specific weight for every historical item and thus largely improves the model expressiveness.

#### 4.3.3.2 Ablation studies on relation modeling

The proposed RCF defines the item relations with relation types and relation values. To demonstrate the effectiveness of these two components, we modify the proposed RCF by masking the corresponding parts. Table 4.5 shows the detail of the masked models. Table 4.6 reports the performance when masking different relation components. We can draw the following conclusions from this table.

1. RCF-type achieves better performance than the single model, demonstrating the importance of relation types. Generally speaking, the type component describes item relations at an abstract level. It helps to model the users' preference on a class of items that share a particular similarity in some macro perspectives.

Table 4.6: Performance of different relation ablations when generating top-10 recommendation. Dec is the average decrease of performance. $*$ denotes the statistical significance for $p < 0.05$.

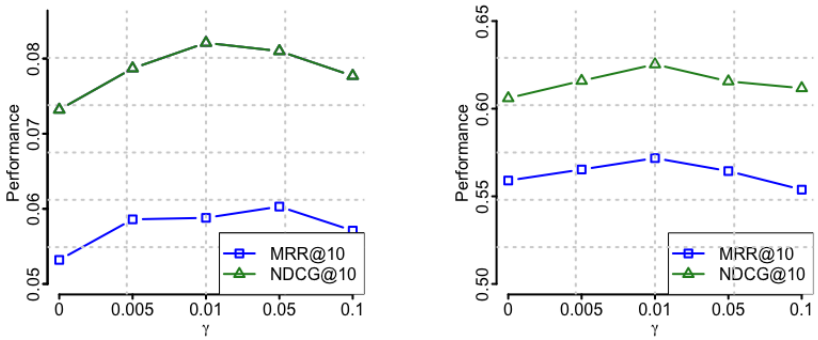
| Ablations | MovieLens | | | |
|---|---|---|---|---|
| | HR@10 | MRR@10 | NDCG@10 | Dec |
| Single | 0.1399 | 0.0481 | 0.0691 | -14.6% |
| RCF-type | 0.1484 | 0.0587 | 0.0804 | -4.5% |
| RCF-value | 0.1548 | 0.0558 | 0.0801 | -3.4% |
| RCF | **0.1591**$^*$ | **0.0598**$^*$ | **0.0821**$^*$ | |
| Ablations | KKBox | | | |
| | HR@10 | MRR@10 | NDCG@10 | Dec |
| Single | 0.6923 | 0.4666 | 0.5207 | -15.7% |
| RCF-type | 0.7523 | 0.5431 | 0.5723 | -6.2% |
| RCF-value | 0.7708 | 0.5579 | 0.5867 | -3.8% |
| RCF | **0.7940**$^*$ | **0.5718**$^*$ | **0.6253**$^*$ | |

2. The performance of RCF-value is also better than the single model. This finding verifies the effectiveness of relation values, which describe the relation between two specific items in a much fine-grained level. The relation value increases the model fidelity and expressiveness largely through capturing the user preference from a micro perspective.

3. RCF achieves the best performance. It demonstrates that both relation types and relation values are necessary to model the user preference. Moreover, it also confirms the effectiveness of the proposed two-level attention mechanism to tackle the hierarchical item relations.

#### 4.3.3.3 Effect of multi-task learning

RCF utilizes the item relational data in two ways: constructing the target-aware user embeddings and introducing the relational structure between item embeddings through the multi-task learning framework. In this part, we conduct experiments to show the effect of the latter.



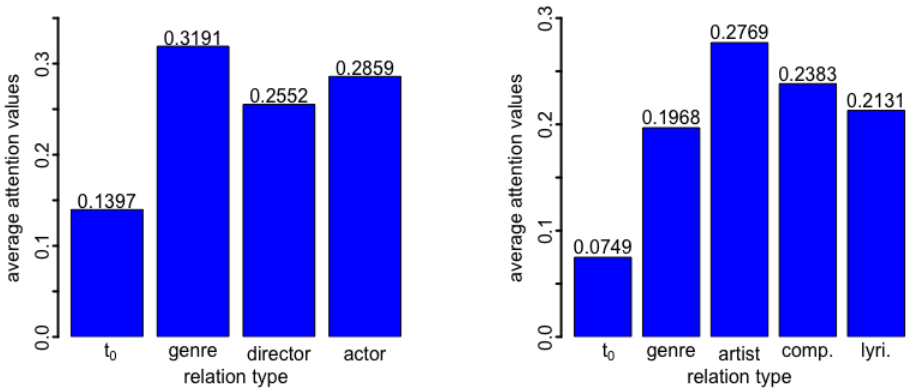

(a) MovieLens (b) KKBox

Figure 4.4: Effect of $\gamma$ on the RCF model.

Figure 4.4 reports the results of MRR@10 and NDCG@10 when changing the multi-task learning weight $\gamma$. It's obvious the performance of RCF boosts when $\gamma$ increases from 0 to positive values on both two datasets. Because $\gamma = 0$ means only the recommendation task (i.e., $L_{rec}$) is considered, we can draw a conclusion that jointly training $L_{rec}$ and $L_{rel}$ can definitely improve the model performance. In fact, the function of $L_{rel}$ is to introduce a constraint that if there is a relation between two items, there must be an inherent structure among their embeddings. This constraint explicitly guides the learning process of both item and relation embeddings and thus helps to improve the model performance. We can also see that with the increase of $\gamma$, the performance improves first and then starts to decrease. Because the primary target of RCF is recommendation other than predicting item relations, we must make sure that $L_{rec}$ accounts for the crucial part in the total loss. Actually, we can see from Table 4.2 that the number of item-item relational triplets is much larger than the number of user-item interactions, leading to a situation that $\gamma$ is commonly set as a small value.

### 4.3.4 Qualitative Analyses (RQ3)

In this part, we conduct qualitative analyses to show how RCF helps us to comprehend user behaviors and generate more explainable recommendation.




(a) MovieLens (b) KKBox

Figure 4.5: Average $a(u,t)$ of RCF on two datasets. $a(u,t)$ denotes the user $u$'s attention on the relation type $t$.

#### 4.3.4.1 Users as a whole

Figure 4.5 illustrates the average $a(u,t)$ for all $u \in \mathcal{U}$ on the two datasets. We can see that on the MovieLens dataset, the largest $a(u,t)$ falls into the genre, which means that users tend to watch movies that share the same genres. The second position is an actor. This finding is in concert with common sense that genres and actors are the most two important elements that affect the users' choices on movies. Director is in the third position. Moreover, we can see that all these three relation types are more important than $t_0$, which denotes the collaborative similarity. It further confirms that only considering collaborative similarity is not enough to model user preference. Multiple and fine-grained item relations should be involved to generate better recommendation.

For the music domain, we can see that the most important relation type falls into the artist. Following that is comp. (short for composer) and lyri. (short for lyricist). They are the most three important factors that affect users when listening to music. Besides, compared with the movie domain, the attention $a(u,t_0)$ in the music domain is much smaller. It indicates that user behavior patterns when listening to music are more explicit than the ones when watching movies. As a result, our proposed RCF achieves a bigger improvement on the KKBox dataset,

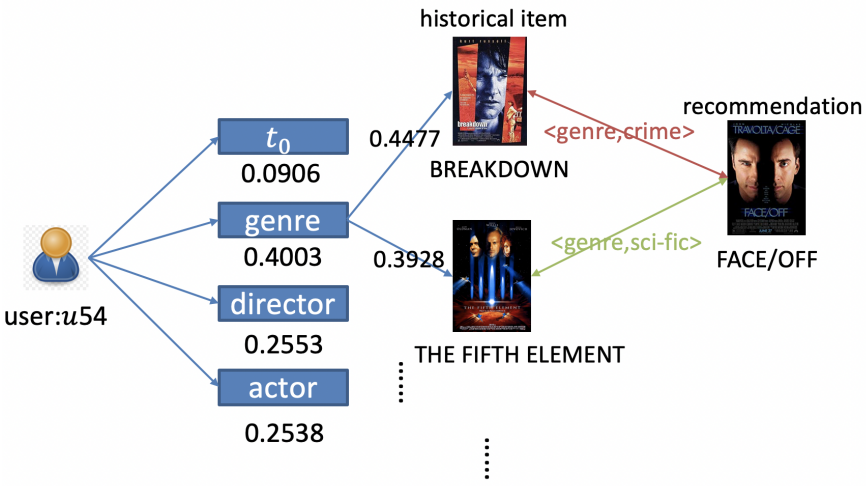



Figure 4.6: Attention visualization of user $u54$ in MovieLens.

as shown in Table 4.3. This attention visualization shows that using deep learning can also help us to get a better comprehension of user behaviors, providing supports for the thesis statement (2).

#### 4.3.4.2 Individual case studies

We randomly select a user $u54$ in the MovieLens dataset to see how RCF helps us to comprehend the individual user behavior. Figure 4.6 shows the attention visualization of this user. We can see that this user pays the most attention (0.4003) to the relation type "shared genres" when watching movies. Among the second-level relation values, he is most interested in "crime" (0.4477) and "sci-fic" (0.3928). Based on his historical interacted movies "Breakdown" and "The Fifth Element", the recommended movie is "Face/Off". From this perspective, we can also generate the explanation as *"Face/Off" is recommended to you because it is a crime movie like "Breakdown" you have watched before*. It's obvious that a side benefit of RCF is that it can generate reasonable explanations for recommendation results, providing strong support for the thesis statement (2).

## 4.4 Chapter Summary

In this chapter, we proposed a deep supervised learning model RCF to exploit multiple item relations for implicit feedback recommendation. RCF extends the item relations of ICF from collaborative similarity to fine-grained and concrete relations. We found that both the relation type and the relation value are crucial for capturing user preference. Based on that, we proposed to utilize a hierarchy attention mechanism to construct user representations. Besides, to maximize the usage of relational data, we further defined another task that aims to preserve the relational structure between item embeddings. We jointly optimize it with the recommendation task in an end-to-end fashion through a multi-task learning framework. Extensive experiments on two real-world datasets show that RCF achieves significant improvement over state-of-the-art baselines. Moreover, RCF also provides us an approach to better comprehend user behaviors and generate more convincing recommendation. The experimental results provide strong support for our thesis statement (1) and statement (2).

# Part III

# Deep Reinforcement Learning for Recommendation

In the previous part, two deep supervised learning models for implicit feedback recommendation are proposed, namely CFM and RCF. Through extensive experiments, we can see that utilizing deep supervised learning models can not only improve the recommendation accuracy but also increase the recommendation interpretability. However, deep supervised learning models are based on instant reward, without considering potential future gains. In this part, we will describe our research on utilizing deep reinforcement learning methods for implicit feedback recommendation, which aims at maximizing the cumulative gains in a whole interaction session. We proposed to combine supervised learning and reinforcement learning to perform more sample efficient off-policy reinforcement learning for implicit recommendation. Experiments on real-world datasets demonstrate the effectiveness of our proposed methods.

# Chapter 5

# Self-Supervised Reinforcement Learning

For a real live recommender system, it is important to consider a number of factors like long-term user engagement, multiple types of user-item interactions such as clicks, purchases, etc. The current supervised approaches target instant feedback and fail to model them appropriately. Casting the next-item recommendation task as a reinforcement learning (RL) problem is a promising direction. A major component of RL approaches is to train the agent through interactions with the environment. However, it is often problematic to train a recommender in an on-line fashion due to the requirement to expose users to irrelevant recommendations. As a result, learning the policy from logged implicit feedback is of vital importance, which is challenging due to the pure off-policy setting and lack of negative rewards (feedback). This conforms with the thesis statement (3).

In this chapter, we propose self-supervised reinforcement learning for recommendation tasks. Our approach augments standard recommendation models with two output layers: one for (self-)supervised learning, and the other for the RL part. The RL part acts as a regularizer to drive the supervised layer focusing on specific rewards (e.g., recommending items that may lead to purchases rather than clicks) and long-term optimization perspective while the self-supervised layer with cross-entropy loss provides strong gradient signals for parameter updates and representation learning. Based on such an approach, we propose two frameworks namely Self-Supervised Q-learning (SQN) and Self-Supervised Actor-Critic (SAC). We integrate the proposed frameworks with four state-of-the-art recommendation models. Experimental results on two real-world datasets demonstrate

the effectiveness of our approach. The results provide strong support for the thesis statement (3).

This chapter is mainly based on my previous work "Self-Supervised Reinforcement Learning for Recommender Systems" published in International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR) 2020 with DOI: https://doi.org/10.1145/3397271.3401147.

## 5.1 Introduction

Generating next item recommendation from sequential user-item implicit interactions in a session (e.g., views, clicks, or purchases) is one of the most common use cases of implicit feedback recommendation and occurs in various domains of live systems, such as e-commerce (Hu et al., 2018b), video (Chen et al., 2019a) and music recommendation (Yuan et al., 2019). Session-based (next item) recommendation models have often been trained with self-supervised learning, in which the model is trained to predict the data itself instead of some "external" labels. For instance, in language modeling, the task is often formulated as predicting the next word given the previous $m$ words (Mikolov et al., 2013). The same procedure can be utilized to predict the next item the user may be interested in given past interactions, see e.g., Hidasi et al. (2015); Yuan et al. (2019); Kang and McAuley (2018). However, this kind of approach can lead to sub-optimal recommendations since the model is purely learned by a loss function defined on the discrepancy between model predictions and the self-supervision signal. Such a loss may not match the expectations from the perspective of recommendation systems (e.g., long-term engagement). Moreover, there can be multiple types of user signals in one session, such as clicks, purchases, etc. How to leverage multiple types of user-item interactions to improve recommendation objectives (e.g., providing users recommendations that lead to real purchases) is also an important research question.

Reinforcement Learning (RL) has achieved impressive advances in game control (Mnih et al., 2015; Silver et al., 2016) and related fields. An RL agent is trained to take actions given the state of the environment it operates in with the objective of getting the maximum long-term cumulative rewards. A recom-

mender system aims (or should aim) to provide recommendations (actions) to users (environment) with the objective of maximizing the long-term user satisfaction (reward) with the system. For example, in one specific timestamp of an interaction session, recommending a mouse to the user could only get small rewards. However, this mouse recommendation could affect the following state of the user in the session and finally lead to the purchase of a new computer, which brings much larger cumulative rewards. Since in principle the reward schema can be designed at will, RL also allows creating models that can serve multiple objectives such as diversity and novelty. As a result, exploiting RL for recommendation has become a promising research direction.

There are two classes of RL methods: model-free RL algorithms and model-based RL algorithms. Model-free RL algorithms need to interact with the environment to observe the feedback and then optimize the policy. Doing this in an on-line fashion is typically unfeasible in commercial recommender systems since interactions with an under-trained policy would affect the user experience. A user may quickly abandon the service if the recommendations don't match her interests. A typical solution is learning off-policy from the logged implicit feedback. However, this poses the following challenges for applying RL-based methods:

- Pure off-policy settings: The policy should be learned from fixed logged data without interactions with the environment (users). Hence the data from which the RL agent is trained (i.e., logged data) will not match its policy. Chen et al. (2019a) proposed to use propensity scores to perform off-policy correction but this kind of method can suffer from unbounded high variances (Munos et al., 2016).

- Lack of data and negative rewards: RL algorithms are data-hungry, traditional techniques overcome this by either simulating the environments or by running RL iterations in controlled environments (e.g. games, robots). This is challenging in the case of recommendations especially considering a large number of potential actions (available items). Moreover, in most cases, learning happens from implicit feedback. The agent only knows which items the user has interacted with but has no knowledge about what the user dislikes. In other words, simply regressing to the Bellman equation (Bellman,

1966) (i.e., Q-learning) wouldn't lead to good ranking performance when there is no negative feedback since the model will be biased towards positive relevance values.

An alternative to off-policy training for recommender systems is the use of model-based RL algorithms. In model-based RL, one first builds a model to simulate the environment. Then the agent can learn by interactions with the simulated environment (Chen et al., 2019b; Shi et al., 2019). These two-stage methods heavily depend on the constructed simulator. Although related methods like generative adversarial networks (GANs) (Goodfellow et al., 2014) achieve good performance when generating images, simulating users' responses is a much more complex task.

In this chapter, we propose self-supervised reinforcement learning for implicit feedback recommendation. The proposed approach serves as a learning mechanism and can be easily integrated with existing recommendation models. More precisely, given a sequential or session-based recommendation model, the (final) hidden state of this model can be seen as its output as this hidden state is multiplied with the last layer to generate recommendations (Hidasi et al., 2015; Yuan et al., 2019; Kang and McAuley, 2018; Tang and Wang, 2018). We augment these models with two final output layers (heads). One is the conventional self-supervised head[1] trained with cross-entropy loss to perform ranking while the second is trained with RL based on the defined rewards such as long-term user engagement, purchases, recommendation diversity, and so on. For the training of the RL head, we adopt double Q-learning which is more stable and robust in the off-policy setting (Hasselt, 2010). The two heads complement each other during the learning process. The RL head serves as a regularizer to introduce desired properties to the recommendations while the ranking-based supervised head can provide negative signals for parameter updates and representation learning. We propose two frameworks based on such an approach: (1) Self-Supervised Q-learning (SQN) co-trains the two layers with a reply buffer generated from the logged implicit feedback; (2) Self-Supervised Actor-Critic (SAC) treats the RL head as a critic measuring the value of actions in a given state while the super-

[1]For simplicity, we use "self-supervised" and "supervised" interchangeable in this chapter. Besides, "head" and "layer" are also interchangeable.

vised head as an actor to perform the final ranking among candidate items. As a result, the model focuses on the pre-defined rewards while maintaining high-ranking performance. We verify the effectiveness of the proposed approaches by integrating SQN and SAC on four state-of-the-art next item recommendation models.

To summarize, this chapter makes the following contributions:

- We propose self-supervised reinforcement learning for the next item recommendation from implicit feedback data. Our approach extends existing recommendation models with a RL layer which aims to introduce reward-driven properties to the recommendation.
- We propose two frameworks SQN and SAC to co-train the supervised head and the RL head. We integrate four state-of-the-art recommendation models into the proposed frameworks.
- We conduct experiments on two real-world e-commerce datasets with both clicks and purchase interactions to validate our proposal. Experimental results demonstrate the proposed methods are effective to improve hit ratios, especially when measured against recommending items that eventually got purchased.

## 5.2 Preliminaries

In this section, we first describe the basic problem of generating next item recommendations from session-based implicit feedback. We then introduce reinforcement learning and analyze its limitations for recommendation. Lastly, we provide a literature review on related work.

### 5.2.1 Next Item Recommendation

Let $\mathcal{I}$ denote the whole item set, then a user-item interaction sequence can be represented as $x_{1:t} = \{x_1, x_2, ..., x_{t-1}, x_t\}$, where $x_i \in \mathcal{I}(0 < i \leq t)$ is the index of the interacted item at timestamp $i$. Note that in a real world scenario there may be different kinds of interactions. For instance, in e-commerce use cases, the interactions can be clicks, purchases, add to basket and so on. In video

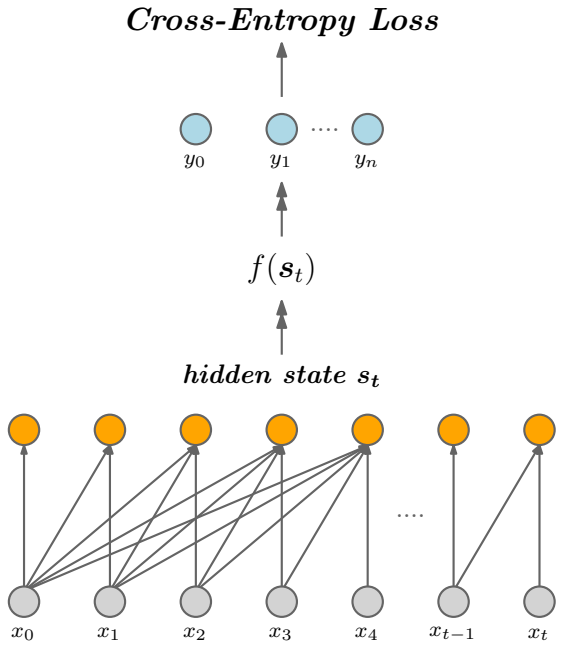



Figure 5.1: The self-supervised learning procedure for next item recommendation from implicit feedback.

recommendation systems, the interactions can be characterized by the watching time of a video. The goal of next item recommendation is recommending the most relevant item $x_{t+1}$ to the user given the sequence of previous interactions $x_{1:t}$.

We can cast this task as a (self-supervised) multi-class classification problem and build a sequential model that generates the classification logits $\mathbf{y}_{t+1} = [y_1, y_2, ...y_n] \in \mathbb{R}^n$, where $n$ is the number of candidate items. We can then choose the top-$N$ items from $\mathbf{y}_{t+1}$ as our recommendation list for timestamp $t+1$. A common procedure to train this type of recommender is shown in Figure 5.1. Typically one can use a generative model $G$ to map the input sequence into a hidden state $\mathbf{s}_t$ as $\mathbf{s}_t = G(x_{1:t})$. This serves as a general encoder function. Plenty of models have been proposed for this task and we have discussed prominent ones in section 2.2.2.5. Based on the obtained hidden state, one can utilize a decoder to map the hidden state to the classification logits as $\mathbf{y}_{t+1} = f(\mathbf{s}_t)$. It is usually defined as a simple fully connected layer or the inner product with candidate item embeddings (Yuan et al., 2019; Hidasi et al., 2015; Tang and Wang, 2018; Kang

and McAuley, 2018). Finally, we can train our recommendation model (agents) by optimizing a loss function based on the logits $\mathbf{y}_{t+1}$, such as the cross-entropy loss or the pair-wise ranking loss (Rendle et al., 2009b).

### 5.2.2 Reinforcement Learning

In terms of RL, we can formulate the next item recommendation problem as a Markov Decision Process (MDP) (Shani et al., 2005), in which the recommendation agent interacts with the environments $\mathcal{E}$ (users) by sequentially recommending items to maximize the long-term cumulative rewards. More precisely, the MDP can be defined by tuples consisting of $(\mathcal{S}, \mathcal{A}, \mathbf{P}, R, \rho_0, \gamma)$ where

- $\mathcal{S}$: a continuous state space to describe the user states. This is modeled based on the user (sequential) interactions with the items. The state of the user can be in fact represented by the hidden state of the sequential model discussed in section 5.2.1. Hence the state of a user at timestamp $t$ can be represented as $\mathbf{s}_t = G(x_{1:t}) \in \mathcal{S}$ $(t > 0)$.

- $\mathcal{A}$: a discrete action space that contains candidate items. The action $a$ of the agent is the selected item to be recommended. In off-line data, we can get the action at timestamp $t$ from the user-item interaction (i.e., $a_t = x_{t+1}$). There are also works that focus on generating slate (set)-based recommendations as discussed in section 2.3.5.

- $\mathbf{P}$: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the state transition probability.

- $R$: $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, where $r(\mathbf{s}, a)$ denotes the immediate reward by taking action $a$ at user state $\mathbf{s}$. The flexible reward scheme is crucial in the utility of RL for recommender systems as it allows for optimizing the recommendation models towards goals that are not captured by conventional loss functions. For example, in the e-commerce scenario, we can give a larger reward to purchase interactions compared with clicks to build a model that assists the user in his purchase rather than the browsing task. We can also set the reward according to item novelty (Bradley and Smyth, 2001) to promote recommendation diversity. For video recommen-

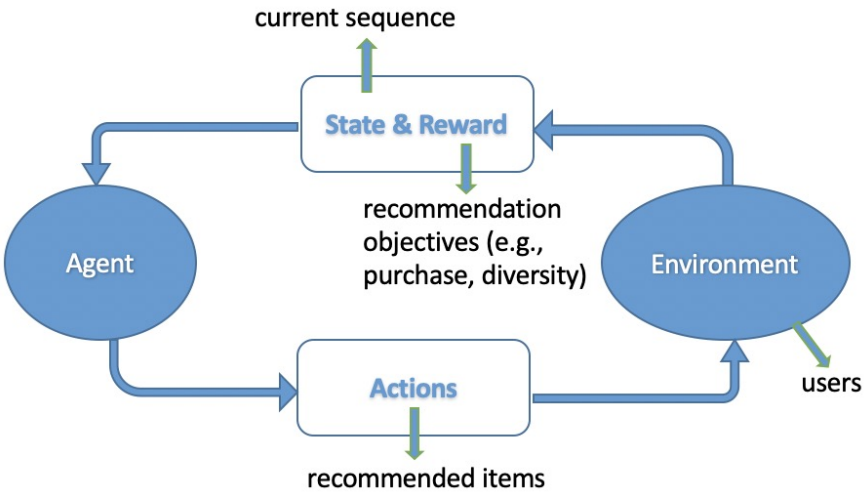



Figure 5.2: A typical RL-based recommender system example.

dation, we can set the rewards according to the watching time (Chen et al., 2019a).

- $\rho_0$ is the initial state distribution with $\mathbf{s}_0 \sim \rho_0$.
- $\gamma$ is the discount factor for future rewards.

A typical RL-based recommendation framework can be shown as Figure 5.2.

RL seeks a target policy $\pi_\theta(a|\mathbf{s})$ which translates the user state $\mathbf{s} \in \mathcal{S}$ into a distribution over actions $a \in \mathcal{A}$, so as to maximize the expected cumulative reward:

$$\max_{\pi_\theta} \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)], \text{ where } R(\tau) = \sum_{t=0}^{|\tau|} \gamma^t r(\mathbf{s}_t, a_t), \tag{5.1}$$

where $\theta \in \mathbb{R}^d$ denotes policy parameters. Note that the expectation is taken over trajectories $\tau = (\mathbf{s}_0, a_0, \mathbf{s}_1, ...)$, which are obtained by performing actions according to the target policy: $\mathbf{s}_0 \sim \rho_0$, $a_t \sim \pi_\theta(\cdot|\mathbf{s}_t)$, $s_{t+1} \sim \mathbf{P}(\cdot|\mathbf{s}_t, a_t)$.

Solutions to find the optimal $\theta$ can be categorized into policy gradient-based approaches (e.g., REINFORCE (Williams, 1992)) and value-based approaches (e.g., Q-learning (Silver et al., 2016)).

Policy-gradient based approaches aim at directly learning the mapping function $\pi_\theta$. Using the "log-trick" (Williams, 1992), gradients of the expected cumu-

lative rewards with respect to policy parameters $\theta$ can be derived as

$$\mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)\nabla_\theta \log \pi_\theta(\tau)]. \tag{5.2}$$

In on-line RL environments, it's easy to estimate the expectation. However, under the recommendation settings, to avoid recommending irrelevant items to users, the agent must be trained using the historical logged data. Even if the RL agent can interact with live users, the actions (recommended items) may be controlled by other recommenders with different policies since many recommendation models might be deployed in a real live recommender system. As a result, what we can estimate from the batched (logged) data is

$$\mathbb{E}_{\tau\sim\beta}[R(\tau)\nabla_\theta \log \pi_\theta(\tau)], \tag{5.3}$$

where $\beta$ denotes the behavior policy that we follow when generating the training data. Obviously, there is a distribution discrepancy between the target policy $\pi_\theta$ and the behavior policy $\beta$. Applying policy-gradient methods for recommendation using this data is thus infeasible.

Value-based approaches first calculate the Q-values (i.e., $Q(\mathbf{s}, a)$, the value of an action $a$ at a given state $\mathbf{s}$) according to the Bellman equation while the action is taken by $a = \text{argmax}\, Q(\mathbf{s}, a)$. The one-step temporal difference (TD) update rule formulates the target $Q(\mathbf{s}_t, a_t)$as Eq.(2.38). One of the major limitations of implicit feedback data is the lack of negative feedback (Rendle et al., 2009b; Yuan et al., 2018), which means we only know which items the user has interacted with but have no knowledge about the missing transactions. Thus there are limited state-action pairs to learn from and Q-values learned purely on this data would be sub-optimal as shown in Figure 5.3. As a result, taking actions using these Q-values by $a = \text{argmax}\, Q(\mathbf{s}, a)$ would result in poor performance. Even though the estimation of $Q(\mathbf{s}, a)$ is unbiased due to the greedy selection of Q-learning[1], the distribution of $\mathbf{s}$ in the logged data is biased. So the distribution discrepancy problem of policy gradient-based methods still exists in Q-learning even though the Q-learning algorithm is "off-policy" (Fujimoto et al., 2018).

From the above analysis, we can see that directly utilizing RL algorithms for implicit recommendation is problematic. The above analysis supports thesis statement (3).

[1]We don't consider the bias introduced by the steps of TD learning. This is not related to our work.

$$x_{1:t} \overset{\textit{click}}{\{x_1,} \overset{\textit{purchase}}{x_2,} \ldots, \overset{\textit{click}}{x_{t-1},} \overset{\textit{click}}{x_t\}}$$

$Q(\boldsymbol{s}_0, x_1) =$ reward of click $+ \max_a Q(\boldsymbol{s}_1, a)$

$Q(\boldsymbol{s}_1, x_2) =$ reward of purchase $+ \max_a Q(\boldsymbol{s}_2, a)$

$Q(\boldsymbol{s}_0, x_1^-) = ?$ $Q(\boldsymbol{s}_1, x_2^-) = ?$ ** *no learning constraints* **

$argmax Q(\boldsymbol{s}, a) = ?$ ** *fails to perform ranking* **

Figure 5.3: Q-learning fails to learn a proper preference ranking because of data sparsity and the lack of negative feedback. $x_1^-$ and $x_2^-$ are unseen (negative) items for the corresponding timestamp.

## 5.3 The Proposed Methods

As discussed in section 5.2.2, directly applying standard RL algorithms to implicit feedback data is essentially unfeasible. In this section, we propose to co-train a RL output layer along with the self-supervised head. The reward can be designed according to the specific demands of the recommendation setting. We first describe the proposed SQN algorithm and then extend it to SAC. Both algorithms can be easily integrated with existing recommendation models.

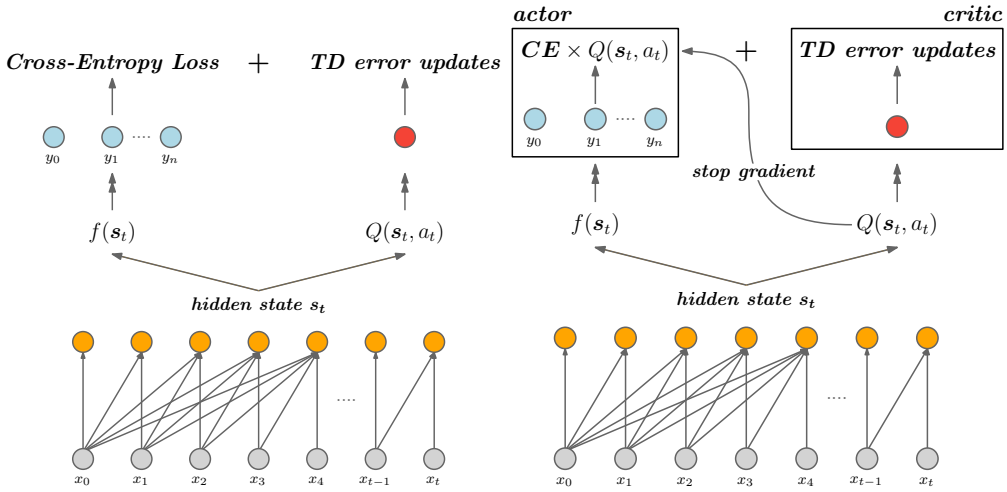
### 5.3.1 Self-Supervised Q-learning

Given an input item sequence $x_{1:t}$ and an existing recommendation model $G$, the self-supervised training loss can be defined as the cross-entropy over the classification distribution:

$$L_s = -\sum_{i=1}^{n} Y_i log(p_i), \text{where } p_i = \frac{e^{y_i}}{\sum_{i'=1}^{n} e^{y_{i'}}}. \tag{5.4}$$

$Y_i$ is an indicator function and $Y_i = 1$ if the user interacted with the $i$-th item in the next timestamp. Otherwise, $Y_i = 0$. Due to the fact that the recommendation model $G$ has already encoded the input sequence into a latent representation $\mathbf{s}_t$, we can directly utilize $\mathbf{s}_t$ as the state for the RL part without introducing another model. What we need is an additional final layer to calculate the Q-values. A concise solution is stacking a fully-connected layer on the top of $G$:

$$Q(\mathbf{s}_t, a_t) = \delta(\mathbf{s}_t \mathbf{h}_t^T + b) = \delta(G(x_{1:t})\mathbf{h}_t^T + b), \tag{5.5}$$




(a) SQN architecture.

(b) SAC architecture. CE stands for cross-entropy.

Figure 5.4: The proposed frameworks in Chapter 5.

where $\delta$ denotes the activation function, $\mathbf{h}_t$ and $b$ are trainable parameters of the RL head.

After that, we can define the loss for the RL part based on the one-step TD error:

$$L_q = (r(\mathbf{s}_t, a_t) + \gamma \max_{a'} Q(\mathbf{s}_{t+1}, a') - Q(\mathbf{s}_t, a_t))^2 \tag{5.6}$$

We jointly train the self-supervised loss and the RL loss on the replay buffer generated from the implicit feedback data:

$$L_{SQN} = L_s + L_q. \tag{5.7}$$

Figure 5.4(a) demonstrates the architecture of SQN.

When generating recommendations, we still return the top-$N$ items from the supervised head. The RL head acts as a regularizer to fine-tune the recommendation model $G$ according to our reward settings. As discussed in section 5.2.2, the state distribution in the logged data is biased, so generating recommendations using the Q-values is problematic. However, due to the greedy selection of $Q(\mathbf{s}_{t+1}, \cdot)$, the estimation of $Q(\mathbf{s}_t, a_t)$ itself is unbiased. As a result, by utilizing Q-learning as a regularizer and keeping the self-supervised layer as the source of recommendations we avoid any off-policy correction issues. The lack of negative rewards in Q-learning does also not affect the methods since the RL output layer

**Algorithm 2** Training procedure of SQN

**Input:** user-item interaction sequence set $\mathcal{X}$, recommendation model $G$, reinforcement head $Q$, supervised head

**Output:** all parameters in the learning space $\Theta$

Initialize all trainable parameters
Create $G'$ and $Q'$ as copies of $G$ and $Q$, respectively
**repeat**
    Draw a mini-batch of $(x_{1:t}, a_t)$ from $\mathcal{X}$, set rewards $r$
    $\mathbf{s}_t = G(x_{1:t})$, $\mathbf{s}'_t = G'(x_{1:t})$
    $\mathbf{s}_{t+1} = G(x_{1:t+1})$, $\mathbf{s}'_{t+1} = G'(x_{1:t+1})$
    Generate random variable $z \in (0, 1)$ uniformly
    **if** $z \leq 0.5$ **then**
        $a^* = \text{argmax}_a \, Q(\mathbf{s}_{t+1}, a)$
        $L_q = (r + \gamma Q'(\mathbf{s}'_{t+1}, a^*) - Q(\mathbf{s}_t, a_t))^2$
        Calculate $L_s$ and $L_{SQN}$ according to Eq.(5.4) and Eq.(6.2)
        Perform updates by $\nabla_\Theta L_{SQN}$
    **else**
        $a^* = \text{argmax}_a \, Q'(\mathbf{s}'_{t+1}, a)$
        $L_q = (r + \gamma Q(\mathbf{s}_{t+1}, a^*) - Q'(\mathbf{s}'_t, a_t))^2$
        Calculate $L_s$ and $L_{SQN}$ according to Eq.(5.4) and Eq.(6.2)
        Perform updates by $\nabla_\Theta L_{SQN}$
    **end if**
**until** converge
return all parameters in $\Theta$

is trained on positive actions and the supervised cross-entropy loss provides the negative gradient signals which come from the denominator of Eq.(5.4).

To enhance the learning stability, we utilize double Q-learning (Hasselt, 2010) to alternatively train two copies of learnable parameters. Algorithm 2 describes the training procedure of SQN.

### 5.3.2 Self-Supervised Actor-Critic

In the previous subsection, we proposed SQN in which the Q-learning head acts as a regularizer to fine-tune the model in line with the reward schema. The learned Q-values are unbiased and learned from positive user-item interactions (feedback). As a result, these values can be regarded as an unbiased measurement of how the recommended item satisfies our defined rewards. Hence the actions with high Q-values should get increased weight on the self-supervised loss, and vice versa.

We can thus treat the self-supervised head which is used for generating recommendations as a type of "actor" and the Q-learning head as the "critic". Based on this observation, we can use the Q-values as weights for the self-supervised loss:

$$L_A = L_s \cdot Q(\mathbf{s}_t, a_t). \tag{5.8}$$

This is similar to what is used in the well-known Actor-Critic (AC) methods (Konda and Tsitsiklis, 2000). However, the actor in AC is based on policy gradient which is on-policy while the "actor" in our methods is essentially self-supervised. Moreover, there is only one base model $G$ in SAC while AC has two separate networks for the actor and the critic. To enhance stability, we stop the gradient flow and fix the Q-values when they are used in that case. We then train the actor and critic jointly. Figure 5.4(b) illustrates the architecture of SAC.

In complex recommendation models (e.g., using the Transformer encoder as $G$ (Kang and McAuley, 2018)), the learning of Q-values can be unstable (Parisotto et al., 2019), particularly in the early stages of training. To mitigate these issues, we set a threshold $T$. When the number of update steps is smaller than $T$, we perform normal SQN updates. After that, the Q-values become more stable and we start to use the critic values in the self-supervised layer and perform updates according to the architecture of Figure 5.4(*b*). The training procedure of SAC is

demonstrated in Algorithm 3.

### 5.3.3 Discussion

The proposed training frameworks can be integrated with existing recommendation models, as long as they follow the procedure of Figure 5.1 to generate recommendations. This is the case for most session-based or sequential recommendation models introduced over the last years. In this paper we utilize the cross-entropy loss as the self-supervised loss, while the proposed methods also work for other losses (Rendle et al., 2009b; Hidasi and Karatzoglou, 2017). The proposed methods are for general purpose recommendation. One can design the reward schema according to his/her own demands and recommendation objectives.

Due to the biased state-action distribution in the off-line setting and the lack of sufficient data, directly generating recommendations from RL is difficult. The proposed SQN and SAC frameworks can be seen as attempts to exploit a RL estimator to "reinforce" existing self-supervised recommendation models. Another way of looking at the proposed approach is as a form of transfer learning whereby the self-supervised model is used to "pretrain" parts of the Q-learning model and vice versa.

## 5.4 Experiments

In this section, we conduct experiments[1] on two real-world next item recommendation datasets to evaluate the proposed SQN and SAC in the e-commerce scenario. To support thesis statement (3), we need to justify whether the proposed learning frameworks can improve the recommendation quality when they are integrated with different base models. Then, we need to see how the reward settings of the RL part affect the learning performance. Lastly, to further demonstrate whether RL can be directly utilized for recommendation, we need to verify the performance when there is only deep reinforcement learning without the supervised part. Based on such motivations, we aim to answer the following research questions:

[1]The implementation can be found at https://drive.google.com/open?id=1nLL3_knhj_RbaP_IepBLkwaT6zNIeD5z

**Algorithm 3** Training procedure of SAC

**Input:** the interaction sequence set $\mathcal{X}$, recommendation model $G$, reinforcement head $Q$, supervised head, threshold $T$
**Output:** all parameters in the learning space $\Theta$

Initialize all trainable parameters
Create $G'$ and $Q'$ as copies of $G$ and $Q$, $t = 0$
**repeat**
    Draw a mini-batch of $(x_{1:t}, a_t)$ from $\mathcal{X}$, set rewards $r$
    $\mathbf{s}_t = G(x_{1:t})$, $\mathbf{s}'_t = G'(x_{1:t})$
    $\mathbf{s}_{t+1} = G(x_{1:t+1})$, $\mathbf{s}'_{t+1} = G'(x_{1:t+1})$
    Generate random variable $z \in (0, 1)$ uniformly
    **if** $z \leq 0.5$ **then**
        $a^* = \text{argmax}_a \, Q(\mathbf{s}_{t+1}, a)$
        $L_q = (r + \gamma Q'(\mathbf{s}'_{t+1}, a^*) - Q(\mathbf{s}_t, a_t))^2$
        **if** $t \leq T$ **then**
            Perform updates by $\nabla_\Theta L_{SQN}$
        **else**
            $L_A = L_s \times Q(\mathbf{s}_t, a_t)$, $L_{SAC} = L_A + L_s$
            Perform updates by $\nabla_\Theta L_{SAC}$
        **end if**
    **else**
        $a^* = \text{argmax}_a \, Q'(\mathbf{s}'_{t+1}, a)$
        $L_q = (r + \gamma Q(\mathbf{s}_{t+1}, a^*) - Q'(\mathbf{s}'_t, a_t))^2$
        **if** $t \leq T$ **then**
            Perform updates by $\nabla_\Theta L_{SQN}$
        **else**
            $L_A = L_s \times Q'(\mathbf{s}'_t, a_t)$, $L_{SAC} = L_A + L_s$
            Perform updates by $\nabla_\Theta L_{SAC}$
        **end if**
    **end if**
    $t = t + 1$
**until** converge
return all parameters in $\Theta$

**RQ1:** How do the proposed methods perform when integrated with existing recommendation models?

**RQ2:** How does the RL component affect performance, including the reward setting and the discount factor.

**RQ3:** What is the performance if we only use Q-leaning for recommendation?

In the following parts, we will describe the experimental settings and answer the above research questions.

## 5.4.1 Experimental Settings

### 5.4.1.1 Datasets

We conduct experiments with two publicly accessible e-commerce datasets: RC15[1] and RetailRocket[2].

**RC15.** This is based on the dataset of RecSys Challange 2015. The dataset is session-based and each session contains a sequence of clicks and purchases. We remove the sessions whose length is smaller than 3 and then sample 200k sessions, resulting in a dataset that contains 1,110,965 clicks and 43,946 purchases over 26702 items. We sort the user-item interactions in one session according to the timestamp.

**RetailRocket.** This dataset is collected from a real-world e-commerce website. It contains sequential events of viewing and adding to the cart. To keep in line with the RC15 dataset, we treat views as clicks and adding to cart as purchases. We remove the items which are interacted less than 3 times and the sequences whose length is smaller than 3. The final dataset contains 1,176,680 clicks and 57,269 purchases over 70852 items.

Table 5.1 summarizes the statistics of the two datasets.

### 5.4.1.2 Evaluation protocols

We adopt cross-validation to evaluate the performance of the proposed methods. The ratio of training, validation, and test set are 8:1:1. We randomly sample 80% of the sequences as the training set. For validation and test sets, the evaluation is done by providing the events of a sequence one-by-one and checking the rank of

[1] https://recsys.acm.org/recsys15/challenge/

[2] https://www.kaggle.com/retailrocket/ecommerce-dataset

Table 5.1: Dataset statistics for Chapter 5.

| Dataset | RC15 | RetailRocket |
|---|---|---|
| #sequences | 200,000 | 195,523 |
| #items | 26,702 | 70,852 |
| #clicks | 1,110,965 | 1,176,680 |
| #purchase | 43,946 | 57,269 |

the item of the next event. The ranking is performed among the whole item set. Each experiment is repeated 5 times, and the average performance is reported.

The recommendation quality is measured with two metrics: hit ratio (HR) and normalized discounted cumulative gain (NDCG). HR@$N$ is a recall-based metric, measuring whether the ground-truth item is in the top-$N$ positions of the recommendation list. We can define HR for clicks as:

$$\text{HR(click)} = \frac{\#\text{hits among clicks}}{\#\text{clicks}}. \tag{5.9}$$

HR(purchase) is defined similarly with HR(click) by replacing the clicks with purchases. NDCG is a rank-sensitive metric that assigns higher scores to top positions in the recommendation list. The more detailed definition can be found in section 2.1.3.2.

#### 5.4.1.3 Baselines

We integrated the proposed SQN and SAC with four state-of-the-art (generative) next item recommendation models:

- GRU (Hidasi et al., 2015): This method utilizes a GRU to model the input sequences. The final hidden state of the GRU is treated as the latent representation for the input sequence.

- Caser (Tang and Wang, 2018): This is a recently proposed CNN-based method that captures sequential signals by applying convolution operations on the embedding matrix of previous interacted items.

- NItNet (Yuan et al., 2019): This method utilizes dilated CNN to enlarge the receptive field and residual connection to increase the network depth, achieving good performance with high efficiency.

- SASRec (Kang and McAuley, 2018): This baseline is motivated from self-attention and uses the Transformer (Vaswani et al., 2017) architecture. The output of the Transformer encoder is treated as the latent representation for the input sequence.

All models are implemented with Tensorflow. More precisely, for NItNet and SASRec, we use the original author implementation. For GRU and Caser, we use the implementation from https://github.com/fajieyuan/nextitnet.

#### 5.4.1.4 Parameter settings

For both of the datasets, the input sequences are composed of the last 10 items before the target timestamp. If the sequence length is less than 10, we complement the sequence with a padding item. We train all models with the Adam optimizer (Kingma and Ba, 2014) through back-propagation. The mini-batch size is set as 256. The learning rate is set as 0.01 for RC15 and 0.005 for RetailRocket. We evaluate the validation set every 2000 batches of updates. For a fair comparison, the item embedding size is set as 64 for all models. For GRU4Rec, the size of the hidden state is set as 64. For Caser, we use 1 vertical convolution filter and 16 horizontal filters whose heights are set from {2,3,4}. The drop-out ratio is set as 0.1. For NextItNet, we utilize the code published by its authors and keep the settings unchanged. For SASRec, the number of heads in self-attention is set as 1 according to its original paper (Kang and McAuley, 2018). For the proposed SQN and SAC, if not mentioned otherwise, the discount factor $\gamma$ is set as 0.5 while the ratio between the click reward ($r_c$) and the purchase reward ($r_p$) is set as $r_p/r_c = 5$.

### 5.4.2 Performance Comparison (RQ1)

Table 5.2 and Table 5.3 show the performance of top-$N$ recommendation on RC15 for purchase and click prediction, respectively. Table 5.4 and Table 5.5 show the performance of top-$N$ recommendation on RetailRocket for purchase and click prediction, respectively.

We observe that on the RC15 dataset, the proposed SQN method achieves consistently better performance than the corresponding baseline when predict-

Table 5.2: Top-$N$ recommendation performance comparison of different models ($N = 5, 10, 20$) in Chapter 5 on RC15 dataset for purchase prediction. NG is short for NDCG. Boldface denotes the highest score. $*$ denotes the significance $p$-value $< 0.01$ compared with the corresponding baseline.

| Models | purchase | | | | | |
|---|---|---|---|---|---|---|
| | HR@5 | NG@5 | HR@10 | NG@10 | HR@20 | NG@20 |
| GRU | 0.3994 | 0.2824 | 0.5183 | 0.3204 | 0.6067 | 0.3429 |
| GRU-SQN | 0.4228* | 0.3016* | 0.5333* | 0.3376* | 0.6233* | 0.3605* |
| GRU-SAC | **0.4394*** | **0.3154*** | **0.5525*** | **0.3521*** | **0.6378*** | **0.3739*** |
| Caser | 0.4475 | 0.3211 | 0.5559 | 0.3565 | 0.6393 | 0.3775 |
| Caser-SQN | 0.4553* | 0.3302* | 0.5637* | 0.3653* | 0.6417* | 0.3862* |
| Caser-SAC | **0.4866*** | **0.3527*** | **0.5914*** | **0.3868*** | **0.6689*** | **0.4065*** |
| NItNet | 0.3632 | 0.2547 | 0.4716 | 0.2900 | 0.5558 | 0.3114 |
| NItNet-SQN | 0.3845* | 0.2736* | 0.4945* | 0.3094* | **0.5766*** | 0.3302* |
| NItNet-SAC | **0.3914*** | **0.2813*** | **0.4964*** | **0.3155*** | 0.5763* | **0.3357*** |
| SASRec | 0.4228 | 0.2938 | 0.5418 | 0.3326 | 0.6329 | 0.3558 |
| SASRec-SQN | 0.4336 | 0.3067* | 0.5505 | 0.3435* | 0.6442* | 0.3674* |
| SASRec-SAC | **0.4540*** | **0.3246*** | **0.5701*** | **0.3623*** | **0.6576*** | **0.3846*** |

Table 5.3: Top-$N$ recommendation performance comparison of different models ($N = 5, 10, 20$) in Chapter 5 on RC15 dataset for click prediction. NG is short for NDCG. Boldface denotes the highest score. $*$ denotes the significance $p$-value $< 0.01$ compared with the corresponding baseline.

| Models | click | | | | | |
|---|---|---|---|---|---|---|
| | HR@5 | NG@5 | HR@10 | NG@10 | HR@20 | NG@20 |
| GRU | 0.2876 | 0.1982 | 0.3793 | 0.2279 | 0.4581 | 0.2478 |
| GRU-SQN | **0.3020*** | **0.2093*** | **0.3946*** | **0.2394*** | **0.4741*** | **0.2587*** |
| GRU-SAC | 0.2863 | 0.1985 | 0.3764 | 0.2277 | 0.4541 | 0.2474 |
| Caser | 0.2728 | 0.1896 | 0.3593 | 0.2177 | 0.4371 | 0.2372 |
| Caser-SQN | **0.2742** | **0.1909** | **0.3613** | **0.2192** | **0.4381** | **0.2386** |
| Caser-SAC | 0.2726 | 0.1894 | 0.3580 | 0.2171 | 0.4340 | 0.2362 |
| NItNet | 0.2950 | 0.2030 | 0.3885 | 0.2332 | 0.4684 | 0.2535 |
| NItNet-SQN | **0.3091*** | **0.2137*** | **0.4037*** | **0.2442*** | **0.4835*** | **0.2645*** |
| NItNet-SAC | 0.2977* | 0.2055* | 0.3906 | 0.2357* | 0.4693 | 0.2557* |
| SASRec | 0.3187 | 0.2200 | 0.4164 | 0.2515 | 0.4974 | 0.2720 |
| SASRec-SQN | **0.3272*** | **0.2263*** | **0.4255*** | **0.2580*** | **0.5066*** | **0.2786*** |
| SASRec-SAC | 0.3130 | 0.2161 | 0.4114 | 0.2480 | 0.4945 | 0.2691 |

Table 5.4: Top-$N$ recommendation performance comparison of different models ($N = 5, 10, 20$) in Chapter 5 on RetailRocket for purchase prediction. NG is short for NDCG. Boldface denotes the highest score. $*$ denotes the significance $p$-value $< 0.01$ compared with the corresponding baseline.

| Models | purchase | | | | | |
|---|---|---|---|---|---|---|
| | HR@5 | NG@5 | HR@10 | NG@10 | HR@20 | NG@20 |
| GRU | 0.4608 | 0.3834 | 0.5107 | 0.3995 | 0.5564 | 0.4111 |
| GRU-SQN | **0.5069**$^*$ | 0.4130$^*$ | **0.5589**$^*$ | 0.4289$^*$ | **0.5946**$^*$ | 0.4392$^*$ |
| GRU-SAC | 0.4942$^*$ | **0.4179**$^*$ | 0.5464$^*$ | **0.4341**$^*$ | 0.5870$^*$ | **0.4428**$^*$ |
| Caser | 0.3491 | 0.2935 | 0.3857 | 0.3053 | 0.4198 | 0.3141 |
| Caser-SQN | 0.3674$^*$ | 0.3089$^*$ | 0.4050$^*$ | 0.3210$^*$ | 0.4409$^*$ | 0.3301$^*$ |
| Caser-SAC | **0.3871**$^*$ | **0.3234**$^*$ | **0.4336**$^*$ | **0.3386**$^*$ | **0.4763**$^*$ | **0.3494**$^*$ |
| NItNet | 0.5630 | 0.4630 | 0.6127 | 0.4792 | 0.6477 | 0.4881 |
| NItNet-SQN | **0.5895**$^*$ | 0.4860$^*$ | **0.6403**$^*$ | 0.5026$^*$ | **0.6766**$^*$ | 0.5118$^*$ |
| NItNet-SAC | **0.5895**$^*$ | **0.4985**$^*$ | 0.6358$^*$ | **0.5162**$^*$ | 0.6657$^*$ | **0.5243**$^*$ |
| SASRec | 0.5267 | 0.4298 | 0.5916 | 0.4510 | 0.6341 | 0.4618 |
| SASRec-SQN | **0.5681**$^*$ | 0.4617$^*$ | **0.6203**$^*$ | 0.4806$^*$ | **0.6619**$^*$ | 0.4914$^*$ |
| SASRec-SAC | 0.5623$^*$ | **0.4679**$^*$ | 0.6127$^*$ | **0.4844**$^*$ | 0.6505$^*$ | **0.4940**$^*$ |

Table 5.5: Top-$N$ recommendation performance comparison of different models ($N = 5, 10, 20$) in Chapter 5 on RetailRocket for click prediction. NG is short for NDCG. Boldface denotes the highest score. $*$ denotes the significance $p$-value $< 0.01$ compared with the corresponding baseline.

| Models | purchase | | | | | |
|---|---|---|---|---|---|---|
| | HR@5 | NG@5 | HR@10 | NG@10 | HR@20 | NG@20 |
| GRU | 0.2233 | 0.1735 | 0.2673 | 0.1878 | 0.3082 | 0.1981 |
| GRU-SQN | **0.2487**$^*$ | **0.1939**$^*$ | **0.2967**$^*$ | **0.2094**$^*$ | **0.3406**$^*$ | **0.2205**$^*$ |
| GRU-SAC | 0.2451$^*$ | 0.1924$^*$ | 0.2930$^*$ | 0.2074$^*$ | 0.3371$^*$ | 0.2186$^*$ |
| Caser | 0.1966 | 0.1566 | 0.2302 | 0.1675 | 0.2628 | 0.1758 |
| Caser-SQN | 0.2089$^*$ | 0.1661$^*$ | 0.2454$^*$ | 0.1778$^*$ | 0.2803$^*$ | 0.1867$^*$ |
| Caser-SAC | **0.2206**$^*$ | **0.1732**$^*$ | **0.2617**$^*$ | **0.1865**$^*$ | **0.2999**$^*$ | **0.1961**$^*$ |
| NItNet | 0.2495 | 0.1906 | 0.2990 | 0.2067 | 0.3419 | 0.2175 |
| NItNet-SQN | **0.2610**$^*$ | **0.1982**$^*$ | **0.3129**$^*$ | **0.2150**$^*$ | **0.3586**$^*$ | **0.2266**$^*$ |
| NItNet-SAC | 0.2529$^*$ | 0.1964$^*$ | 0.3010$^*$ | 0.2119$^*$ | 0.3458$^*$ | 0.2233$^*$ |
| SASRec | 0.2541 | 0.1931 | 0.3085 | 0.2107 | 0.3570 | 0.2230 |
| SASRec-SQN | **0.2761**$^*$ | **0.2104**$^*$ | **0.3302**$^*$ | **0.2279**$^*$ | **0.3803**$^*$ | **0.2406**$^*$ |
| SASRec-SAC | 0.2670$^*$ | 0.2056$^*$ | 0.3208$^*$ | 0.2230$^*$ | 0.3701$^*$ | 0.2355$^*$ |

ing both clicks and purchases. SQN introduces a Q-learning head that aims to model the long-term cumulative reward. The additional learning signal from this head improves both clicks and purchase recommendation performance because the models are now trained to select actions that are optimized not only for the current state but also for future interactions. We can see that on this dataset, the best performance when predicting purchase interactions is achieved by SAC. Since the learned Q-values are used as weights for the supervised loss function, the model is "reinforced" to focus more on purchases. As a result, the SAC method achieves significantly better results when recommending purchases. We can assume that the strong but sparse signal that comes with a purchase is better utilized by SAC.

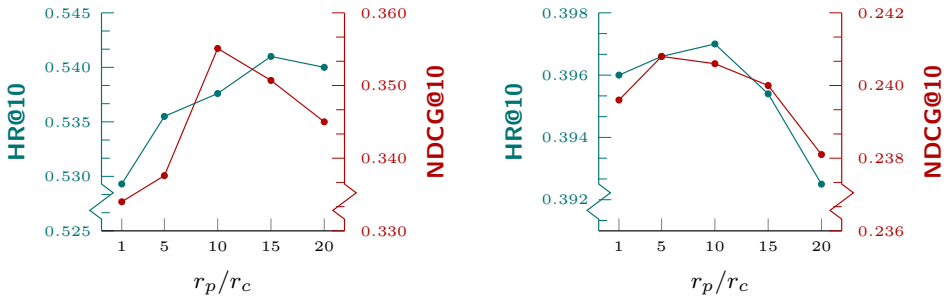
On the RetailRocket dataset, we can see that both SQN and SAC achieve consistently better performance than the corresponding baseline in terms of predicting both clicks and purchases. This further verifies the effectiveness of the proposed methods. Besides, we can also see that even though SQN sometimes achieves the highest HR(purchase), SAC always achieves the best performance with respect to the NDCG of purchase. This demonstrates that the proposed SAC is actually more likely to push the items which may lead to purchase to the top positions of the recommendation list.

To conclude, it's obvious that the proposed SQN and SAC achieve consistent improvement with respect to the selected baselines. This demonstrates the effectiveness and the generalization ability of our methods, supporting thesis statement (3).

### 5.4.3 RL Investigation(RQ2)

#### 5.4.3.1 Effect of reward settings.

In this part, we conduct experiments to investigate how the reward setting of RL affects the model performance. Figure 5.5 and Figure 5.6 show the results of HR@10 and NDCG@10 when changing the ratio between $r_p$ and $r_c$ (i.e., $r_p/r_c$) on RC15 and RetailRocket, respectively. We show the performance when choosing GRU as the base model. We can see from Figure 5.5(a) and Figure 5.6(a) that the performance of SQN when predicting purchase interactions start to improve when $r_p/r_c$ increases from 1. It shows that when we assign a higher reward



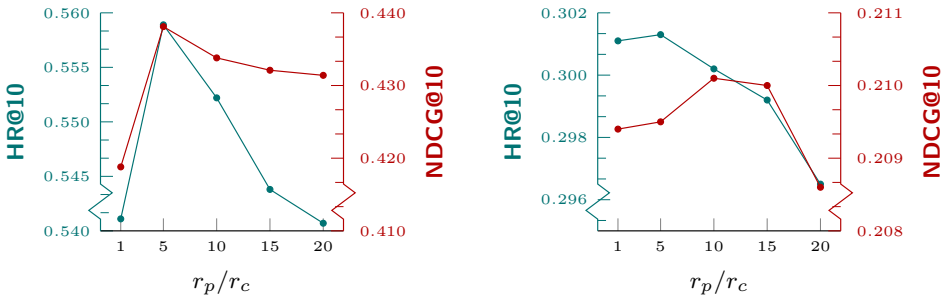

(a) SQN for purchase

(b) SQN for click

(c) SAC for purchase

(d) SAC for click

Figure 5.5: Effect of reward settings on RC15




(a) SQN for purchase

(b) SQN for click

(c) SAC for purchase

(d) SAC for click

Figure 5.6: Effect of reward settings on RetailRocket

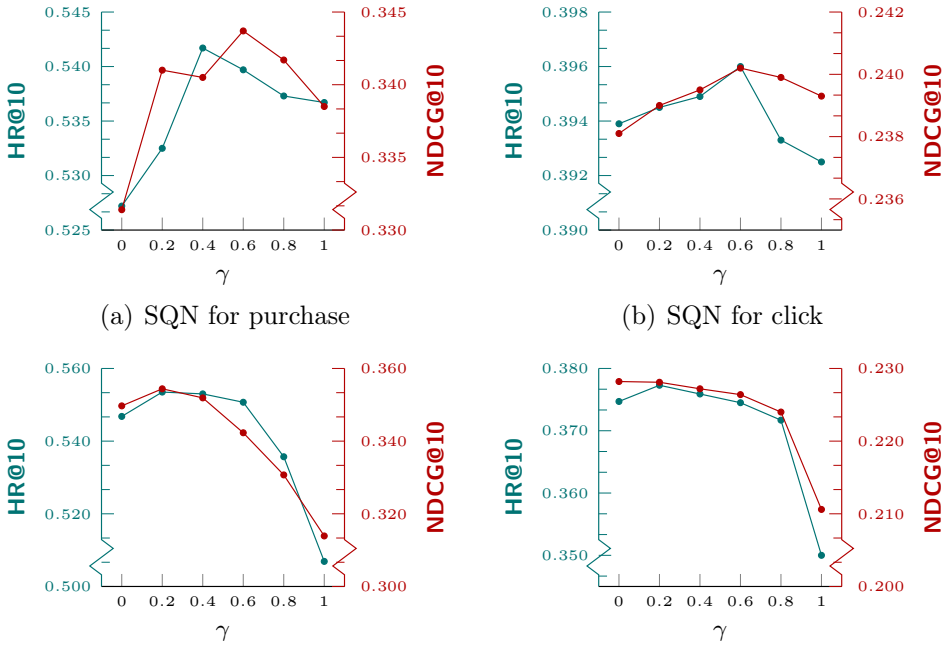
to purchases, the introduced RL head successfully drives the model to focus on the desired rewards. Performance starts to decrease after reaching higher ratios. The reason may be that high reward differences might cause instability in the TD updates and thus affects the model performance. Figure 5.5(c) and Figure 5.6(c) shows that the performance of SAC when predicting purchase behaviors also improves at the beginning and then drops with the increase of $r_p/r_c$. It's similar to SQN.

For click recommendation, we can see from Figure 5.5(b) and Figure 5.6(b) that the performance of SQN is actually stable at the beginning (even increases a little) and then starts to decrease. There are two factors for this performance drop. The first is the instability of RL as discussed before. The second is that too much reward discrepancy might diminish the relative importance of clicks which constitute the vast majority of the data points. This also helps to explain the performance drop of SAC as shown in Figure 5.5(d) and Figure 5.6(d).

In addition, observing the performance of SQN and SAC when $r_p/r_c = 1$, we can find that they still perform better than the basic GRU. For example, when predicting purchases on the RC15 dataset, the HR@10 of SAC is around 0.54 according to Figure 5.5(c) while the basic GRU method only achieves 0.5183 according to Table 5.2. This means that even if we don't distinguish between clicks and purchases, the proposed SQN and SAC still work better than the basic model. The reason is that the introduced RL head successfully adds additional learning signals for long-term rewards.

#### 5.4.3.2 Effect of the discount factor

In this part, we show how the discount factor affects the recommendation performance. Figure 5.7 illustrates the HR@10 and NDCG@10 of SQN and SAC with different discount factors on the RC15 dataset. We choose GRU as the base recommendation model. We can see that the performance of SQN and SAC improves when the discount factor $\gamma$ increases from 0. $\gamma = 0$ means that the model doesn't consider long-term reward and only focuses on immediate feedback. This observation leads to the conclusion that taking long-term rewards into account does improve the overall HR and NDCG on both click and purchase recommendations. However, we can also see the performance decreases when the discount




(a) SQN for purchase

(b) SQN for click

(c) SAC for purchase

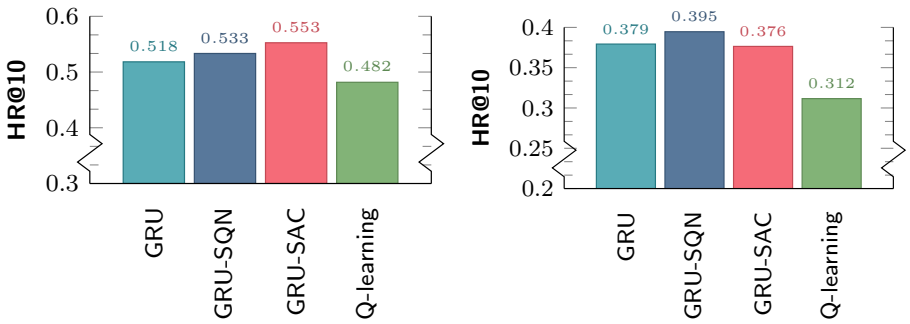
(d) SAC for click

Figure 5.7: Effect of discount factor

factor is too large. Compared with the game control domain in which there may be thousands of steps in one MDP, the user interaction sequence is much shorter. The average sequence length of the two datasets is only 6. As a result, although $\gamma = 0.95$ or 0.99 is a common setting for game control, a smaller discount factor should be applied under the recommendation settings.

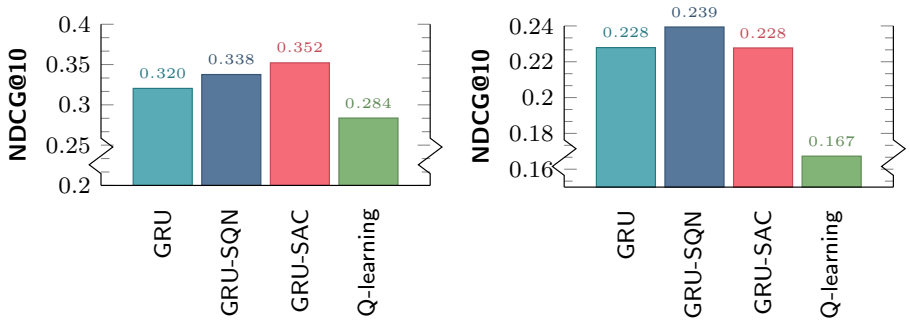
### 5.4.4 Q-learning for Recommendation (RQ3)

We also conduct experiments to examine the performance if we generate recommendations only using Q-learning. To make Q-learning more effective when perform ranking, we explicitly introduce uniformly sampled unseen items to provide negative rewards (Rendle et al., 2009b; Zhao et al., 2018). Figure 5.8 and Figure 5.9 show the results in terms of HR@10 and NDCG@10 on the RC15 dataset, respectively. The base model is GRU. We can see that the performance of Q-learning is even worse than the basic GRU method. As discussed in section 5.2.2, directly utilizing Q-learning for recommendation is problematic and




(a) Purchase predictions (b) Click predictions

Figure 5.8: Comparison of HR when only using Q-learning for recommendations.




(a) Purchase predictions (b) Click predictions

Figure 5.9: Comparison of NDCG when only using Q-learning for recommendations.

off-policy correction needs to be considered in that situation. This observation provides support for the thesis statement (3). However, the estimation of Q-values based on the given state is unbiased, and exploiting Q-learning as a regularizer or critic doesn't suffer from the above problem. Hence the proposed SQN and SAC achieve better performance.

## 5.5 Chapter Summary

In this chapter, we propose self-supervised reinforcement learning for recommender systems. We first formalize the next item recommendation task from implicit feedback data and then analyze the difficulties when exploiting RL for this task. The first is the pure off-policy setting which means the recommender

agent must be trained from logged data without interactions between the environment. The second is the lack of negative rewards. To address these problems, we propose to augment the existing recommendation model with another RL head. This head acts as a regularizer to introduce our specific desires into the recommendation. The motivation is to utilize the unbiased estimator of RL to fine-tune the recommendation model according to our own reward settings. Based on that, we propose SQN and SAC to perform joint training of the supervised head and the RL head. To verify the effectiveness of our methods, we integrate them with four state-of-the-art recommendation models and conduct experiments on two real-world e-commerce datasets. Experimental results demonstrate that the proposed SQN and SAC are effective to improve the hit ratio, especially when predicting the real purchase interactions. The experimental results and analysis provide support for the thesis statement (3).

# Chapter 6

# Self-Supervised Advantage Actor-Critic

In Chapter 5, we have observed that using reinforcement learning (RL) for next-item recommendation from implicit feedback through reward signals is a promising research direction towards recommender systems (RS) that maximize cumulative gains. However, the direct use of RL algorithms in the RS setting is impractical due to challenges like off-policy training, huge action spaces, and lack of sufficient reward signals. Chapter 5 attempts to tackle these challenges by combining RL and self-supervised learning, but the proposed methods still suffer from certain limitations from e.g., biased estimation of the Q-values due to the existence of only positive rewards and the length of the sequence.

To address such problems, in this chapter we propose to use negative sampling for the RL training procedure and then combine it with self-supervised learning, namely Self-Supervised Negative Q-learning (SNQN). Based on the sampled negative actions (items), we can then calculate the "advantage" of a positive action over the average case, which can be further utilized as a normalized weight for learning the self-supervised part. This leads to another learning framework: Self-Supervised Advantage Actor-Critic (SA2C). We integrate SNQN and SA2C with four state-of-the-art next-item recommendation models and conduct experiments on two real-world datasets. Experimental results show that the proposed approaches achieve significantly better performance than state-of-the-art supervised models and the methods proposed in Chapter 5 (i.e., SQN and SAC). The results provide strong support for the thesis statement (3). This chapter is mainly based on my under-review submission on SIGIR 2021.

## 6.1 Introduction

As described in section 5.1, generating next-item recommendation from implicit user-item interaction sequences is a common use case of RS. Using RL for next-item recommendation is a promising direction to introduce cumulative reward-based optimization objectives. However, unlike game control and robotics, directly utilizing RL for RS comes with a set of unique difficulties and challenges, like off-policy training and lack of sufficient reward signals.

In Chapter 5, we proposed self-supervised reinforcement learning for RS, achieving promising results on off-line evaluation metrics. Two learning frameworks namely Self-Supervised Q-learning (SQN) and Self-Supervised Actor-Critic (SAC) are proposed. The key insight of self-supervised RL is to utilize the RL component as a form of a regularizer to fine-tune the recommendation model according to pre-defined rewards. Although SQN and SAC achieve good performance, they still suffer from some limitations which need to be addressed. For example, the RL head[1] in SQN and SAC is only defined on positive (observed) actions (items), so the negative comparison signals only come from the supervised part. As a result, the RL head can only act as a regularizer but cannot be used to generate recommendations, as it lacks negative feedback to remove the bias introduced by the existence of only positive reward signals. Moreover, SAC uses the output Q-values as the critic to re-weight the actor (supervised part). However, the Q-values depend heavily on the length of the sequence, which further introduces bias to the learning procedure.

In this chapter, to address the above issues of SQN and SAC, we first propose to introduce a negative sampling strategy for training RL in a RS setting and then combine it with self-supervised learning. We call this Self-Supervised Negative Q-learning (SNQN). Different from SQN, which only performs RL on positive actions (clicks, views, etc.), the RL output head of SNQN is learned on both positive actions and a set of sampled negative actions. This design allows the RL part of the SNQN model to not only act as a regularizer but also as a good ranking model, which can be used to generate recommendations. Based on the sampled

[1]For simplicity, in this chapter we make "head" and "output layer" interchangeable in this paper. Moreover, "self-supervised" and "supervised" are also interchangeable.

negative actions and the unbiased estimate of the Q values, we can moreover calculate the "advantage" of a positive action over the other actions. We propose the Self-Supervised Advantage Actor-Critic (SA2C), which uses this advantage instead of the raw Q-values and re-weights the supervised output layer loss. The advantage values can be seen as normalized Q-values that help us alleviate the influence of sequence length on the estimate of the Q-values.

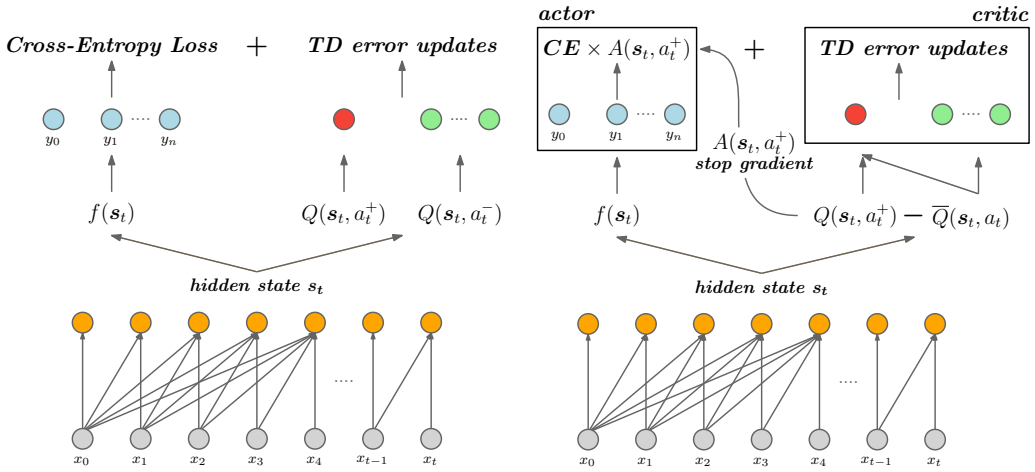
To summarize, this chapter makes the following contributions:

- We propose SNQN to introduce negative sampling into the RL training of the RS model and then combine it with self-supervised learning. As a result, both the supervised head and the RL head can be used to generate recommendations. We show that joint training of the two heads with shared base models helps to achieve better performance than independent learning.
- We propose SA2C to calculate the advantage of a positive action over the others. This advantage can be seen as a normalized Q-value and is further utilized to re-weight the supervised component.
- We integrate the proposed SNQN and SA2C with four state-of-the-art recommendation models and conduct experiments on two real-world e-commerce datasets. Experimental results demonstrate the proposed methods are effective and improve the performance of RS compared to existing methods.

## 6.2 The Proposed Methods

### 6.2.1 Self-Supervised Negative Q-learning

Given an input user-item interaction sequence $x_{1:t}$ and an existing recommendation model $G(\cdot)$, the self-supervised training loss can be defined as the cross-entropy over the classification distribution, as shown in Eq.(5.4). The cross-entropy loss will push the positive logits to high values. Meanwhile, the cross-entropy loss can also provide negative learning signals by pushing down the output values of items that the user has not interacted with. This is particularly helpful in an RS setting where ranking items that are likely to be interacted by the user in the top-$N$ positions is the main goal.




(a) SNQN architecture.

(b) SA2C architecture. CE is short for cross-entropy.

Figure 6.1: The learning framework architectures of SNQN and SA2C.

Since $G(\cdot)$ already encodes the input sequence into a latent state, we can directly utilize Eq.(5.5) to calculate $Q(\mathbf{s}_t, a_t)$ given the state and action pair. When learning from logged implicit feedback data, it's often the case that there are no negative reward signals (Rendle et al., 2009b). Performing Q-learning solely based on positive reward signals (clicks, views, etc.), where the negative interaction signals are not provided, would lead to a model with a positive bias. As a result, the RL component of the model can only act as a regularizer but can't be used for generating recommendation. To address this issue, we propose to introduce a negative reward sampling strategy for the RL training procedure. More precisely, the Q-learning loss function of SNQN is defined not only on positive actions and rewards but also on the sampled negative ones. Based on this motivation, we define the one-step time difference (TD) Q-loss of SNQN as:

$$\begin{aligned} L_q = & \underbrace{(r(\mathbf{s}_t, a_t^+) + \gamma \max_{a'} Q(\mathbf{s}_{t+1}, a') - Q(\mathbf{s}_t, a_t^+))^2}_{L_p\text{: positive TD error}} \\ & + \underbrace{\sum_{a_t^- \in N_t} (r(\mathbf{s}_t, a_t^-) + \gamma \max_{a'} Q(\mathbf{s}_t, a') - Q(\mathbf{s}_t, a_t^-))^2}_{L_n\text{: negative TD error}}, \end{aligned} \tag{6.1}$$

where $a_t^+$ and $a_t^-$ are the positive action and negative action at timestamp $t$,

respectively. $N_t$ denotes the set of sampled unobserved (negative) actions.

Another interpretation of negative sampling in RL is imitation learning under sparse reward settings (Reddy et al., 2019). Reddy et al. (2019) demonstrated that imitation learning through expert demonstrations can be achieved with promising performance, by assigning a constant positive reward for matching the demonstrated actions and a constant negative reward for other behaviors. In the recommendation settings, we can regard the observed (interacted) actions as expert behaviors while sampled unobserved actions as non-expert behaviors with a constant negative reward. From that perspective, the introduced negative sampling strategy of SNQN makes the RL component a good imitation learning agent. In our implementation, we assign a constant reward value $r_n$ for negative actions (i.e.,$r(\mathbf{s}_t, a_t^-) = r_n$).

Note that in the negative TD error, the maximum operation is performed in $Q(\mathbf{s}_t, a')$. This is because when learning from off-line data, we can make an assumption that only positive actions can affect the user state. In other words, taking a negative (unobserved) action doesn't update the user state. This is a widely adopted assumption in the literature (Zhao et al., 2018; Ie et al., 2019; Chen et al., 2019b). While for the positive reward $r(\mathbf{s}_t, a_t^+)$, we can define according to the specific demands of the recommendation domain. For example, in e-commerce we can assign a higher reward to actions which lead to purchases rather than just clicks. As in SQN, we jointly train the self-supervised loss and the RL loss on the replay buffer generated from the implicit feedback data:

$$L_{snqn} = L_s + L_q. \tag{6.2}$$

Figure 6.1(a) demonstrates the architecture of SNQN.

We utilize double Q-learning for better learning stability (Hasselt, 2010) and alternately train two copies of model parameters. Algorithm 4 describes the training procedure of SNQN.

### 6.2.2 Self-Supervised Advantage Actor-Critic

Actor-Critic (AC) methods have been successfully used in the RL research area. The key idea of AC methods is the introduction of a critic that evaluates the goodness of an action taken and assigns higher weights to actions with high cumulative rewards. In the proposed SNQN method, the supervised component

**Algorithm 4** Training procedure of SNQN

**Input:** user-item interaction sequence set $\mathcal{X}$, recommendation model $G(\cdot)$, reinforcement head $Q(\cdot)$, supervised head $f(\cdot)$, pre-defined reward function $r(\mathbf{s}, a)$
**Output:** all parameters in the learning space $\Theta$

Initialize all trainable parameters
Create $G'(\cdot)$ and $Q'(\cdot)$ as copies of $G(\cdot)$ and $Q(\cdot)$, respectively
**repeat**
    Draw a mini-batch of $(x_{1:t}, a_t^+)$ from $\mathcal{X}$
    Draw negative actions set $N_t$ for $x_{1:t}$
    $\mathbf{s}_t = G(x_{1:t})$, $\mathbf{s}'_t = G'(x_{1:t})$
    $\mathbf{s}_{t+1} = G(x_{1:t+1})$, $\mathbf{s}'_{t+1} = G'(x_{1:t+1})$
    Generate random variable $z \in (0, 1)$ uniformly
    **if** $z \leq 0.5$ **then**
        $a_*^+ = \text{argmax}_a\ Q(\mathbf{s}_{t+1}, a)$, $a_*^- = \text{argmax}_a\ Q(\mathbf{s}_t, a)$
        $L_p = (r(\mathbf{s}_t, a_t^+) + \gamma Q'(\mathbf{s}'_{t+1}, a_*^+) - Q(\mathbf{s}_t, a_t^+))^2$
        $L_n = \sum_{a_t^- \in N_t} (r(\mathbf{s}_t, a_t^-) + \gamma Q'(\mathbf{s}'_t, a_*^-) - Q(\mathbf{s}_t, a_t^-))^2$
        Calculate $L_s$ and $L_{snqn} = L_s + L_p + L_n$
        Perform updates by $\nabla_\Theta L_{snqn}$
    **else**
        $a_*^+ = \text{argmax}_a\ Q'(\mathbf{s}_{t+1}, a)$, $a_*^- = \text{argmax}_a\ Q'(\mathbf{s}_t, a)$
        $L_p = (r(\mathbf{s}_t, a_t^+) + \gamma Q(\mathbf{s}_{t+1}, a_*^+) - Q'(\mathbf{s}'_t, a_t^+))^2$
        $L_n = \sum_{a_t^- \in N_t} (r(\mathbf{s}_t, a_t^-) + \gamma Q(\mathbf{s}_t, a_*^-) - Q'(\mathbf{s}'_t, a_t^-))^2$
        Calculate $L_s$ and $L_{snqn} = L_s + L_p + L_n$
        Perform updates by $\nabla_\Theta L_{snqn}$
    **end if**
**until** converge
return all parameters in $\Theta$

can be seen as the actor which aims at imitating the logged user behavior. A simple solution for the critic is to use the output Q-values from the RL head, as these Q-values measure the cumulative rewards the system gains given the state-action pair. However, these Q-values are sensitive to the sequence length. For example, a bad action in a long sequence could also have a high Q-value if it occurs in an early position because Q-values have a bias with respect to the sequence position.

Instead of the absolute Q-value, what we actually would like to measure is how much "advantage" we can get by taking an action compared to the average case (i.e. average Q-values). This advantage can help us alleviate the bias introduced from the sequence length. However, calculating the average Q-values along the whole action space would introduce additional computation cost, especially when the candidate item set is large. Luckily, we have already introduced negative samples in the proposed SNQN methods. As a result, a concise solution is to calculate the average among the sampled actions (including both positive and negative examples) as an approximation. Based on this motivation, the average Q-values can be defined as:

$$\overline{Q}(\mathbf{s}_t, a) = \frac{\sum_{a' \in a_t^+ \cap N_t} Q(\mathbf{s}_t, a')}{|N_t| + 1}. \tag{6.3}$$

The advantage of an observed (positive) action is formulated as:

$$A(\mathbf{s}_t, a_t^+) = Q(\mathbf{s}_t, a_t^+) - \overline{Q}(\mathbf{s}_t, a). \tag{6.4}$$

We can then use this advantage to re-weight the actor (i.e. the supervised head). If a positive action has higher advantage over the average, we increase its weight in the supervised training procedure, and vice versa.

To enhance stability, we stop the gradient flow and fix the Q-values when they are used to calculate the average and advantage. We then train the actor and critic jointly. The training loss of SA2C is formulated as:

$$L_{sa2c} = L_a + L_q, \text{ where } L_a = L_s \cdot A(\mathbf{s}_t, a_t^+). \tag{6.5}$$

Figure 6.1(b) illustrates the architecture of SA2C. During the training procedure, the learning of Q-values can be unstable (Parisotto et al., 2019), particularly in the early stage. To mitigate these issues, we pre-train the model using SNQN in the first $T$ training steps (batches). When the Q-values become more stable, we start to use the advantage to re-weight the actor and perform updates according

to the architecture of Figure 6.1(b). We also use double Q-learning and the training procedure of SA2C is shown in Algorithm 5.

### 6.2.3 Discussion

We provide a brief discussion about the connections between our algorithms and some related methods.

By assigning a negative reward $r_n$ to unobserved actions, SNQN explicitly introduces negative action signals in the RL head. As a result, both the supervised head and the RL head of SNQN can be used to generate recommendations[1]. Compared to SQN which can only use the supervised head to generate recommendation, SNQN provides a more flexible choice to switch between the two heads. For example, if we want the agent to imitate more the logged user data, we can use the supervised head. On the contrary, if we want the RS to be more reward-driven, we can use the RL head.

A related work to SA2C is (Chen et al., 2019a), in which the authors propose the use of an off-policy corrected policy-gradient method. Policy-gradient uses the cumulative reward to re-weight the cross-entropy loss. However, it's a Monte Carlo (MC)-based method which needs the interaction session to end first and then calculate the cumulative rewards at each timestamp. In contrast, SA2C calculates the advantage of an action through the RL output layer, which is a more fine-grained estimate of the value of a potential action during the session. Furthermore, the cumulative reward can also introduce bias from the sequence length, while the advantage estimates in SA2C can be seen as normalized Q-values which help to alleviate this influence. The effect of off-policy correction will be discussed in the experimental section.

## 6.3 Experiments

In this section, we report experiments on two real-world datasets to evaluate the proposed SNQN and SA2C in the e-commerce scenario. Both datasets contain click and purchase interactions. We use the supervised head to generate recommendations without special mention. To support thesis statement (3), we need

[1]Recommendation can be generated from the RL head by selecting highest Q-values.

**Algorithm 5** Training procedure of SA2C

**Input:** the interaction sequence set $\mathcal{X}$, recommendation model $G$, reinforcement head $Q$, supervised head, threshold $T$

**Output:** all parameters in the learning space $\Theta$

Initialize all trainable parameters
Create $G'$ and $Q'$ as copies of $G$ and $Q$, $t = 0$
**repeat**
  Draw a mini-batch of $(x_{1:t}, a_t)$ from $\mathcal{X}$, set rewards $r$
  $\mathbf{s}_t = G(x_{1:t})$, $\mathbf{s}'_t = G'(x_{1:t})$
  $\mathbf{s}_{t+1} = G(x_{1:t+1})$, $\mathbf{s}'_{t+1} = G'(x_{1:t+1})$
  Generate random variable $z \in (0, 1)$ uniformly
  **if** $z \leq 0.5$ **then**
    $a^+_* = \text{argmax}_a\ Q(\mathbf{s}_{t+1}, a)$, $a^-_* = \text{argmax}_a\ Q(\mathbf{s}_t, a)$
    $L_p = (r(\mathbf{s}_t, a^+_t) + \gamma Q'(\mathbf{s}'_{t+1}, a^+_*) - Q(\mathbf{s}_t, a^+_t))^2$
    $L_n = \sum_{a^-_t \in N_t} (r(\mathbf{s}_t, a^-_t) + \gamma Q'(\mathbf{s}'_t, a^-_*) - Q(\mathbf{s}_t, a^-_t))^2$
    Calculate $L_s$ and $L_{snqn} = L_s + L_p + L_n$
    **if** $t \leq T$ **then**
      Perform updates by $\nabla_\Theta L_{snqn}$
    **else**
      Calculate $A(\mathbf{s}_t, a^+_t)$ according to Eq.(6.3) and Eq.(6.4)
      $L_a = L_s \times A(\mathbf{s}_t, a^+_t)$, $L_{sa2c} = L_A + L_s$
      Perform updates by $\nabla_\Theta L_{sa2c}$
    **end if**
  **else**
    $a^+_* = \text{argmax}_a\ Q'(\mathbf{s}_{t+1}, a)$, $a^-_* = \text{argmax}_a\ Q'(\mathbf{s}_t, a)$
    $L_p = (r(\mathbf{s}_t, a^+_t) + \gamma Q(\mathbf{s}_{t+1}, a^+_*) - Q'(\mathbf{s}'_t, a^+_t))^2$
    $L_n = \sum_{a^-_t \in N_t} (r(\mathbf{s}_t, a^-_t) + \gamma Q(\mathbf{s}_t, a^-_*) - Q'(\mathbf{s}'_t, a^-_t))^2$
    Calculate $L_s$ and $L_{snqn} = L_s + L_p + L_n$
    **if** $t \leq T$ **then**
      Perform updates by $\nabla_\Theta L_{snqn}$
    **else**
      Calculate $A(\mathbf{s}_t, a^+_t)$ according to Eq.(6.3) and Eq.(6.4)
      $L_a = L_s \times A(\mathbf{s}_t, a^+_t)$, $L_{sa2c} = L_A + L_s$
      Perform updates by $\nabla_\Theta L_{sa2c}$
    **end if**
  **end if**
  $t = t + 1$
**until** converge
return all parameters in $\Theta$

to see how the proposed learning frameworks perform when they are integrated with existing based models, compared with the normal training and SQN, SAC. Because the SQNQ enables the Q-learning head to generate recommendation, so we also justify the performance when using Q-learning for recommendation. Finally, we show how the hyper-parameter setting affects the model performance. Based on such motivations, in this section we aim to answer the following research questions:

**RQ1:** How do the proposed methods perform when integrated with different base models?

**RQ2** What is the performance if we use the Q-leaning head to generate recommendation?

**RQ3:** What is the performance if we introduce an additional off-policy correction term in the actor of SA2C?

**RQ4:** How does the negative sampling strategy affect the performance?

## 6.3.1 Experimental Settings

### 6.3.1.1 Datasets

We conduct experiments with two publicly accessible datasets: RC15 and Retail-Rocket. The two datasets are the same as the datasets used in Chapter 5. The datasets detail and statistics can be found in section 5.4.1.1 and Table 5.1.

### 6.3.1.2 Evaluation protocols

We use the same cross-validation protocols as Chapter 5. The ratio of training, validation, and test set are 8:1:1. The ranking is performed among the whole item set. Each experiment is repeated 5 times with different random seeds, and the average performance is reported. The recommendation quality is measured with two metrics: hit ratio (HR) and normalized discounted cumulative gain (NDCG). A more detailed procedure can be found in section 5.4.1.2.

### 6.3.1.3 Baselines

We integrated the proposed SNQN and SA2C with the same baseline models in Chapter 5 which are GRU (Hidasi et al., 2015), Caser (Tang and Wang, 2018), NItNet (Yuan et al., 2019) and SASRec (Kang and McAuley, 2018). A more

detailed description of baseline models can be found in section 5.4.1.3. To further demonstrate the effectiveness of the proposed methods, we also compare SNQN, SA2C with SQN, SAC (Xin et al., 2020), respectively.

#### 6.3.1.4 Parameter settings

For both datasets, the input sequences are composed of the last 10 interacted items before the target timestamp. If the sequence length is less than 10, we complement the sequence with a padding item. We train all models with the Adam optimizer (Kingma and Ba, 2014). The mini-batch size is set as 256. For SNQN, the learning rate is set as 0.01 on RC15 and 0.005 on RetailRocket, which is the same setting as SQN (Xin et al., 2020). For SA2C, we use the same learning rate with SNQN at the early pre-training stage. After that, the learning rate is set as 0.001 on both datasets. For a fair comparison, we use the basic uniform distribution to sample negative examples and eliminate influence from the sampler. The item embedding size is set as 64 for all models. For GRU, the size of the hidden state is set as 64. For Caser, we use 1 vertical convolution filter and 16 horizontal filters whose heights are set from {2,3,4}. The drop-out ratio is set as 0.1. For NextItNet, we use the published implementation (Yuan et al., 2019) with the predefined settings. For SASRec, the number of heads in self-attention is set as 1, according to the original paper (Kang and McAuley, 2018). Note that, when SNQN and SA2C are integrated with a base model, the hyper-parameter setting of the base model remains unchanged, for a fair comparison.

For the proposed SNQN and SA2C, the discount factor $\gamma$ is set as 0.5. The ratio between the click reward ($r_c$) and the purchase reward ($r_p$) is set as $r_p/r_c = 5$. These settings are the same as Chapter 5 for a fair comparison. For one positive action, we sample 10 negative actions in the training procedure, if without special mention. The reward for negative actions is set as $r_n = 0$.

### 6.3.2 Performance Comparison (RQ1)

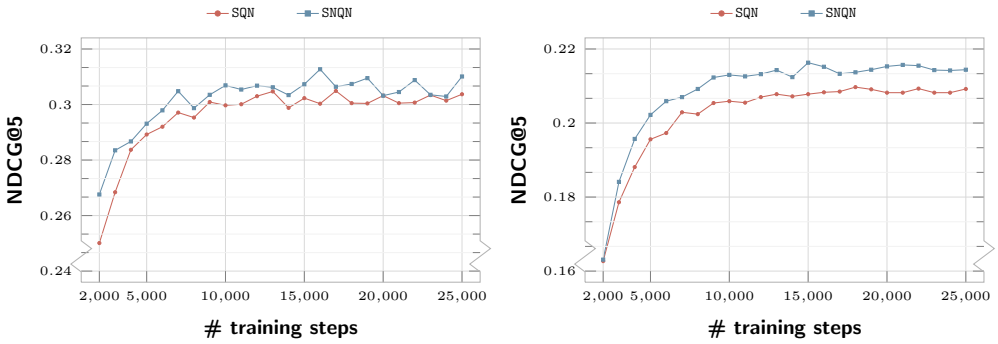
Table 6.1 and Table 6.2 show the performance of top-$N$ recommendations on RC15 for purchase and click prediction, respectively. Table 6.3 and Table 6.4 show the performance of top-$N$ recommendations on RetailRocket. We can obtain the following observations from the results.

Table 6.1: Top-$N$ recommendation performance comparison of different models in Chapter 6 ($N = 5, 10, 20$) on RC15 dataset for purchase prediction. Recommendations are generated from the supervised head. NG is short for NDCG. Boldface denotes the highest score.

| Models | purchase | | | | | |
|---|---|---|---|---|---|---|
| | HR@5 | NG@5 | HR@10 | NG@10 | HR@20 | NG@20 |
| GRU | 0.3994 | 0.2824 | 0.5183 | 0.3204 | 0.6067 | 0.3429 |
| GRU-SQN | 0.4228 | 0.3016 | 0.5333 | 0.3376 | 0.6233 | 0.3605 |
| GRU-SNQN | 0.4368 | 0.3115 | 0.5428 | 0.3460 | 0.6316 | 0.3686 |
| GRU-SAC | 0.4394 | 0.3154 | 0.5525 | 0.3521 | 0.6378 | 0.3739 |
| GRU-SA2C | **0.4514** | **0.3297** | **0.5606** | **0.3652** | **0.6420** | **0.3859** |
| Caser | 0.4475 | 0.3211 | 0.5559 | 0.3565 | 0.6393 | 0.3775 |
| Caser-SQN | 0.4553 | 0.3302 | 0.5637 | 0.3653 | 0.6417 | 0.3862 |
| Caser-SNQN | 0.4781 | 0.3460 | 0.5876 | 0.3816 | 0.6657 | 0.4015 |
| Caser-SAC | 0.4866 | 0.3527 | 0.5914 | 0.3868 | 0.6689 | 0.4065 |
| Caser-SA2C | **0.4917** | **0.3635** | **0.6000** | **0.3989** | **0.6796** | **0.4192** |
| NItNet | 0.3632 | 0.2547 | 0.4716 | 0.2900 | 0.5558 | 0.3114 |
| NItNet-SQN | 0.3845 | 0.2736 | 0.4945 | 0.3094 | 0.5766 | 0.3302 |
| NItNet-SNQN | 0.3969 | 0.2803 | 0.5039 | 0.3152 | 0.5876 | 0.3363 |
| NItNet-SAC | 0.3914 | 0.2813 | 0.4964 | 0.3155 | 0.5763 | 0.3357 |
| NItNet-SA2C | **0.4382** | **0.3171** | **0.5403** | **0.3505** | **0.6259** | **0.3722** |
| SASRec | 0.4228 | 0.2938 | 0.5418 | 0.3326 | 0.6329 | 0.3558 |
| SASRec-SQN | 0.4336 | 0.3067 | 0.5505 | 0.3435 | 0.6442 | 0.3674 |
| SASRec-SNQN | 0.4435 | 0.3163 | 0.5581 | 0.3535 | 0.6450 | 0.3742 |
| SASRec-SAC | 0.4540 | 0.3246 | 0.5701 | 0.3623 | 0.6576 | 0.3846 |
| SASRec-SA2C | **0.4705** | **0.3385** | **0.5756** | **0.3728** | **0.6648** | **0.3956** |

Table 6.2: Top-$N$ recommendation performance comparison of different models ($N = 5, 10, 20$) in Chapter 6 on RC15 dataset for click prediction. Recommendations are generated from the supervised head. NG is short for NDCG. Boldface denotes the highest score.

| Models | click | | | | | |
|---|---|---|---|---|---|---|
| | HR@5 | NG@5 | HR@10 | NG@10 | HR@20 | NG@20 |
| GRU | 0.2876 | 0.1982 | 0.3793 | 0.2279 | 0.4581 | 0.2478 |
| GRU-SQN | 0.3020 | 0.2093 | 0.3946 | 0.2394 | 0.4741 | 0.2587 |
| GRU-SNQN | 0.3124 | 0.2164 | 0.4067 | 0.2469 | 0.4856 | 0.2669 |
| GRU-SAC | 0.2863 | 0.1985 | 0.3764 | 0.2277 | 0.4541 | 0.2474 |
| GRU-SA2C | **0.3287** | **0.2307** | **0.4214** | **0.2606** | **0.5000** | **0.2806** |
| Caser | 0.2728 | 0.1896 | 0.3593 | 0.2177 | 0.4371 | 0.2372 |
| Caser-SQN | 0.2742 | 0.1909 | 0.3613 | 0.2192 | 0.4381 | 0.2386 |
| Caser-SNQN | 0.2800 | 0.1951 | 0.3682 | 0.2237 | 0.4465 | 0.2436 |
| Caser-SAC | 0.2726 | 0.1894 | 0.3580 | 0.2171 | 0.4340 | 0.2362 |
| Caser-SA2C | **0.2948** | **0.2068** | **0.3835** | **0.2356** | **0.4596** | **0.2549** |
| NItNet | 0.2950 | 0.2030 | 0.3885 | 0.2332 | 0.4684 | 0.2535 |
| NItNet-SQN | 0.3091 | 0.2137 | 0.4037 | 0.2442 | 0.4835 | 0.2645 |
| NItNet-SNQN | 0.3153 | 0.2176 | 0.4098 | 0.2482 | 0.4896 | 0.2686 |
| NItNet-SAC | 0.2977 | 0.2055 | 0.3906 | 0.2357 | 0.4693 | 0.2557 |
| NItNet-SA2C | **0.3410** | **0.2395** | **0.4348** | **0.2699** | **0.5113** | **0.2897** |
| SASRec | 0.3187 | 0.2200 | 0.4164 | 0.2515 | 0.4974 | 0.2720 |
| SASRec-SQN | 0.3272 | 0.2263 | 0.4255 | 0.2580 | 0.5066 | 0.2786 |
| SASRec-SNQN | 0.3284 | 0.2267 | 0.4271 | 0.2588 | 0.5083 | 0.2794 |
| SASRec-SAC | 0.3130 | 0.2161 | 0.4114 | 0.2480 | 0.4945 | 0.2691 |
| SASRec-SA2C | **0.3444** | **0.2407** | **0.4402** | **0.2719** | **0.5194** | **0.2920** |




(a) Purchase prediction. (b) Purchase prediction.

Figure 6.2: Model convergence of SNQN on RC15.

(1) On both datasets, the proposed SNQN achieves consistently better performance than the supervised base model and SQN, on both click and purchase recommendations. This demonstrates that the introduced negative sampling strategy on the RL head does improve the learning performance also on the supervised component. This can be attributed to the shared recommendation model $G(\cdot)$ between the supervised part and the RL part. We also observed in our experiments that SNQN achieves faster convergence than SQN. Figure 6.2 shows the comparison between model convergence under the same learning rate on the validation set of RC15, using GRU as the base model $G(\cdot)$. This further demonstrates that the introduced negative sampling helps the model to learn faster and improves its performance.

(2) SA2C achieves better performance than SAC in most cases (except for some HR values in RetailRocket with Caser as the base model). This demonstrates that the advantage estimate used in SA2C is a more effective critic compared with the raw Q-values used in SAC. This can be attributed to the fact that the advantage estimation helps to alleviate the sequence length bias.

(3) SA2C achieves the best performance in most cases, except for a few values of HR. However, SA2C always achieves the highest NDCG, which demonstrates that SA2C is more effective in pushing good actions (recommended items) to top-ranking positions. This is due to the fact that positive actions are weighted (advantaged) in a more effective manner during the training procedure of SA2C.

Table 6.3: Top-$N$ recommendation performance comparison of different models ($N = 5, 10, 20$) in Chapter 6 on RetailRocket for purchase prediction. Recommendations are generated from the supervised head. NG is short for NDCG. Boldface denotes the highest score.

| Models | purchase | | | | | |
|---|---|---|---|---|---|---|
| | HR@5 | NG@5 | HR@10 | NG@10 | HR@20 | NG@20 |
| GRU | 0.4608 | 0.3834 | 0.5107 | 0.3995 | 0.5564 | 0.4111 |
| GRU-SQN | 0.5069 | 0.4130 | 0.5589 | 0.4289 | 0.5946 | 0.4392 |
| GRU-SNQN | 0.5232 | 0.4376 | 0.5713 | 0.4544 | 0.6175 | 0.4650 |
| GRU-SAC | 0.4942 | 0.4179 | 0.5464 | 0.4341 | 0.5870 | 0.4428 |
| GRU-SA2C | **0.5526** | **0.4754** | **0.5963** | **0.4897** | **0.6313** | **0.4985** |
| Caser | 0.3491 | 0.2935 | 0.3857 | 0.3053 | 0.4198 | 0.3141 |
| Caser-SQN | 0.3674 | 0.3089 | 0.4050 | 0.3210 | 0.4409 | 0.3301 |
| Caser-SNQN | 0.3757 | 0.3179 | 0.4181 | 0.3317 | 0.4595 | 0.3422 |
| Caser-SAC | 0.3871 | 0.3234 | 0.4336 | 0.3386 | **0.4763** | 0.3494 |
| Caser-SA2C | **0.3971** | **0.3446** | **0.4381** | **0.3578** | 0.4733 | **0.3667** |
| NItNet | 0.5630 | 0.4630 | 0.6127 | 0.4792 | 0.6477 | 0.4881 |
| NItNet-SQN | 0.5895 | 0.4860 | 0.6403 | 0.5026 | 0.6766 | 0.5118 |
| NItNet-SNQN | 0.6016 | 0.5062 | 0.6543 | 0.5234 | **0.6921** | 0.5330 |
| NItNet-SAC | 0.5895 | 0.4985 | 0.6358 | 0.5162 | 0.6657 | 0.5243 |
| NItNet-SA2C | **0.6226** | **0.5422** | **0.6573** | **0.5534** | 0.6842 | **0.5603** |
| SASRec | 0.5267 | 0.4298 | 0.5916 | 0.4510 | 0.6341 | 0.4618 |
| SASRec-SQN | 0.5681 | 0.4617 | 0.6203 | 0.4806 | 0.6619 | 0.4914 |
| SASRec-SNQN | 0.5776 | 0.4846 | 0.6310 | 0.5020 | 0.6719 | 0.5123 |
| SASRec-SAC | 0.5623 | 0.4679 | 0.6127 | 0.4844 | 0.6505 | 0.4940 |
| SASRec-SA2C | **0.5929** | **0.5080** | **0.6437** | **0.5246** | **0.6798** | **0.5337** |

Table 6.4: Top-$N$ recommendation performance comparison of different models ($N = 5, 10, 20$) in Chapter 6 on RetailRocket for click prediction. Recommendations are generated from the supervised head. NG is short for NDCG. Boldface denotes the highest score.

| Models | click | | | | | |
|---|---|---|---|---|---|---|
| | HR@5 | NG@5 | HR@10 | NG@10 | HR@20 | NG@20 |
| GRU | 0.2233 | 0.1735 | 0.2673 | 0.1878 | 0.3082 | 0.1981 |
| GRU-SQN | 0.2487 | 0.1939 | 0.2967 | 0.2094 | 0.3406 | 0.2205 |
| GRU-SNQN | 0.2662 | 0.2065 | 0.3181 | 0.2233 | **0.3656** | 0.2353 |
| GRU-SAC | 0.2451 | 0.1924 | 0.2930 | 0.2074 | 0.3371 | 0.2186 |
| GRU-SA2C | **0.2720** | **0.2150** | **0.3208** | **0.2308** | **0.3656** | **0.2422** |
| Caser | 0.1966 | 0.1566 | 0.2302 | 0.1675 | 0.2628 | 0.1758 |
| Caser-SQN | 0.2089 | 0.1661 | 0.2454 | 0.1778 | 0.2803 | 0.1867 |
| Caser-SNQN | 0.2160 | 0.1721 | 0.2530 | 0.1841 | 0.2895 | 0.1934 |
| Caser-SAC | **0.2206** | 0.1732 | **0.2617** | 0.1865 | **0.2999** | 0.1961 |
| Caser-SA2C | 0.2170 | **0.1759** | 0.2528 | **0.1875** | 0.2873 | **0.1963** |
| NItNet | 0.2495 | 0.1906 | 0.2990 | 0.2067 | 0.3419 | 0.2175 |
| NItNet-SQN | 0.2610 | 0.1982 | 0.3129 | 0.2150 | 0.3586 | 0.2266 |
| NItNet-SNQN | 0.2699 | 0.2065 | 0.3236 | 0.2240 | 0.3703 | 0.2358 |
| NItNet-SAC | 0.2529 | 0.1964 | 0.3010 | 0.2119 | 0.3458 | 0.2233 |
| NItNet-SA2C | **0.2787** | **0.2197** | **0.3271** | **0.2354** | **0.3719** | **0.2468** |
| SASRec | 0.2541 | 0.1931 | 0.3085 | 0.2107 | 0.3570 | 0.2230 |
| SASRec-SQN | 0.2761 | 0.2104 | 0.3302 | 0.2279 | 0.3803 | 0.2406 |
| SASRec-SNQN | 0.2815 | 0.2171 | 0.3381 | 0.2355 | 0.3888 | 0.2483 |
| SASRec-SAC | 0.2670 | 0.2056 | 0.3208 | 0.2230 | 0.3701 | 0.2355 |
| SASRec-SA2C | **0.2873** | **0.2242** | **0.3409** | **0.2416** | **0.3893** | **0.2538** |

Table 6.5: Recommendation performance from the RL head. NG is short for NDCG. Boldface denotes the highest score. DQN denotes only a Q-learning head is stacked upon the base model without any supervised head.

| | Methods | purchase | | click | |
|---|---|---|---|---|---|
| | | HR@5 | NG@5 | HR@5 | NG@5 |
| RC15 | DQN | 0.3642 | 0.2476 | 0.2096 | 0.1353 |
| | SNQN | **0.3698** | **0.2497** | **0.2286** | **0.1495** |
| Retail Rocket | DQN | 0.2952 | 0.2204 | 0.1368 | 0.0961 |
| | SNQN | **0.3124** | **0.2422** | **0.1546** | **0.1103** |

To conclude, the proposed SNQN and SA2C introduce significant improvements compared to existing methods, especially SA2C which provides the best performance in most cases.

### 6.3.3 Recommendation from Q-learning (RQ2)

Table 6.5 shows the performance comparison when we use the Q-learning head to generate recommendations. We compare the performance of SNQN with a simple double Q-learning (DQN) algorithm with the same negative sampling strategy but without a supervised head upon the base model. The performance of SA2C is not significantly different from SNQN as the two methods are essentially identical with regards to the Q-learning head. We use the same base model GRU and the same hyper-parameters for DQN and SNQN. We observe that SNQN achieves better performance than DQN in all evaluation metrics on both purchase and click predictions. Combined with the results in section 6.3.2, we observe that joint training of supervised learning and RL with shared base models helps to improve the performance of each component. Based on this finding, we believe that transfer learning between self-supervised learning and RL would be a promising research direction. RL makes supervised models be more reward-driven, while supervised learning improves the data efficiency of RL. This observation provides strong support for the thesis statement (3).

### 6.3.4 Effect of Off-Policy Correction (RQ3)

Chen et al. (2019a) introduced an off-policy correction term (propensity score) for the policy-gradient method. The propensity score is defined as $\rho = \frac{\pi_\theta(a|\mathbf{s})}{\beta(a|\mathbf{s})}$. In this subsection, we investigate the effect of this propensity score when introduced into the actor component of SA2C. In that case, the training loss of the actor becomes:

$$L_{a-off} = L_s \cdot A(\mathbf{s}_t, a_t^+) \cdot \rho. \tag{6.6}$$

We also introduce another NDCG-based off-policy corrected evaluation metric (Vlassis et al., 2019) which is formulated as

$$NG_{off} = \frac{\sum \frac{NDCG}{\beta}}{\sum \frac{1}{\beta}}. \tag{6.7}$$

In this implementation, we use the item frequency to approximate the behavior policy $\beta$, which is also adopted in Strehl et al. (2010). Table 6.6 shows the result when generating top-10 recommendations with GRU as the base model. We note the following observations:

(1) On the standard evaluation metric NDCG, the off-policy correction doesn't improve the score. The reason for this is that the normal NDCG is actually defined on non-corrected data, so the non-corrected actor performs better at this evaluation metric.

(2) On the metric $NG_{off}$, the off-policy correction helps the model to achieve better performance for click predictions but not for purchases. The reason for this is that clicks account for the biggest part of the dataset. Hence the off-policy correction term is actually better defined to correct the click data, leading to a better performance of $NG_{off}$ for clicks, while the high variance of the off-policy correction for the small portion of purchase data leads to less of an improvement. This observation suggests that perhaps we should design different correction terms for different kinds of interactions.

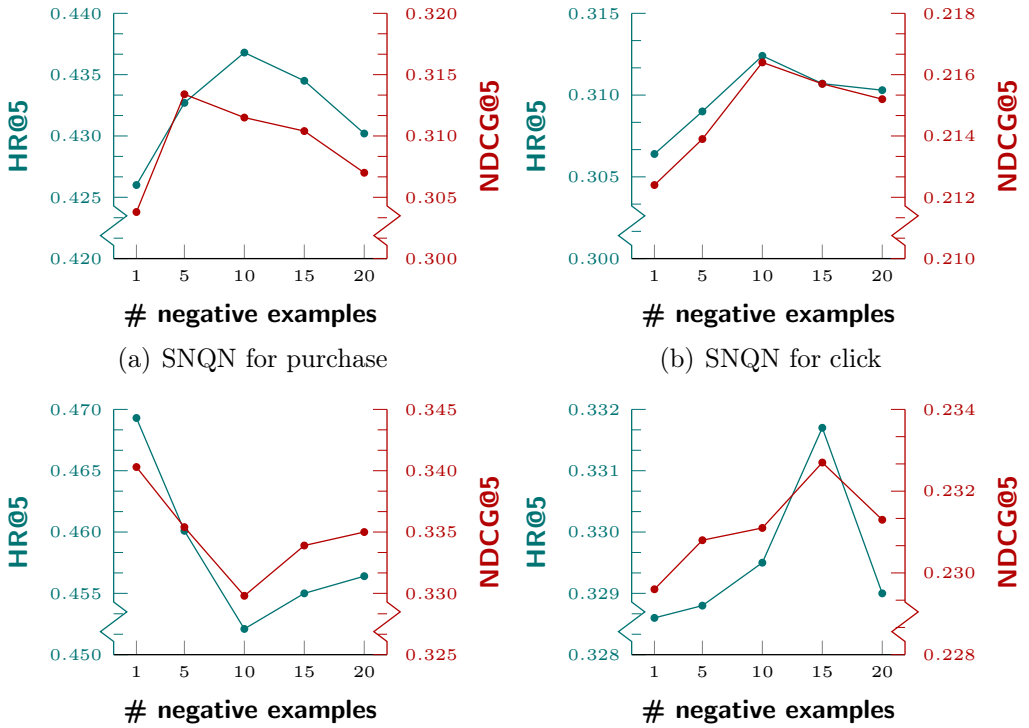
In our experiment, we found that computing the off-policy correction term involves a lot of normalization techniques (e.g., clipping and smoothing) (Chen et al., 2019a) and that the behavior policy $\beta$ can also be a long-tail distribution (Strehl et al., 2010). This introduces substantial noise and high variance into the training procedure. Designing more effective and stable off-policy correction terms remains an open research problem.

Table 6.6: Effect of off-policy correction. w/o means without off-policy correction in the actor while w means the opposite. Boldface denotes the highest score.

| | Methods | purchase | | click | |
|---|---|---|---|---|---|
| | | NDCG | $NG_{off}$ | NDCG | $NG_{off}$ |
| RC15 | w/o | **0.3652** | **0.1077** | **0.2606** | 0.0767 |
| | w | 0.3551 | 0.1064 | 0.2595 | **0.0781** |
| Retail Rocket | w/o | **0.4897** | **0.2171** | **0.2308** | 0.0861 |
| | w | 0.4771 | 0.2147 | 0.2238 | **0.0872** |

### 6.3.5 Hyperparameter Study (RQ4)

In this section, we conduct a series of experiments to demonstrate the effect of negative sampling on recommendation performance. Figure 6.3 and Figure 6.4 show the recommendation accuracy with different sizes of negative examples (i.e. $|N_t|$) on RC15 and RetailRocket, respectively (the base model is GRU). Here, it is evident that, on both click and purchase predictions, the recommendation performance initially increases and then decreases (except in Figure 6.3(c)). When more negative actions are introduced, the model has more data to learn from. By introducing negative actions, the model does not only learn that actions leading to purchases are better than actions leading to clicks but also learns to draw a contrast between negative and positive actions. Increasing the sample size means that the model can have access to more diverse negative signals and, thus, leads to better performance. However, a very large negative sample size may bias the model to negative values and degrade performance. Regarding Figure 6.3(c), we observe that the model also achieves good performance with small sample sizes. The reason could be attributed to that a small sample size would introduce more noise into the estimation of the advantage. This noise may help the model to find a better local optimal with higher performance but also requires more update steps to converge. We have observed in our experiments that SA2C needs more iterations to converge when the sample size is small.



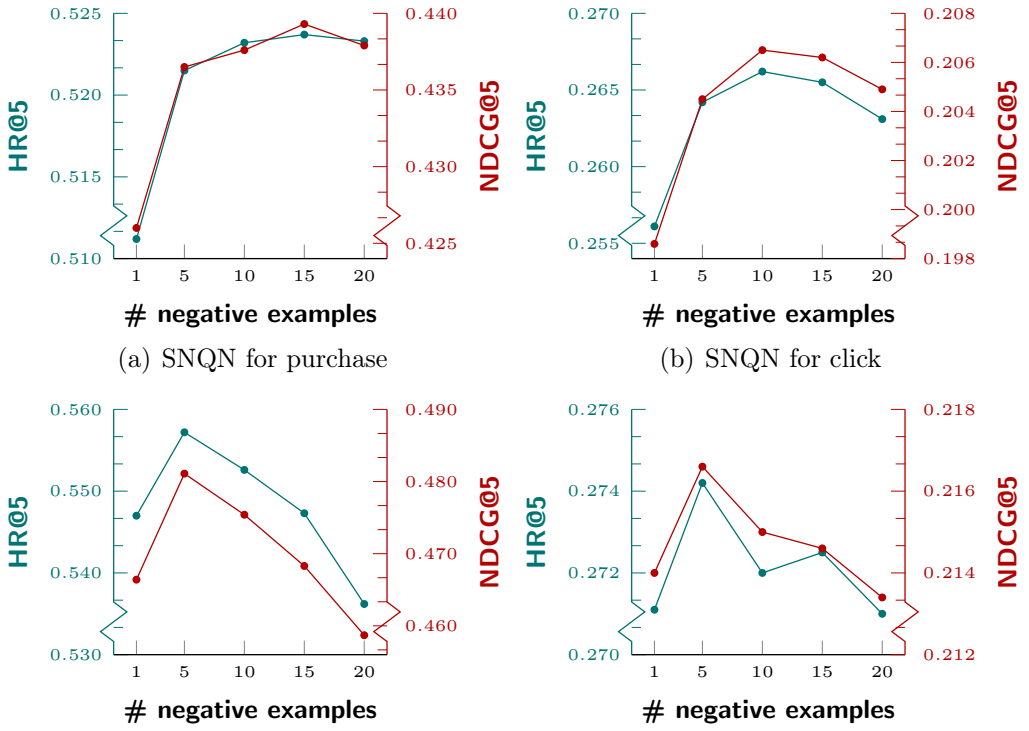

(a) SNQN for purchase

(b) SNQN for click

(c) SA2C for purchase

(d) SA2C for click

Figure 6.3: Effect of negative samples on RC15




(a) SNQN for purchase

(b) SNQN for click

(c) SA2C for purchase

(d) SA2C for click

Figure 6.4: Effect of negative samples on Kaggle

## 6.4 Chapter Summary

In this chapter, we propose two learning frameworks (SNQN and SA2C) to explore the usage of RL under recommendation settings. SNQN combines supervised learning and RL with the shared base model and introduces negative sampling into the RL training procedure. The explicitly introduced negative comparison signals help the RL output layer to perform a good ranking. Based on the sampled actions, SA2C first computes the advantage of actions which can be seen as normalized Q-values, and then use this advantage estimate as a critic to re-weight the actor. To verify the effectiveness of our methods, we integrate them into four state-of-the-art recommendation models and conduct experiments on two real-world e-commerce datasets. Our experimental findings demonstrate that the proposed SNQN and SA2C are effective in further improving the recommendation performance, compared to existing self-supervised RL methods. The experimental results provide sufficient support for our thesis statement (3).

Generally speaking, we hold the belief that combining supervised learning and reinforcement learning is a promising direction for off-line (off-policy) learning solutions. Supervised learning can provide knowledge for representation learning, which can help to reduce the sample complexity of RL. On the other hand, RL can introduce the desired reward expectations to supervised learning. Recently, Agarwal et al. (2020) also obtained similar observations.

# Part IV
# Conclusion

In the previous chapters of this thesis, we have developed deep learning-based recommendation methods for implicit feedback data from two perspectives: supervised learning and reinforcement learning. For supervised learning, we proposed two deep recommendation models, namely CFM and RCF which tackle the implicit feedback data as feature vectors and relations, correspondingly. Both the proposed models have shown improved recommendation accuracy compared with related baselines. This demonstrates that a good use case of deep learning techniques is effective to improve the recommendation accuracy. Besides, RCF also shows the advantage to generate more explainable recommendation. For reinforcement learning, we cast the recommendation task in one interaction session as a reinforcement learning problem and analyzed the challenges in that procedure. We then proposed self-supervised reinforcement learning frameworks to address the challenges. We show that combining supervised learning and reinforcement learning is a promising direction for future recommendation agents.

# Chapter 7

# Conclusion and Future Work

## 7.1 Conclusion

In this thesis, we focus on utilizing advanced deep learning techniques for top-$k$ recommendation on implicit feedback data. The recommendation quality is measured by the recommendation accuracy. The proposed recommendation methods focus on perspectives of both conventional supervised learning and reinforcement learning.

Firstly, from the perspective of supervised learning, we developed deep learning models for implicit feedback data with side information. To better describe user-item interactions, besides the user ID and item ID, a lot of data also contains rich side information, such as item attributes, location, and daytime of the interaction. A concise solution to represent this kind of user-item interaction is to use one/multi-hot feature vectors to encode the IDs and essential information. Based on such feature vectors, previous models (Rendle, 2010) are limited to their linearity and model fidelity to capture high-order feature interaction signals. This motivates the proposed CFM (Xin et al., 2019a) model in Chapter 3. CFM utilizes the outer product between feature embeddings to represent the interaction of the two features. Then it stacks the pair-wise feature interactions to form an interaction cube and utilizes convolutional neural networks to learn high-order interaction signals. A pooling layer is also proposed to reduce the computational cost. Experimental results demonstrated that CFM achieves better recommendation accuracy compared with related baselines.

Given the rich side information, we can also infer multiple item relations. For example, two movies may share the same director, two products may comple-

ment each other. Compared with the collaborative signal which is indicated by the co-interaction patterns, item relations inferred from side information are fine-grained and contain more semantic meaning. Incorporating such item relations may help to improve both the recommendation accuracy and recommendation interpretability. Based on such motivation, we can re-organize the implicit feedback data as relational data. This leads to the proposed RCF (Xin et al., 2019b) model in Chapter 4. In RCF, we define item relations as tuples consisting of relation types and relation values. Relation types describe high-level relations such as shared genres between movies while relation values depict more fine-grained detail of which genres do they share. Then RCF utilizes two attention networks to compute the importance of relation types and relation values for user decision making. Experimental results on movie and music recommendation demonstrate that RCF cannot only improve the recommendation accuracy but also help to better understand the user behavior patterns and thus increase the interpretability of recommendation.

The above sections focus on developing supervised deep learning models for recommendation, through utilizing the implicit feedback data in different forms: feature vectors and relations. The experimental results show that deep learning methods can not only help to improve the recommendation accuracy but also provide more explainable recommendation. This observation supports the thesis statement (1) and statement (2).

However, there could be factors that can not be appropriately modeled by simple supervised learning, such as long-term user engagement in one interaction session and different recommendation objectives. This motivates the utilize of reinforcement learning for recommendation.

Regarding reinforcement learning, we first formulated the session-based recommendation as a reinforcement learning task and gave the definition of the corresponding MDP. We then analyzed the challenges to applying reinforcement learning algorithms for implicit-feedback based recommendation. The major challenges come from the pure off-policy setting and the lack of negative feedback. To address the problem, we proposed two self-supervised reinforcement learning algorithms namely SQN and SAC for recommendation in Chapter 5. The main idea is to utilize reinforcement learning as a regularizer to introduce the desired

long-term reward to supervised learning and conduct transfer learning between the two components. Based on SQN and SAC, we further developed SNQN and SA2C in Chapter 6 to introduce the negative sampling strategy. In that case, the reinforcement learning component is not just a regularizer but also a good ranking player which can also be used to generate recommendation. We integrated the proposed learning frameworks with different deep recommendation models and conducted experiments on two real-world e-commerce implicit feedback datasets. Experimental results demonstrated the effectiveness of the proposed methods, which provides support for the thesis statement (3).

## 7.2 Future Work

In this section, we present several promising research directions for the deep learning-based recommendation.

- Residual learning for recommendation: Recently, residual learning with skip connections on convolutional neural networks has achieved great success in the field of computer vision (He et al., 2016b). The introduced skip connections help to alleviate the gradient issue of deep networks. With residual skip connection, ResNet (He et al., 2016a) can achieve an extreme deep architecture of hundreds of layers with high performance. We are also interested in introducing residual learning in the proposed CFM model. It may help to improve the model fidelity and further improve the recommendation accuracy. However, it may also introduce more computational costs. How to improve the model performance and meanwhile keep high inference efficiency is an interesting research direction.

- Multiple user relationships could further improve the proposed RCF model: The proposed RCF model introduces item relations into recommendation. This is motivated by item-based collaborative filtering. However, besides the item relations, there are also multiple user relations in the real-world scenario. Exploiting social relations have been verified to be effective to improve the recommendation performance (Ma et al., 2011; Tang et al., 2013, 2016; Guo et al., 2015). Compared with item relations, user relations

are more complex and dynamic. How to incorporate user relations in the framework of RCF is also a promising direction. When both item relations and user relations are integrated, the user-item interactions can be represented as graphs whose nodes are users and items, links are interactions and relations. We can then utilize graph neural networks for recommendation. Besides, incorporating context information into the learning process and then formulate the context-aware user-item relation graph learning could be interesting. We have done some attempts in supporting publications 8 and 9. Because the first author of these publications is not me, so they are not included in the main content of this thesis.

- Graphs and reinforcement learning for better negative sampler: Under the setting of implicit feedback, plenty of state-of-the-art recommendation models utilize the negative sampling strategy to sample negative examples for pair-wise comparison. Research (Chen et al., 2018; Yuan et al., 2016; Zhang et al., 2013) has shown that a well-designed negative sampler can further improve the model performance compared with the simple uniform sampler. The major idea is to select hard and diverse negative examples for pair-wise comparison. When the implicit feedback data is presented as interaction graphs, the topology information of the graph can help us to infer the hard training examples. On the other hand, reinforcement learning also provides solutions for the design of a negative sampler. For example, we can regard the selection of negative examples as the action of the reinforcement learning problem and the gradient we get as the reward. By solving this reinforcement learning problem, we aim to get the maximum cumulative gradient updates. It conforms to the analysis in Rendle and Freudenthaler (2014).

- Reinforcement learning as a teacher model: As discussed in Chapter 5, directly utilizing reinforcement learning for recommendation is infeasible due to the pure off-policy setting. Another solution is to use reinforcement learning as a teacher model and supervised learning as a student model. For example, in the training procedure, the supervised model (i.e., the student) encodes the input into a hidden state, then the reinforcement learning model

(i.e., the teacher) can update this hidden state to a new state. Then we can generate two recommendation lists according to the above two hidden states and the difference between them can be seen as the reward. In that case, the teacher is trained to update the student's outcome to fit the defined reward. What's more, because now the reinforcement learning is not directly utilized to generate recommendation, we can avoid the off-policy challenge. We have done some initial attempts on supporting publication 7 regarding this topic.

- Reinforcement learning for re-ranking: In an industry recommender system, there are usually multiple recommendation models. Each model can generate a recommendation list and the final output is decided by a re-ranking procedure. Utilizing reinforcement learning for this re-ranking procedure is also a promising direction. Firstly, reward-driven reinforcement learning provides more flexibility to customize the final recommendation list according to the specific demands. Secondly, the re-ranking scenario provides the potential possibility to perform on-policy learning. Because in the re-ranking procedure, the candidate items have already been selected according to the user preference. In that case, performing on-policy learning may not affect the user experience too much. It means we can afford the price to make the reinforcement learning agents interact with the environment (i.e., the users) and observe the rewards.

## 7.3 Closing Remarks

Recommender systems play an important role in Web life to solve the problem of information overload. It can benefit both service providers by increasing traffic and users by providing personalized items. Implicit feedback is one of the most common data used to train an effective recommender agent. In recent years, deep learning methods have achieved great success in a variety of fields due to their high model fidelity and expressiveness. In this thesis, we conducted research to develop deep learning-based methods for implicit feedback recommendation. We focus on both conventional supervised learning and reinforcement learning. For supervised learning, we developed two deep models to tackle implicit feedback

as different data structures. For reinforcement learning, we analyzed the specific challenges encountered in the field of recommendation and then provided solutions by combing supervised learning and reinforcement learning. Generally speaking, supervised learning methods focus on imitating the historical records and optimizes the immediate feedback while reinforcement learning is more reward-driven and optimizes the cumulative gains in a whole interaction session. We believe that combing supervised learning and reinforcement learning is a promising direction for future recommendation agents. In that case, supervised learning helps to alleviate the data-hungry problem of reinforcement learning while reinforcement learning introduces reward-driven properties and long-term optimization perspectives into supervised learning.

# References


Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749, 2005. 10, 12

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, 2020. 136

Charu C Aggarwal et al. *Recommender systems*, volume 1. Springer, 2016. 19

Qingyao Ai, Vahid Azizi, Xu Chen, and Yongfeng Zhang. Learning heterogeneous knowledge base embeddings for explainable recommendation. *Algorithms*, 11 (137), 2018. 74

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. 30, 63

Linas Baltrunas, Karen Church, Alexandros Karatzoglou, and Nuria Oliver. Frappe: Understanding the usage and perception of mobile app recommendations in-the-wild. *arXiv preprint arXiv:1505.03014*, 2015. 53

Immanuel Bayer, Xiangnan He, Bhargav Kanagal, and Steffen Rendle. A generic coordinate descent framework for learning from implicit feedback. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1341–1350. International World Wide Web Conferences Steering Committee, 2017. 16, 43

Nicholas J Belkin and W Bruce Croft. Information filtering and information retrieval: Two sides of the same coin? *Communications of the ACM*, 35(12): 29–38, 1992. 2

Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966. 36, 90

Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017. xi, 31, 32, 33

Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. Latent cross: Making use of context in recurrent recommender systems. In *CIKM*, pages 46–54. ACM, 2018. 44

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013. 70, 73

Keith Bradley and Barry Smyth. Improving recommendation diversity. In *Proceedings of the Twelfth Irish Conference on Artificial Intelligence and Cognitive Science, Maynooth, Ireland*, pages 85–94. Citeseer, 2001. 94

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. 5

Yixin Cao, Xiang Wang, Xiangnan He, Tat-Seng Chua, et al. Unifying knowledge graph learning and recommendation: Towards a better understanding of user preferences. In *Proceedings of the 30th international conference on World Wide Web*. ACM, 2019. 74

Pablo Castells, Saúl Vargas, and Jun Wang. Novelty and diversity metrics for recommender systems: choice, discovery and relevance. 2011. http://ir.ii.uam.es/rim3/publications/ddr11.pdf. 18

Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 335–344. ACM, 2017. 30, 76

Long Chen, Fajie Yuan, Joemon M Jose, and Weinan Zhang. Improving negative sampling for word representation using self-embedded features. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 99–107, 2018. 14, 141

Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H Chi. Top-k off-policy correction for a reinforce recommender system. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 456–464. ACM, 2019a. 39, 89, 90, 95, 122, 132

Tianqi Chen, Weinan Zhang, Qiuxia Lu, Kailong Chen, Zhao Zheng, and Yong Yu. Svdfeature: a toolkit for feature-based collaborative filtering. *Journal of Machine Learning Research*, 13(Dec):3619–3622, 2012. 24, 52

Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. Generative adversarial user model for reinforcement learning based recommendation system. In *International Conference on Machine Learning*, pages 1052–1061, 2019b. 39, 91, 119

Chen Cheng, Haiqin Yang, Michael R Lyu, and Irwin King. Where you like to go next: Successive point-of-interest recommendation. In *International Joint Conference on Artificial Intelligence*, 2013. 31

Chen Cheng, Fen Xia, Tong Zhang, Irwin King, and Michael R Lyu. Gradient boosting factorization machines. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 265–272. ACM, 2014. 25

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st*

*Workshop on Deep Learning for Recommender Systems*, pages 7–10. ACM, 2016. 29, 43

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014. 31

Evangelia Christakopoulou and George Karypis. Local latent space models for top-n recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1235–1243. ACM, 2018. 23

Alberto Costa and Fabio Roda. Recommender systems by means of information retrieval. In *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, page 57. ACM, 2011. 2

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 191–198. ACM, 2016. 62

Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46. ACM, 2010. 25

Luis M De Campos, Juan M Fernández-Luna, Juan F Huete, and Miguel A Rueda-Morales. Combining content-based and collaborative recommendations: A hybrid approach based on bayesian networks. *International Journal of Approximate Reasoning*, 51(7):785–799, 2010. 13

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 5

Xiaoyu Du, Xiangnan He, Fajie Yuan, Jinhui Tang, Zhiguang Qin, and Tat-Seng Chua. Modeling embedding dimension correlations via convolutional neural collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 37 (4):1–22, 2019. 30, 49

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011. 55, 78

Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *Proceedings of the 27th International Conference on World Wide Web*, pages 1775–1784. ACM, 2018. 62

Asmaa Elbadrawy and George Karypis. User-specific feature-based similarity models for top-n recommendation of new items. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):33, 2015. 23

Christoph Freudenthaler, Lars Schmidt-Thieme, and Steffen Rendle. Bayesian factorization machines. In *NIPS-WS*, 2011. 25

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*, 2018. 38, 39, 96

Anupriya Gogna and Angshul Majumdar. A comprehensive recommender system model: Improving accuracy for both warm and cold start users. *IEEE Access*, 3:2803–2813, 2015. 22

Yu Gong, Yu Zhu, Lu Duan, Qingwen Liu, Ziyu Guan, Fei Sun, Wenwu Ou, and Kenny Q Zhu. Exact-k recommendation via maximal clique optimization. *arXiv preprint arXiv:1905.07089*, 2019. 39

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 91

Guibing Guo, Jie Zhang, and Neil Yorke-Smith. Trustsvd: Collaborative filtering with both the explicit and implicit influence of user trust and of item ratings. In *Aaai*, volume 15, pages 123–125, 2015. 140

Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for ctr prediction. In *IJCAI*, 2017. 29, 42, 54, 55

Weiyu Guo, Shu Wu, Liang Wang, and Tieniu Tan. Personalized ranking with pairwise factorization machines. *Neurocomputing*, 214:191–200, 2016. 25

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018. 38

Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947, 2000. 50

F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (TIIS)*, 5(4):19, 2016. 75

Hado V Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010. 36, 91, 100, 119

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016a. 5, 140

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016b. 5, 26, 140

Ruining He and Julian McAuley. Fusing similarity models with markov chains for sparse sequential recommendation. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 191–200. IEEE, 2016a. 31

Ruining He and Julian McAuley. Vbpr: Visual bayesian personalized ranking from implicit feedback. In *AAAI*, pages 144–150, 2016b. 29

Ruining He, Chen Fang, Zhaowen Wang, and Julian McAuley. Vista: A visually, socially, and temporally-aware model for artistic recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 309–316. ACM, 2016c. 31

Ruining He, Wang-Cheng Kang, and Julian McAuley. Translation-based recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 161–169. ACM, 2017a. 73

Xiangnan He and Tat-Seng Chua. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 355–364, 2017. 5, 42, 43, 51, 54, 55, 77, 78

Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 549–558. ACM, 2016d. 16

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on Wold Wide Web*, pages 173–182. International World Wide Web Conferences Steering Committee, 2017b. 4, 5, 26, 27, 54

Xiangnan He, Xiaoyu Du, Xiuli Wang, Feng Tian, Jinhui Tang, and Tat-Seng Chua. Outer product-based neural collaborative filtering. In *IJCAI*, 2018a. 29, 30, 49, 52, 54

Xiangnan He, Zhankui He, Xiaoyu Du, and Tat-Seng Chua. Adversarial personalized ranking for recommendation. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 355–364. ACM, 2018b. 23

Xiangnan He, Zhankui He, Jingkuan Song, Zhenguang Liu, Yu-Gang Jiang, and Tat-Seng Chua. Nais: Neural attentive item similarity model for recommendation. *TKDE*, 30(12):2354–2366, 2018c. 11, 23, 30, 54, 62, 68, 69, 73, 76, 77

Balázs Hidasi and Alexandros Karatzoglou. Recurrent neural networks with top-k gains for session-based recommendations. *arXiv preprint arXiv:1706.03847*, 2017. 101

Balázs Hidasi and Domonkos Tikk. General factorization framework for context-aware recommendations. *Data Mining and Knowledge Discovery*, 30(2):342–371, 2016. 31

Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015. 31, 89, 91, 93, 104, 124

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016. 40

Jonathan Ho, Jayesh Gupta, and Stefano Ermon. Model-free imitation learning with policy optimization. In *International Conference on Machine Learning*, pages 2760–2769, 2016. 40

Binbin Hu, Chuan Shi, Wayne Xin Zhao, and Philip S Yu. 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*, pages 1531–1540. ACM, 2018a. 74

Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. Ieee, 2008. 16

Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 368–377. ACM, 2018b. 89

Jin Huang, Wayne Xin Zhao, Hongjian Dou, Ji-Rong Wen, and Edward Y Chang. Improving sequential recommendation with knowledge-enhanced memory networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 505–514. ACM, 2018. 74

Neil Hurley and Mi Zhang. Novelty and diversity in top-n recommendation–analysis and evaluation. *ACM Transactions on Internet Technology (TOIT)*, 10(4):14, 2011. 19

Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Tushar Chandra, and Craig Boutilier. Slateq: A tractable decomposition for reinforcement learning with recommendation sets. 2019. 39, 119

Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *TOIS*, 20(4):422–446, 2002. 19, 20, 54

Santosh Kabbur, Xia Ning, and George Karypis. Fism: factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 659–667, 2013. 11, 15, 22, 62, 76, 77

Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 197–206. IEEE, 2018. 30, 31, 89, 91, 93, 100, 105, 124, 125

Wang-Cheng Kang, Mengting Wan, and Julian McAuley. Recommendation through mixtures of heterogeneous item relationships. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1143–1152. ACM, 2018. 73, 74, 77, 78

Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 79–86, 2010. 26

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 105, 125

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000. 38, 100

Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434, 2008. 23

Yehuda Koren and Robert Bell. Advances in collaborative filtering. In *Recommender systems handbook*, pages 77–118. Springer, 2015. 23

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009. 5, 12, 22, 31, 73, 77

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436, 2015. 28

Joonseok Lee, Seungyeon Kim, Guy Lebanon, and Yoram Singer. Local low-rank matrix approximation. In *International Conference on Machine Learning*, pages 82–90, 2013. 23

Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670. ACM, 2010. 39, 40

Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 297–306. ACM, 2011. 39

Mu Li, Ziqi Liu, Alexander J. Smola, and Yu-Xiang Wang. Difacto: Distributed factorization machines. In *WSDM*, pages 377–386, 2016. 25

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 38

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, volume 15, pages 2181–2187. AAAI Press, 2015. 70, 73, 77

Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, (1):76–80, 2003. 11, 62

Qiang Liu, Feng Yu, Shu Wu, and Liang Wang. A convolutional click prediction model. In *CIKM*, pages 1743–1746. ACM, 2015. 30, 49

Hao Ma, Dengyong Zhou, Chao Liu, Michael R Lyu, and Irwin King. Recommender systems with social regularization. In *ACM International Conference on Web Search and Data Mining*, pages 287–296, 2011. 140

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013. 89

Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014. 30

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. 5, 36, 39, 89

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016. 38

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062, 2016. 90

Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 502–511. IEEE, 2008. 16

Emilio Parisotto, H Francis Song, Jack W Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant M Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. *arXiv preprint arXiv:1910.06764*, 2019. 100, 121

Chanyoung Park, Donghyun Kim, Jinoh Oh, and Hwanjo Yu. Do also-viewed products help user rating prediction? In *Proceedings of the 26th International Conference on World Wide Web*, pages 1113–1122. ACM, 2017. 74

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing*, pages 1532–1543, 2014. 22

Siddharth Reddy, Anca D Dragan, and Sergey Levine. Sqil: Imitation learning via reinforcement learning with sparse rewards. *arXiv preprint arXiv:1905.11108*, 2019. 119

Steffen Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010. 4, 12, 24, 25, 28, 31, 44, 54, 77, 138

Steffen Rendle. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57, 2012. 24

Steffen Rendle. Scaling factorization machines to relational data. *VLDB Journal*, 6(5):337–348, 2013. 25

Steffen Rendle and Christoph Freudenthaler. Improving pairwise learning for item recommendation from implicit feedback. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 273–282. ACM, 2014. 3, 14, 141

Steffen Rendle, Leandro Balby Marinho, Alexandros Nanopoulos, and Lars Schmidt-Thieme. Learning optimal ranking with tensor factorization for tag recommendation. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 727–736. ACM, 2009a. 21, 25

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009b. 3, 4, 14, 30, 50, 54, 69, 94, 96, 101, 112, 118

Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 811–820. ACM, 2010. 31

Steffen Rendle, Dennis Fetterly, Eugene J Shekita, and Bor-yiing Su. Robust large-scale machine learning in the cloud. In *SIGKDD*, pages 1125–1134. ACM, 2016. 25

Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*. 2011. 3

David Rohde, Stephen Bonner, Travis Dunlop, Flavian Vasile, and Alexandros Karatzoglou. Recogym: A reinforcement learning environment for the problem of product recommendation in online advertising. *arXiv preprint arXiv:1808.00720*, 2018. 40

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001. 11

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015. 37

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 37

Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. Deep crossing: Web-scale modeling without manually crafted combinatorial

features. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 255–262. ACM, 2016. 44

Wenjie Shang, Yang Yu, Qingyang Li, Zhiwei Qin, Yiping Meng, and Jieping Ye. Environment reconstruction with hidden confounders for reinforcement learning based recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 566–576. ACM, 2019. 39

Guy Shani, David Heckerman, and Ronen I Brafman. An mdp-based recommender system. *Journal of Machine Learning Research*, 6(Sep):1265–1295, 2005. 94

Swapneel Sheth, Nipun Arora, Christian Murphy, and Gail Kaiser. wehelp: a reference architecture for social recommender systems. In *IEEE/ACM International Conference on Automated Software Engineering workshops. IEEE/ACM International Conference on Automated Software Engineering*. NIH Public Access, 2010. 3

Leily Sheugh and Sasan H Alizadeh. A note on pearson correlation coefficient as a metric of similarity in recommender system. In *AI & Robotics (IRANOPEN), 2015*, pages 1–6. IEEE, 2015. 12

Jing-Cheng Shi, Yang Yu, Qing Da, Shi-Yong Chen, and An-Xiang Zeng. Virtual-taobao: Virtualizing real-world online retail environment for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4902–4909, 2019. 91

Yue Shi. Ranking and context-awareness in recommender systems. 2013. http://homepage.tudelft.nl/c7c8y/theses/PhDThesisShi.pdf. 3

Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Nuria Oliver, and Alan Hanjalic. Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 139–146. ACM, 2012. 15, 19, 20

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016. 89, 95

Brent Smith and Greg Linden. Two decades of recommender systems at amazon.com. *IEEE Internet Computing*, 21(3):12–18, May 2017. 62

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 69

Alex Strehl, John Langford, Lihong Li, and Sham M. Kakade. Learning from logged implicit exploration data. In *NIPS*, 2010. 132

Zhu Sun, Jie Yang, Jie Zhang, Alessandro Bozzon, Long-Kai Huang, and Chi Xu. Recurrent knowledge graph embedding for effective recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, pages 297–305. ACM, 2018. 74

Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *ACM International Conference on Web Search and Data Mining*, 2018. 30, 31, 49, 91, 93, 104, 124

Jiliang Tang, Xia Hu, Huiji Gao, and Huan Liu. Exploiting local and global social context for recommendation. In *International Joint Conference on Artificial Intelligence*, volume 13, pages 2712–2718, 2013. 140

Jiliang Tang, Suhang Wang, Xia Hu, Dawei Yin, Yingzhou Bi, Yi Chang, and Huan Liu. Recommendation with social dimensions. In *AAAI Conference on Artificial Intelligence*, pages 251–257, 2016. 140

Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. Latent relational metric learning via memory-based a ention for collaborative ranking. In *Proceedings of the 27th International Conference on World Wide Web*. ACM, 2018. 30, 73, 76

Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018. 40

Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966. 26

Saúl Vargas and Pablo Castells. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 109–116. ACM, 2011. 18, 19

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, ukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30:5998–6008, 2017. 5, 26, 30, 31, 105

Nikos Vlassis, Aurélien Bibaut, Maria Dimakopoulou, and Tony Jebara. On the design of estimators for bandit off-policy evaluation. In *ICML*, 2019. 132

Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. Ripplenet: Propagating user preferences on the knowledge graph for recommender systems. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 417–426. ACM, 2018a. 74

Hongwei Wang, Fuzheng Zhang, Xing Xie, and Minyi Guo. Dkn: Deep knowledge-aware network for news recommendation. *Proceedings of the 27th International Conference on World Wide Web*, pages 1835–1844, 2018b. 74

Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*, page 12. ACM, 2017. 43

Xiang Wang, Xiangnan He, Fuli Feng, Liqiang Nie, and Tat-Seng Chua. Tem: Tree-enhanced embedding model for explainable recommendation. In *WWW*, pages 1543–1552, 2018c. 43

Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *SIGIR*, 2019a. 62

Xiang Wang, Dingxian Wang, Canran Xu, Xiangnan He, Yixin Cao, and Tat-Seng Chua. Explainable reasoning over knowledge graphs for recommendation. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*. AAAI Press, 2019b. 74, 76

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. 37, 95

Joost Wit. Evaluating recommender systems: an evaluation framework to predict user satisfaction for recommender systems in an electronic programme guide context. Master's thesis, University of Twente, 2008. https://essay.utwente.nl/59711/1/MA_thesis_J_de_Wit.pdf. 12

Libing Wu, Cong Quan, Chenliang Li, Qian Wang, Bolong Zheng, and Xiangyang Luo. A context-aware user-item representation learning for item recommendation. *TOIS*, 37(2):22, 2019. 43

Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 153–162. ACM, 2016. 23

Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. Attentional factorization machines: Learning the weight of feature interactions via attention networks. *arXiv preprint arXiv:1708.04617*, 2017. 30

Xin Xin, Dong Wang, Yue Ding, and Chen Lini. Fhsm: factored hybrid similarity methods for top-n recommender systems. In *Asia-Pacific Web Conference*, pages 98–110. Springer, 2016. 15, 23

Xin Xin, Yuan Fajie, He Xiangnan, and Jose Joemon. Batch is not heavy: Learning word embeddings from all samples. *The Annual Meeting of the Association for Computational Linguistics*, 2018. 16, 17

Xin Xin, Bo Chen, Xiangnan He, Dong Wang, Yue Ding, and Joemon M Jose. Cfm: convolutional factorization machines for context-aware recommendation.

In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 3926–3932. AAAI Press, 2019a. 138

Xin Xin, Xiangnan He, Yongfeng Zhang, Yongdong Zhang, and Joemon Jose. Relational collaborative filtering: Modeling multiple item relations for recommendation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 125–134, 2019b. 139

Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis, and Joemon M Jose. Self-supervised reinforcement learning for recommender systems. *arXiv preprint arXiv:2006.05779*, 2020. 17, 125

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015. 30

Feng Xue, Xiangnan He, Xiang Wang, Jiandong Xu, Kai Liu, and Richang Hong. Deep item-based collaborative filtering for top-n recommendation. 2018. 62

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014. 70, 71

Wenhui Yu, Huidi Zhang, Xiangnan He, Xu Chen, Li Xiong, and Zheng Qin. Aesthetic-based clothing recommendation. In *WWW*, pages 649–658, 2018. 30

Fajie Yuan. *Learning implicit recommenders from massive unobserved feedback*. PhD thesis, University of Glasgow, 2018. xi, 4, 14

Fajie Yuan, Guibing Guo, Joemon M Jose, Long Chen, Haitao Yu, and Weinan Zhang. Lambdafm: learning optimal ranking with factorization machines using lambda surrogates. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 227–236. ACM, 2016. 25, 54, 141

Fajie Yuan, Guibing Guo, Joemon M Jose, Long Chen, Haitao Yu, and Weinan Zhang. Boostfm: Boosted factorization machines for top-n feature-based recommendation. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces*, pages 45–54. ACM, 2017. 16

Fajie Yuan, Xin Xin, Xiangnan He, Guibing Guo, Weinan Zhang, Tat-Seng Chua, and Joemon Jose. fbgd: Learning embeddings from positive unlabeled data with bgd. *Association for Uncertainty in Artificial Intelligence*, 2018. 16, 17, 96

Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. A simple convolutional generative network for next item recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 582–590, 2019. 17, 31, 49, 89, 91, 93, 104, 124, 125

Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 353–362. ACM, 2016a. 74, 77

Shuai Zhang, Lina Yao, and Aixin Sun. Deep learning based recommender system: A survey and new perspectives. *arXiv preprint arXiv:1707.07435*, 2017. 29

Weinan Zhang, Tianqi Chen, Jun Wang, and Yong Yu. Optimizing top-n collaborative filtering via dynamic negative item sampling. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 785–788. ACM, 2013. 141

Weinan Zhang, Tianming Du, and Jun Wang. Deep learning over multi-field categorical data. In *ECIR*, pages 45–57. Springer, 2016b. 29

Yongfeng Zhang, Guokun Lai, Min Zhang, Yi Zhang, Yiqun Liu, and Shaoping Ma. Explicit factor models for explainable recommendation based on phrase-level sentiment analysis. In *SIGIR*, pages 83–92. ACM, 2014. 29, 44

Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. Recommendations with negative feedback via pairwise deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1040–1048. ACM, 2018. 39, 112, 119

Lei Zheng, Vahid Noroozi, and Philip S Yu. Joint deep modeling of users and items using reviews for recommendation. In *WSDM*, pages 425–434. ACM, 2017. 30

Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In *SIGKDD*, pages 1059–1068. ACM, 2018. 43

Lixin Zou, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu, and Dawei Yin. Reinforcement learning to optimize long-term user engagement in recommender systems. *arXiv preprint arXiv:1902.05570*, 2019. 39