Learning from User Interactions for Recommending Queries and Items

Wanyu Chen

Learning from User Interactions for Recommending Queries and Items

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. K.I.J. Maex
ten overstaan van een door het College voor Promoties ingestelde
commissie, in het openbaar te verdedigen in
de Agnietenkapel
op 2 februari 2021, te 12:00 uur

door

Wanyu Chen

geboren te Anhui

Promotiecommissie

Promotor:		
	Prof. dr. M. de Rijke	Universiteit van Amsterdam
Co-promotor:		
	Dr. F. Cai	National University of Defense Technology
Overige leden:		
	Prof. dr. H. Haned	Universiteit van Amsterdam
	Prof. dr. E. Kanoulas	Universiteit van Amsterdam
	Dr. C. Monz	Universiteit van Amsterdam
	Prof. dr. M.A. Larson	Technische Universiteit Delft
	Prof. dr. M. Zhang	Tsinghua University

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

The research was supported by the China Scholarship Council.

Cover by Wanyu Chen
Printed by Off Page, Amsterdam, The Netherlands

ISBN: 978-94-93197-43-5

Acknowledgements

Doing a Ph.D. has been an unforgettable journey. There is not only the joy of success but also the frustration caused by failures on this journey. However, I am grateful for all of those experiences since they have helped to constitute who I am now. They have given me memories of happiness and taught me important lessons. There are many people who have helped and accompanied me on this journey. I could not have completed this book without all of you and I am truly thankful to you for your support along the way. You and all the experiences are a great wealth to me.

My deepest gratitude is to my promotor Maarten de Rijke. As my supervisor, you have taught me much and always given me guidance and support when I felt confused during my Ph.D. I am fully impressed by your patience, attitudes towards research and immense knowledge. Also, you have built a great research group as well as a warm family, ILPS. I feel very lucky to be part of this family and to work with you. Thank you, Maarten.

I am also deeply grateful to my co-promotor Fei Cai, who has contributed much effort to help me in my research and told me how to overcome obstacles. You have always supported me to finish my Ph.D. at the University of Amsterdam. Besides, I am thankful to Prof.dr. Honghui Chen who gave me long-term endorsement over the past years.

I am very honored and grateful to have Hinda Haned, Min Zhang, Martha Larson, Christof Monz, and Evangelos Kanoulas as my committee members.

I thank all the members of the Information and Language Processing Systems (ILPS) group at the University of Amsterdam. You have helped me to overcome the loneliness when studying aboard. I have really enjoyed the time we worked together, the coffee breaks, the Friday lunches and so many other events together. Thanks to: Ali A, Ali V, Amir, Ana, Anna, Antonis, Arezoo, Alexey, Andreas, Artem, Bob, Boris, Chang, Chuan, Christof, Christophe, Dan, Dat, David D, Evangelos, Georgios, Gabriel, Hamid, Harrie, Hinda, Hosein, Ilya, Jiahuan, Jie, Jin, Julia, Julien, Katya, Ke, Maarten dR, Maarten M, Maartje, Mahsa, Mariya, Marlies, Marzieh, Maurits, Mohammad, Mostafa, Mozhdeh, Nikos, Olivier, Peilei, Pengjie, Petra, Praveen, Ridho, Rolf, Sam, Sami, Sebastian, Shaojie, Spyretta, Svitlana, Trond, Vera, Xiaohui, Xiangsheng, Xiaomeng, Yangjun, Yifan, and Ziming.

Many thanks to Jiahuan and Dan for helping me to begin my life in Amsterdam. Thanks a lot to Jin and Yangjun for your company; I will never forget the happy times that we spent together making dinner. I also want to thank Pengjie who has always been patient and willing to discuss my research with me; discussion with you have enriched my ideas, which really helped me in my work. Thanks to Harrie for spending time and effort to translate my summary to Dutch. I especially thank Petra for helping me deal with countless practical details. You are so nice and patient with us.

I would like to acknowledge the China Scholarship Council (CSC) for the financial support that funded parts of the research discussed in this dissertation. I also appreciate the generosity of my home university, the National University of Defense Technology, for giving me this excellent opportunity to study aboard.

Last but not least, I would like to thank my parents for their long-term encourage-

ment and love. No matter what happens, you are always my strong shield behind me. The love from you is my source of strength in my whole life.

Wanyu Chen
December 1st, 2020

Contents

1 Introduction

People use search engines and recommender systems for multiple purposes in daily life – they may search for some specific information with queries, or they may just want to watch movies and do online shopping for entertainment. In many cases it is not clear what exactly users want: what information they are looking for or what items they are interested in. Thus, search engines and recommender systems need to become proactive and provide users with auxiliary information so as to gain a better understanding of users' goals. E.g., search engines could suggest queries and recommender systems could offer lists of options that may address the users' needs. Since users are diverse and they may have different purposes due to their current situation or personal preference, providing such recommendations to satisfy a certain user's information need can be a challenging task for search engines and recommender system. To this end, collecting and analyzing user interactions can be an effective way to learn their long-term preferences or their short-term intents.

Regarding search engines, query suggestions help users to refine their queries and thus improve the efficiency of expressing their information needs. We can generate query suggestions by mining different users' search behavior saved in query logs. Traditional methods relying on query co-occurrence are not able to satisfy all users in all contexts as they always provide the same list of suggestions to different users [13, 21]. However, previously submitted queries of a user may provide a useful search context to reduce ambiguity of the current query and to produce more focused suggestions [16, 41, 60]. Since a query log can be partitioned into query sessions, i.e., sequences of queries issued by a unique user within a short time interval, users' long-term preferences as well as their short-term intents can be learned from those interactions. Neural networks, such as Recurrent Neural Networks (RNNs), provide us with effective ways to model the complex relationship between those interactions within a session and between sessions.

Recommender systems collect user-item interactions, such as whether a user rated, clicked, or bought an item. Such signals are then used for learning user preferences and providing recommendations. Collaborative Filtering (CF) is a widely used approach that makes use of long-term historical user-item interactions for so-called top-N recommendation. This is the task of finding a few specific items that are supposed to be most appealing to the user. Many traditional CF techniques are based on Matrix Factorization (MF) [148]. They characterize users and items by latent factors that are

extracted from the user-item rating matrix and then a user's preference towards an item is modeled as an inner product of these latent factors. The main idea behind CF is that users with similar interactions may have similar preferences, thus we can recommend a user items that are preferred by like-minded people. However, a linear projection, i.e., a dot product, may not be able to give an accurate description of the characteristics of users (items) and user-item interactions: previous work has pointed out that non-linearities have potential advantages for improving the performance of recommender systems with extensive experiments [76, 110, 140].

General recommender systems often discard sequential information and focus on mining the static relevancy of an item for a user from interactions [47, 119, 149]. For instance, MF may be effective at modeling a user's general preferences by learning from their entire interaction history but it does not model the order of the user's interactions. To this end, session-based recommender systems have been proposed to model the evolution of a user's short-term preferences reflected by sequential interactions. Today's session-based recommender systems successfully capture users' short-term decision making process with RNN- or Markov chain (MC)-based methods. But they do not capture variations in the relative importance of a user's long-term vs. short-term interests for session-based recommendation. Users with different shopping preferences may prefer different next items even under the same session context. Also, in practice users interactions are heterogeneous, which may indicate different motivations of the user and hence trigger different follow-up actions. Capturing users' dynamic preferences with implicit preference data can help to improve the sequential recommendation performance.

So far, we have discussed personalized recommendations based on learning from user interactions, which focuses on improving recommendation accuracy only. However, in real scenarios, users may prefer more diverse lists of recommended items [147]. This is especially true in Sequential Recommendation (SR) as users may have multiple intents, e.g., different topics or categories of items. In addition, user intents are occasionally exploratory which means they do not have a specific goal in mind. A homogeneous list of recommendations cannot satisfy such users, leading to a boring user experience [111]. It is challenging to design an optimal solution for diversified sequential recommendation mainly due to two facts. First, some of the current approaches assume that user intents are static and they require that user intents are prepared beforehand, which is unrealistic in most SR scenarios [5, 20]. Second, most current approaches to diversified recommendation belong to the post-processing paradigm, where the recommendation models are not aware of diversity during training and it is hard to design ideal diversity strategies for different recommendation models.

In this thesis, we investigate how to learn from user interaction data and thus make satisfactory recommendations for both search engines and recommender systems.

1.1 Research Outline and Questions

How to improve the recommendation performance by learning from user interactions is a very broad question. In this thesis, we investigate the following four themes:

(1) learning users' search intents and recommending personalized queries based on

user historical submitted queries in search engines;

(2) learning users' general preferences and recommending personalized items using their historical interactions;

(3) learning users' dynamic preferences and recommending personalized items considering users' long-term as well as short-term behavior; and

(4) learning users' multiple intents from their sequential behavior and recommending personalized as well as diversified items.

1.1.1 Learning users' search intent and recommending personalized queries

Search engines and recommender system are solutions to help users cope with an increasingly complex information landscape [91]. Regarding search engines, users submit queries related to their intent [128]. However, most of the users may not have a clear idea about their information needs, thus the submitted queries may be ambiguous which leads to unsatisfactory search results [4]. Modern search engines offer query suggestions to help users formulate a good query and thus get the intended search results to address their information needs [4]. In most commercial search engines such as Baidu, Bing, Google, Yahoo! and Yandex provide query suggestions to improve their system's usability [21]. By predicting a user's search intent, a search engine recommends queries that reflect the user's information needs based on his input. A significant amount of work has gone into methods for formulating a better understandable query submitted by users [11, 12, 15, 114, 128].

Previous work on query suggestion has mainly focused on incorporating directly observable features such as query co-occurrence and semantic similarity. The structure of such features is often set manually, as a result of which hidden dependencies between queries and users may be ignored. This has motivated us to find a better way to learn such dependencies, which leads to the following research question:

(RQ1) How to capture users' search intent by learning from their historical submitted queries?

To answer **RQ1**, we develop a neural query suggestion model that is able to capture the user's search intent by capturing both their short-term interests, as manifested during an ongoing search session, and their long-term interests, as manifested during earlier sessions. The model combines a session-level neural network and a user-level neural network into a hierarchical structure to model the short- and long-term search history of a user. We also apply an attention mechanism inside the hierarchical structure that is meant to capture a user's preference towards different queries in a session.

Based on experimental results on the AOL query log dataset [94], we find that: (1) the proposed Attention-based Hierarchical Neural Query Suggestion (AHNQS) model helps to boost query suggestion performance in terms of MRR and Recall across sessions with various lengths; (2) using the combined session state in the AHNQS model achieves better performance than only using the local session state; and (3) the AHNQS model yields better performance than the best baseline for inactive, active, as well has highly active users.

1.1.2 Learning users' general preferences and recommending personalized items

Besides search engines, recommender systems also play an important role in fulfilling the information needs of users without submitting any queries. Over the past years, various approaches have been proposed to predict users' general preferences based on long-term user-item interactions. Collaborative Filtering (CF) approaches have been widely investigated and used for personalized recommendation [3, 148]. Traditional CF methods, such as the Latent Factor Model (LFM) [68], often predict a user's preference for an item with a linear kernel, i.e., a dot product of their latent factors, which may not be able to capture the complex structure of user-item interactions well. Recently introduced Deep Learning (DL)-based approaches to recommender systems overcome shortcomings of conventional approaches and are able to achieve high recommendation quality.

We hypothesize that DL should be able to effectively capture both non-linear and non-trivial user-item relationships as well as users' (items') characteristics with multi-layer projections [148]. Hence, this has led to the following research question:

RQ2 Can we learn users general preference by modeling non-linear user-item relationships as well as characteristics based on their interactions?

To answer **RQ2**, we propose a Joint Neural Collaborative Filtering (J-NCF) model that enables two processes–feature extraction and user-item interaction modeling–to be trained jointly in a unified DL structure. The J-NCF model contains two main networks for recommendation. The first network uses the rating information of a user (an item) as the network input, and outputs a vector representation for the user (the item). Then, using the connection between a user's and an item's vector as input, the second neural network models the user-item interactions and outputs the prediction of the corresponding rating of the user and item. Thus, these two networks can be coupled tightly and trained jointly in a unified structure. Besides, we take both implicit and explicit feedback, point-wise and pair-wise loss into account to enhance the prediction performance.

We analyze the recommendation performance of J-NCF as well as baseline models on several datatsets and find that J-NCF consistently yields the best performance. J-NCF also shows competitive improvements over the best baseline model when applied with inactive users and different degrees of data sparsity.

1.1.3 Learning users' dynamic preferences for sequential recommendation

Conventional recommender systems often ignore sequential information behind user-item interactions and assume the user preferences to be static [47, 119, 149]. For instance, a typical conventional recommender system based on matrix factorization [67] may be effective at modeling a user's general preferences by learning from their entire interaction history but it does not model the order of the user's interactions. Unlike conventional recommender systems, session-based recommender systems model the evolution of a user's short-term preference implied by sequential interactions in a ses-

sion with the aim of recommending the next item a user may be interested in [138]. An effective session-based recommender should be able to exploit a user's evolving preferences, which we assume to be a mixture of her short- and long-term interests.

Existing session-based recommendation methods often embed a user's long-term preference into a static representation, which plays a fixed role when dealing with the user's current short-term interests. This is problematic because long-term preferences may be more or less important for predicting the next conversion depending on the user's short-term interests. Therefore, we are motivated to address the following research question:

RQ3 How can we incorporate users' long- and short-term interaction behavior for session-based recommendation?

To answer **RQ3**, we propose a Dynamic Co-attention Network for Session-based Recommendation (DCN-SR). DCN-SR applies a co-attention network to capture the dynamic interactions between the user's long- and short-term interaction behavior and generates co-dependent representations of the user's long- and short-term interests. For modeling a user's short-term interaction behavior, we design a Contextual Gated Recurrent Unit (CGRU) network to take actions like "click," "collect" and "buy" into account.

We analyze the performance of our model with experiments on two e-commerce datasets. The results confirm the effectiveness and robustness of DCN-SR with different session lengths and varying numbers of users' historical interactions. DCN-SR outperforms the best performing state-of-the-art model Short-Term Attention/Memory Priority Model (STAMP) across different session lengths, especially for short sessions. As to users with different numbers of historical interactions, DCN-SR shows more competitive recommendation performance on all users than the state-of-the-art baseline model STAMP. In addition, the improvements of DCN-SR are higher for users with more historical interactions.

1.1.4 Learning users' multiple intents for diversified sequential recommendation

Previous studies on Sequential Recommendation (SR) mostly focus on optimizing the recommendation accuracy while ignoring the diversity of recommended items. This strategy risks over-specialization, i.e., the set of recommended items may become very homogeneous [139]. However, it has been shown that diversity is an important metric to consider in recommender systems, as users prefer more diverse lists of recommended items [147]. This is especially true in SR as users may have multiple intents. For example, a user may want to buy various categories of items for a festival, including food, clothes or decorations at the same time. In addition, user intents are occasionally exploratory, which means they do not have a specific goal in mind. A homogeneous list of recommendations cannot satisfy such users with an exploratory intent and often leads to a boring user experience [111].

Many existing methods for improving the diversity of recommended items are not applicable to SR because they assume that user intents are static and rely on post-processing the list of recommended items to promote diversity. An effective solution

for sequential recommendation that considers personalization as well as diversification is called for. This leads to the following research question:

RQ4 How can we address the challenge of diversified sequential recommendation in an end-to-end framework?

To answer **RQ4**, we consider both accuracy and diversity by reformulating SR as a list generation task, and propose an integrated approach with an end-to-end neural model, called intent-aware diversified sequential recommendation (IDSR). Specifically, we introduce an implicit intent mining (IIM) module to capture multiple user intents as reflected in sequences of user behavior. The IIM module employs multiple attention areas to mine user's multiple intents, with each attention area capturing a particular latent user intent. Then, we apply an intent-aware recommendation decoder to generate a list of recommendations by selecting one item at a time. In order to supervise the learning of the IIM module and force the model to take recommendation diversity into account during training, we design an intent-aware diversity promoting (IDP) loss function that evaluates recommendation accuracy and diversity based on the generated lists of recommended items.

For this new task of sequential recommendation, we conduct experiments on four datatsets. The results demonstrate that IDSR significantly outperforms the state-of-the-art baselines in terms of recommendation diversity while maintaining competitive accuracy scores.

1.2 Main Contributions

In this section, the main contributions of this thesis are summarized as follows.

1.2.1 Algorithmic contributions

The algorithmic contributions of this thesis come in the form of four models:

(1) *An attention-based hierarchical neural architecture:* a new approach to query suggestion, Attention-based Hierarchical Neural Query Suggestion (AHNQS) (Chapter 2). It adopts a hierarchical structure containing an attention mechanism to better capture the user's search intent. The hierarchical structure contains two parts: a session-level RNN and a user-level RNN. The first captures queries in the current session and is used to model the user's short-term search context to predict their next query. The second captures the past search sessions for a given user and is applied to model their long-term search behavior to output a user state vector representing their preferences. We apply an attention mechanism inside the hierarchical structure of AHNQS that is used to capture a user's preference towards different queries in a session.

(2) *A joint neural collaborative filtering approach:* a neural approach for recommender systems, named Joint Neural Collaborative Filtering (J-NCF) (Chapter 3). It enables deep feature learning and deep user-item interaction modeling to be coupled tightly and jointly optimized in a single neural network. As

for objective functions, point-wise objectives aim at obtaining accurate ratings, which is more applicable in rating prediction tasks [61]. Pair-wise objectives are usually focused on users' preferences towards pairs of items and are usually considered more suitable for top-N recommendation [103]. We design a new loss function that explores the information contained in both point-wise and pair-wise loss as well as implicit and explicit feedback.

(3) *A dynamic co-attention network model:* a model for session-based recommendation named Dynamic Co-attention Network for Session-based Recommendation (DCN-SR) (Chapter 4). It is able to integrate users' long-term and short-term preferences. Specifically, we apply a co-attention network to capture interactions between actions in a user's long-term and short-term interaction histories and generate co-dependent representations of his long-term and short-term preferences. Thus, DCN-SR cannot only exploit the benefits of incorporating long-term and short-term knowledge, but also consider the dynamic properties in the relations between a user's long-term and short-term preferences. We also design a Contextual Gated Recurrent Unit (CGRU) to incorporate different types of short-term user actions so as to better estimate a user's next consumption interests.

(4) *An intent-aware end-to-end neural approach:* a novel framework for diversified sequential recommendation named intent-aware diversified sequential recommendation (IDSR) (Chapter 5). It reformulates SR as a list generation task so as to model the relationship among recommended items. To the best of our knowledge, this is the first end-to-end list generation based neural framework that considers diversification for SRs. IDSR employs an *implicit intent mining* (IIM) module to automatically capture multiple latent user intents reflected in sequences of user behavior, and an *intent-aware diversity promoting* (IDP) decoder to directly generate accurate and diverse lists of recommendations for the latent user intents. In order to supervise the learning of the IIM module and force the model to take recommendation diversity into account during training, we also design an IDP loss function that evaluates recommendation accuracy and diversity based on the generated lists of recommended items.

1.2.2 Experimental contributions

(1) The experimental contributions made in Chapter 2 are: (a) An empirical comparison of AHNQS with other state-of-the-art question suggestion models. (b) An analysis of the impact of the number of users' sessions on the performance of query suggestion. (c) An analysis of the impact of lengths of users' sessions on the performance of query suggestion.

(2) In Chapter 3 our experimental contributions are: (a) An empirical comparison of J-NCF with other state-of-the-art recommendation models. (b) An analysis of the impact of the number of users' interactions on the performance of recommendation. (c) An analysis of the impact of different loss functions as well as

the number of layers on the performance of the proposed model. (d) An analysis of the impact of data sparsity on the performance of recommendation.

(3) The experimental contributions made in Chapter 4 are: (a) An empirical comparison of DCN-SR with its variants as well as other state-of-the-art session-based recommendation models. (b) An analysis of the impact of the number of users' historical interactions on the performance of recommendation. (c) An analysis of the impact of the length of users' sessions on the performance of recommendation.

(4) And, finally, in Chapter 5 the experimental contributions are: (a) An empirical comparison of DCN-SR with other state-of-the-art session-based recommendation models as well as diversified recommendation models. (b) An analysis of the impact of some hyper-parameters in the proposed model on the performance of recommendation. (c) A case study to show that the proposed model can recommend a list that contains accurate as well as diversified items.

1.3 Thesis Overview

We give a brief overview of the content of each chapter in this thesis. In this thesis, we investigate how to understand user goals and provide suitable recommendations by learning from user interactions.

First, in Chapter 2 we consider the task of query suggestion for search engines. Here we work with logs that record queries and clicks that a user submitted. We propose an attention-based hierarchical neural approach to improve the performance for personalized query suggestion.

Then, in Chapter 3 we consider the general item recommendation task for recommender systems. Here the data we work with consists of interactions of users with items such as watched movies, bought or clicked products. We analyze users' long-term behavior with the recommender system and propose a joint neural collaborative filtering method for personalized recommendations.

Next, in Chapter 4 we deal with the task of personalized session-based recommendation in which we consider users' sequential behavior. We design a dynamic co-attention network model combining user's long- as well as short-term behavior to help improve the performance of personalized session-based recommendation.

In Chapter 5 we consider both personalization and diversification for sequential recommendation by proposing an end-to-end neural model, i.e., an intent-aware diversified sequential recommendation model.

Finally, in Chapter 6: we give a brief summary of the whole thesis, discuss limitations, and provide insights to some future directions.

1.4 Origins

The following publications form the basis of the chapters in this thesis.

- **Chapter 2** is based on W. Chen, F. Cai, H. Chen, and M. de Rijke. Attention-based hierarchical neural query suggestion. In *SIGIR '18*, pages 1093–1096, 2018 [22], and its extension W. Chen, F. Cai, H. Chen, and M. de Rijke. Hierarchical neural query suggestion with an attention mechanism. *Information Processing and Management*, page 102040, 2019 [24]. All authors contributed to the design of the algorithms and of the experiments. WC did most of the model design, experiments and writing; FC, MdR and HC contributed to the writing.
- **Chapter 3** is based on W. Chen, F. Cai, H. Chen, and M. de Rijke. Joint neural collaborative filtering for recommender systems. *ACM Transactions on Information Systems*, 37(4), 2019 [25]. All authors contributed to the design of the algorithms and of the experiments. WC did most of the model design, experiments and writing; FC, MdR and HC contributed to the writing.
- **Chapter 4** is based on W. Chen, F. Cai, H. Chen, and M. de Rijke. A dynamic co-attention network for session-based recommendation. In *CIKM '19*, pages 1461–1470, 2019 [23]. All authors contributed to the design of the algorithms and of the experiments. WC did most of the model design, experiments and writing; FC, MdR and HC contributed to the writing.
- **Chapter 5** is based on W. Chen, P. Ren, F. Cai, F. Sun, and M. de Rijke. Improving end-to-end sequential recommendations with intent-aware diversification. In *CIKM '20*, page 175–184, 2020 [27]. All authors contributed to the design of the algorithms and of the experiments. WC did most of the model design, experiments and writing; PR, FC and FS helped with the model design; PR, FC, FS, and MdR contributed to the writing.

The writing of the thesis also benefited from work based on the following papers:

- W. Chen, F. Cai, H. Chen, and M. de Rijke. Personalized query suggestion diversification. In *SIGIR '17*, pages 817–820, 2017 [21].
- W. Chen, F. Cai, H. Chen, and M. de Rijke. Personalized query suggestion diversification in information retrieval. *Frontiers of Computer Science*, 14(3): 143602, 2020 [26].

2

Attention-based Hierarchical Neural Query Suggestion

In this chapter, we address **RQ1**: How to capture users' search intent by learning from their historical submitted queries? We propose an attention-based hierarchical neural approach for query suggestion, which learns users' search intent from their submitted queries.

2.1 Introduction

Query suggestion has been an effective way offered by modern search engines which aims to help users express their information needs effectively. Previous work on query suggestion mainly relies on features indicating dependencies between queries and users, such as clicks, co-occurrence and dwell time with some probabilistic methods [21]. However, the structure of those dependencies is usually modeled manually, while some hidden relationship between those queries cannot be captured. A query log can be treated as sequential data that can be modeled to predict the next input query with Recurrent Neural Network (RNN)-based approaches, which have been proposed to deal with the aforementioned challenges. However, existing neural based methods only consider so-called current sessions (in which a query suggestion is being generated) as the search context for query suggestion [91]. Our *research goal* is to develop a neural query suggestion method that is able to capture the user's search intent by capturing both their short-term interests, as manifested during an ongoing search session, and their long-term interests, as manifested during earlier sessions. To this end we propose an Attention-based Hierarchical Neural Query Suggestion (AHNQS) model that applies a user attention mechanism inside a hierarchical neural structure for query suggestion. The hierarchical structure contains two parts: a session-level Recurrent Neural Network (RNN) and a user-level RNN. The first captures queries in the current session and is used to model the user's short-term search context to predict their next query. The second captures the past search sessions for a given user and is applied to model their long-term search behavior to output a user state vector representing their preferences. We use the hidden state of the session-level RNN as the input to the user-

This chapter was published as [24].

level RNN; the user state of the latter is then used to initialize the first hidden state of the next session-level RNN.

In addition, we apply an attention mechanism inside the hierarchical structure of AHNQS that is meant to capture a user's preferences towards different queries in a session. This addition is based on the assumption that different queries in the same session may express different aspects of the user's search intent [6]. For example, queries with subsequent click behavior are more likely to represent the user's information need than those without. An attention mechanism can automatically assign different weights to hidden states of queries in the session-level RNN. The attentive hidden states together compose the session state, which we regard as a local session state. The local session state has the advantage of adaptively focusing on more important queries to capture a user's main purpose in the current session. Besides, we also consider the final hidden state of the session-level RNN as a global session state, which acts as a vertical summary of the full sequence behavior. Then we use a combination of the global and local session state as the input for the user-level RNN.

We compare the performance of AHNQS against a state-of-the-art query suggestion baseline and variants of RNN-based query suggestion methods using the AOL query log [94]. In terms of query suggestion ranking accuracy, we establish improvements of AHNQS over the best baseline model of up to 9.66% and 12.51% in terms of Recall@10 and MRR@10, respectively. In addition, we investigate the impact on query suggestion performance of different session states, i.e., global vs. local vs. combined. The results show the effectiveness of the AHNQS model with the combined session state. Furthermore, we test the scalability of the AHNQS model across users with different numbers of sessions in their interaction history. Our experimental results show that the performance of AHNQS is better than the best baseline model for users with varying degrees of activity.

Our contributions in this chapter are:

(1) We tackle the challenge of query suggestion in a novel way by proposing an Attention-based Hierarchical Neural Query Suggestion model, i.e., AHNQS, which adopts a hierarchical structure containing a user attention mechanism to better capture the user's search intent.

(2) We analyse the impact of session length on query suggestion performance and find that AHNQS consistently yields the best performance, especially with short search contexts.

(3) We examine the performance of AHNQS with different numbers of users' sessions. We find that AHNQS always yields better performance over the best baseline model, especially for users with few search sessions.

We describe related work in Section 2.2. Details of the attention-based hierarchical query suggestion model are described in Section 2.3. Section 2.4 presents our experimental setup. In Section 2.5, we report and discuss our results. Finally, we conclude in Section 2.6, where we also suggest future research directions.

2.2 Related Work

Query suggestion can support users of search engines during their search tasks. A significant amount of work has gone into methods for formulating a better understandable query submitted by users [11, 12, 15, 114, 128, 128]. In recent years, deep learning techniques have been applied to a range of information retrieval tasks, often leading to a better understanding of users' search behavior [10, 91]. In this section, we summarize traditional query suggestion methods in Section 2.2.1 and neural methods for query suggestion in Section 2.2.2.

2.2.1 Traditional query suggestion methods

Query suggestion methods that only rely on query co-occurrence are not able to satisfy all users in different conditions since they always provide the same list of suggestions to different users [13, 21]. Thus, search query logs are an important resource to mine different users' search behavior. A query log can be partitioned into query sessions, i.e., sequences of queries issued by a unique user within a short time interval. Previously submitted queries may provide a useful search context to reduce ambiguity of the current query and to produce more focused suggestions [60]. He et al. [41] propose a context-aware method that uses a Variable Memory Markov model (QVMM) and builds a suffix tree to model the user query sequence. The method proposed by Cao et al. [16] is similar but they build a suffix tree on clusters of queries and model transitions between clusters. However, for both methods, the number of parameters increases with the depth of the tree, which naturally leads to sparsity. Instead, our model can consider a flexible length of contexts with a fixed number of parameters. In addition, Santos et al. [107] and Ozertem et al. [92] focus on learning to rank approaches for query suggestion. However, those approaches are trained with pairwise features, which cannot consider previous queries effectively.

Other work mainly focuses on personalized query suggestion methods based on a query-URL bipartite graph, which is constructed from click data with one type of vertex corresponding to queries and another type corresponding to URLs. Such query suggestion methods typically use a click graph representing the information flow in query logs with a Markov random walk model [33, 122]. For instance, Ma et al. [85] develop a two-level query recommendation method based on two bipartite graphs (user-query and query-URL bipartite graphs) extracted from click data. Li et al. [75] use the connectivity of a query-URL bipartite graph through a novel two-phrase algorithm to recommend relevant queries that can improve the effectiveness of personalized query recommendation. Mei et al. [88] propose a personalized query suggestion method by employing hitting time and creating pseudo query nodes in a click graph. Click graph-based approaches cannot extract features of users' long-term and short-term search histories, and thus fail to combine different types of user behavior effectively.

Unlike the work listed above, our method uses a session-level RNN and a user-level RNN to model a user's short-term and long-term search history, respectively, thereby extracting sequential behavior features in an effective way. In addition, we integrate this two level RNN structure into a hierarchical architecture to combine short-term and long-term search behavior simultaneously. An attention mechanism is applied inside

the hierarchical structure to capture the user's search intent.

2.2.2 Neural networks in query suggestion

Neural networks have been applied in a variety of tasks in Information Retrieval, ranging from document ranking, and query understanding to user modeling [9, 57, 91, 112]. To some extent, we share similar targets with [9], who use distributed representations to model user browsing behavior in web search with a neural click model. We represent a user's short-term and long-term search behavior as a sequence of vector states that can describe a user's search intent. This kind of representation is richer than traditional methods and thereby enables us to capture more complex patterns of user search behavior.

Similar work has been done on neural query auto completion (QAC), i.e., a special type of query suggestion that suggests queries according to the input query prefix. Deep learning-based QAC models can be categorized into two groups: semantic models with convolutional neural networks and language models with recurrent neural networks [90, 93]. For RNN-based models, the probability for predicting the completed query is related to the length of the query, consequently favoring short query predictions [36]. Using the RNN and CNN-based features developed in the work mentioned, a learning to rank-based method has been shown to achieve state-of-the-art performance on the neural query auto completion task, but with high computational costs [36].

In addition, Sordoni et al. [115] propose a hierarchical recurrent encoder-decoder model for generative query suggestion. The authors formulate a novel hierarchical neural network architecture and use it to produce query suggestions. It differs from our work in two important ways. On the one hand, it incorporates the neural query suggestion model as a feature into a learning to rank approach and thus belongs to the feature engineering approaches. On the other hand, their approach only considers users' short-term search history and, hence, it ignores long-term search behavior.

Another line of work similar to ours is due to [98], where a neural hierarchical session-based approach for recommender systems is proposed. In particular, the authors extend previous RNN-based session modeling with an additional RNN level that models the user activity across sessions and the evolution of their interests over time. We apply an attention mechanism inside the hierarchical structure that can capture a user's preference towards certain queries over others in a session. In addition, we propose a combined session state to capture both the user's sequential behavior and their main purpose in the current session, which we show to be useful in improving the query suggestion performance.

2.3 Approach

Before introducing the AHNQS model, we introduce a neural query suggestion (NQS) model with session-level RNNs, and a hierarchical neural query suggestion (HNQS) model with hierarchical user-session RNNs.

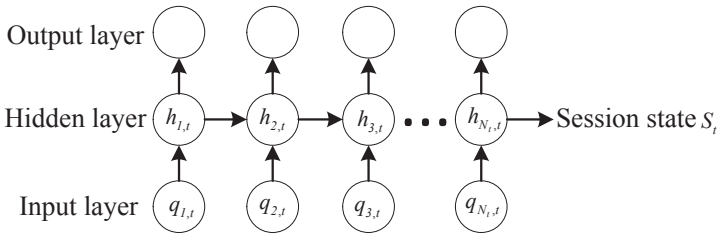


Figure 2.1: Structure of the NQS model.

2.3.1 Session-level RNN for query suggestion

As in [115], session-level RNNs are our starting point. In the neural-based query suggestion model (NQS), queries in the current session are taken as sequential input and used to output the probability of being the next query for the query suggestion candidates.

Formally, we assume that a query session $session_t$ contains N_t queries, denoted as $session_t = (q_{1,t}, q_{2,t}, q_{3,t}, \ldots, q_{N_t,t})$. As shown in Figure 2.1, for generating the input vector of the network, we use a 1-of-N encoding of q_i, i.e., the vector length equals the number of unique queries V and only the coordinate corresponding to the i-th query is one, the others are zero. We choose to use the Gated Recurrent Unit (GRU) [31] as our non-linear transformation. The hidden state h_n can be calculated by using the previous hidden state h_{n-1} and the candidate update state $\widehat{h}_n$:

$$h_n = (1 - u_n)h_{n-1} + u_n\widehat{h}_n, \tag{2.1}$$

where the update gate u_n can be generated by:

$$u_n = \sigma(I_u q_{n,t} + H_u h_{n-1}). \tag{2.2}$$

The candidate update state $\widehat{h}_n$ is calculated by:

$$\widehat{h}_n = \tanh(I q_{n,t} + H(r_n \cdot h_{n-1})), \tag{2.3}$$

where the reset gate r_n is:

$$r_n = \sigma(I_r q_{n,t} + H_r h_{n-1}), \tag{2.4}$$

where $I, I_u, I_r \in \mathbb{R}^{d_h \times V}$, $H, H_u, H_r \in \mathbb{R}^{d_h \times d_h}$, and d_h is the number of dimensions of the hidden state; $\sigma(\cdot)$ denotes the sigmoid function. The H matrices are used to keep or forget the information in h_{n-1}. Finally, $q_{n,t}$ is the representation of the n-th query in $session_t$.

We use $RNN_{session}$ and RNN_{user} to denote the GRU function. The final hidden state of a session-level RNN is used to indicate the session state, $S_t = h_{N_t,t}$. The output of the session-level RNN are the scores of query suggestion candidates being predicted as the next query:

$$s_{n,t} = g(\mathbf{W}_i h_{n,t}), \tag{2.5}$$

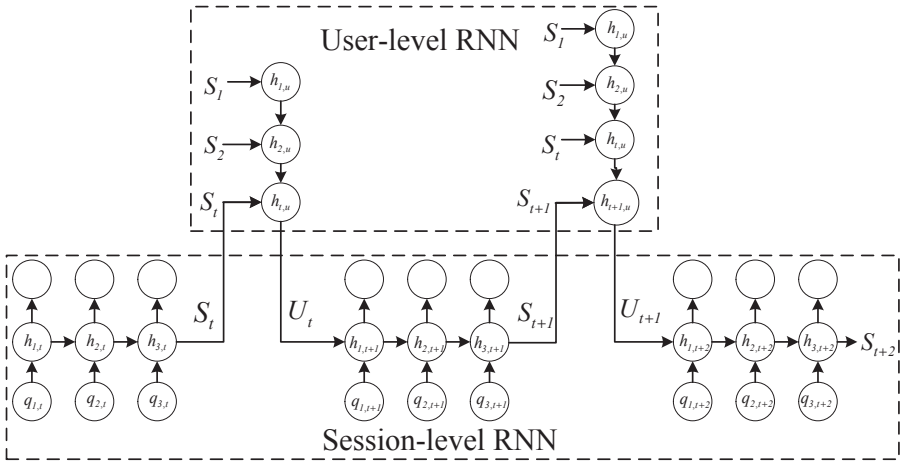


Figure 2.2: Structure of the HNQS model.

where $g(\cdot)$ is the activation function of the output layer, which can be either a softmax or a $\tanh$, depending on the loss function of the neural network. In addition, $\mathbf{W}_i \in \mathbb{R}^{V \times d_h}$ is a parameter that can keep the dimension of the output equal to the number of queries. Thus we can generate a list of query suggestions in the test set according to the scores of query candidates.

We choose a pairwise loss function that forces positive query suggestion samples to be ranked higher than negative ones. There are several pairwise ranking loss functions, including cross-entropy and TOP1 [51]. In the field of recommender systems, TOP1 has proved to outperform others, so we set:

$$Loss = \frac{1}{N_S} \cdot \sum_{j=1}^{N_S} \sigma(s_{j,t} - s_{i,t}) + \sigma(s_{j,t}^2), \tag{2.6}$$

where $s_{j,t}$ and $s_{i,t}$ denote the score of a negative query candidate and a ground truth query, respectively, and N_S is the number of negative sampled query candidates, which are the queries that a user does not submit or input; as before $\sigma(\cdot)$ denotes the sigmoid function.

2.3.2 Hierarchical user-session RNN for query suggestion

Clearly, the NQS model only models the short-term search context. Next, we model the long-term search behavior of a given user with a user-level RNN, thereby producing the hierarchical NQS model (HNQS).

We assume that a user u has N_u query sessions, which is denoted as: $u(session) = (\mathit{session}_{1,u}, \mathit{session}_{2,u}, \ldots, \mathit{session}_{N_u,u})$. In a user-level RNN, the input is the session state S_t and the hidden state in user-level RNN can be calculated as:

$$h_{n,u} = \mathit{RNN}_{user}(h_{n-1,u}, S_{n,u}), \tag{2.7}$$

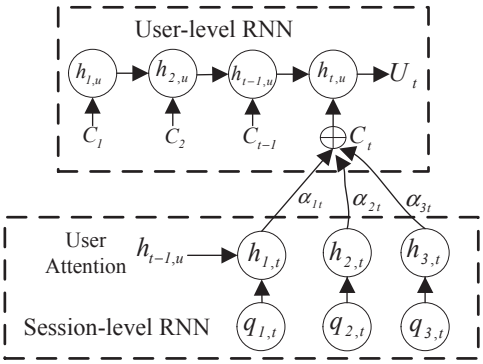


Figure 2.3: The attention mechanism in the AHNQS model.

where $S_{n,u}$ is the session state of the n-th query session of user u, which is equal to the last hidden state of the session-level RNN.

As shown in Figure 2.2, we use the final hidden state of the user-level RNN to denote the user state, $U_t = h_{t,u}$, that contains the information about the search behavior from a user's past sessions and thus can be applied in the session-level RNN. In HNQS, the session-level RNN is initialized with a user state as follows:

$$h_{0,t} = \tanh(\mathbf{W} \cdot U_{t-1} + \mathbf{b_0}). \tag{2.8}$$

Updating:

$$h_{n,t} = \mathit{RNN}_{\mathit{session}}(h_{n-1,t}, q_{n,t}). \tag{2.9}$$

Output:

$$s_{n,t} = g(\mathbf{W}_i h_{n,t}). \tag{2.10}$$

With this initialization strategy, the user's short-term search intent can be incorporated with his general preference. We choose to use only the initialization strategy for session-level RNN with user state U_{t-1}, instead of transporting the user information from U_{t-1} throughout the whole session-level RNN including initialization, updating and output. The GRU unit has both long and short term memory [53] and can automatically transport the user state information within the network, which leads to a better performance when combined with our initialization strategy. If we choose to transport the user state throughout the whole session-level RNN, the user state information will be overloaded, which limits the performance of the model.

2.3.3 Attention-based hierarchical RNN for query suggestion

We assume that submitted queries that trigger subsequent click behavior have a better expression of the user's search intent. We hypothesize that queries in a session should have different weights to reflect the user's information need and employ an attention mechanism on top of the HNQS model to capture the user's preference for different queries in a session and then aggregate the representations of informative queries.

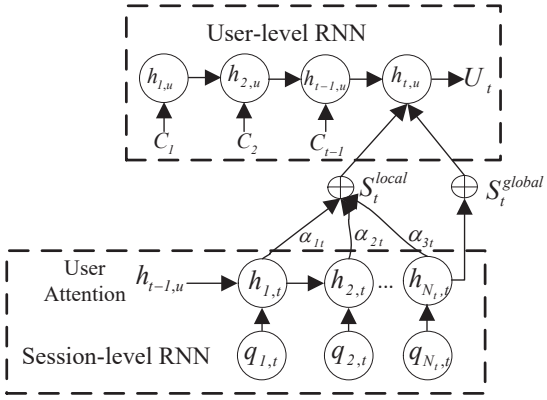


Figure 2.4: Structure of the AHNQS model with combined session state.

Figure 2.3 shows how we update the user-level RNN in AHNQS with an attention mechanism as follows:

$$h_{t,u} = \mathit{RNN}_{\mathit{user}}(h_{t-1,u}, C_t), \tag{2.11}$$

where C_t is the attentive representation of the session state, a weighted sum of the hidden states $h_{j,t}$ from the session-level RNN, which is generated as:

$$C_t = \sum_{j=1}^{N_t} \alpha_{jt} h_{j,t}, \tag{2.12}$$

where α_{jt} is the normalized attention score for the j-th query in session $\mathit{session}_t$, which is interpreted as the contribution of the query to the preference of the user:

$$\alpha_{jt} = \frac{\exp(e_{jt})}{\sum_{k=1}^{N_t} \exp(e_{kt})}, \tag{2.13}$$

where $e_{jt} = h_{t-1,u}^{\mathrm{T}} \mathbf{W}_a h_{jt}$ is the initial attention score computed with user state $h_{t-1,u}$ and hidden state h_{jt} in a session-level RNN. The parameters $\mathbf{W}_a$ can be jointly trained with the other components of AHNQS as the attention mechanism allows the gradient of the loss function to be back propagated.

For the task of session-based query suggestion, the final hidden state of the session-level RNN can express a summary of the whole sequence behavior, which is called the *global session state*. As the attentive representation of the session state can adaptively select the important items in the current session to capture the user's main purpose, we regard C_t in Eq. (2.11) as the local session state. While the global session state indicates users' current search intents explicitly, the local session state denotes users' general preferences towards the queries in the current session. Thus, we combine the global and local session state by concatenating them to form an extended representation of a session as shown in Figure 2.4. We denote the global session state as $S_t^{global} = h_{N_t,t}$ and the local session state as $S_t^{local} = C_t$. Then, the combined session state can

be generated by

$$S_t^{comb} = \left[S_t^{global}; S_t^{local}\right] \mathbf{W}_b^{\mathrm{T}} = \left[h_{N_t,t}; \sum_{j=1}^{N_t} \alpha_{jt} h_{j,t}\right] \mathbf{W}_b^{\mathrm{T}}, \tag{2.14}$$

where $\mathbf{W}_b$ is used to keep the dimension of S_t^{comb} the same as S_t^{global} and S_t^{local}. Then, the hidden states in the user-level RNN in Eq. (2.11) can be updated with the combined session state as:

$$h_{t,u} = \mathit{RNN}_{\mathit{user}}(h_{t-1,u}, S_t^{comb}), \tag{2.15}$$

So far, we have developed the AHNQS model, which combines a hierarchical user-session RNN and an attention mechanism for query suggestion; the hierarchical structure models the user's short- and long-term search behavior, while the attention mechanism captures the user's search preference.

The training process of AHNQS with combined session states is outlined in Algorithm 1. We first initialize the parameters in the session-level RNN and user-level RNN in step 1. Then, for a session-level RNN, we initialize the first hidden state in $session_t$ with the former user state U_{t-1} in step 6 and produce the hidden states in step 7. The prediction score for the next query is generated in step 9 and the loss of the whole network is calculated in step 10, which can be used during backpropagation to optimize the parameters. As for the user-level RNN, we calculate the attention weights with step 12 and step 13. Then we generate the combined session state S_t^{comb} for $session_t$ from step 14 and step 16. Finally, the t-th hidden state $h_{t,u}$ and user state U_t in the user-level RNN are calculated in step 17 and step 18.

2.4 Experiments

We conduct our experiments on the AOL dataset to examine the effectiveness of AHNQS. We first list the research questions and the models used for comparison. After that, the datasets and experimental setup are described.

2.4.1 Research questions

To answer **RQ1**, we decompose it in four more fine-grained questions:

(RQ1.1) Do the hierarchical structure and attention mechanism incorporated in AHNQS help to improve the performance of the neural query suggestion model and outperform the state-of-the-art?

(RQ1.2) What is the impact on query suggestion performance of session length, i.e., short vs. medium length vs. long sessions?

(RQ1.3) What is the impact on query suggestion performance of different session states, i.e., global vs. local vs. combined?

(RQ1.4) How does the performance of AHNQS vary across users with different numbers of search sessions?

Algorithm 1 Attention-based Hierarchical Neural Query Suggestion

Input: Epoches: training iterations;
U: user set;
$u(session)$: the session set of user u;
$session_t$:the t-th session in $u(session)$;
$query_{i,t}$: the i-th query in $session_t$;
N_t: the number of queries in $session_t$;
N_S: the number of negative sampled query candidates;
Output: the optimized parameters in session-level RNN and user-level RNN.

randomly initialize the parameters and the hidden states in session-level RNN and user-level RNN.
for epoch in range(Epoches) **do**
  for $u \in U$ **do**
    for $session_t \in u(session)$ **do**
      for $query_{i,t} \in session_t$ **do**
        $h_{0,t} = \tanh(\mathbf{W} \cdot U_{t-1} + \mathbf{b_0})$; %% initialization strategy for session-level RNN
        $h_{i,t} = \mathit{RNN}_{\mathit{session}}(h_{i-1,t}, q_{i,t})$;
        $s_{i,t} = g(\mathbf{W}_i h_{i,t})$; %% prediction score of next query
        $Loss = \frac{1}{N_S} \cdot \sum_{j=1}^{N_S} \sigma(s_{j,t} - s_{i,t}) + \sigma(s_{j,t}^2)$; %% loss calculation
        use back propagation to optimize the parameters.
      end for
      $e_{jt} = h_{t-1,u}^{\mathrm{T}} \mathbf{W}_a h_{jt}$;
      $\alpha_{jt} = \frac{\exp(e_{jt})}{\sum_{k=1}^{N_t} \exp(e_{kt})}$; %% attention mechanism
      $S_t^{local} = \sum_{j=1}^{N_t} \alpha_{jt} h_{j,t}$;
      $S_t^{global} = h_{N_t,t}$;
      $S_t^{comb} = [S_t^{global}, S_t^{local}] \mathbf{W}_b^{\mathrm{T}}$;
      $h_{t,u} = \mathit{RNN}_{\mathit{user}}(h_{t-1,u}, S_t^{comb})$;
      $U_t = h_{t,u}$;
    end for
  end for
end for
return the parameters in session-level RNN and user-level RNN.

2.4.2 Model summary

As AHNQS is based on a neural network to capture the dependencies between queries and users, we compare it with the state-of-the-art neural models for query suggestion. As an aside, Sordoni et al. [115] incorporate a neural query suggestion model as a feature into a learning to rank approach and, hence, their approach belongs to the feature engineering approaches we do not compare with. We consider the following baselines for comparison:

ADJ original co-occurrence-based query suggestion method [56];

Table 2.1: An overview of models discussed in this chapter.

Model	Description	Source
ADJ	An original co-occurrence-based query suggestion method.	[56]
NQS	A simple session-based RNN method for query suggestion.	[51], Section 2.3.1
HNQS	A hierarchical structure with user-session-level RNN for query suggestion.	[98], Section 2.3.2
AHNQS$_{local}$	An attention-based hierarchical RNN model for query suggestion, using the local session state S_t^{local}.	This chapter, Section 2.3.3
AHNQS$_{combined}$	An attention-based hierarchical RNN model for query suggestion, using the combined session state $S_t^{combined}$.	This chapter, Section 2.3.3

NQS a simple session-based RNN method for query suggestion [51], see Section 2.3.1;

HNQS a hierarchical structure with user-session-level RNN for query suggestion, see Section 2.3.2.

In addition, we consider two variants of our attention-based hierarchical neural query suggestion model:

AHNQS$_{local}$ an attention-based hierarchical RNN model for query suggestion, using the local session state S_t^{local}, see Section 2.3.3;

AHNQS$_{combined}$ an attention-based hierarchical RNN model for query suggestion, using the combined session state $S_t^{combined}$, see Section 2.3.3.

We list all the models used for comparison in Table 2.1. It should be noticed that if we only use a global session state to update the user-level RNN, AHNQS is the same as the HNQS model. Thus, in the experiments on which we report below, we use HNQS to denote the model that only has a global session state.

2.4.3 Datasets and experimental setup

Dataset

We use the AOL query log and preprocess the dataset following [40]. Queries are separated into sessions by 30 minutes of inactivity. We remove queries with less than 20 occurrences and keep sessions whose length is larger than 5 as well as users with at least 5 sessions to provide sufficient user-session information. The training set consists all but the last 30 days in the search history; the test set consists of the last 30 days in

Table 2.2: Dataset statistics.

Variable	Training	Test
# Queries	1,545,543	576,817
# Unique queries	61,641	33,519
# Sessions	166,414	67,716
# Users	23,308	19,255
Average # queries per session	9.28	8.52
Average # sessions per user	7.14	3.52

Table 2.3: Parameters used for each model.

Model	Batch	Dropout	Learning rate	Momentum
NQS	50	0.5	0.01	0.0
HNQS	50	0.1	0.1	0.0
AHNQS	50	0.1	0.1	0.0

the log after filtering out queries that do not exist in the training set. Table 2.2 details the statistics of the dataset used.

Parameter settings

We use GRUs as the RNN units and optimize the neural models using the TOP1 loss function and AdaGrad with momentum for 20 epochs. The number of hidden units is set to 100 in all cases and we use dropout regularization. We optimize the hyperparameters by running 100 experiments at randomly selected points of the parameter space. Optimization is done on a validation set, which is partitioned from the training set with the same procedure as the test set. We summarize the best performing parameters in Table 2.3.

In addition, in order to answer **RQ1.4**, we plot distributions of users with different numbers of sessions in the AOL dataset in Figure 2.5. The x-axis denotes the number of sessions while the y-axis indicates the number of users corresponding to the sessions. We see that although the maximum number of sessions that a user has is more than 150, the majority of users in the dataset only have a small number of sessions, which we regard as "inactive users." In detail, 60.49% of the users have fewer than 8 sessions, 32.33% from 8 to 20 sessions, and only 7.18% of the users have more than 20 sessions. Thus, it is meaningful to investigate how the performance of the AHNQS models varies with different numbers of users' sessions.

Training and evaluation

Research in natural language processing tasks often uses a fixed number of sequential words in a sentence to form a session and then put those sessions next to each other to form mini-batches. However, in query suggestion, the lengths of sessions are different and our goal is to capture how a session evolves over time, thus it is not suitable to set a fixed length for sessions. Besides, different users also have different numbers of

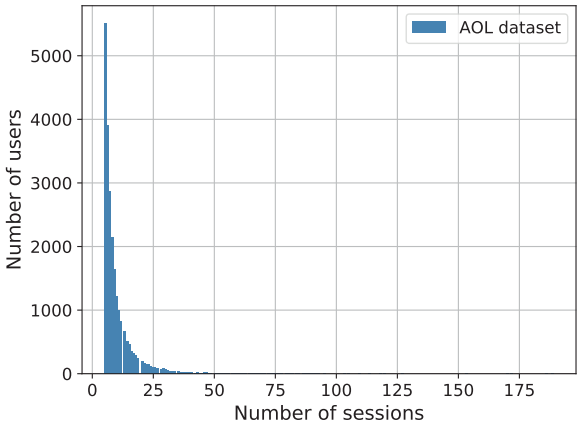


Figure 2.5: Distribution of users with varying numbers of sessions in the AOL dataset.

sessions and we need to deal with sessions of the same user in order. Hence, we use parallel mini-batches with the identification of users and sessions following [51]. We first group sessions by users and then sort sessions within each group by time. Then we order the users at random. Next, the first item of the first session of the first N users constitutes the first mini-batch. The second item in each session serves as the output for the first mini-batch and also constitutes the next mini-batch. When a session ends, its next session of the user comes for training. When the sessions of a user have been processed completely, the next user is put in its place in the next mini-batch.

During training, we apply dropout in three ways: the calculation of the hidden states in the session-level RNN, the calculation of the hidden states in the user-level RNN, and the initialization for the first hidden state in the session-level RNN with a user state.

We evaluate the models by providing queries in a session one by one and measure the ranking performance of query suggestions with MRR and Recall on the test set. The source codes for our model are available on our public repository.[1]

2.5 Results and Discussion

2.5.1 Performance of query suggestion models

To answer RQ1.1, we examine the query suggestion performance of the baselines as well as the AHNQS$_{local}$ and AHNQS$_{combined}$ models. Table 2.4 presents the results. As shown in Table 2.4, amongst the baselines, ADJ outperforms NQS, with 9.74% and 12.58% improvements in terms of Recall@10 and MRR@10, respectively. This may be due to the fact that the NQS model (without knowing about individual users) fails to capture information from the past search history. HNQS shows improvements

[1] `https://bitbucket.org/WanyuChen/ahnqs/`

Table 2.4: Performance of query suggestion models. The results by the best baseline and the best performer in each column are underlined and in boldface, respectively. Statistical significance of pairwise differences of AHNQS$_{local}$ and AHNQS$_{combined}$ vs. the best baseline) is determined by a t-test ($^{\blacktriangle}$/$^{\blacktriangledown}$ for α = .01).

Model	Recall@10	MRR@10
ADJ	.7072	.6922
NQS	.6444	.6148
HNQS	.8138	.7874
AHNQS$_{local}$	.8618$^{\blacktriangle}$	.8514$^{\blacktriangle}$
AHNQS$_{combined}$	**.8924**$^{\blacktriangle}$	**.8859**$^{\blacktriangle}$

over ADJ of up to 15.07% and 13.75% in terms of Recall@10 and MRR@10, respectively. This demonstrates that the hierarchical structure can effectively incorporate a given user's previous search behavior and then improve the accuracy. Thus, in general, HNQS performs best among the baselines we consider and, hence, it is used as the baseline of choice in our later experiments.

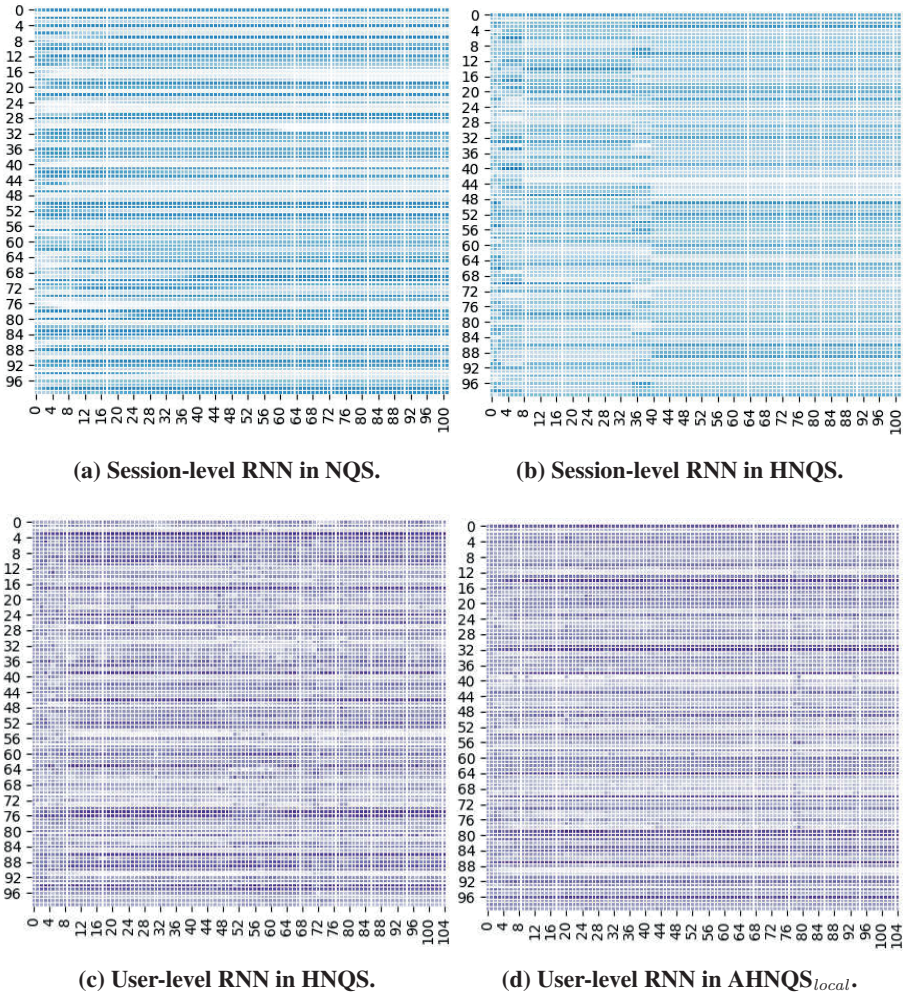
Regarding our newly introduced models, the AHNQS models display a competitive performance compared to HNQS and ADJ. In particular, AHNQS$_{local}$ outperforms ADJ by 21.86% in terms of Recall@10 and 22.99% in terms of MRR@10, and beats HNQS by 5.9% in terms of Recall@10 and 8.13% in terms of MRR@10, respectively. The improvements are significant when tested with a t-test at the $\alpha = .01$ level. These findings indicate that attention can strengthen the model's ability to rank query suggestion candidates effectively. The best performance is obtained by AHNQS$_{combined}$; its improvements over AHNQS$_{local}$ are 3.55% in terms of Recall@10 and 4.05% in terms of MRR@10, respectively. This can be attributed to the utility of the combined session state.

Interestingly, we see that the improvements of our AHNQS models against the baselines in terms of MRR@10 are more obvious than those in terms of Recall@10. For example, AHNQS$_{combined}$ shows an improvement over HNQS by 12.51% in terms of MRR@10 and 9.66% in terms of Recall@10. This means that the attention mechanism cannot only help to suggest the right query, but has a competitive performance in ranking it at the top position in the list of query suggestion.

2.5.2 Visualization of the hierarchical structure and attention mechanism

To determine the impact of the hierarchical structure and attention mechanism, we consider a sample session and user, and compare the hidden states of an RNN in NQS and HNQS, respectively in Figure 2.6a and 2.6b, as well as the hidden states of an RNN in HNQS and AHNQS$_{local}$, respectively in Figure 2.6c and 2.6d. The lighter the area in the plot, the more important the information is.

In Figure 2.6a and 2.6b, the session contains 102 queries (x-axis); the number of hidden units is 100 (y-axis). Compared with Figure 2.6a, we can see that the hierarchical structure modifies the user's search intent especially at the first positions in



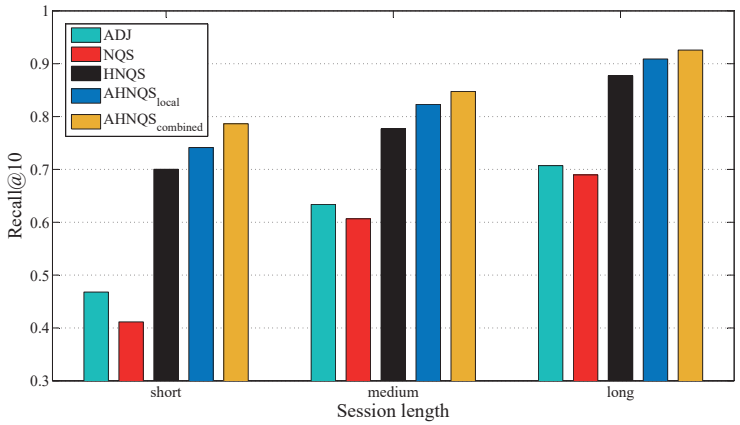
(a) Session-level RNN in NQS. **(b) Session-level RNN in HNQS.**

(c) User-level RNN in HNQS. **(d) User-level RNN in AHNQS$_{local}$.**

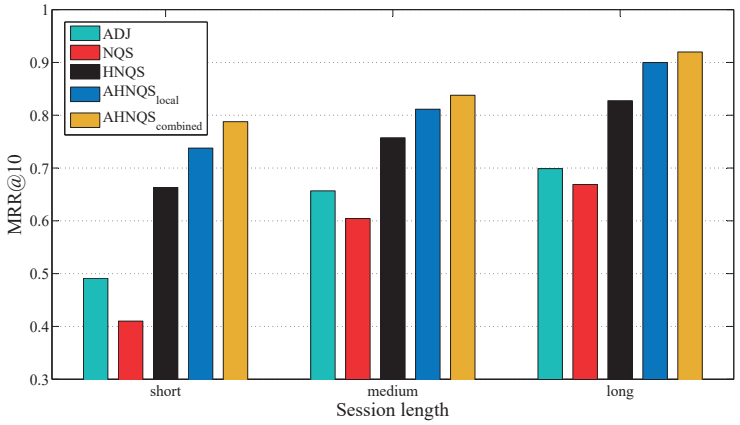
Figure 2.6: Visualizing the hierarchical structure ((a) and (b)) and attention mechanism ((c) and (d)). The lighter the area in the plot, the more important the information is.

a session in Figure 2.6b. This is because we initialize the session-level RNN with a user state U_t in HNQS. There is a fluctuation around the 36-th to 40-th queries in Figure 2.6b, which may be due to the fact that we use a GRU unit inside the session-level RNN to transport the user state information within the network.

Turning to the attention mechanism (Figure 2.6c and 2.6d), we select a user with 105 sessions (x-axis) and the number of hidden units in the user-level RNN is set to 100 (y-axis). Compared with Figure 2.6c, the user's preference towards different information is more equally distributed inside the user-level RNN in Figure 2.6d. Moreover, going from left to right there are fewer abrupt shifts form high interest (light) to low



(a) Performance in terms of Recall@10.



(b) Performance in terms of MRR@10.

Figure 2.7: Effect on performance of five models in terms of Recall@10 and MRR@10 with different session lengths, tested on the AOL log.

interest (dark) areas, or vice versa, in Figure 2.6d than in Figure 2.6c: the attention mechanism can help to describe a user's long-term search preferences towards different topics.

2.5.3 Impact of the current session length

For RQ1.2, we expect the current session length to have an impact on the performance of query suggestion models. We report separate results for short (2 queries), medium (3 or 4 queries), and long current sessions (at least 5 queries) on the test set in Figure 2.7.

Clearly, as the session length increases, the performance in terms of Recall@10 of all query suggestion models improves and our AHNQS$_{combined}$ model always achieves

the highest scores. As for the baselines, ADJ outperforms NQS; the margin between them goes down as the session length increases. HNQS performs better than both ADJ and NQS across all session lengths. The improvements of HNQS and AHNQS against ADJ (as well as NQS) are more obvious for short sessions than for long sessions. This is due to the fact that when predicting a user's search intent at the first position of a session, the hierarchical structure within RNN models can provide effective information from a user's past search history and thus can improve the accuracy for query suggestion.

For MRR@10, a similar trend is shown in Figure 2.7b. Compared with the results in Figure 2.7a, the AHNQS models show a larger improvement over HNQS. For AHNQS$_{local}$, the improvements are 11.23%, 7.13% and 8.74% in terms of MRR@10, for short, medium and long sessions, respectively, vs. improvements of 5.88%, 5.97% and 3.61% for Recall@10. As for AHNQS$_{combined}$, the improvements are 18.77%, 10.61% and 11.15% in terms of MRR@10, for short, medium and long sessions, respectively, vs. improvements of 12.31%, 9.12% and 5.53% for Recall@10. This confirms our intuition about attention mechanisms, i.e., that they help to improve the precision for query suggestion.

2.5.4 Impact of different session states

For RQ1.3, in order to investigate the impact of different session states used in our hierarchical structure, i.e., the global session state, the local session state, and the combined session state, we compare the performance of HNQS, AHNQS$_{local}$ and AHNQS$_{combined}$ by varying the hidden state dimension in the session-level RNN from 50 to 100. Here, we should remind the reader that HNQS can be regarded as the model that only has a global session state, as we have explained in Section 2.4.2. We present the Recall and MRR scores when the cutoff N is set to 5, 10 and 15 in Table 2.5 as tested on the AOL log.

As shown in Table 2.5, we find that the HNQS and AHNQS$_{local}$ models, which only use a single way to represent the session state, do not perform well in terms of the two metrics. However, AHNQS$_{combined}$ yields the best performance under all experimental settings and its improvements over HNQS are significant when tested with a t-test at $\alpha = .05$. This indicates that merely considering the final hidden state or the sequential behavior in the current session may not allow the network to learn a good query suggestion model. In particular, AHNQS$_{local}$ performs better than HNQS on different hidden state dimensions over the three cutoff settings. This demonstrates that the local session state can give a better description of a user's search intent than the global session state.

Regarding the impact of different hidden state dimensions, we can see that the query suggestion performance of the three models improves when the hidden state dimension increases from 50 to 100. However, the AHNQS$_{combined}$ model shows less improvements than the other two models. For example, the improvements of $D = 100$ over $D = 50$ are 1.97%, 1.21% and 0.98% of HNQS, AHNQS$_{local}$ and AHNQS$_{combined}$ in terms of Recall@10, and 2.98%, 2.56% and 2.33% in terms of MRR@10, respectively. This can be explained by the fact that AHNQS$_{combined}$ takes advantage both of the local and global session state, and thus can generate a better

Table 2.5: Performance comparison among query suggestion models with different session states when the cutoff N is 5, 10 and 15, respectively. D denotes the hidden state dimension in session-level RNN. The results by the best performer in each column are in boldface. Statistical significance of pairwise differences of AHNQS$_{combined}$ vs. HNQS is determined by a t-test ($^{\blacktriangle}$/$^{\blacktriangledown}$ for α = .05).

(a) Performance comparison at $N = 5$

	$D = 50$		$D = 100$	
Model	Recall@5	MRR@5	Recall@5	MRR@5
HNQS	.7819	.7537	.7977	.7816
AHNQS$_{local}$	.8279	.8208	.8421	.8499
AHNQS$_{combined}$	**.8701**$^{\blacktriangle}$	**.8629**$^{\blacktriangle}$	**.8790**$^{\blacktriangle}$	**.8837**$^{\blacktriangle}$

(b) Performance comparison at $N = 10$

	$D = 50$		$D = 100$	
Model	Recall@10	MRR@10	Recall@10	MRR@10
HNQS	.7981	.7646	.8138	.7874
AHNQS$_{local}$	.8515	.8301	.8618	.8514
AHNQS$_{combined}$	**.8837**$^{\blacktriangle}$	**.8657**$^{\blacktriangle}$	**.8924**$^{\blacktriangle}$	**.8859**$^{\blacktriangle}$

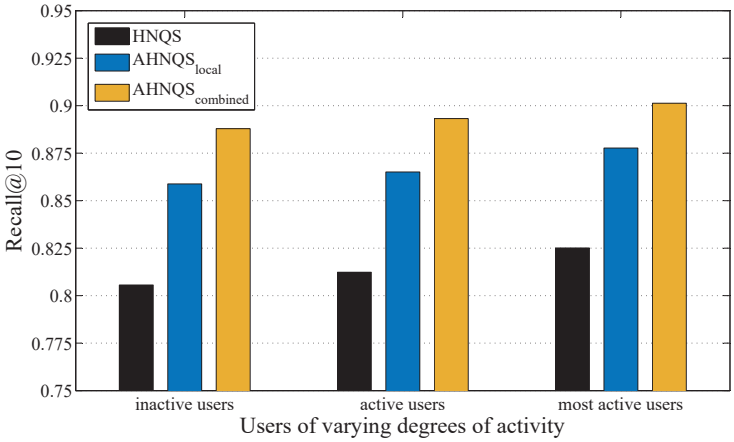
(c) Performance comparison at $N = 15$

	$D = 50$		$D = 100$	
Model	Recall@15	MRR@15	Recall@15	MRR@15
HNQS	.8036	.7673	.8178	.7882
AHNQS$_{local}$	.8524	.8336	.8653	.8537
AHNQS$_{combined}$	**.8866**$^{\blacktriangle}$	**.8671**$^{\blacktriangle}$	**.8978**$^{\blacktriangle}$	**.8864**$^{\blacktriangle}$

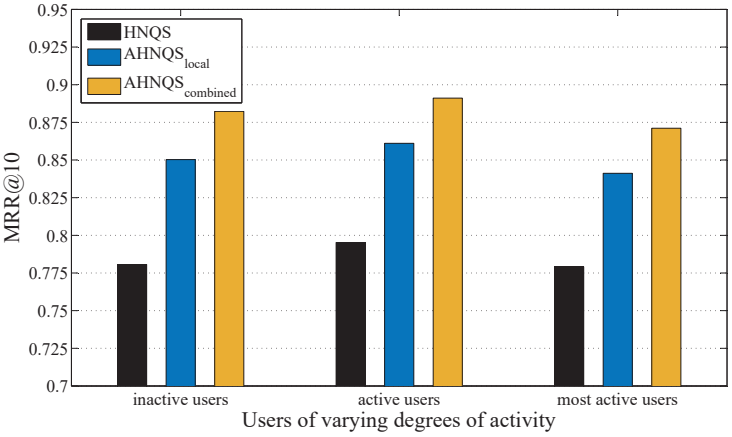
performance with a smaller hidden state dimension. In addition, comparing different cutoff settings, the performances of all models get improved with N increasing from 5 to 15. In addition, the improvements of $N = 15$ over $N = 5$ of our AHNQS models are smaller than of HNQS in terms of both metrics. This can be explained by the fact that the AHNQS models can rank the right query at the top position of the query suggestion list and thus the performance is affected little by an increase in the cutoff.

2.5.5 Scalability with different numbers of users' sessions

In Figure 2.5, we have shown that most users have a very limited number of sessions (≤ 17) while there are some that have a particularly large number of sessions (≥ 150), which we regard as "inactive users" and "most active users," respectively. It is meaningful to investigate how the performance of AHNQS and HNQS varies across users with different numbers of sessions (RQ1.4). Following [96], we look at the perfor-



(a) Performance in terms of Recall@10.



(b) Performance in terms of MRR@10.

Figure 2.8: Performance of three models across users with different numbers of search sessions, tested on the AOL log.

mance for users of varying degrees of activity, measured by percentile. In Figure 2.8, we first rank the users according to their number of sessions. The "inactive users" mark shows the mean performance across the bottom 10% of users, who are least active but constitute the majority of the dataset; the "active users" mark shows the mean performance for the remaining users except the top 10% most active users; the "most active users" mark shows the mean performance of the top 10% most active users, which form the smallest subset of the dataset.

As shown in in Figure 2.8, the AHNQS models outperform the best baseline model HNQS for users across all activity levels, i.e., inactive users, active users as well as

most active users. In addition, the AHNQS$_{combined}$ model always achieves the best performance in terms of Recall@10 and MRR@10. In particular, AHNQS$_{combined}$ shows larger improvements over the HNQS model for the "inactive users" than for the "active users" as well as the "most active users." For example, when the number of user sessions increases, i.e., testing on the group of "inactive users" to "active users" and "most active users" accordingly, the improvements decrease from 10.22% to 9.95% and 9.23% in terms of Recall@10, and from 13.01% to 12.04% and 11.81% in terms of MRR@10 on the AOL dataset, respectively. This demonstrates that AHNQS$_{combined}$ can effectively model a user's search intent even with a small number of user sessions.

We zoom in on the scores in terms of two metrics. In Figure 2.8a, the performance of the three models in terms of Recall@10 monotonically improves when the number of user sessions goes up. However, as shown in Figure 2.8b, the performance in terms of MRR@10 decreases when the number of user sessions increases, cf. "active users" vs. "most active users." That may be due to the fact that users with lots of search sessions may have very diverse search intents and thus it is difficult to improve the ranking accuracy of the models for such users. This naturally suggests that in future work we should consider extending the AHNQS$_{combined}$ model by integrating other strong signals for personalized query suggestion, such as dwell time or user profiles.

2.6 Conclusion

We propose an attention-based hierarchical neural query suggestion model (AHNQS) that combines a hierarchical user-session RNN with an attention mechanism. The hierarchical structure, which incorporates a session-level and a user-level RNN, can model both the user's short-term and long-term search behavior effectively, while the attention mechanism captures a user's preference towards certain queries over others. For the session-level RNN, a combined session state is applied to capture both the user's sequential behavior and their main purpose in the current session, which is then used as the input for the user-level RNN. For the user-level RNN, we use the final hidden state to initialize the next session-level RNN, which can automatically transport the user information within the network.

Our experimental results show that: (1) the proposed AHNQS model helps to boost query suggestion performance in terms of MRR and Recall across sessions with various lengths; (2) using the combined session state in the AHNQS model achieves better performance than only using the local session state; (3) the AHNQS model yields better performance than the best baseline for inactive, active, as well has highly active users. In summary, our answer to **RQ1** is that learning from a user's submitted queries in a hierarchical structure with an attention mechanism can help to capture his current search intent.

The theoretical implication of our research is that a hierarchical model that is combined with an attention mechanism for query suggestion can capture the dynamic search intent of a user. The practical implication of our research is that the improvements of AHNQS over the best baseline model are significant; they are especially prominent for short sessions and for inactive users with few search sessions, which is a realistic setup that online services are always confronted with [101]. Compared to

the state-of-the-art, AHNQS achieves improvements of 9.66% and 12.51% in terms of Recall@10 and MRR@10, respectively, on average over all users, and of 10.22% and 13.01% for inactive users.

As to future work, we plan to evaluate our model on other datasets so as to verify its robustness. We also want to investigate the performance of AHNQS when combining semantic similarity within the hierarchical structure [89], e.g., with different encoding methods for input queries [63, 71, 72]. As for attention strategies, we plan to apply different attention mechanisms to explore users' search intents, e.g., self-attention. In addition, signals for personalized query suggestion that we do not consider in this chapter, such as user profiles and dwell time, could also be incorporated in our AHNQS model.

Next in Chapter 3, we consider recommendation instead of search and focus on items instead of queries.

3

Joint Neural Collaborative Filtering for Recommendation

In Chapter 2, we have learned users' search intent through their interaction logs and thus make query suggestions in search engines. In this chapter, we investigate how to learn users' preferences from user-item interactions in recommender systems. We aim to make general recommendations by analyzing users' long-term behavior. The proposed method helps us to provide a positive answer to the following research question asked in Chapter 1: **RQ2**: Can we learn users' general preferences by modeling non-linear user-item relationships as well as characteristics based on their interactions?

3.1 Introduction

Recommender systems are an effective solution to help people cope with an increasingly complex information landscape. Collaborative Filtering (CF) approaches have been widely investigated and used for personalized recommendation [3, 148]. Many traditional CF techniques are based on Matrix Factorization (MF) [148]. They characterize users and items by latent factors that are extracted from the user-item rating matrix. In the latent space, traditional CF methods, such as the Latent Factor Model (LFM) [68], often predict a user's preference for an item with a linear kernel, i.e., a dot product of their latent factors, which may not be able to capture the complex structure of user-item interactions well.

Recently introduced Deep Learning (DL)-based approaches to recommender systems overcome shortcomings of conventional approaches to recommender systems, such as dynamic user preferences and intricate relationships within the data itself, and are able to achieve high recommendation quality. Today's DL-based approaches to recommender systems mostly use DL to explore auxiliary information, e.g., textual descriptions of items or audio features of music, which is then used to model item features [64, 130, 131]. For the user-item rating matrix, recent work mostly continues to use traditional MF-based approaches. Restricted Boltzmann Machines (RBMs) [106] seem to have been the first model to use neural networks to model the user-item rating matrix and obtain competitive results over traditional methods; it is a two-layer

This chapter was published as [25].

network rather than a deep learning structure. Another recent approach, Collaborative Denoising Auto-Encoder (CDAE) [140], is mainly designed for rating prediction with a one-hidden layer neural network. Neural Collaborative Filtering (NCF) [46] uses deep neural networks for learning the interaction function from data with multi-layer perceptrons, yet it does not explore users' and items' features that are known to be helpful in improving CF recommendation performance. CDAE and NCF only exploit implicit feedback for recommendations instead of explicit rating feedback. Deep Matrix Factorization (DMF) [55] models the user-item rating matrix with a neural network that maps the users' and items' features into a low-dimensional space with non-linear projections; it uses an inner product to compute interactions between users and items, and applies the same linear kernel (i.e., dot product) as LFM [68].

We hypothesize that DL should be able to effectively capture both non-linear and non-trivial user-item relationships as well as users' (items') characteristics [148]. We propose a Joint Neural Collaborative Filtering (J-NCF) model that enables two processes, i.e., feature extraction and user-item interaction modeling, to be trained jointly in a unified DL framework. The J-NCF model contains two main networks for recommendation. The first network uses the rating information of a user (an item) as the network input, and outputs a vector representation for the user (the item). Then, using the connection of a user's and an item's vectors as input, the second neural network models the user-item interactions and outputs the prediction of the corresponding rating of the user and item. Thus, these two networks can be coupled tightly and trained jointly in a unified structure. Interaction modeling can optimize the feature learning process and more accurate feature representations can, in turn, improve the user-item interaction prediction. We take both implicit and explicit feedback, point-wise and pair-wise loss into account to enhance the prediction performance. In contrast, previous neural approaches such as CDAE, NCF and DMF are all optimized only with point-wise loss functions and leave dealing with pair-wise loss as future work.

To the best of our knowledge, in the area of recommender systems ours is the first attempt to use a joint neural network to tightly couple feature learning and interaction modeling with the rating matrix. J-NCF allows these two processes to optimize each other through joint training and thereby improve the recommendation performance.

Our experiments on real-world datasets, including the MovieLens dataset and the Amazon Movies dataset, show that J-NCF outperforms the state-of-the-art baselines in prediction accuracy, with improvements of up to 8.24% on the MovieLens 100K dataset, 10.81% on the MovieLens 1M dataset, and 10.21% on the Amazon Movies dataset in terms of HR@10. NDCG@10 improvements are 12.42% on the MovieLens 100K dataset, 14.24% on the MovieLens 1M dataset, and 15.06% on the Amazon Movies dataset, respectively, over the best baseline model. In addition, we investigate the scalability and sensitivity of J-NCF with different degrees of sparsity and different numbers of users' ratings. Our experimental results indicate that J-NCF achieves competitive recommendation performance when compared to the best state-of-the-art model.

Our contributions in this chapter are:

(1) We design a Joint Neural Collaborative Filtering model (J-NCF) for recommendation, which enables deep feature learning and deep user-item interaction mod-

eling to be coupled tightly and jointly optimized in a single neural network.

(2) We design a new loss function that explores the information contained in both point-wise and pair-wise loss as well as implicit and explicit feedback.

(3) We analyse the recommendation performance of J-NCF as well as baseline models and find that J-NCF consistently yields the best performance. J-NCF also shows competitive improvements over the best baseline model when applied with inactive users and different degrees of data sparsity.

We summarize related work in Section 3.2. Our approach, J-NCF, is described in Section 3.3. Section 3.4 presents our experimental setup. In Section 3.5, we report our results to demonstrate the recommendation performance of J-NCF. We also investigate the scalability and sensitivity of our model as well as other baselines in Section 3.6. Finally, we conclude our work in Section 3.7, where we also suggest future research directions.

3.2 Related Work

We first look back to traditional approaches to recommender systems in Section 3.2.1, that focus on modeling the similarity between users (items) for recommendation. Then, as applying deep learning techniques into recommender systems is gaining momentum due to its state-of-the-art performance and high-quality recommendations, we summarize recent work on deep learning-based recommender systems in Section 3.2.2 that can provide a better understanding of user's demands, item's characteristics as well as historical interactions between them by extracting the features of items with auxiliary information, e.g., the content of movies.

3.2.1 Traditional recommender systems

In many commercial systems, "best bet" recommendations are shown, but the predicted rating values are not. This is usually referred to as a top-N recommendation task, where the goal of the recommender system is to find a few specific items that are supposed to be most appealing to the user. A similar prediction schema, denoted as Top Popular (Item-pop), recommends the top-N items with the highest popularity (largest number of ratings).

Most top-N recommender systems are based on collaborative filtering [3], where recommendations rely on past behavior (ratings) from users, regardless of domain knowledge [116]. We group these CF approaches into two categories, i.e., neighborhood-based methods [79, 108] and latent factor-based models [61, 68]. Neighborhood-based models share the typical merits of CF, which concentrate on exploring the similarity among either users or items. For instance, two users are similar because they have rated similarly the same set of items. A dual concept of similarity can be defined among items. Latent factor-based approaches generally model users and items as vectors in the same "latent factor" space by means of a reduced number of hidden factors. In such a space, users and items are directly comparable: the rating of a user u on

an item i is predicted by the proximity (e.g., inner-product) between the related latent factor vectors.

For neighborhood-based models, algorithms that are centered around user-user similarity typically predict the rating by a user based on the ratings expressed by other users similar to her about such item. On the other hand, algorithms centered around item-item similarity compute the user preference to an item based on her own ratings to similar items. The similarity between item i and item j is measured as the tendency of users to rate items i and j similarly. It is typically based either on the cosine, the adjusted cosine, or (most commonly) the Pearson correlation coefficient [108]. The kNN (k-nearest-neighborhood) approach is a representative enhanced neighborhood model [2], which considers only the k items rated by user u that are the most similar to the item i when predicting the rating r_{ui}. kNN-based approaches discard items that are poorly correlated to the target item, thus decreasing noise for improving the quality of recommendations. Neighborhood-baesd approaches are similar to the item-item model for user personalization, which is different from our approach based on the user-item model [108]. Thus, we focus on the latent factor modeling approach.

Most research on latent factor modeling is based on factoring the user-item rating matrix, which is known as Singular Value Decomposition (SVD) [68]. SVD factorizes the user-item rating matrix to a product of two lower rank matrices, one containing the "user factors," the other containing the "item-factors." Then, with an inner product and biases (b_{ui}), the user's preference towards an item can be generated, i.e.,

$$\hat{y}_{ui} = b_{ui} + \mathbf{z_u}\mathbf{z_i}^{\mathrm{T}}, \tag{3.1}$$

where $\mathbf{z_u}$ and $\mathbf{z_i}$ denote the "user factors" and "item-factors," respectively.

Since the conventional SVD is undefined in the presence of unknown values, i.e., missing ratings, several solutions have been proposed. Earlier work addresses this issue by filling the missing ratings with a baseline estimation [109]. However, this leads to a very large, dense user rating matrix, where the factorization process becomes computationally infeasible. Recent work learns factor vectors directly on known ratings through a suitable objective function that minimizes a prediction error. The proposed objective functions are usually regularized in order to avoid overfitting [95]. Typically, gradient descent is applied to minimize the objective function. An advantage of SVD-based approaches is that they can provide recommendations for new users after given their ratings towards some items without reconstructing the parameters of the models. Thus for a new user, SVD-based approaches can provide recommendations immediately according to his current ratings.

Another model based on SVD, SVD++ [67], incorporates both explicit and implicit feedback, and shows improved performance over many MF models. This is consistent with our motivation of combining explicit and implicit feedback in J-NCF. However, applying traditional MF methods to sparse ratings matrices can be a non-trivial challenge with high computational costs for decomposing the rating matrix.

Many traditional recommender systems apply a linear kernel with an inner product of user and item vectors to model user-item interactions. Linear functions may not be able to give an accurate description of the characteristics of users (items) and user-item interactions: previous work has pointed out that non-linearities have potential advantages for improving the performance of recommender systems with extensive

experiments [76, 110, 140].

3.2.2 Deep learning-based recommender system

DL-based recommender systems can be divided into two categories, i.e., single neural network models and deep integration models, depending on whether they rely solely on deep learning techniques or integrate traditional recommendation models with deep learning [7, 43, 57, 80, 91, 116, 137, 148, 152].

For the first category, RBM [83, 106, 124] is an early neural recommender system. It uses a two-layer undirected graph to model tabular data, such as users' explicit ratings of movies. RBM targets rating prediction, not top-N recommendation, and its loss function considers only the observed ratings. It is technically challenging to incorporate negative sampling into the training of RBMs [140], which would be required for top-N recommendation. AutoRec [110] uses an Auto-Encoder for rating prediction. It only considers the observed ratings in the loss function, which does not guarantee good performance for top-N recommendation. To prevent the Auto-Encoder from learning an identity function and failing to generalize to unseen data, Denoising Auto-Encoders (DAEs) [76] have been applied to learn from intentionally corrupted inputs. Most of the publications listed so far focus on explicit feedback and, hence, fail to learn users' preference from implicit feedback. CDAE [140] extends DAEs; its input is a user's partially observed implicit feedback. Unlike our work, both DAEs and CDAE use an item-item model for personalization that represents a user with their rated items [108] and the outputs are the item scores decoded from the learned user's representation. Our work is a kind of user-item model, which learns users' as well as items' representations first and then calculates the relevance between them. The proposed J-NCF model is a user-item model that personalizes by modeling user-item interactions. Also, CDAE applies a linear kernel to model the relationship between users and items, whereas a J-NCF applies a non-linear kernel.

Several Convolution Neural Network (CNN)-based recommendation models have been proposed [64, 125, 130]. They primarily use CNNs to extract item features with auxiliary information, e.g., review text or contextual information, which we will incorporate in our future work. As for Recurrent Neural Networks, they are used in recommender systems that address the temporal dynamics of ratings and sequential features [51, 123].

Most closely related to our model is Neural Collaborative Filtering (NCF) [46]. It uses multi-layer perceptrons to model the two-way interaction between users and items, which is meant to capture the non-linear relationship between users and items. Let v_u^{user} and v_u^{item} denote the side information (e.g., the feature information), then, the prediction rule of NCF is formulated as follows:

$$\hat{y}_{ui} = f(U^{\mathrm{T}} \cdot v_u^{user}, V^{\mathrm{T}} \cdot v_u^{item} \mid U, V, \theta), \tag{3.2}$$

where the function $f(\cdot)$ defines the multilayer perceptron, and θ are the parameters of the network. However, NCF randomly initializes the representation of users and items, with just a one-hot identifier of user u and item i respectively, which only explores the users' and items' features in a limited manner. J-NCF adopts a joint neural network structure to capture both user and item features, and user-item relationships, as

we hypothesize that the two parts can be optimized through tight coupling and joint training. In addition, NCF only exploits implicit feedback for item recommendations and ignores explicit feedback.

An extension based on NCF is CCCFNet (Cross-domain Content-boosted Collaborative Filtering neural Network) [77]. The basic building block of CCCFNet is also a dual network (for users and items, respectively). It models the user-item interactions in the last layer with the dot product. Unlike our work, it applies content information with a neural network to capture the user's preferences and item features. In addition, DeepFM (Deep Factorization Machine) [39] is an end-to-end model that seamlessly integrates factorization machine and MLP. However, it also applies content information and thus models higher-order feature interactions via a deep neural network and low-order interactions via a factorization machine. In contrast, J-NCF adopts the rating information to explore both user and item features, which are easier to collect.

As to deep integration models, Collaborative Deep Learning (CDL) [131] is a hierarchical Bayesian model that integrates stacked DAEs into traditional probabilistic MF. It differs from our work in two ways: First, it extracts deep feature representations of items from the content information which we do not explore, and then it uses a linear kernel to model relations between users and items with the dot product of user and item vectors. A well-known integration model is DeepCoNN (Deep Cooperative Neural Network) [151], which adopts two parallel convolutional neural networks to model user behavior and item properties from review texts. In the final layer, a factorization machine is applied to capture their interactions from rating predictions. It alleviates the sparsity problem and enhances model interpretability by exploiting a rich semantic representation of the reviews, which could be investigated in J-NCF as future work.

Wide & Deep learning [29] and DeepFM [39] are two state-of-the-art recommendation approaches with deep learning techniques. While they focus on incorporating various features of users and items, we aim at exploring deep learning methods for pure collaborative filtering systems. Another integration model that is directly relevant to our work is Deep Matrix Factorization (DMF) [55]. It uses a deep MF model with a neural network that maps users and items into a common low-dimensional space. It follows the LFM, which uses the inner product to compute interactions between users and items. This may partially explain why using deep layers does not help to improve the performance of DMF. Unlike DMF, we apply multi-layer perceptrons to model user-item interactions using a combination of user and item feature vectors as input. This does not only help our model to be more expressive in modeling user-item interactions than linear products, but it also helps to improve the accuracy of user and item feature extraction.

On top of the previous work discussed above, our proposed model J-NCF combines feature learning and interaction modeling into an end-to-end trainable neural network, which enables the two processes to be optimized jointly. Besides this, we design a new loss function that combines point-wise and pair-wise losses to explore the integration of different types of information, i.e., both implicit and explicit feedback.

Table 3.1: Main notation used in the chapter.

Notation	Description
U	the set of users
I	the set of items
R_{ui}	an explicit rating of user u to item i
$\mathbf{v_u}$	a vector containing a user's ratings; serves as input to Net$_{user}$
$\mathbf{v_i}$	a vector containing an item's ratings; serves as input to Net$_{item}$
M	the number of unique users
N	the number of unique items
$\mathbf{W_u^x}$	the weight matrix for the x-th layer in Net$_{user}$
$\mathbf{b_u^x}$	the bias for the x-th layer in Net$_{user}$
f_u^x	the activation function for the x-th layer in Net$_{user}$
X	the number of layers in DF network
$\mathbf{W_{ui}^y}$	the weight matrix for the y-th layer in the DI network
$\mathbf{a_{ui}}$	a combination of user and item vectors; serves as input to the DI network
$\mathbf{b_{ui}^y}$	the bias for the y-th layer in the DI network
f_{ui}^y	the activation function for the y-th layer in the DI network
Y	the number of layers in the DI network
$\hat{y}_{ui}$	the predicted score of the interaction between user u and item i
V^+	the set of items that a user rates
V^-	the set of items that are not rated by a user
α	a tradeoff parameter controlling the contributions of the point-wise loss and pair-wise loss

3.3 Approach

The proposed model, J-NCF, has a joint structure with a layer used for modeling users' and items' features (the DF network) and a higher layer used for modeling user-item interactions (the DI network). These two layers can be trained in a joint manner to give a predicted score of a user's interactions with an item with minimum prediction error. We first describe the notation used and then detail J-NCF. We also describe the loss function that we use for optimization.

3.3.1 Problem formulation and notation

First we describe the task of top-N recommendation that we study in this chapter. Suppose that there are M users and N items, denoted as $U = \{user_1, \ldots, user_M\}$ and $I = \{item_1, \ldots, item_N\}$. $R \in \mathbb{R}^{M \times N}$ denotes the rating information, where R_{ui} is the rating given by user $user_u$ to item $item_i$. The task for top-N recommendation is to return a list containing a set of items for an individual user to maximize the user's satisfaction.

The main notation we use in this chapter is listed in Table 3.1.

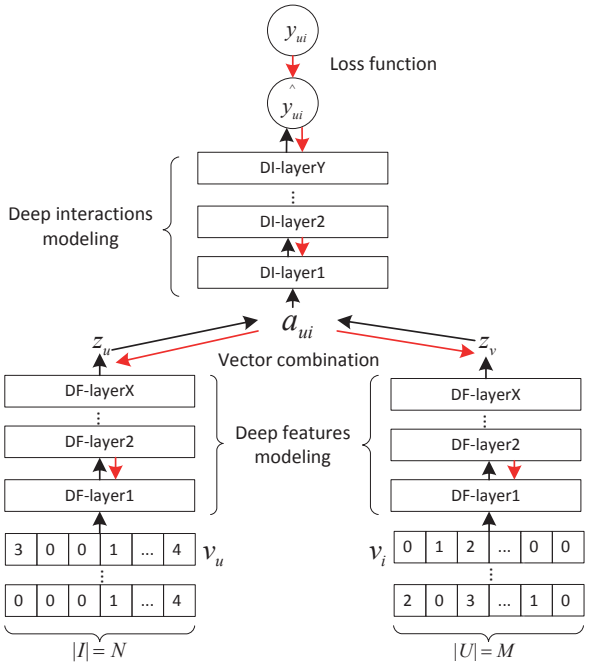


Figure 3.1: Structure of the J-NCF model. Black arrows indicate the forward propagation for calculating the predictions. Red arrows indicate the back propagation for optimizing the parameters. (Best viewed in color.)

3.3.2 Joint neural collaborative filtering

The joint architecture of the proposed J-NCF model is shown in Figure 3.1. The model contains two main networks: a DF network for modeling features and a DI network for modeling interactions between items and users, where the output of the first network serves as the input of the second.

The DF network is used for modeling users' and items' features. It contains two parallel neural networks coupled in the last layer, one network for users (Net$_{user}$) and another for items (Net$_{item}$). We give the ratings of a user and an item as inputs to Net$_{user}$ and Net$_{item}$, respectively, which are defined as $\mathbf{v_u} = \langle y_{u1}, \ldots, y_{uN} \rangle$ and $\mathbf{v_i} = \langle y_{1i}, \ldots, y_{Mi} \rangle$, where

$$y_{ui} = \begin{cases} 0, & \text{for unknown ratings,} \\ R_{ui}, & \text{when explicit feedback is available.} \end{cases} \tag{3.3}$$

We think of ratings as non-trivial explicit feedback from users as different ratings indicate different levels of users preference towards items. Obviously, there are many unknown ratings between users and items indicating non-preference of a user towards an item. Following [46, 55], we regard these unknown ratings as a kind of implicit feedback and mark them as zeroes. When pursuing a top-N recommendation task, we are interested only in a correct item ranking and care less about the exact rating

scores. This grants us some flexibility, like considering all missing values in the user rating matrix as zeros [32]. Thus we can take both explicit and implicit feedback into consideration with Eq. (3.3).

Then, with multi-layer perceptrons (MLP), the initial high-dimensional rating vectors of users and items are mapped to lower-dimensional vectors. Since Net_{user} and Net_{item} only differ in their inputs, we focus on illustrating the process for Net_{user}; the same process is applied for Net_{item} with similar layers. The MLP model in the DF network is defined as:

$$\begin{aligned} \mathbf{z_u^1} &= f_u^1(\mathbf{W_u^1 v_u} + \mathbf{b_u^1}) \\ \mathbf{z_u^2} &= f_u^2(\mathbf{W_u^2 z_u^1} + \mathbf{b_u^2}) \\ &\vdots \\ \mathbf{z_u} &= f_u^X(\mathbf{W_u^X z_u^{X-1}} + \mathbf{b_u^X}), \end{aligned} \tag{3.4}$$

where $\mathbf{W_u^x}$, $\mathbf{b_u^x}$ and f_u^x denote the weight matrix, the bias vector and the activation function for the x-th layer. Here, we use a ReLU as the activation function, as it has been shown to be more expressive than others and can effectively deal with the vanishing gradient problem [46, 55]. X indicates the number of layers used in the DF network. The output of the final layer z_u is a deep representation of the user features; likewise, z_i is the deep representation for the item features.

As to modeling user-item interactions, traditional LFM methods have been widely used. Such methods are based on the dot product of user and item vectors, which models a user's preference with a linear kernel. In order to investigate the differences between non-linear and linear functions in modeling user-item interactions, we propose two ways to obtain fused users' and items' feature vectors $\mathbf{a_{ui}}$ as the input of the DI network:

$$\mathbf{a_{ui}} = \begin{cases} \begin{bmatrix} \mathbf{z_u} \\ \mathbf{z_i} \end{bmatrix}, & \text{concatenation, or} \\ \mathbf{z_u} \odot \mathbf{z_i}, & \text{multiplication.} \end{cases} \tag{3.5}$$

The first way is to concatenate the two input vectors $\mathbf{z_u}$ and $\mathbf{z_i}$, which we regard as a non-linear fusion. The second way is to use the element-wise product of vectors, which uses a linear kernel to generate user-item interactions. Based on these two ways of fusing the input vectors $\mathbf{z_u}$ and $\mathbf{z_i}$, we propose two versions of J-NCF, which we discuss in detail in our experiments.

Generating $\mathbf{a_{ui}}$ is the first step for modeling user-item interactions. However, it is insufficient for modeling the complex relationship between users and items. Thus, we adopt intermediate hidden layers to which $\mathbf{a_{ui}}$ is fed so as to obtain a multi-layer non-linear projection of user-item interactions:

$$\begin{aligned} \mathbf{z_{ui}^1} &= f_{ui}^1(\mathbf{W_{ui}^1 a_{ui}} + \mathbf{b_{ui}^1}) \\ \mathbf{z_{ui}^2} &= f_{ui}^2(\mathbf{W_{ui}^2 z_{ui}^1} + \mathbf{b_{ui}^2}) \\ &\vdots \\ \mathbf{z_{ui}} &= f_{ui}^Y(\mathbf{W_{ui}^Y z_{ui}^{Y-1}} + \mathbf{b_{ui}^Y}), \end{aligned} \tag{3.6}$$

where $\mathbf{W_{ui}^y}$, $\mathbf{b_{ui}^y}$ and f_{ui}^y denote the weight matrix, the bias vector and the activation function for the y-th layer in the DI network. A ReLU is applied again as the activation

function. Y indicates the number of layers used in the network. The output of the network is the predicted score of the interaction between user u and item i:

$$\hat{y}_{ui} = \sigma(\mathbf{h}^{\mathrm{T}}\mathbf{z_{ui}}), \tag{3.7}$$

where the sigmoid function σ can restrict the output in (0,1). $\mathbf{h}$ can be learnt through the training process with back propagation to control the weight of each dimension in $\mathbf{z_{ui}}$.

3.3.3 Loss function

Objective functions for training recommender systems can be divided into three groups: point-wise, pair-wise and list-wise. Point-wise objectives aim at obtaining accurate ratings, which is more applicable in rating prediction tasks [61]. Pair-wise objectives are usually focused on users' preferences towards pairs of items and are usually considered more suitable for top-N recommendation [45, 46, 61, 103]. List-wise objectives are focused on users' interests towards a list of items, which are also used in some deep learning algorithms. We briefly summarize the three groups of loss functions.

We use $\ell(\cdot)$ to denote a loss function and $\Omega(\theta)$ to represent a regularization term that controls the model complexity and encodes prior information such as sparsity, non-negativity, or graph regularization.

For a *point-wise* loss function, the general calculation is:

$$L = \sum_{u \in U} \sum_{i \in I} \ell_{\textit{point-wise}}(y_{ui}, \hat{y}_{ui}) + \lambda\Omega(\theta), \tag{3.8}$$

There are several types of point-wise loss function. E.g., squared loss is more suitable for explicit feedback than implicit feedback, as it is calculated with:

$$\ell_{squ} = \sum_{u \in U} \sum_{i \in I} w_{ui}(y_{ui} - \hat{y}_{ui})^2, \tag{3.9}$$

where w_{ui} is a hyper-parameter denoting the weight of training instance (u, i). The use of squared loss is based on the assumption that observations are generated from a Gaussian distribution, however, it may not tally well with implicit data [105]. For implicit feedback, there is a point-wise loss function that is mainly used for classification tasks [46, 55], named log loss [61], which can perform better with implicit feedback than squared loss:

$$\ell_{\log} = -\sum_{u \in U} \sum_{i \in I} y_{ui} \log \hat{y}_{ui} + (1 - y_{ui}) \log(1 - \hat{y}_{ui}). \tag{3.10}$$

Pair-wise loss considers the relative order of the prediction for pairs of items, which is a more reliable kind of information for top-N recommendation. Hidasi and Karatzoglou [50] investigate several popular pair-wise loss functions, i.e., TOP1, BPR-max and TOP1-max. We give a brief introduction of them. TOP1 is the regularized approximation of the relative rank of the relevant item, which can be calculated as:

$$\ell_{\text{TOP1}} = \frac{1}{|N_S|} \sum_{j \in N_S} \sigma(\hat{y}_{uj} - \hat{y}_{ui}) + \sigma(\hat{y}_{uj}^2), \tag{3.11}$$

where $\hat{y}_{uj}$ and $\hat{y}_{ui}$ denote the prediction scores for a negative item j and a positive item i, respectively; N_S is the set of negative samples. The first part of TOP1 aims to ensure that the target score is higher than the score of the negative samples, while the second part pushes the score of the negative samples down. As for BPR-max and TOP1-max, they have been proposed by Hidasi and Karatzoglou [50] to overcome the vanishing gradients as the number of negative samples increases. The idea is to have the target score compared with the most relevant sample score, which is the maximum score amongst the samples. As the maximum operation is non-differentiable, softmax scores are used to preserve differentiability. By summing over the individual losses weighted by the corresponding softmax scores s_j, TOP1-max can be calculated as:

$$\ell_{\text{TOP1-max}} = \sum_{j \in N_S} s_j (\sigma(\hat{y}_{uj} - \hat{y}_{ui}) + \sigma(\hat{y}_{uj}^2)). \tag{3.12}$$

And the BPR-max loss function can be calculated as:

$$\ell_{\text{BPR-max}} = -\log \sum_{j \in N_S} s_j \sigma(\hat{y}_{ui} - \hat{y}_{uj}). \tag{3.13}$$

For *list-wise* loss, some approaches combine cross-entropy loss with softmax, which introduces list-wise properties into the loss. We refer to it as softmax+cross-entropy (XE) loss, which can be calculated with the following function:

$$\ell_{\text{XE}} = -\log s_i = -\log \frac{e^{\hat{y}_{ui}}}{\sum_{j \in N_S} e^{\hat{y}_{uj}}} \tag{3.14}$$

Most deep learning-based models only use the point-wise loss function for optimization and leave the pair-wise loss function for future work [46, 55]. Point-wise loss only uses the rating information and ignores the information contained in the relative order of pairs of items. Pair-wise loss, in contrast, ignores the information of a user's individual preference for a certain item. Thus, unlike previous work, NCF and DMF, our proposed J-NCF model considers both point-wise and pair-wise loss for the top-N recommendation task and combines them into a new loss function:

$$L = \alpha L_{\textit{pair-wise}} + (1 - \alpha) L_{\textit{point-wise}}, \tag{3.15}$$

where α is used to control the weights of the two parts.

For point-wise loss, we adopt the log loss (Eq. (3.10)), which can integrate both implicit and explicit feedback. As to pair-wise loss, combining with different pair-wise losses yields different new loss functions, i.e., point-wise+TOP1, point-wise+BPR-max, and point-wise+TOP1-max. We analyze the performance of these different combined loss functions with experiments in Section 3.5.

Acknowledging that explicit and implicit feedback both contain information about a user's preference towards items, we combine both kinds of feedback in our loss function for optimization and rewrite Eq. (3.15) in detail as

$$L = \alpha L_{\textit{pair-wise}} + (1 - \alpha)(-Y_{ui} \log \hat{y}_{ui} - (1 - Y_{ui}) \log(1 - \hat{y}_{ui})), \tag{3.16}$$

Algorithm 2 Joint Neural Collaborative Filtering.

Input: Epochs: training iterations;
R: the original rating matrix;
U: user set;
I: item set;
Output: $\mathbf{W_u^x}$ $(x = 1, \ldots, X)$: Weight matrix of Net$_{user}$;
$\mathbf{b_u^x}$ $(x = 1, \ldots, X)$: Bias of Net$_{user}$;
$\mathbf{W_i^x}$ $(x = 1, \ldots, X)$: Weight matrix of Net$_{item}$;
$\mathbf{b_i^x}$ $(x = 1, \ldots, X)$: Bias of Net$_{item}$;
$\mathbf{W_{ui}^y}$ $(y = 1, \ldots, Y)$: Weight matrix of DI network;
$\mathbf{b_{ui}^y}$ $(y = 1, \ldots, Y)$: Bias of DI network.

1: randomly initialize $\mathbf{W_u}$, $\mathbf{W_i}$,$\mathbf{W_{ui}}$, $\mathbf{b_u}$, $\mathbf{b_i}$ and $\mathbf{b_{ui}}$;
2: $y_{ui} \leftarrow$ use Eq. (3.3) with R;
3: $V^+ \leftarrow$ all none zero interactions pairs;
4: **for** epoch in range(Epochs) **do**
5: random shuffle of V^+
6: **for** $\langle u, i \rangle \in V^+$ **do**
7: sample the set of negative samples N_S
8: **for** $j \in N_S$ **do**
9: $\mathbf{v_u}$, $\mathbf{v_i}$, $\mathbf{v_j} \leftarrow y_{ui}$ with Eq. (3.3);
10: $\mathbf{z_u}$, $\mathbf{z_i}$, $\mathbf{z_j} \leftarrow$ use Eq. (3.4) with $\mathbf{v_u}$, $\mathbf{v_i}$, $\mathbf{v_j}$ as inputs;
11: $\mathbf{a_{ui}}$, $\mathbf{a_{uj}} \leftarrow$ use Eq. (3.5) with $\mathbf{z_u}$, $\mathbf{z_i}$, $\mathbf{z_j}$;
12: $\hat{y}_{ui}$, $\hat{y}_{uj} \leftarrow$ use Eq. (3.6) and Eq. (3.7);
13: $L \leftarrow$ use Eq. (3.16) with y_{ui}, $\hat{y}_{ui}$ and $\hat{y}_{uj}$ as inputs;
14: use back propagation to optimize the parameters;
15: **end for**
16: **end for**
17: **end for**
18: **return** $\mathbf{W_u}$, $\mathbf{W_i}$, $\mathbf{W_{ui}}$, $\mathbf{b_u}$, $\mathbf{b_i}$ and $\mathbf{b_{ui}}$.

where $Y_{ui} = \frac{y_{ui}}{Max(R_u)}$, and $Max(R_u)$ denotes the largest rating score of user u given to items, so that different values of y_{ui} have a different influence on the loss. For example, if the largest rating score of a user u given to items is 4, when he rates an item i with 2, we can generate $Y_{ui} = \frac{y_{ui}}{Max(R_u)} = \frac{2}{4}$. We refer to our loss function Eq. (3.16) as a "*hybrid*" loss function.

We have developed the joint neural network structure of the J-NCF model. The training process of J-NCF is shown in Algorithm 2. We first initialize the parameters in the network and modify the rating matrix from step 1 to 3. Then, in step 9 and 10, we generate deep feature representations for both users and items with the DF network. In step 11 and 12, we calculate the predicted scores for the user-item interactions with the DI network. Finally, we use the hybrid loss function in Eq. (3.16) and back propagation to optimize the network parameters with step 13 and 14.

3.4 Experiments

We design experiments on a variety of datasets to examine the effectiveness of J-NCF. We first explain the research questions and the models we use for comparison in Section 3.4.1. The datasets and experiments are described in Section 3.4.2.

3.4.1 Model summary and research questions

We decompose **RQ2** into a number of more fine-grained research questions that guide our experiments:

(RQ2.1) Does our proposed J-NCF method outperform state-of-art collaborative filtering baselines for recommender systems?

(RQ2.2) How is the performance of J-NCF impacted by different choices for the pairwise loss in Eq. (3.16)?

(RQ2.3) Does the hybrid loss function Eq. (3.15), which combines point-wise and pair-wise loss, help to improve the performance of J-NCF?

(RQ2.4) Are deeper layers of hidden units in the DF network and DI network helpful for the recommendation performance of J-NCF?

(RQ2.5) Does the combination of explicit and implicit feedback help to improve the performance of J-NCF?

(RQ2.6) How does the performance of J-NCF vary across users with different numbers of interactions?

(RQ2.7) Is J-NCF sensitive to different degrees of data sparsity?

(RQ2.8) How does J-NCF perform on a large and sparse dataset?

(RQ2.9) How do the training and inference times of J-NCF compare against those of other neural models?

We compare J-NCF against a number of traditional collaborative filtering baselines and against state-of-the-art deep learning based models:

Item-pop This method ranks items based on the number of interactions, which is a non-personalized approach to determine recommendation scores [3].

BPR This method uses a pairwise loss function to optimize a MF model based on implicit feedback. We use it as a strong baseline for traditional collaborative filtering method [103].

NCF This is a state-of-the-art neural network-based method for recommender systems. It aims to capture the non-linear relationship between users and items. Unlike J-NCF, it simply uses one-hot vectors representing users and items as the input for modeling user-item interactions. And it only uses implicit feedback and a point-wise loss function [46].

Table 3.2: An overview of the models discussed in the chapter.

Model	Description	Source
Item-pop	A typical recommendation approach, which ranks items based on the number of interactions.	[3]
BPR	A recommendation method using a pairwise loss function to optimize an MF model based on implicit feedback.	[103]
NCF	A state-of-the-art neural based method for recommender systems.	[46]
DMF	A method using multi-layer perceptrons for rating matrix factorization.	[55]
J-NCF$_m$	A J-NCF model using element-wise multiplication for combining a user and an item feature vector as the input for the DI layer.	This chapter
J-NCF$_c$	A J-NCF model using concatenation for combining a user and an item feature vector as the input for the DI layer.	This chapter
J-NCF$_{point}$	A J-NCF model with only point-wise loss based on Eq. (3.10).	This chapter
J-NCF$_{pair}$	A J-NCF model with only pair-wise loss based in Eq. (3.11).	This chapter
J-NCF$_{hybrid}$	A J-NCF model with our designed loss function in Eq. (3.16).	This chapter
J-NCF$_{ex}$	A J-NCF model with both explicit and implicit feedback in the input and the loss function.	This chapter
J-NCF$_{im}$	A J-NCF model with only implicit feedback in the input and the loss function.	This chapter

DMF This method uses multi-layer perceptrons for rating matrix factorization. Unlike our work, after projecting users and items into low dimensional vectors, it applies an inner product to calculate interactions between users and items, which is a linear kernel. It uses a point-wise loss function for optimization [55].

In addition, following the choices that we identified in Eq. (3.5), we consider two versions of J-NCF:

J-NCF$_m$ This is J-NCF using element-wise multiplication for combining a user and an item feature vector as the input for the DI layer, which has a linear kernel inside.

J-NCF$_c$ This is J-NCF using concatenation for combining a user and an item feature vector as the input for the DI layer, which is a non-linear way.

We list all the models to be discussed in Table 3.2.

Table 3.3: Dataset statistics. "Density" is the density of each dataset (i.e., $\#Density = \#Ratings/(\#Uses \times \#Items)$).

Dataset	#Users	#Items	#Ratings	#Density(%)
ML100K	943	1,682	100,000	6.3047
ML1M	6,040	3,706	1,000,209	4.4685
AMovies	15,067	69,629	877,736	0.0837
AEle	1,221,341	157,003	4,486,501	0.00234

3.4.2 Datasets and experimental setup

Dateset

We use three publicly available datasets to evaluate our models and the baselines:

(1) **MovieLens**, which contains several rating datasets from the MovieLens web site. The datasets are collected over various periods of time, depending on the size of the set [46, 55]. We use two sets for our experiments, i.e., MovieLens 100K (ML100K) containing 100,000 ratings from 943 users on 1,682 movies, and MovienLens 1M (ML1M) containing more than 1 million ratings from 6,040 users on 3,706 movies.[1]

(2) **Amazon Movies (AMovies)**, which contains 4,607,047 ratings for movies from Amazon, which is bigger and sparser than the MovieLens datasets and used widely in the recommender systems literture for evaluation [55, 148].[2]

(3) **Amazon Electronics (AEle)**, which is a larger and sparser dataset than the other datasets used in this chapter. It contains 7,824,482 ratings of users on different electronics. We use it to test the performance of our model when applied on a large and sparse dataset.[3]

For the two MovieLens datasets, we do not process them because they are already filtered. For the AMovies dataset, following [46, 55], we filter the dataset so that, similar to the MovieLens data, only users with at least 20 interactions and items with at least 5 interactions are retained. For the larger dataset AEle, we only do minor filtering on the data, i.e., filtering out users with less than 2 interactions and items with less than 5 interactions. To answer RQ2.1 to RQ2.7, we use the ML100K, ML1M, and AMovies datasets to evaluate our models and baselines. As for RQ2.8 to RQ2.9, we test the models on all of the datasets. The characteristics of the datasets after preprocessing are summarized in Table 3.3.

In order to answer RQ2.6, we plot distributions of users with different numbers of interactions in the ML100K, ML1M, and AMovies datasets in Figure 3.2. The x-axis denotes the number of ratings while the y-axis indicates the number of users corresponding to the ratings. We see that the majority of users in the three datasets only have a few ratings, which we regard as "inactive users," and few "active users"

[1] https://grouplens.org/datasets/movielens/
[2] http://jmcauley.ucsd.edu/data/amazon/
[3] http://jmcauley.ucsd.edu/data/amazon/

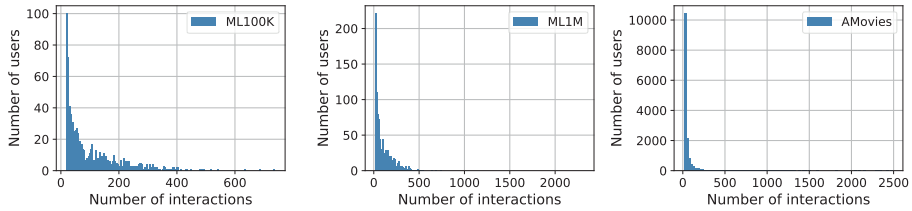


Figure 3.2: Distribution of users with varying numbers of interactions in the ML100K, ML1M, and AMovies datasets, respectively.

have far more ratings. E.g., in the ML100K dataset, 61.72% of the users have fewer than 100 ratings, 32.66% have between 100 and 300 ratings, and only 5.6% of the users have more than 300 ratings.

As we will see below, the models being considered in this chapter achieve different scores when used on datasets with different characteristics, i.e., number of users and number of items (see Section 3.5). Thus, for RQ2.7, in order to evaluate the performance of our model on datasets with different degrees of sparsity, we keep the number of users and items the same. Namely, following [61], for each of the three datasets, i.e., ML100K, ML1M, and AMovies, we create three versions at different sparsity levels with the the following steps:

(Step 1) We start by randomly choosing a subset of users and items from the original dataset. This dataset is represented with a '-1' suffix.

(Step 2) We randomly choose a rating record and make a judgment if the numbers of users as well as items are unchanged of the sub-dataset after removing this record. If unchanged, we remove this record; otherwise repeat Step 2.

(Step 3) After several repetitions of Step 2, the first sparser version of the dataset with the '-2' suffix is created.

(Step 4) Repeat Step 2 and Step 3 based on the dataset with a '-2' suffix, the second sparser version of the dataset with the '-3' suffix is created in the same way.

The characteristics of the datasets are summarized in Table 3.4.

Experimental setup

For evaluation, we use a *leave-one-out* strategy, which has been used widely in DL-based recommender systems [45, 46, 55]. The training set consists of all but the last interaction of every user; the test set contains the latest interaction of every user. When testing, it is time-consuming to give ranking predictions to all items for every user. Thus, following He et al. [46], Hong-Jian et al. [55], we randomly sample 100 items with which the user has not interacted and then give the test item ranking predictions among the 100 samples. Although using this sampling strategy during evaluation may overestimate the performance of all algorithms, Bellogin et al. [8], Hidasi and Karatzoglou [50] have pointed out that the comparison among algorithms still remains fair.

Table 3.4: Dataset statistics with different degrees of sparsity.

Dataset	#Users	#Items	#Ratings	#Density(%)
ML100K-1	943	1,682	69,999	4.4132
ML100K-2	943	1,682	39,999	2.2522
ML100K-3	943	1,682	9,999	0.6304
ML1M-1	3,706	6,040	850,208	3.7982
ML1M-2	3,706	6,040	350,207	1.5645
ML1M-3	3,706	6,040	167,870	0.7499
AMovies-1	7,402	12,080	87,807	0.0982
AMovies-2	7,402	12,080	37,823	0.0423
AMovies-3	7,402	12,080	18,867	0.0211

The majority of the recommender system literature applies error metrics for evaluation, i.e., *Root Mean Squared Error* (RMSE) and *Mean Absolute Error* (MAE). Such classical error criteria do not really measure the top-N recommendation performance [32]. An extensive evaluation of several state-of-the-art recommender algorithms suggests that algorithms optimized for minimizing RMSE do not necessarily perform as expected in terms of the top-N recommendation task [32, 49]. Experimental results also show that improvements in terms of RMSE often do not translate into accuracy improvements [49]. Thus, here we choose to use accuracy metrics to examine the recommendation performance [46]. Specifically, we use HR and NDCG to evaluate the performance of our models. *Hit Ratio* (HR) is used to evaluate the precision of the recommender system, i.e., whether the test item is contained in the top-N list. The *Normalized Discount Cumulative Gain* (NDCG) measures the ranking accuracy of the recommender system, i.e., whether the test item is ranked at the top of the list.

As for parameters, we optimize the hyperparameters by running 100 experiments at randomly selected points of the parameter space. Optimization is done on a validation set, which is partitioned from the training set with the same procedure as the test set. As for the loss function, we test the parameter α from 0 to 1 with step size of 0.1 in our experiment. For the neural networks, we randomly initialize model parameters with a Gaussian distribution (mean of 0 and standard deviation of 0.01), optimizing the model with mini-batch Adam [66]. The batch size and learning rate are set to 256 and 0.0001. For the baselines, we set the parameters of DMF as well as NCF following [46, 55], respectively. For DMF and NCF, we set the batch size to 256, and the learning rate to 0.0001 and 0.001. For the DF network in DMF model, we apply two layers and the sizes of them are [128, 64]. For the DI network in the NCF model, we employ three hidden layers with size [128, 64, 8]. For the DF and DI networks in J-NCF, without special mention, we employ three layers in DF network with the size of [256, 128, 64] and two layers in DI network with size of [128, 8]. Thus the embedding sizes of users as well as items are the same in all baseline models as well as J-NCF. We also keep the size of the last hidden layer of the DI network in J-NCF the same as NCF, which may determine the model capability. We also test our model as well as the baseline models with different numbers of layers to see if deep layers are beneficial to the overall performance of these models. Unless specified, for all the results presented in this chapter, the number of recommendations (N) is equal to 10 [46, 55]. The source

Table 3.5: Performance of recommendation models. The results produced by the best baseline and the best performer in each column are underlined and bold-faced, respectively. Statistical significance of pairwise differences of J-NCF$_m$ and J-NCF$_c$ vs. the best baseline) is determined by a t-test ($^{\blacktriangle}$/$^{\blacktriangledown}$ for α = .01, or $^{\triangle}$/$^{\triangledown}$ for α = .05).

	ML100K		ML1M		AMovies	
Model	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10
Item-pop	.3832	.2018	.4513	.2315	.5925	.3493
BPR	.5762	.3021	.6097	.3711	.6288	.3903
NCF	.6066	.3488	.6498	.3951	.6782	.4135
DMF	.6309	.3616	.6748	.4221	.7151	.4616
J-NCF$_m$	.6627$^{\triangle}$	.3877$^{\triangle}$	.7127$^{\blacktriangle}$	.4485$^{\blacktriangle}$	.7666$^{\blacktriangle}$	.5098$^{\blacktriangle}$
J-NCF$_c$	**.6829**$^{\blacktriangle}$	**.4065**$^{\blacktriangle}$	**.7377**$^{\blacktriangle}$	**.4822**$^{\blacktriangle}$	**.7881**$^{\blacktriangle}$	**.5311**$^{\blacktriangle}$

codes for our model are available on our public repository.[4]

3.5 Results and Discussion

3.5.1 Overall performance

To answer RQ2.1, we examine the recommendation performance of the baselines and the J-NCF$_m$ and J-NCF$_c$ models. See Table 3.5.

Let us first consider the baselines. From Table 3.5, we see that DMF achieves a better performance than the other baselines in terms of HR@10 and NDCG@10. Hence, we only use DMF as the best baseline for comparisons in later experiments. Bayesian Personalized Ranking (BPR) clearly shows higher improvements over the Item-pop baseline in terms of NDCG@10 than in terms of HR@10, which shows that pairwise loss has a strong performance for ranking prediction. The NCF and DMF models both show better performance than the two traditional CF models, which indicates the utility of DL techniques in improving recommendation performance.

Next, we compare the baselines against the J-NCF models. NCF and DMF both lose against the J-NCF models in terms of HR@10 and NDCG@10. This shows that a joint neural network structure that tightly couples deep feature learning and deep interaction modeling helps to improve the recommendation performance. Regarding the J-NCF models, independent of the choice of combining the users' and items' vectors, J-NCF achieves a better performance than the DMF baseline, resulting in HR@10 improvements ranging from 5.04% to 8.24% on the ML100K dataset, 5.62% to 10.81% on the ML1M dataset, and 7.21% to 10.21% on the AMovies dataset. NDCG@10 improvements range from 7.22% to 12.42% on the ML100K dataset, 6.25% to 14.24% on the ML1M dataset, and 10.44% to 15.06% on the AMovies dataset. Significant improvements against the baseline in terms of HR@10 and NDCG@10 are observed for both J-NCF$_c$ and J-NCF$_m$ at the $\alpha = .01$ level, except for J-NCF$_m$ on the ML100K

[4] https://bitbucket.org/WanyuChen/jncf/

dataset, for which we observe significant improvements at the $\alpha = .05$ level in terms of HR@10 and NDCG@10. The higher improvements in NDCG@10 over HR@10 may be due to the fact that we incorporate pair-wise loss in our loss function, which motivates us to conduct a further investigation to answer RQ2.3.

Comparing J-NCF$_c$ and J-NCF$_m$, we see that J-NCF$_c$ achieves the best performance, with improvements of 3.05%, 3.51% and 2.81% in terms of HR@10, and 4.85%, 7.51% and 4.18% in terms of NDCG@10 over J-NCF$_m$ on the three datasets, respectively. The complex relationship between users and items can be described better with a non-linear kernel than linear kernel, which is consistent with the findings in [46, 83].

3.5.2 Impact of different loss functions

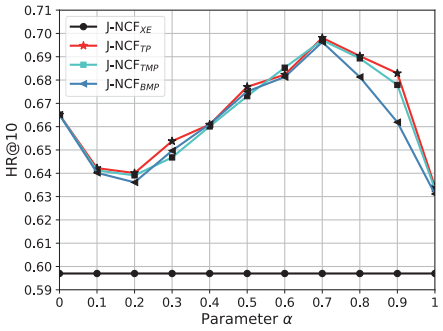
As we have mentioned in Section 3.3.3, there are several kinds of pair-wise loss functions that can be incorporated in Eq. (3.16). When J-NCF combines the point-wise loss, i.e., log loss, with TOP1, TOP1-max, and BPR-max pair-wise losses, it gives rise to the J-NCF$_{TP}$, J-NCF$_{TMP}$ and J-NCF$_{BMP}$ models, respectively. Additionally, list-wise loss, i.e., softmax+cross-entropy (XE), can also be applied with J-NCF, which gives rise to the J-NCF$_{XE}$ model. In order to investigate the impact of various loss functions on J-NCF (RQ2.2), we examine the recommendation performance of J-NCF$_{TP}$, J-NCF$_{TMP}$, J-NCF$_{BMP}$ as well as J-NCF$_{XE}$ models where the parameter α in Eq. (3.16) ranges from 0 to 1 with a step size of 0.1. Figure 3.3 shows the results.

As for the overall performance, we can see that when applied with a list-wise loss function, J-NCF$_{XE}$ has the worst performance among the four models. The other three models, which combine pair-wise and point-wise losses, show relatively similar results in terms of HR@10 and NDCG@10. When $\alpha = 0$, it results in J-NCF$_{point}$. When $\alpha = 1$, it leads to J-NCF, a model with only corresponding pair-wise loss functions. It is obvious that solely based on point-wise loss, J-NCF has better performance in terms of HR@10 while worse performance regarding NDCG@10 than J-NCF with only pair-wise loss. This can be explained by the fact that pair-wise loss can help J-NCF learn to rank items in right positions.

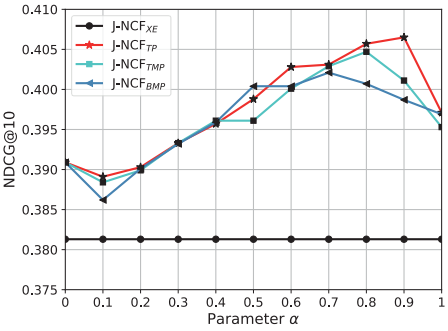
In Figure 3.3a, the performance of all models increases from $\alpha = 0.2$ to $\alpha = 0.7$ before a short-term decrease and then a dramatic drop after reaching the peak at $\alpha = 0.7$. The performance of J-NCF$_{TP}$, J-NCF$_{TMP}$ and J-NCF$_{BMP}$ is comparable in terms of HR@10. As for NDCG@10, shown in Figure 3.3b, J-NCF$_{TP}$ shows better performance than the other two models and achieves the highest point at $\alpha = 0.9$.

Regarding the performance on the ML1M dataset, similar trends can be found in Figure 3.3c and Figure 3.3d as in Figure 3.3a and Figure 3.3b, respectively. For the AMovies dataset shown in Figure 3.3e and Figure 3.3f, J-NCF$_{BMP}$ shows slightly better performance than both J-NCF$_{TP}$ and J-NCF$_{TMP}$ in terms of HR@10, while the performance of J-NCF$_{BMP}$ and J-NCF$_{TP}$ is similar in terms of NDCG@10, which is a little better than that of J-NCF$_{TMP}$.

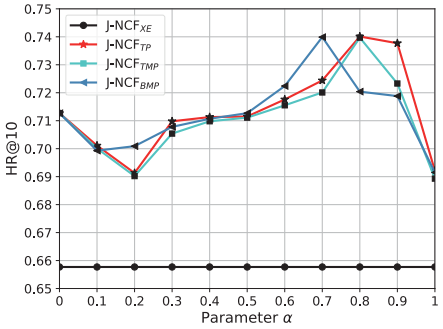
As discussed in [50], the BPR-max and TOP1-max loss functions have been proposed to overcome vanishing gradients as the number of negative samples increases. Since we use a small number of negative samples in this chapter, the performance is relatively similar between the three models, J-NCF$_{TP}$, J-NCF$_{TMP}$ and J-NCF$_{BMP}$. As



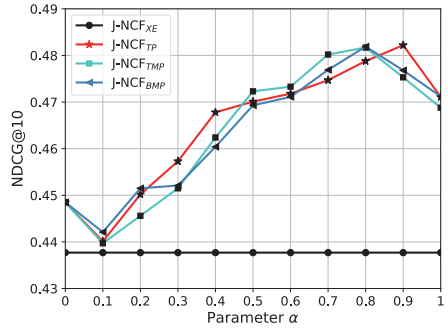
(a) Performance in terms of HR@10 on the ML100K dataset.



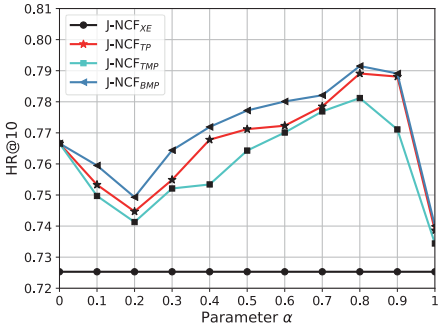
(b) Performance in terms of NDCG@10 on the ML100K dataset.



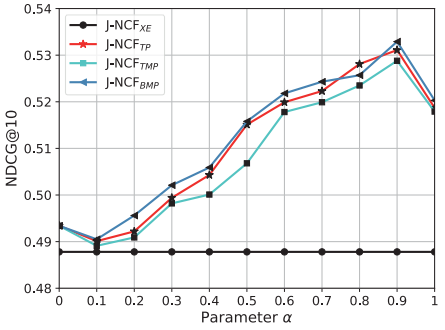
(c) Performance in terms of HR@10 on the ML1M dataset.



(d) Performance in terms of NDCG@10 on the ML1M dataset.



(e) Performance in terms of HR@10 on the AMovies dataset.



(f) Performance in terms of NDCG@10 on the AMovies dataset.

Figure 3.3: Performance of the J-NCF models applied with different loss functions where the parameter α in Eq. (3.16) ranges from 0 to 1 with a step size of 0.1.

BPR-max and TOP1-max losses need additional softmax calculations for all negative samples, we apply the TOP1 pair-wise loss in Eq. (3.16) for J-NCF in the experiments

on which we report below.

3.5.3 Utility of hybrid loss function

For RQ2.3, in order to investigate the utility of the hybrid loss function (Eq. (3.16)), we examine the recommendation performance of the J-NCF$_c$ models under different settings, i.e., J-NCF$_{point}$ with only point-wise loss based on Eq. (3.10) (we incorporate explicit feedback in the same way as Eq. (3.16)), J-NCF$_{pair}$ with only pair-wise loss based on Eq. (3.11), and J-NCF$_{hybrid}$ with our designed loss function from Eq. (3.16). Figure 3.4 shows the results.

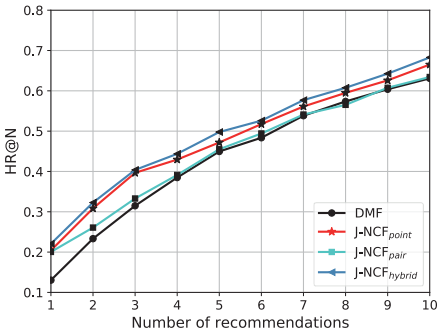
The overall performance in terms of HR and NDCG increases when the size of the top-N recommended list ranges from 1 to 10, as a large value of N increases the probability of including a user's preferred item in the recommendation list. J-NCF$_{hybrid}$ consistently achieves improvements over DMF as well as the two models with a single loss function across positions, which demonstrates the utility of our newly designed loss function. Based on the ML100K dataset, J-NCF$_{hybrid}$ improves by 2.68% and 7.61%, respectively, over J-NCF$_{point}$ and J-NCF$_{pair}$ in terms of HR@10; improvements of NDCG@10 over J-NCF$_{point}$ and J-NCF$_{pair}$ are 3.99% and 2.36%, respectively.

Comparing J-NCF$_{point}$ and J-NCF$_{pair}$, we find that J-NCF$_{point}$ beats J-NCF$_{pair}$ in terms of HR, while J-NCF$_{pair}$ shows more competitive performance in terms of NDCG than J-NCF$_{point}$. This confirms the findings in [45, 103] that a pair-wise ranking-aware learner has a strong performance for ranking prediction. This finding motivates us to incorporate both point-wise loss and pair-wise loss into the hybrid loss function. Clearly, J-NCF$_c$ based models, i.e., J-NCF$_{point}$, J-NCF$_{pair}$ and J-NCF$_{hybrid}$, show a better performance than DMF, which also proves that the joint neural structure is effective, i.e., deep interaction modeling can optimize neural matrix factorization and thus improve the recommendation performance.

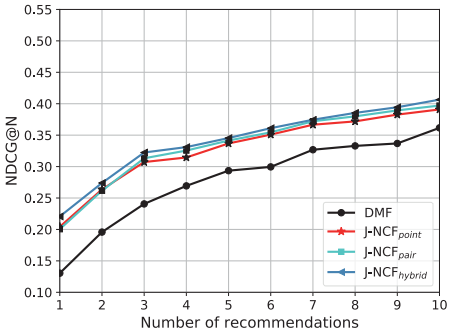
Comparing the left and right hand sides of Figure 3.4, we see that the improvements of J-NCF$_{hybrid}$ in terms of NDCG are more significant than those in terms of HR, as indicated by the relative improvements over DMF with different sizes of the recommendation list. In Figure 3.4a, J-NCF$_{hybrid}$ shows a 8.78% improvement over DMF in terms of HR at cutoff $N = 6$, a 5.91% improvement at $N = 8$ and a 8.24% improvement at $N = 10$ on the ML100K dataset. In Figure 3.4b, the improvements in terms of NDCG at cutoff $N = 6$, $N = 8$ and $N = 10$ are 19.01%, 15.72% and 12.42%, respectively. J-NCF$_c$ with the hybrid loss function cannot only recommend the correct item to a user, but is also competitive in terms of ranking it at the top of the list.

3.5.4 Number of layers in the networks

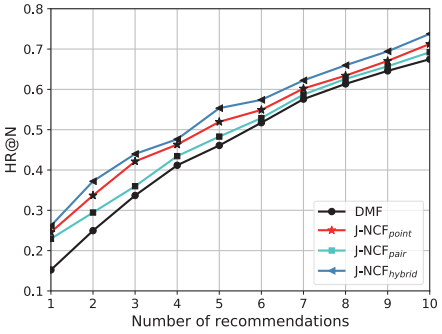
In J-NCF$_c$, we not only learn features of users and items through the DF neural network with multiple hidden layers, but also model user-item interactions with multi-layer perceptrons in the DI network. Thus it is crucial to see whether DL is helpful in our model. We conduct experiments to examine the performance of J-NCF$_c$ with various numbers of layers in the DF and DI networks, respectively (RQ2.4). In addition, we also test the



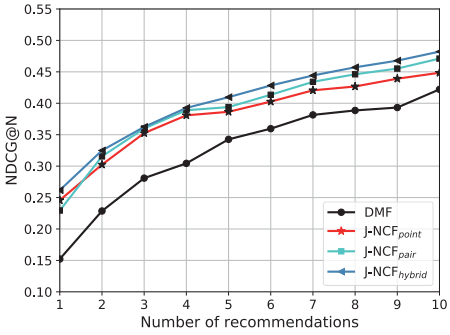
(a) Performance in terms of HR@N on the ML100K dataset.



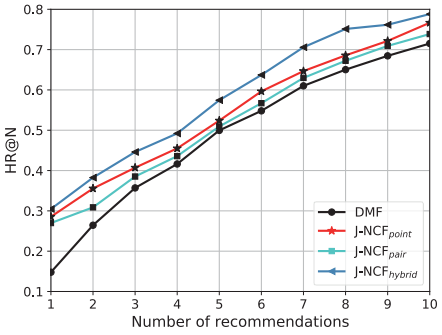
(b) Performance in terms of NDCG@N on the ML100K dataset.



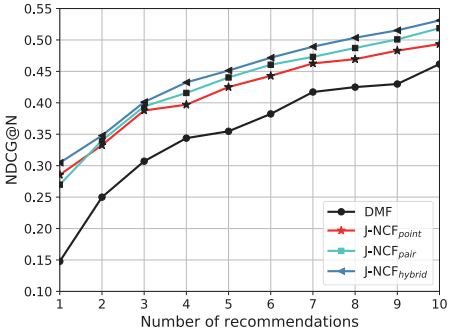
(c) Performance in terms of HR@N on the ML1M dataset.



(d) Performance in terms of NDCG@N on the ML1M dataset.



(e) Performance in terms of HR@N on the AMovies dataset.



(f) Performance in terms of NDCG@N on the AMovies dataset.

Figure 3.4: Performance of Top-N item recommendation where N ranges from 1 to 10. The left and right plots show the performance in terms of HR@N and NDCG@N, respectively.

performance of the best baseline model, i.e., DMF, with different DF networks. The results are shown in Table 3.6. The i in DF-i and DI-i in Table 3.6 denotes the number of layers in the DF network and DI network of J-NCF$_c$, respectively.

As shown in Table 3.6, in terms of HR@10, we can see that with the number of layers increasing, the recommendation performance of J-NCF is improved, which verifies the effectiveness of DL techniques for recommender systems.

Comparing the number of layers in the DI and DF networks, we can find that stacking more layers in the DF network of J-NCF$_c$ seems more helpful than in the DI network in enhancing the recommendation performance. For example, based on the ML100K dataset, the improvements of the configuration (DF-3, DI-2) over (DF-2, DI-2) are 2.82% and 4.31% in terms of HR@10 and NDCG@10, while the improvements are 1.05% and 2.62% for (DF-2, DI-3) over (DF-2, DI-2). When we stack more than 4 layers in the DI network (e.g., DI-5), the performance of J-NCF$_c$ no longer increases. However, stacking more layers in the DF network (e.g., DF-5) still seems helpful and the best results produced for each dataset are all based on J-NCF$_c$ with the (DF-5, DI-4) configuration. This may be because deep layers are more helpful in extracting users' as well as items' features and thus enhancing the user-item interactions predictions. It motivates us to incorporate more auxiliary information for exploring users' and items' features with deep learning techniques in future work.

As for NDCG@10, a similar phenomenon can be found. However, when comparing the scores of HR@10 and NDCG@10 under the same configurations, we can find that deeper layers can lead to more obvious improvements in terms of NDCG@10 than HR@10 on all of the three datasets. The best performance of J-NCF with (DF-5, DI-4) outperforms the worst performance of J-NCF with (DF-1, DI-1) by 20.52%, 25.37% and 34.52% in terms of HR@10 on the three datasets, respectively. However, the improvements are 28.96%, 63.05% and 53.37% in terms of NDCG@10 on the three datasets.

As for the baseline model DMF shown in the bottom rows in Table 3.6, when applied with DF-1, J-NCF$_c$ with DI-1 loses to DMF on all datasets. Similar results can be found with DF-2, except on ML100K dataset. This can be explained by the fact that the simple concatenation of user's and item's embeddings with only one MLP layer in J-NCF$_c$ is not efficient for modeling user-item interactions. When applied with more DI layers, J-NCF$_c$ has better performance than DMF with the same number of DF layers. Additionally, we can find that DMF achieves the best performance with DF-2 and deeper layers do not seem useful for DMF model, which corresponds to the results in [55]. However, J-NCF$_c$ achieves further improvements when stacking more layers in either the DI or DF network, or both.

3.5.5 Impact of feedback

In J-NCF, we consider different kinds of user feedback. On the one hand, we use the interaction matrix as the input of the network with Eq. (3.3), which contains not only implicit feedback but also explicit feedback. On the other hand, our loss function in Eq. (3.16) employs a normalized strategy in the form of $Y_{ui} = \frac{y_{ui}}{Max(R_u)}$, where $Max(R_u)$ denotes the largest rating score of user u given to items, to incorporate the explicit feedback. In order to answer RQ2.5, we conduct experiments to investigate

Table 3.6: Performance of J-NCF$_c$ and DMF with different numbers of layers in terms of HR@10 and NDCG@10. The results produced by the best performing setting on each dataset are boldfaced.

		HR@10					NDCG@10				
		DF-1	DF-2	DF-3	DF-4	DF-5	DF-1	DF-2	DF-3	DF-4	DF-5
ML100K	DI-1	.6242	.6511	.6713	.6955	.7213	.3581	.3721	.3971	.4123	.4313
	DI-2	.6351	.6642	.6829	.7183	.7388	.3694	.3899	.4067	.4277	.4426
	DI-3	.6493	.6712	.7144	.7309	.7479	.3811	.4001	.4197	.4388	.4535
	DI-4	.6571	.6832	.7277	.7411	**.7523**	.3945	.4183	.4311	.4481	**.4618**
	DI-5	.6501	.6799	.7254	.7408	.7501	.3903	.4111	.4287	.4433	.4587
	DMF	.6285	.6309	.6301	.6297	.6298	.3598	.3616	.3614	.3607	.3598
ML1M	DI-1	.6451	.6671	.7121	.7389	.7619	.3622	.3911	.4399	.4893	.5301
	DI-2	.6531	.6999	.7377	.7531	.7814	.3889	.4233	.4822	.5211	.5525
	DI-3	.6766	.7198	.7589	.7728	.7929	.4195	.4601	.5177	.5437	.5777
	DI-4	.7134	.7472	.7683	.7834	**.8088**	.4581	.5101	.5389	.5663	**.5906**
	DI-5	.7099	.7411	.7653	.7821	.8021	.4517	.5078	.5333	.5644	.5878
	DMF	.6673	.6748	.6738	.6722	.6725	.3955	.4221	.4201	.4197	.4199
AMovies	DI-1	.6611	.6922	.7481	.7911	.8188	.4041	.4533	.5004	.5413	.5622
	DI-2	.6872	.7378	.7881	.8101	.8411	.4327	.4911	.5311	.5597	.5803
	DI-3	.6989	.7633	.8078	.8378	.8787	.4632	.5204	.5501	.5714	.6102
	DI-4	.7414	.7999	.8293	.8612	**.8893**	.5137	.5461	.5644	.5966	**.6198**
	DI-5	.7379	.7922	.8201	.8589	.8821	.5111	.5402	.5599	.5934	.6145
	DMF	.7478	.7515	.7491	.7483	.7479	.4551	.4616	.4612	.4603	.4591

whether the combination of explicit and implicit feedback works for J-NCF with different settings, i.e., J-NCF$_{ex}$ with both kinds of feedback in the input and the loss function as well as J-NCF$_{im}$ with only implicit feedback by labeling 1 for the interactions and 0 for unknown ratings in the input and the loss function. Figure 3.5 shows the recommendation performance of J-NCF$_{ex}$, J-NCF$_{im}$, DMF and NCF across different numbers of training iterations, respectively.

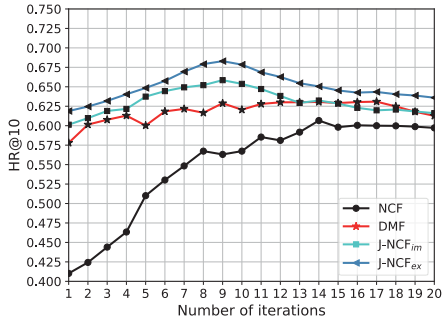
First, from Figure 3.5 we can see that J-NCF$_{ex}$ with both kinds of feedback achieves a competitive performance across all iterations in terms of HR and NDCG on the three datasets. It indicates that the combination of explicit and implicit feedback in the input and the specially designed loss function of J-NCF does help to improve the recommendation performance. Second, as the number of training iterations increases, the recommendation performance of all models is improved and then degraded after reaching a peak. More iterations may lead to overfitting, which hurts the recommendation performance. However, comparing J-NCF model with the baselines, i.e., DMF and NCF, we find that J-NCF converges to the best performance faster than other models. For example, on the ML100K dataset, the best result of J-NCF is generated after the first 9 effective iterations, while DMF and NCF need more training iterations to obtain the best results, i.e., 16 and 14 iterations respectively. The same phenomenon can be observed on the other two datasets. The optimal number of updates needed for J-NCF, DMF and NCF are around 10, 17 and 19 on the ML1M dataset, and 14, 18 and 19 on the AMovies dataset, respectively. Third, comparing the performance in terms of HR@10 and NDCG@10, we find that J-NCF$_{ex}$ shows larger improvements over J-NCF$_{im}$ in terms of NDCG@10 than HR@10. For example, the improvements are 3.72%, 5.22% and 4.89% in terms of HR@10, on the ML100K, ML1M and AMovies datasets, respectively, vs. improvements of 4.61%, 5.58% and 5.31% in terms of NDCG@10. This confirms our hypothesis that incorporating both explicit and implicit feedback can improve the ranking precision for recommendation.

3.6 Scalability and Sensitivity

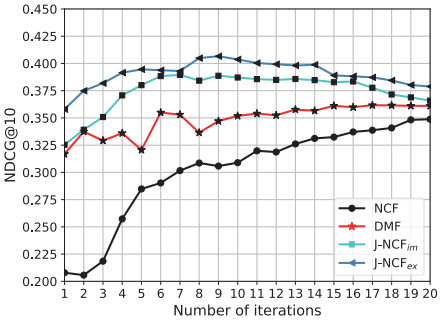
In order to answer RQ2.6 to RQ2.9, we study the scalability and sensitivity of J-NCF as well as the best baseline DMF when applied in different settings, i.e., with users with various numbers of ratings in Section 3.6.1, and with datasets with different levels of sparsity in Section 3.6.2. In addition, we also investigate the performance of the deep learning-based approaches, i.e., J-NCF, DMF and NCF, when applied with a large and sparse dataset in Section 3.6.3. Moreover, the training and inference time needed for these models on all datasets is discussed in Section 3.6.4.

3.6.1 Model scalability with user ratings

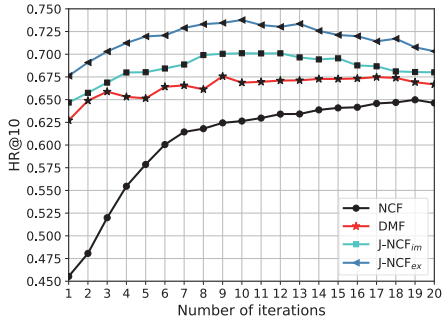
In Figure 3.2, we have shown that in every dataset most users only have a few ratings, thus it is meaningful to investigate how the performance of J-NCF and DMF varies with different numbers of user ratings. Following [96], we look at the performance for users of varying degrees of activity, measured by percentile. For example, in Table 3.7, we first rank the users according to their numbers of their activities. 10% shows the



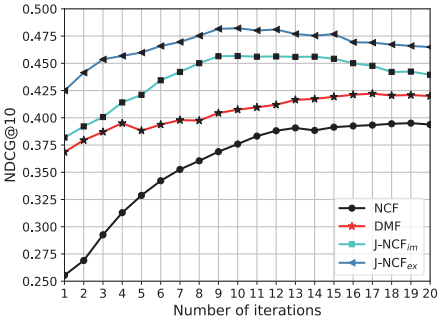
(a) Performance in terms of HR@10 on the ML100K datasets.



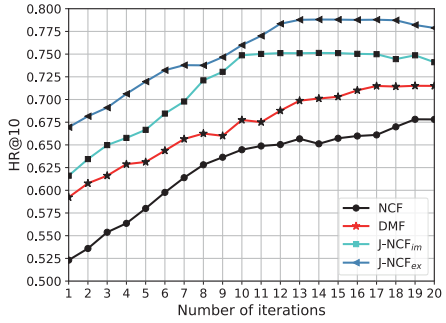
(b) Performance in terms of NDCG@10 on the ML100K datasets.



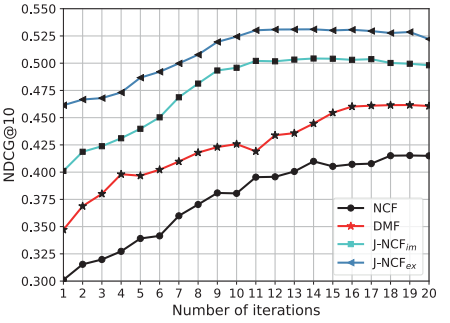
(c) Performance in terms of HR@10 on the ML1M datasets.



(d) Performance in terms of NDCG@10 on the ML1M datasets.



(e) Performance in terms of HR@10 on the AMovies datasets.



(f) Performance in terms of NDCG@10 on the AMovies datasets.

Figure 3.5: Recommendation performance across different numbers of iterations. The left and right plots show the performance in terms of HR@10 and NDCG@10, respectively.

mean performance across the bottom 10% of users, who are least active; the 90% mark shows the mean performance for all but the top 10% most active users.

Table 3.7: Recommendation performance across users who are ranked by the number of activities. The results produced by the best performing recommender system in each row are boldfaced. Statistical significance of pairwise differences of J-NCF$_m$ and J-NCF$_c$ vs. DMF is determined by a t-test ($^{\blacktriangle}$/$^{\blacktriangledown}$ for α = .01, or $^{\triangle}$/$^{\triangledown}$ for α = .05).

		HR@10			NDCG@10		
		DMF	J-NCF$_m$	J-NCF$_c$	DMF	J-NCF$_m$	J-NCF$_c$
ML100K	10%	.7001	.7400$^{\blacktriangle}$	**.8015**$^{\blacktriangle}$	.4358	.4786$^{\blacktriangle}$	**.5001**$^{\blacktriangle}$
	50%	.6813	.7349$^{\triangle}$	**.7568**$^{\blacktriangle}$	.4200	.4379$^{\triangle}$	**.4602**$^{\blacktriangle}$
	90%	.6279	.6585$^{\triangle}$	**.6772**$^{\blacktriangle}$	.3813	.3897$^{\triangle}$	**.4092**$^{\blacktriangle}$
ML1M	10%	.7548	.7927$^{\blacktriangle}$	**.8511**$^{\blacktriangle}$	.5111	.5417$^{\blacktriangle}$	**.5952**$^{\blacktriangle}$
	50%	.7211	.7532$^{\blacktriangle}$	**.7982**$^{\blacktriangle}$	.4855	.5266$^{\blacktriangle}$	**.5587**$^{\blacktriangle}$
	90%	.6601	.6981$^{\blacktriangle}$	**.7277**$^{\blacktriangle}$	.4217	.4432$^{\blacktriangle}$	**.4751**$^{\blacktriangle}$
AMovies	10%	.7851	.8611$^{\blacktriangle}$	**.9191**$^{\blacktriangle}$	.5349	.5998$^{\blacktriangle}$	**.6611**$^{\blacktriangle}$
	50%	.7519	.7855$^{\blacktriangle}$	**.8411**$^{\blacktriangle}$	.5033	.5466$^{\blacktriangle}$	**.5821**$^{\blacktriangle}$
	90%	.7013	.7411$^{\blacktriangle}$	**.7732**$^{\blacktriangle}$	.4597	.5038$^{\blacktriangle}$	**.5301**$^{\blacktriangle}$

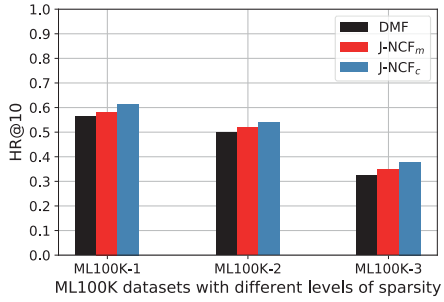
As shown in Table 3.7, J-NCF$_c$ outperforms the best baseline model DMF for users across all activity levels, i.e., both the "inactive" users who constitute the majority, and the relatively few "very active" users who give more ratings. In addition, J-NCF$_c$ always achieves the best performance in terms of HR@10 and NDCG@10. In order to test the robustness of J-NCF under different settings, i.e., J-NCF$_c$ and J-NCF$_m$, we conduct t-tests between the two versions of J-NCF with DMF, respectively. Significant improvements against the baseline DMF in terms of HR@10 and NDCG@10 are observed for both J-NCF$_m$ and J-NCF$_c$ at the $\alpha = .01$ level across all activity levels, except for J-NCF$_m$ on the ML100K dataset with 50% and 90% users, for which we observe significant improvements at the $\alpha = .05$ level in terms of HR@10 and NDCG@10.

Specifically, J-NCF shows larger improvements over the DMF model for "inactive" users than for "very active" users. For example, when incorporating users with more interactions, i.e., from 50% to 90%, the improvements change from 11.08% to 7.85% in terms of HR@10, and 9.57% to 7.32% in terms of NDCG@10 on the ML100K dataset. This may be because the "very active" users have many interactions with the items that have few ratings and collaborative filtering lacks information for recommending items based only on the rating matrix. This naturally suggest a line of future work in which one would extend J-NCF with more auxiliary information, such as content information, to explore more accurate relationships between items.

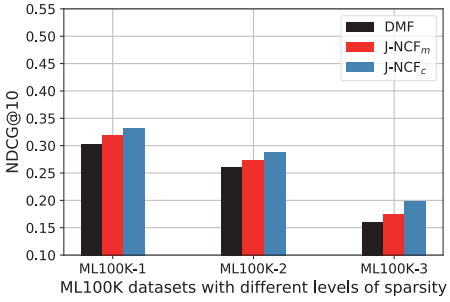
To conclude and answer RQ2.6, the J-NCF models can beat the best baseline model for users across all activity levels. J-NCF$_c$ shows the best performance in all datasets. In addition, for "inactive" users, J-NCF shows larger improvements over DMF than for "very active" users.

3.6.2 Sensitivity to data sparsity

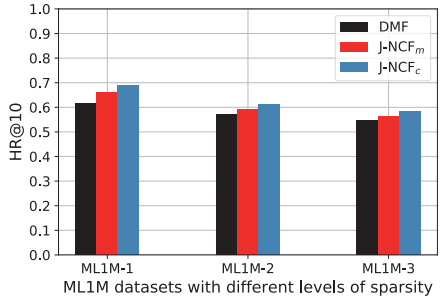
To investigate the sensitivity of J-NCF to different levels of data sparsity, we examine the recommendation performance on datasets with different levels of sparsity, as presented in Table 3.4. Figure 3.6 shows the results. The overall performance of all



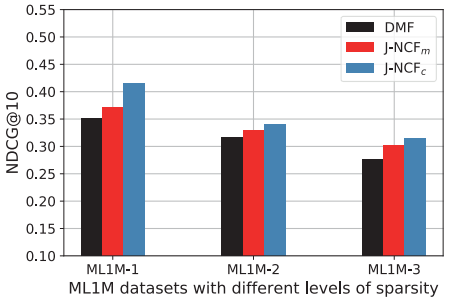
(a) Performance in terms of HR@10 on the ML100K datasets.



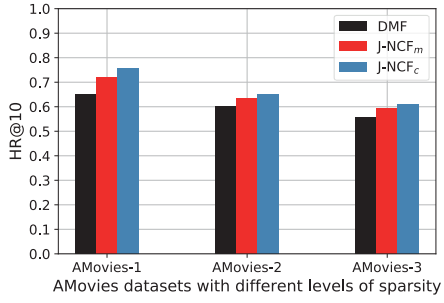
(b) Performance in terms of NDCG@10 on the ML100K datasets.



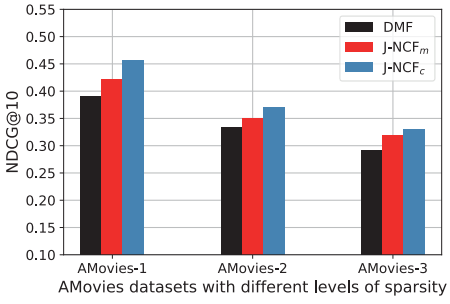
(c) Performance in terms of HR@10 on the ML1M datasets.



(d) Performance in terms of NDCG@10 on the ML1M datasets.



(e) Performance in terms of HR@10 on the AMovies datasets.



(f) Performance in terms of NDCG@10 on the AMovies datasets.

Figure 3.6: Recommendation performance across datasets with different levels of sparsity. The left and right plots show the performance in terms of HR@10 and NDCG@10, respectively.

models on the AMovies dataset is better than that on the other two datasets. That is to say, the recommendation performance may be influenced by the size of a dataset. Thus, in order to investigate the model sensitivity across datasets with different degrees of sparsity, it is essential to keep the number of users and items in the same scale for the datasets.

From Figure 3.6, in particular, for the ML100K dataset, the ML1M dataset and the AMovies dataset respectively, we see that the J-NCF models outperform the baseline model DMF across all sub datasets with different degrees of sparsity in terms of HR@10 and NDCG@10. In addition, we find that when the density of those datasets goes down, the performance of all models decreases. Thus it is interesting to investigate the robustness of J-NCF when it is applied to sparse datasets. We find that when applied on small datasets, e.g., subsets of ML100K, our best model, i.e., J-NCF$_c$, shows higher improvements against DMF on sparser datasets. For example, J-NCF$_c$ achieves 4.91% and 9.12% improvements over DMF in terms of HR@10 and NDCG@10 on the ML100K-1 subset (Density $= 4.413\%$), while the improvements on the ML100K-3 subset (Density $= 0.630\%$) are 7.77% and 12.02% in terms of HR@10 and NDCG@10, respectively. However, when applied on larger datasets with more users and items, i.e., subsets of ML1M and AMovies, J-NCF$_c$ shows higher improvements against DMF on denser datasets. For instance, J-NCF$_c$ achieves 11.13% improvements over DMF in terms of HR@10 on the ML1M-1 subset (Density $= 3.7982\%$), while the improvements on the ML1M-3 subset (Density $= 0.7499\%$) are 6.53% in terms of HR@10. These results may indicate that when the dataset becomes larger and sparser, it will be more difficult for models to improve their recommendation performances, which motivates us to conduct a further investigation to answer RQ2.8; see Section 3.6.3 below.

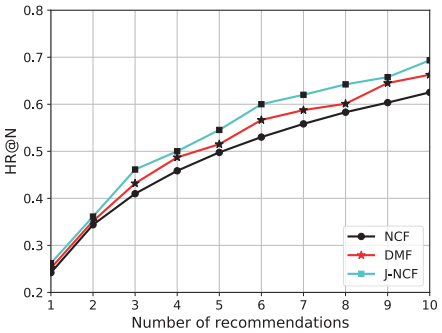
In addition, comparing the left and right-hand side plots in Figure 3.6, we find that J-NCF$_c$ shows a better performance in terms of NDCG@10 than HR@10. For example, the improvements of J-NCF$_c$ over DMF are 9.19%, 8.28% and 15.11% in terms of HR@10 on ML100K-1, ML100K-2 and ML100K-3 datasets, respectively, while the improvements are 10.11%, 10.65% and 20.55% in terms of NDCG@10. This result is consistent with our findings in Section 3.5.3.

Thus in answer to RQ2.7, the J-NCF models outperform the best baseline model DMF across all datasets with different degrees of sparsity in terms of both metrics. Specifically, when applied on large datasets, i.e., ML1M and AMovies, J-NCF$_c$ shows higher improvements against DMF on denser datasets. In addition, the improvements of J-NCF$_c$ over DMF in terms of NDCG@10 are larger than in terms of HR@10.

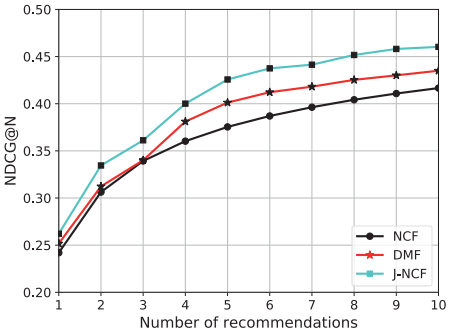
3.6.3 Performance with a large and sparse dataset

For RQ2.8, in order to see if our model is able to work well on a large and sparse dataset, we examine our model as well as two baseline models, i.e., NCF and DMF, on the Amazon Electronic (AEle) dataset, which is larger and sparser than the MovieLens and Amazon Movies datasets. Figure 3.7 shows the performance of the three models with different sizes of top-N recommended lists.

It is clear that J-NCF outperforms DMF as well as NCF in terms of HR and NDCG across different numbers of recommendations. With the size of top-N recommended



(a) Performance in terms of HR@N on AEle dataset.



(b) Performance in terms of NDCG@N on AEle dataset.

Figure 3.7: Performance of Top-N item recommendation where N ranges from 1 to 10, tested on AEle dataset.

lists ranging from 1 to 10, the overall performances of all models increase, which is consistent with the conclusion in Section 3.5.3. Comparing the results shown in Figure 3.7a and Figure 3.7b, the improvements of J-NCF over DMF in terms of NDCG are more significant than those in terms of HR. For example, when $N = 5$ and $N = 10$, the improvements of J-NCF over DMF in terms of HR are 5.88% and 4.62%, while the improvements are 6.12% and 5.82% in terms of NDCG, respectively. To conclude and answer RQ2.8, J-NCF can also work well with large and sparse datasets, especially in ranking items correctly.

3.6.4 Training and inference time

To answer RQ2.9, we investigate the scalability of J-NCF regarding training and inference time in Table 3.8. As shown in Table 3.8, in the "Training" part, "Total time" denotes the time needed for training the model to the best performance. And the "Average epoch" means the average training time for a single epoch in the training process. In the "Prediction" part, "Total time" denotes the prediction time needed for the whole test set. Since the test set contains the latest interaction of every user, the "Average ranking" indicates the time needed for providing a ranked list containing top 10 recommendations for a single user.

As we can see in Table 3.8, when the size of the dataset becomes larger, the time needed for both training and prediction increases significantly for all models. NCF consistently costs the least time among the three models for both training and prediction processes on all datasets. For the training process, the average training time for one epoch of J-NCF is slightly higher than DMF. However, the total training time for J-NCF is less than for DMF. It can be explained by the fact that J-NCF needs fewer iterations to obtain the best results than DMF, as indicated in Section 3.5.5. Thus, J-NCF costs less time for training to the best performance than DMF. For the prediction process, although the total time needed for J-NCF and DMF is more than NCF,

Table 3.8: Training and prediction time needed for baseline models as well as J-NCF on all datasets.

		Training		Prediction	
		Total time (s)	Avg. epoch (s)	Total time (s)	Avg. ranking (s)
ML100K	NCF	46.344	1.943	1.389	0.00147
	DMF	180.017	9.587	1.558	0.00165
	J-NCF	116.023	10.925	1.607	0.00170
ML1M	NCF	494.038	17.751	8.251	0.00137
	DMF	5,451.671	320.687	12.376	0.00205
	J-NCF	3,539.059	340.048	13.858	0.00229
AMovies	NCF	977.265	25.836	25.599	0.00170
	DMF	39,249.657	2,180.537	34.955	0.00232
	J-NCF	31,414.628	2,206.084	37.818	0.00251
AEle	NCF	61,812.187	326.828	2,919.005	0.00239
	DMF	788,138.604	43,785.478	4,360.187	0.00357
	J-NCF	723,586.192	45,224.137	4,775.443	0.00391

the three models cost roughly similar amounts of time for providing a top 10 ranked list for a single user, which is around a few milliseconds.

3.7 Conclusion

We have proposed a joint neural collaborative filtering model, J-NCF, for recommender systems. J-NCF uses a unified deep neural network to tightly couple two important parts in a recommender system, i.e., deep feature learning of users and items, and deep modeling of user-item interactions. For the user and item feature extraction, we use a deep neural network with matrix factorization and a combination of explicit and implicit feedback as input. Then we adopt another neural network for modeling user-item interactions using the feature vectors as inputs. Thus, J-NCF enables the two parts to be optimized with each other through a joint training process. In order to make J-NCF fit the top-N recommendation task, we design a new loss function that incorporates information from both pair-wise and point-wise loss.

The experimental results confirm the effectiveness of J-NCF. In addition, we have also experimentally investigated the performance of J-NCF under various settings, e.g., with different loss functions, with varying numbers of layers in the networks, and with using different feedback as inputs. The results confirm the effectiveness of our hybrid loss function and demonstrate that J-NCF performs better with more layers in the networks and using the combination of implicit and explicit feedback as input.

In addition, we have investigated the robustness of J-NCF with different degrees of data sparsity and different numbers of user ratings. J-NCF outperforms the best baseline model DMF for users across all activity levels, especially for “inactive users”

who constitute the majority of users in the datasets. As for datasets with different levels of sparsity, in general, J-NCF shows more competitive recommendation performance on all datasets than the state-of-the-art baseline model DMF. Moreover, we have also tested J-NCF model with a large and sparse dataset, i.e., AEle, and the results show that J-NCF also outperforms state-of-the-art baseline models on the dataset. Thus in answer to **RQ2**, our proposed J-NCF modeling non-linear user-item relationships as well as characteristics in a unified structure can help to learn users' general preferences.

As to future work, first, we plan to extend J-NCF with more auxiliary information [11, 12, 134, 151], such as the content information of items as well as reviews, to get a more informed expression of users as well as items. As collaborative filtering usually suffers from limited performance due to the sparsity of user-item interactions [113], auxiliary information could be used to boost the performance. It would also be interesting to explore heterogeneous information in a knowledge base to improve the quality of recommender systems with deep learning [146]. Second, we plan to explore the context information of a user in a session with recurrent neural networks to deal with dynamic aspects recommender systems [14, 15, 18, 52]. In addition, an attention mechanism could be applied to J-NCF, which can filter out uninformative content and select the most representative items while providing good interpretability [19]. Finally, as we have found that J-NCF is computationally more expensive than NCF, we plan to optimize the structure and implementation details of our model to make it more efficient.

Next, we will investigate how to learn users' dynamic preferences from their sequential behavior and make session-based recommendations in Chapter 4.

4

Session-based Recommendation with a Dynamic Co-attention Network

In Chapter 3, we have proposed a collaborative filtering approach to learn users' general preferences while ignoring the time orders of their interactions. In this chapter, we aim to learn users' dynamic preferences from their sequential behavior. We propose a dynamic co-attention network approach which incorporates users' long- and short-term preferences for session-based recommendation. The proposed model provides an answer to the following research question asked in Chapter 1: **RQ3**: How can we incorporate users' long- and short-term interaction behavior for session-based recommendation?

4.1 Introduction

Conventional recommender systems often discard sequential information and focus on mining the static relevancy between users and items from interactions [47, 119, 149]. For instance, a typical conventional recommender system based on matrix factorization [67] may be effective at modeling a user's general preferences by learning from their entire interaction history but it does not model the order of the user's interactions. Unlike conventional recommender systems, session-based recommender systems model the evolution of a user's short-term preference implied by sequential interactions in a session with the aim of recommending the next item a user may be interested in [138]. Popular modeling choices for session-based recommender systems include Markov chains and Recurrent Neural Networks (RNNs) [42]. For instance, the Factorizing Personalized Markov Chain (FPMC) model combines Markov chains with matrix factorization to achieve good recommendation performance [104]. Wang et al. [133] propose a Hierarchical Representation Model (HRM) model that extends FPMC by employing a two-layer structure to construct a hybrid representation. Markov chain-based methods only model local sequential patterns between adjacent interactions. RNN-based models can model multi-step sequential behaviors: the Hierarchical Recurrent Neural Network (HRNN) model [98] and Dynamic REcurrent bAsket Model (DREAM) [144] model embed all of a user's historical interactions into

This chapter was published as [23].

the final hidden state of an RNN to represent their current preferences; both achieve significant improvements over HRM and FPMC.

Today's session-based recommender systems successfully capture users' short-term decision making process. But they do not capture variations in the relative importance of a user's long-term vs. short-term interests for session-based recommendation. Users with different shopping preferences may prefer different next items even under the same session context. Thus, how to better capture individual users' dynamic consumption motivations is critical [58, 135].

Our working hypothesis is that the relative importance of events in a user's long-term interaction history depends on events in their short-term interaction history, and vice versa. Let us consider an example. Take a user who has searched for a camera in the current session; her long-term interactions related to electronic products should probably be given a higher weight than her interactions related to clothing when deciding what to recommend next. Conversely, if the user's past interactions indicate a strong interest in the Sony brand, then, during the current session, interactions related to this brand may be more important than others when predicting the next item. But there is more that should be modeled than the relation between past and present interactions. Different user actions, e.g., clicks, add-to-cart, or buy, provide different types of information about the user's interest and, hence, should trigger different follow-up actions. For example, a click on a camera may indicate that the current recommendation is not satisfactory so that substitute offerings should be recommended; adding an item to the cart may show a strong consumption motivation of a user for the item; and while repeat purchases are important [102], a purchase action involving a camera should probably be followed by a recommendation of complementary items [126].

In summary, the main challenges facing session-based recommendation are [58, 135]:

- *How to incorporate user's long-term as well as short -term preferences for session-based recommendation?* and
- *How to capture users' dynamic preferences with implicit preference data?*

To address these questions, we propose a *Dynamic Co-attention Network for Session-based Recommendation* (DCN-SR). Dynamic Co-attention Network for Session-based Recommendation (DCN-SR) has three main components:

(1) The first is a Contextual Gated Recurrent Unit (CGRU) network to model a user's short-term preferences, which we represent as a combination of hidden states of interactions in the current session.

(2) The second is a Multi-Layer Perceptron (MLP) to deal with a user's historical interactions and infer long-term preferences.

(3) The third is a co-attention network that uses the outputs of the first two components to capture interactions between actions in a user's long-term and short-term interaction histories and generate co-dependent representations of their long-term and short-term preferences.

To the best of our knowledge, in the field of session-based recommender systems, ours is the first attempt to use a co-attention network to exploit the relation between a user's long-term and short-term preferences learned from their long-term and short-term interaction history.

Experiments on two e-commerce datasets, the Tmall dataset and the Tianchi dataset, show that DCN-SR outperforms state-of-the-art baselines in prediction accuracy. In addition, we investigate the scalability and sensitivity of DCN-SR with different lengths of search sessions and different numbers of user historical interactions.

In summary, our key technical contributions in this chapter are:

(1) We design a dynamic co-attention network model for session-based recommendation (DCN-SR) that is able to integrate users' long-term and short-term preferences.

(2) We design a contextual gated recurrent unit CGRU to incorporate different types of short-term user actions so as to better estimate a user's next consumption interests.

(3) We analyze the recommendation performance of DCN-SR and find that DCN-SR consistently meets or beats the state-of-the-art, especially with short sessions and active users.

4.2 Related Work

We summarize related work in two areas – sequential recommender systems and attention-based models.

4.2.1 Sequential recommendation models

Interactive systems log users' behavior along with the associated timestamps [54]. Many models have been proposed to leverage this kind of sequential data for modeling users' dynamic preferences and for sequential recommendation. Markov chains have been a popular choice. Following the Factorizing Personalized Markov Chain (FPMC) [104] model, Feng et al. [35] apply metric embeddings with a low dimensional vector for playlist and successive location recommendation. He and McAuley [42] fuse similarity models with Markov chains for sequential recommendation to solve sparse recommendation problems. In order to better capture both users' general taste and sequential behavior, Wang et al. [133] extend FPMC by using a hierarchical structure to learn user representations (HRM). Those Markov chain-based methods only model the local sequential patterns between adjacent interaction events.

Deep neural networks have improved the performance on the sequential recommendation task. Hidasi et al. [51] propose an RNN-based model for session-based recommendation that consists of Gated Recurrent Unit (GRU) units and uses a session-parallel mini-batch training process. With user profiles available, Quadrana et al. [98] develop hierarchical RNNs with cross-session information transfer and Yu et al. [144] propose a dynamic recurrent basket model (DREAM) to capture global sequential patterns for learning a user's dynamic interest representations based on RNNs, which

outperforms HRM and FPMC. DREAM embeds all of a users' historical interactions into the final hidden state of an RNN to represent their current preferences. To improve the performance of RNN-based approaches to sequential recommendation, Tan et al. [118] adopt data augmentation and a method to account for shifts in the input data distribution. The RNN-based approaches listed above usually implicitly encode a user's long-term and short-term interactions into a latent factor or hidden state without distinguishing between the roles that each event may play when making recommendations.

Memory-based approaches leverage user memory networks to store and manipulate a user's previous interactions. Chen et al. [28] propose a Recommendation with User Memory Network (RUM) model to leverage external memory networks integrated with collaborative filtering. It uses a static latent vector to represent users' general preferences and the memory can only store and distinguish users' short-term interactions with a fixed size, which ignores the possibility that different historical interactions may have different degrees of importance.

Our approach to sequential recommendation differs from the work listed above because we do not only exploit the benefits of incorporating long-term and short-term interests, but also consider dynamic aspects of the relation between a user's long-term and short-term preferences. In addition, unlike the work listed above, we explore the information contained in users' different actions.

4.2.2 Neural attention based models

Attention mechanisms have been applied to recommendation tasks to help models exploit users' preferences [48, 84, 120]. Li et al. [73] propose a neural attentive session-based recommendation machine (Neural Attentive Recommendation Machine (NARM)) that takes the last hidden state from the session-based RNN as the sequential behavior, and uses the other hidden states of previous clicks for computing attention to capture users' current preferences in a given session. Although NARM achieves significant improvements over traditional RNN-based approaches, it does not consider users' long-term preferences based on their historical interactions. Ying et al. [143] adopt a hierarchical attention network for sequential recommendation (SHAN). The first attention layer in SHAN learns users' long-term preferences based on the historical purchased item representations, while the second one outputs the final user representation as a combination of the user's long-term and short-term preferences. It is worth pointing out that SHAN generates its attentive representation of user's long-term and short-term preferences independently and thus ignores the relations between them. As to memory-based models, Liu et al. [82] propose a short-term attention memory priority model (STAMP), in which the attention weights are calculated from the session context and enhanced with the final records in the current session.

Our approach to sequential recommendation differs from the aforementioned models in two ways. First, the attention mechanism used in recent sequential recommendation models deals with users' historical and recent interactions separately. In contrast, DCN-SR applies a co-attention network to calculate the correlated importance of actions in both a user's historical and recent interactions, and generates co-dependent representations for their long-term and short-term preferences. Second, previous work considers a user's long-term preferences to be a static vector when dealing with the

user's different short-term interests. In contrast, in DCN-SR, an event in a user's historical interactions may have different degrees of importance when combined with different recent sessions.

4.3 Approach

The DCN-SR model we propose has three main components: a short-term preference generator, a long-term preference generator, and a co-attention network with short-term and long-term preferences. As shown in Figure 4.1, these three components can be trained in a joint manner and give a predicted score of a user's preference for an item through a trilinear composition. We first describe the notation used and then detail the three components in DCN-SR.

4.3.1 Problem formulation and notation

Given a user and their sequential interactions, we aim to recommend their next purchase based on her long-term and short-term preferences learned from those interactions.

For a user u, we denote her current session as $Session_u = \{(x_1, a_1), (x_2, a_2), \ldots, (x_T, a_T)\}$, where x_i is the i-th item in the session and a_i denotes an action (e.g., click, cart or purchase) along with the item; T denotes the number of events in the current session. In addition, we consider the items that u interacts with in her historical sessions and denote them as $History_u = \{x_1, x_2, \ldots, x_N\}$. Here, N denotes the number of events in the user's historical interactions. For exploring the user's long-term preferences, not all actions necessarily depict the user's preference. Therefore, we only retain items with actions that can clearly reveal the user's preference, such as buy or collect. As shown in Figure 4.1, there is an embedding layer at the bottom of the network used for generating the item embeddings as well as the action embeddings. We use $\mathbf{x}_i$ and $\mathbf{a}_i$ to indicate the embeddings of x_i and a_i.

4.3.2 Short-term preference generator

A user's discriminative actions, such as click, collect or purchase, can help to explore sequential interactions as prior knowledge to predict the items that the user is mostly like to access. Since different actions may imply different consumption motivations in a short session, we take all types of actions in the current session into account when learning a user's short-term preferences.

As shown in Figure 4.2, we model a user's sequential interactions in a session with a Contextual GRU network (CGRU) considering the action along with each item as a contextual feature. We modify the operations in a traditional GRU cell and add the action embedding $\mathbf{a}_i$ to the input gate, forget gate and update gate, respectively, shown as the purple arrows in Figure 4.2. The hidden state $\mathbf{h_t}$ in CGRU can be a linear interpolation between the previous hidden state $\mathbf{h}_{t-1}$ and the candidate hidden state $\hat{\mathbf{h}}_t$:

$$\mathbf{h}_t = \mathbf{z}_t \mathbf{h}_{t-1} + (1 - \mathbf{z}_t)\hat{\mathbf{h}}_t, \tag{4.1}$$

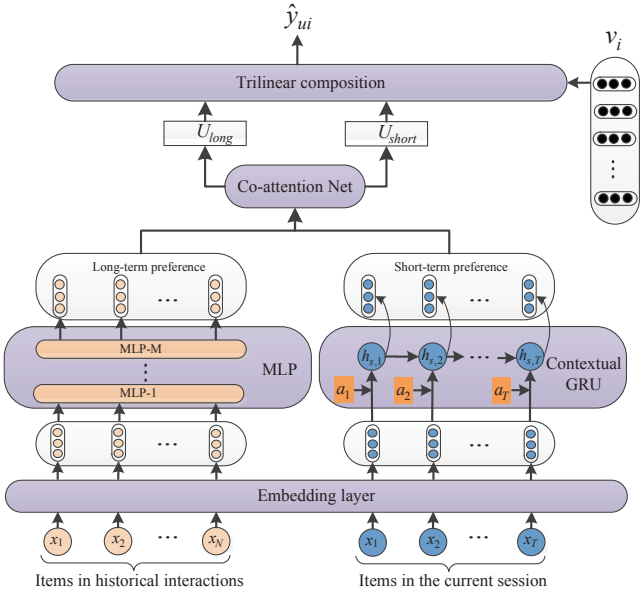


Figure 4.1: Structure of the DCN-SR model.

where the update gate $\mathbf{z}_t$ is given by:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{V}_z \mathbf{a}_t + \mathbf{U}_z \mathbf{h}_{t-1}), \tag{4.2}$$

where $\mathbf{W}_z$, $\mathbf{V}_z$ and $\mathbf{U}_z$ are update parameters for $\mathbf{x}_t$, $\mathbf{a}_t$ and $\mathbf{h}_{t-1}$, respectively. The candidate hidden state can be computed as:

$$\hat{\mathbf{h}}_t = \tanh(\mathbf{W}\mathbf{x}_t + \mathbf{V}\mathbf{a}_t + \mathbf{r}_t \odot \mathbf{U}\mathbf{h}_{t-1}), \tag{4.3}$$

where the reset gate $\mathbf{r}_t$ can be calculated by:

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{V}_r \mathbf{a}_t + \mathbf{U}_r \mathbf{h}_{t-1}), \tag{4.4}$$

where $\mathbf{W}_r$, $\mathbf{V}_r$ and $\mathbf{U}_r$ are reset parameters for $\mathbf{x}_t$, $\mathbf{a}_t$ and $\mathbf{h}_{t-1}$, respectively.

As each hidden state contains the information of a user's search intent in the current session, we use a collection of hidden states to represent a user's initial short-term preference as $\mathbf{U}_s = \{\mathbf{h}_{s,1}, \mathbf{h}_{s,2}, \ldots, \mathbf{h}_{s,T}\}$ and $\mathbf{U}_s \in \mathbb{R}^{D \times T}$, where D is the dimension of each hidden state in $\mathbf{U}_s$. We will future explore the user's interest drift across these hidden states with a co-attention network in Section 4.3.4.

4.3.3 Long-term preference generator

As we discussed above, when exploring a user's long-term preference with their historical interactions, we only retain the items with actions that could depict users' preferences (e.g., buy or collect). We feed the dense low-dimensional embedding of each

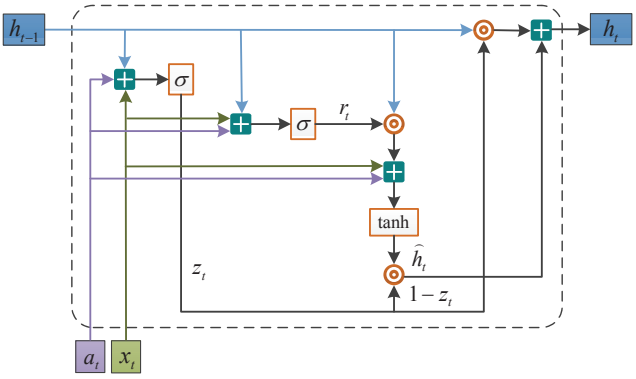


Figure 4.2: Structure of the Contextual GRU network.

item in a user's historical interactions $History_u = \{x_1, x_2, \ldots, x_N\}$ through a multi-layer perceptron (MLP) to generate hidden representations of those events:

$$\begin{aligned}
\mathbf{z}_{1,i} &= \phi(\mathbf{W}_1\mathbf{x}_i + \mathbf{b}_1) \\
\mathbf{z}_{2,i} &= \phi(\mathbf{W}_2\mathbf{z}_{1,i} + \mathbf{b}_2) \\
&\vdots \\
\mathbf{z}_{M,i} &= \tanh(\mathbf{W}_M\mathbf{z}_{M-1,i} + \mathbf{b}_M) \\
\mathbf{X}_i &= \mathbf{z}_{M,i},
\end{aligned} \tag{4.5}$$

where $\mathbf{W}_m$, $\mathbf{b}_m$ and ϕ denote the weight matrix, the bias vector and the activation function in the m-th layer. Here, we use a ReLU as the activation function, as it has been shown to be more expressive than others and can deal with the vanishing gradient problem effectively [46, 55]. M indicates the number of layers used in MLP network. The output of the final layer $\mathbf{X}_i$ is the hidden representation of the i-th event. We also apply a collection of these event representations to indicate a user's initial long-term preference as $\mathbf{U}_l = \{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_N\}$ and $\mathbf{U}_l \in \mathbb{R}^{D\times N}$. We use a MLP network because of its non-linear modeling capability, which has been applied in many neural collaborative filtering works and shows reliable performance [46].

4.3.4 Co-attention network

It is beneficial to incorporate the short-term and long-term preference of a user when making recommendations. However, traditional methods treat these two types of preference as independent [143], which ignores the (potential) mutual dependence between them.

In addition, conventional attention mechanisms assign weights for the events in a user's historical and recent interactions separately. We argue that historical interactions and recent interactions can provide context for each other when calculating the importance of each event. Thus, we design a co-attention network to explore correlations between historical and current interactions of a user.

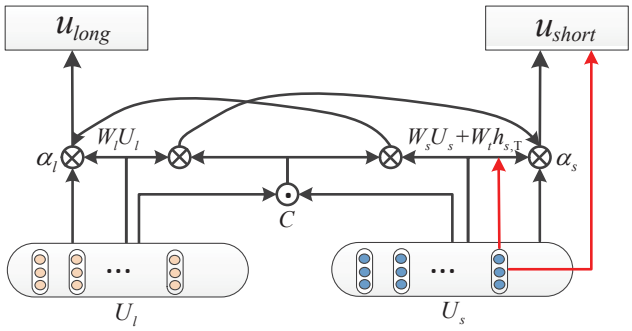


Figure 4.3: Structure of the co-attention network.

As shown in Figure 4.3, after generating a user's initial short-term and long-term preferences, we use them as the inputs of the co-attention network and calculate the affinity matrix $\mathbf{C}$:

$$\mathbf{C} = \tanh(\mathbf{U}_l^{\mathrm{T}} \mathbf{W}_c \mathbf{U}_s), \tag{4.6}$$

where $\mathbf{W}_c \in \mathbb{R}^{D \times D}$ contains the weights. After computing the affinity matrix, we consider it as a feature and use it to transform the short-term attention space into the long-term attention space with:

$$\mathbf{H}^l = \tanh(\mathbf{W}_l \mathbf{U}_l + (\mathbf{W}_s \mathbf{U}_s + \mathbf{W}_t \mathbf{h}_{s,T})\mathbf{C}^{\mathrm{T}}) \tag{4.7}$$

$$\alpha_l = \mathrm{softmax}(\mathbf{w}_{hl}^{\mathrm{T}} \mathbf{H}^l) \tag{4.8}$$

and vice versa:

$$\mathbf{H}^s = \tanh(\mathbf{W}_s \mathbf{U}_s + \mathbf{W}_t \mathbf{h}_{s,T} + (\mathbf{W}_l \mathbf{U}_l)\mathbf{C}) \tag{4.9}$$

$$\alpha_s = \mathrm{softmax}(\mathbf{w}_{hs}^{\mathrm{T}} \mathbf{H}^s), \tag{4.10}$$

where $\mathbf{W}_l$, $\mathbf{W}_s$, $\mathbf{W}_t \in \mathbb{R}^{K \times D}$, $\mathbf{w}_{hl}$, $\mathbf{w}_{hs} \in \mathbb{R}^K$ are weight parameters for long-term and short-term preferences, respectively. Here, $\alpha_l \in \mathbb{R}^N$ and $\alpha_s \in \mathbb{R}^T$ are the attention probabilities for the events in historical and current interactions, respectively.

It should be noted in Eq. (4.7) and Eq. (4.9) that besides the collection of the hidden states in current session, i.e., $\mathbf{U}_s$, we explicitly consider the final hidden state in the current session, i.e., $\mathbf{h}_{s,T}$, shown as the red arrows in Figure 4.3. Importantly, $\mathbf{h}_{s,T}$ summarizes the complete sequential behavior, which contains different information from $\mathbf{U}_s$ when exploring user's short-term preferences [73, 82]. Both NARM [73] and STAMP [82] have shown that the explicit use of $\mathbf{h}_{s,T}$ improves the performance of session-based recommendations.

Based on the attention weights, the co-dependent representations of a user's long-term and short-term preferences can be calculated as the weighted sum of their interactions representations:

$$\mathbf{U}_{co-l} = \textstyle\sum_{n=1}^{N} \alpha_l^n \mathbf{X}_n, \tag{4.11}$$

and

$$\mathbf{U}_{co-s} = \sum_{t=1}^{T} \alpha_s^t \mathbf{h}_{s,t}. \tag{4.12}$$

In order to take $\mathbf{h}_{s,T}$ into consideration, we use $\mathbf{U}_{long} = \mathbf{U}_{co-l}$, $\mathbf{U}_{short} = [\mathbf{U}_{co-s}; \mathbf{h}_{s,T}]$ to represent the final representations of a user's long-term and short-term preference. And then, for a given candidate item x_i, the scoring function that produces a prediction can be a trilinear combination:

$$\hat{z}_{ui}^l = \mathbf{v}_i^{\mathrm{T}} \mathbf{B}_l \mathbf{U}_{long} \quad \hat{z}_{ui}^s = \mathbf{v}_i^{\mathrm{T}} \mathbf{B}_s \mathbf{U}_{short} \quad \hat{z}_{ui} = \sigma(< \hat{z}_{ui}^l, \hat{z}_{ui}^s >), \tag{4.13}$$

where $\mathbf{B}_l \in \mathbb{R}^{E \times D}$, and $\mathbf{B}_s \in \mathbb{R}^{E \times 2D}$, E is the dimension of each item embedding. The trilinear combination incorporates the user's long-term preferences as well as their short-term preferences towards an item. Moreover, $\hat{z}_{ui}$ represents the unnormalized cosine similarity between the user's preference and the i-th candidate item. We use $\hat{\mathbf{z}}_u \in \mathbb{R}^V$ to denote the vector that consists of $\hat{z}_{ui} (i \in [1, \ldots, V])$, where V is the number of candidate items. It is then processed by a softmax function:

$$\hat{\mathbf{y}}_u = \mathrm{softmax}(\hat{\mathbf{z}}_u), \tag{4.14}$$

where $\hat{\mathbf{y}}_u$ denotes the output vector of our model, which represents a probability distribution over the candidate items, and each element $\hat{y}_{ui}$ denotes the probability of the item v_i being the next purchase.

We adopt the cross-entropy loss as our loss function:

$$L(\hat{\mathbf{y}}_u) = -\sum_{i=1}^{V} \mathbf{y}_u \log(\hat{\mathbf{y}}_u), \tag{4.15}$$

where $\mathbf{y}_u$ is the true distribution.

Finally, a Back-Propagation Through Time (BPTT) method with a fixed number of time steps is adopted to train our DCN-SR model based on Eq. (4.15).

4.4 Model Analysis

To provide insights into DCN-SR, we discuss its connection to previous work on session-based recommendation. By choosing appropriate settings, DCN-SR can subsume several existing methods, including session-based recommendations with recurrent neural networks (GRU4Rec) and an attention-based model, i.e., the Neural Attentive Recommendation Machine (NARM).

4.4.1 DCN-SR vs. GRU4Rec

GRU4Rec is an RNN-based approach that uses the final hidden state to represent a user's preference:

$$\mathbf{h}_T = GRU_{sess}(\mathbf{v}_T, \mathbf{h}_{T-1}), \tag{4.16}$$

and predict the score for a candidate item v_i as:

$$\hat{z}_{ui} = \sigma(\mathbf{v}_i^{\mathrm{T}} \mathbf{h}_T). \tag{4.17}$$

As shown in Figure 4.1, when we do not consider the user's historical interactions and different actions, DCN-SR will reduce to an RNN-based approach. To show how our model degenerates to GRU4Rec, we set the historical interactions empty and the weight parameter $\mathbf{w}_{hs} = 0$; then, in the co-attention component, the affinity matrix $\mathbf{C}$, $\mathbf{U}_{co-l}$ and $\mathbf{U}_{co-s}$ will be 0. The user's preferences will be equal to the final hidden state of the session, i.e., $\mathbf{U}_{short} = [\mathbf{U}_{co-s}; \mathbf{h}_{s,T}] = \mathbf{h}_{s,T}$. And the prediction score is calculated as:

$$\hat{z}_{ui} = \sigma(\hat{z}^s_{ui}) = \sigma(\mathbf{v}_i^{\mathrm{T}} \mathbf{U}_{short}) = \sigma(\mathbf{v}_i^{\mathrm{T}} \mathbf{h}_{s,T}). \tag{4.18}$$

This is the same as the prediction function (Eq. (4.17)) of GRU4Rec. However, by enabling a user's historical interactions, DCN-SR is able to collect valuable information for her long-term preference. In addition, with the weight parameter $\mathbf{w}_{hs}$, DCN-SR can adaptively select important items in the current session to capture the user's short-term interest, which can bring improved performance in the task of sequential recommendation as shown in our experiments.

4.4.2 DCN-SR vs. NARM

Both DCN-SR and NARM apply an attention mechanism to capture a user's main interest. In NARM, the attention mechanism takes the last hidden state $\mathbf{h}_T$ from the RNN as the sequential behavior, which denotes the global encoder of the current session:

$$\mathbf{c}_g = \mathbf{h}_T. \tag{4.19}$$

It then uses the hidden states of previous clicks in the current session for computing attention scores, which is a local encoder combining different parts of the sequence:

$$\mathbf{c}_l = \sum_{i=1}^{T} \alpha_i \mathbf{h}_i, \tag{4.20}$$

where α is the weighted factor calculated by:

$$\alpha_i = \mathbf{v}^{\mathrm{T}} \sigma(\mathbf{A}_1 \mathbf{h}_i + \mathbf{A}_2 \mathbf{h}_T), \tag{4.21}$$

where σ is an activation function, and $\mathbf{A}_1$ and $\mathbf{A}_2$ are used to transform $\mathbf{h}_i$ and $\mathbf{h}_T$ into a latent space.

By concatenating the global and local encoder, NARM adopts a unified representation $\mathbf{c}$ to model the user's short-term preference:

$$\mathbf{c} = [c_g; c_l] = [\mathbf{h}_T; \textstyle\sum_{i=1}^{T} \alpha_i \mathbf{h}_i]. \tag{4.22}$$

The prediction score for a candidate item v_i is calculated as:

$$\hat{z}_{ui} = \mathbf{v}_i^{\mathrm{T}} \mathbf{B} \mathbf{c}, \tag{4.23}$$

where $\mathbf{B}$ is a latent parameter.

To see the connection between DCN-SR and NARM, we set the user's historical interactions empty and ignore different actions in RNN. Thus DCN-SR can be reduced to:

$$\mathbf{C} = \tanh(\mathbf{U}_l^{\mathrm{T}} \mathbf{W}_c \mathbf{U}_s) = 0, \tag{4.24}$$

and

$$\begin{aligned}\mathbf{H}^s &= \tanh(\mathbf{W}_s\mathbf{U}_s + \mathbf{W}_t\mathbf{h}_{s,T} + \mathbf{W}_l\mathbf{U}_l\mathbf{C}) \\ &= \tanh(\mathbf{W}_s\mathbf{U}_s + \mathbf{W}_t\mathbf{h}_{s,T}).\end{aligned} \tag{4.25}$$

Because $\mathbf{U}_s$ is a set of hidden states in RNN, i.e., $\mathbf{U}_s = \{\mathbf{h}_{s,1}, \mathbf{h}_{s,2}, \ldots, \mathbf{h}_{s,T}\}$, we divide Eq. (4.25) for each hidden state as:

$$\mathbf{H}_i^s = \tanh(\mathbf{W}_s\mathbf{h}_{s,i} + \mathbf{W}_t\mathbf{h}_{s,T}). \tag{4.26}$$

Then, the attention weight for each event in the current interactions is calculated by:

$$\begin{aligned}\alpha_s^i &= \text{softmax}(\mathbf{w}_{hs}^{\mathrm{T}}\mathbf{H}_i^s) \\ &= \text{softmax}(\mathbf{w}_{hs}^{\mathrm{T}}\tanh(\mathbf{W}_s\mathbf{h}_{s,i} + \mathbf{W}_t\mathbf{h}_{s,T})).\end{aligned} \tag{4.27}$$

We can see that the attention weights calculated in our model are the same as NARM. Based on these attention weights, we can rewrite a user's final short-term preference as:

$$\mathbf{U}_{short} = [\mathbf{U}_{co-s}; \mathbf{h}_{s,T}] = \left[\sum_{i=1}^{T}\alpha_s^i\mathbf{h}_{s,i}; \mathbf{h}_{s,T}\right]. \tag{4.28}$$

The prediction score is generated by:

$$\hat{z}_{ui} = \sigma(\hat{z}_{ui}^s) = \sigma(\mathbf{v}_i^{\mathrm{T}}\mathbf{B}_s\mathbf{U}_{short}). \tag{4.29}$$

According to these derivations, we see that by choosing proper activation functions in Eq. (4.27) and (4.29), DCN-SR and NARM have the same representations of users' preferences and prediction function.

Based on our analysis, we see that DCN-SR is a very general model for session-based recommendation. On the one hand, by introducing different settings (i.e., parameters and activation functions) DCN-SR can be seen as a generalization of many existing models. On the other hand, DCN-SR enables us to explore more information from users' historical interactions and the dynamic correlations between long-term and short-term preferences.

4.5 Experiments

We refine **RQ4** into the following more fine-grained questions:

(RQ3.1) Does DCN-SR outperform state-of-the-art baselines for session-based recommendation?

(RQ3.2) Does the Contextual GRU, which incorporates different user actions, contribute to the performance of DCN-SR?

(RQ3.3) How is the performance of DCN-SR impacted by sessions of different lengths?

(RQ3.4) How does the performance of DCN-SR vary across users with different numbers of historical interactions?

(RQ3.5) How can we visualize the co-attention mechanism?

4.5.1 Model summary

As DCN-SR considers users' long-term and short-term preferences, we mainly compare our method with personalized session-based recommendation models, i.e., HRNN and SHAN. In addition, we also consider some traditional models, i.e., FPMC and Item-pop, as well as neural models with or without an attention mechanism, i.e., NARM, STAMP and GRU4Rec. These are our baselines:

Item-pop A method that ranks items based on the number of interactions, which is a non-personalized approach [3].

FPMC A state-of-the-art hybrid model for sequential recommendation, based on Markov chains and collaborative filtering. Both sequential behaviors and general taste are taken into account [104].

GRU4Rec An RNN-based model for session-based recommendation, which contains GRUs and utilizes session-parallel mini-batches as well as a pair-wise loss function for training [51].

NARM An RNN-based model that applies an attention mechanism to capture users' main purposes from the hidden states and combines it with sequential behavior as final representations of users' current preferences [55].

STAMP A memory-based model with attention mechanism that explicitly considers correlations between each click and the last click in a session. It combines the weighted events and the last click to model users' current preferences [82].

HRNN A hierarchical RNN for personalized session-based recommendation which uses a session- and a user-level RNN to model users' short- and long-term preferences [98].

SHAN A personalized session-based recommendation method that adopts a hierarchical attention network, in which the first attention layer learns users' long-term preferences while the second one outputs the final user representation as a combination of the user's long-term and short-term preferences [143].

4.5.2 Datasets and experimental setup

Datasets. We use two publicly available real-world datasets to evaluate our models and the baselines. *Tmall* is a dataset released by Taobao.[1] It contains records of online transactions, with 884 users, 9,531 brands and 182,880 interactions. Customer action types include click, collect, cart, and purchase. *Tianchi* is a dataset provided by Alibaba.[2] It is based on user-commodity behavior data of Alibaba's M-Commerce platforms. It contains 23,291,027 interactions of 20,000 customers on 4,758,484 items within a month plus category information of each item. Customer actions include click, collect, cart, and purchase.

[1] http://102.alibaba.com/competition/addDiscovery/index.htm

[2] https://tianchi.aliyun.com/getStart/information.htm?spm=5176.100067.5678.2.30a8b6d933N6Rr&raceId=231522

Table 4.1: Dataset statistics.

Dataset	Tmall	Tianchi
#of users	822	14,080
#of items	5,823	40,886
#of interactions	157,709	3,782,379
#of action types	4	4
Average interactions per user	192.13	272.03
Average interactions per item	27.03	93.68
# interactions in training set	147,735	2,897,330
# interactions in test set	9,974	885,049

For the Tmall dataset, we filter out users with fewer than 3 interactions and items that appear less than 3 times [144]. For the Tianchi dataset, we filter out users with fewer than 20 interactions and items with fewer than 50 interactions. The characteristics of the datasets after preprocessing are summarized in Table 4.1.

Settings and parameters. For evaluation, we divide the Tmall and Tianchi datasets into training and test sets according to the users' search time. The training set consists of all but the last 7 days of interactions; the test set contains the remaining 7 days of interactions. As collaborative filtering methods cannot recommend an item that has not appeared before, we filter out interactions from the test set with items that do not appear in the training set.

For the Tmall and Tianchi datasets, we treat user records in one day as a session to model short-term preferences, following [81, 143]. For the Tmall dataset, although there is no detailed time information beyond one day, the sequential information of user behaviors on items still exists, so we can also model it with an RNN [81].

Unless specified differently, for all the results that we presented, the number of recommendations (N) equals 10 [46, 55]. We use *Recall@10* and *MRR@10* to evaluate the performance of models [73, 82]. Recall@10 is used to evaluate the recall of the recommender system, i.e., whether the test item is contained in the top 10 list. MRR@10 measures the ranking accuracy of the recommender system, i.e., whether the test item is ranked at the top of the list.

We optimize the hyperparameters using Adam [66] with the initial learning rate set to 0.01, and the mini-batch size fixed at 512. The dimension of the item embeddings is set to 50 and we use one GRU layer with 100 hidden units. Optimization is done on a validation set, which is partitioned from the training set with the same procedure as the test set. The source codes for our model are available on our public repository.[3]

[3] `https://bitbucket.org/WanyuChen/dsr/`

4.6 Results and Discussion

4.6.1 Overall performance

To answer RQ3.1, we examine the recommendation performance of the baselines and DCN-SR. See Table 4.2.

Table 4.2: Performance of recommendation models. The results produced by the best baseline and the best performer in each column are underlined and bold-faced, respectively. Statistical significance of pairwise differences of DCN-SR vs. the best baseline is determined by a t-test ($^{\blacktriangle}$ for α = .01, or $^{\triangle}$ for α = .05).

	Tmall		Tianchi	
Model	Recall@10	MRR@10	Recall@10	MRR@10
Item-pop	.1058	.0455	.0022	.0011
FPMC	.1813	.1227	.0594	.0377
GRU4Rec	.5852	.5613	.1117	.0875
NARM	.7237	.6781	.3155	.1909
STAMP	<u>.7246</u>	<u>.6872</u>	<u>.3185</u>	<u>.1955</u>
HRNN	.6894	.6617	.1971	.1801
SHAN	.7101	.6687	.2208	.1843
DCN-SR	**.7433**$^{\triangle}$	**.7132**$^{\blacktriangle}$	**.3283**$^{\triangle}$	**.2034**$^{\blacktriangle}$

Let us first consider the baselines. From Table 4.2, we see that neural-based approaches outperform traditional methods, i.e., Item-pop and FPMC. As to non-personalized session-based approaches, i.e., GRU4Rec, NARM and STAMP, we see that NARM and STAMP both improve over GRU4Rec, which indicates the utility of using an attention mechanism. This result can also be proved by comparing the results of the personalized models, i.e., HRNN and SHAN, where SHAN with a hierarchical attention structure shows better performance than HRNN. The results of HRNN are higher than of a simple RNN-based approach such as GRU4Rec, which means that incorporating users' historical and recent interactions together can help to boost the recommendation performance. STAMP and NARM show better results than SHAN. The explicit use of the last hidden state seems to improve the performance for session-based recommendations, as the last behavior in a short session can reveal users' current consumption motivations better. STAMP outperforms other baselines in terms of Recall@10 and MRR@10. Hence, we use STAMP as our baseline in later experiments.

Next, we compare the baselines against the DCN-SR model. Personalized and non-personalized models, i.e., SHAN and STAMP, both lose against DCN-SR in terms of Recall@10 and MRR@10. This shows that using the co-attention network helps to improve the recommendation performance. This may be due to two factors: one is that with the co-attention network, DCN-SR can capture the mutual dependence between users' historical and recent interactions and learn dynamic representations of users' long- and short-term preferences; the other is that DCN-SR integrates both a user's long- and short-term preferences to predict their next interactions.

The improvements of DCN-SR over the best baseline model in terms of Recall@10 are 2.58% on the Tmall dataset and 3.08% on the Tianchi dataset. MRR@10 improvements are 3.78% on the Tmall dataset and 4.05% on the Tianchi dataset. Significant improvements against the best performing baseline are observed for the DCN-SR model at the $\alpha = .01$ level in terms of MRR@10 on both datasets. For Recall@10, we observe significant improvements at the $\alpha = .05$ level on both datasets. The fact that improvements in terms of MRR@10 are bigger than in terms of Recall@10 suggests that the main effect of DCN-SR's architecture is to boost the ranking of relevant items rather than the number of relevant items found.

4.6.2 The Contextual GRU network

For RQ3.2, in order to demonstrate the utility of the CGRU network, which considers users' actions as search context in a short session, we examine the recommendation performance of DCN-SR under different settings, i.e., DCN-SR$_{GRU}$ (with a simple GRU network) vs. DCN-SR$_{CGRU}$ (with the Contextual GRU network). Table 4.3 contrasts their performance against the best baseline model (STAMP), with different numbers of recommended items N.

Table 4.3: Recommendation performance with different numbers of recommended items N. The results produced by the best performer in each column with a certain N are boldfaced. Statistical significance of pairwise differences is determined by a t-test ($^{\blacktriangle}$ for $\alpha = .01$ and $^{\triangle}$ for $\alpha = .05$ when comparing DCN-SR$_{CGRU}$ or DCN-SR$_{GRU}$ vs. STAMP; $^{\bullet}$ for $\alpha = 0.01$ and $^{\circ}$ for $\alpha = 0.05$ when comparing DCN-SR$_{CGRU}$ vs. DCN-SR$_{GRU}$).

		Tmall		Tianchi	
N		MRR	Recall	MRR	Recall
5	STAMP	.6859	.7148	.1895	.2685
	DCN-SR$_{GRU}$	.6964$^{\triangle}$	.7256$^{\triangle}$	.1939$^{\triangle}$	.2744$^{\triangle}$
	DCN-SR$_{CGRU}$	**.6999**$^{\blacktriangle\bullet}$	**.7272**$^{\triangle\circ}$	**.1955**$^{\blacktriangle\bullet}$	**.2752**$^{\triangle\circ}$
10	STAMP	.6872	.7246	.1955	.3184
	DCN-SR$_{GRU}$	.7016$^{\triangle}$	.7395$^{\triangle}$	.2016$^{\triangle}$	.3278$^{\triangle}$
	DCN-SR$_{CGRU}$	**.7132**$^{\blacktriangle\bullet}$	**.7433**$^{\triangle\circ}$	**.2034**$^{\blacktriangle\bullet}$	**.3283**$^{\triangle\circ}$
15	STAMP	.6878	.7314	.1985	.3473
	DCN-SR$_{GRU}$	.7032$^{\triangle}$	.7470$^{\triangle}$	.2046$^{\triangle}$	.3575$^{\triangle}$
	DCN-SR$_{CGRU}$	**.7147**$^{\blacktriangle\bullet}$	**.7501**$^{\triangle\circ}$	**.2067**$^{\blacktriangle\bullet}$	**.3581**$^{\triangle\circ}$

DCN-SR$_{GRU}$, which lacks users' action information, still beats the best baseline model, i.e., STAMP, which indicates that the dynamic co-attention network helps to improve the performance of sequential recommendations. DCN-SR$_{CGRU}$ consistently achieves improvements over DCN-SR$_{GRU}$, which demonstrates the utility of the Contextual GRU network. Improvements of DCN-SR$_{GRU}$ over STAMP are significant at the $\alpha = .05$ level in terms of MRR and Recall, on both datasets. For DCN-SR$_{CGRU}$, we observe significant improvements at the $\alpha = .05$ level in terms of Recall, and at the

$\alpha = .01$ level in terms of MRR on the two datasets.

Regarding different numbers of recommendations, we see that the overall performance in terms of Recall and MRR increases when N ranges from 5 to 15, as a large value of N increases the probability of including a user's preferred item in the list.

The improvements of DCN-SR$_{CGRU}$ in terms of MRR are more significant than those in terms of Recall, as indicated by the relative improvements over DCN-SR$_{GRU}$ with different numbers of recommendations. We further conduct paired t-tests, verifying that these improvements are statistically significant for $\alpha = .05$ in terms of Recall and $\alpha = .01$ in terms of MRR. These improvements can prove that incorporating the information contained in users' different actions helps to learn more accurate representations of users' short-term preferences.

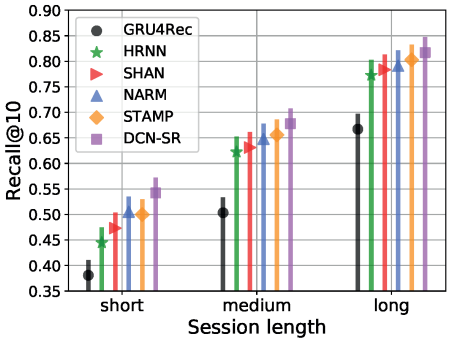
4.6.3 Session length

In order to understand the scalability of sequential recommendation models when applied with sessions of different lengths (RQ3.3), we divide the sessions in the datasets into *short* (no more than 5 items), *medium* (6 to 15 items) and *long* sessions (more than 15 items) on the test set and report separate results in Figure 4.4. We do not Item-pop and FPMC in the comparison, as their performance is worse than that of the RNN-based models, especially with short sessions.

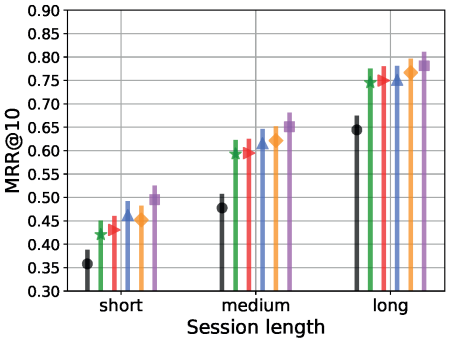
From Figure 4.4 we can see that as the session length increases, the performance of all models improves on the Tmall dataset while it decreases on the Tianchi dataset. The DCN-SR model always achieves the highest scores, on both datasets, across different session lengths. Specifically, for Recall@10, as shown in Figure 4.4a and Figure 4.4c, among the baselines, NARM and STAMP perform better than the model without attention mechanism, i.e., GRU4Rec, across all three session lengths. As for the personalized methods, although both have a hierarchical structure, SHAN shows better performance than HRNN across all session lengths, which demonstrates the utility of an attention mechanism that combines long- and short-term preferences. STAMP outperforms NARM except when applied with short sessions; this may be due to the fact that short sessions contain less information than long sessions, thus RNN-based model, i.e., NARM, can provide positional and sequential information as a supplementary for recommendation, while STAMP lacks information for predicting users' preferences with few interactions.

For MRR@10, a similar trend is shown in Figure 4.4b and 4.4d. Particularly, DCN-SR shows larger improvements over STAMP in terms of MRR@10 than Recall@10, which is consistent with our findings in Table 4.2. For the Tmall dataset, the improvements are 9.03%, 4.54% and 1.83% in terms of MRR@10, for short, medium and long sessions, respectively, vs. improvements of 8.02%, 3.21% and 1.72% in terms of Recall@10. For Tianchi dataset, the improvements are 7.33%, 6.42% and 3.82% in terms of MRR@10, for short, medium and long sessions, respectively, vs. 5.02%, 3.69% and 1.74% for Recall@10.

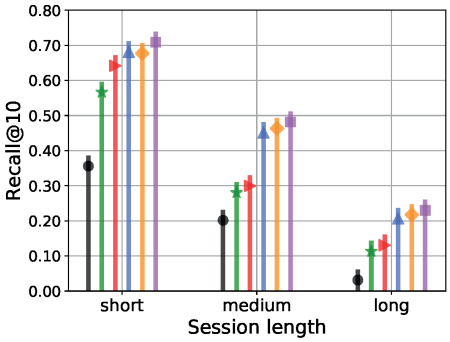
The improvements of DCN-SR over STAMP are more obvious for short sessions than for long sessions. This may be because (1) we consider the users' historical interactions with the co-attention network, which can provide us with users' long-term preferences when making recommendations for short sessions; (2) the CGRU network



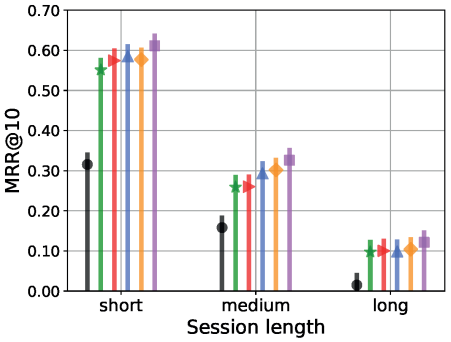
(a) Performance in terms of Recall@10 on Tmall.



(b) Performance in terms of MRR@10 on Tmall.



(c) Performance in terms of Recall@10 on Tianchi.



(d) Performance in terms of MRR@10 on Tianchi.

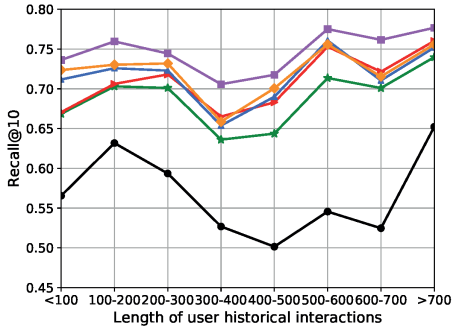
Figure 4.4: Effect on the performance of six models in terms of Recall@10 and MRR@10 of different session lengths.

incorporates users' action information, which supplies additional information on users' consumption motivations.

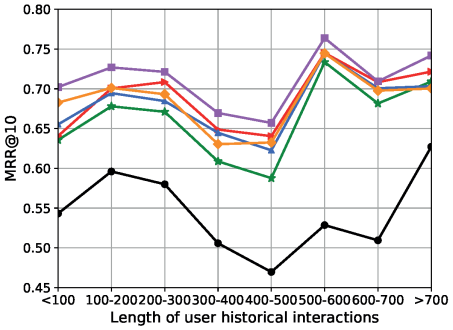
4.6.4 The length of historical interactions

To answer RQ3.4, we evaluate the sequential recommendation models that we consider with different volumes of users' historical interactions. This time, we group results by the length of users' historical interactions, which is denoted as H. That is, we use both datasets and partition the users into eight groups: $H < 100$, $H \in [100, 200)$, $H \in [200, 300)$, $H \in [300, 400)$, $H \in [400, 500)$, $H \in [500, 600)$, $H \in [600, 700]$, and $H > 700$. In order to see the impact of the length of a user's historical interactions on the recommendation performance, we compare the performance of DCN-SR with five baseline models except Item-pop and FPMC; see Figure 4.5.

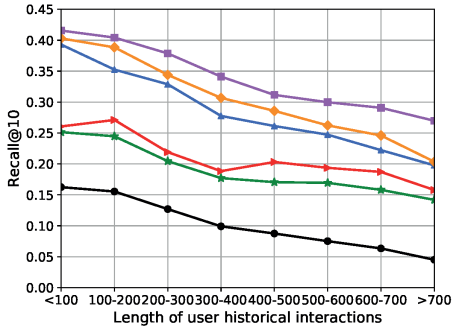
DCN-SR achieves the best performance for all of the eight groups on both datasets. For the Tmall dataset, when the number of users' historical interactions increases, the



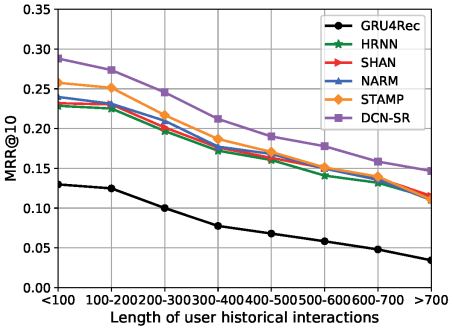
(a) Performance in terms of Recall@10 on Tmall.



(b) Performance in terms of MRR@10 on Tmall.



(c) Performance in terms of Recall@10 on Tianchi.



(d) Performance in terms of MRR@10 on Tianchi.

Figure 4.5: Effect on the performance of six models in terms of Recall@10 and MRR@10 with different numbers of historical interactions, tested on the Tmall and Tianchi datasets.

performance of all models begins to fluctuate at first but shows an upward trend overall. In particular, as the number of interactions increases, the performance of DCN-SR, SHAN and HRNN improves more noticeably than of STAMP and NARM. For example, SHAN shows better performance than STAMP and NARM in terms of Recall@10 and MRR@10 when the number of interactions is more than 700. The performance gap between DCN-SR and STAMP in terms of MRR@10 increases when the number of interactions increases from the seventh group ($[600, 700]$) to the eighth group (> 700).

For the Tianchi dataset, the performance of all models decreases in terms of both metrics as we consider longer histories. The results for DCN-SR, SHAN and HRNN decline more slowly than for the STAMP and NARM model, which is consistent with our findings in Figure 4.5a and 4.5b. E.g., the improvements of DCN-SR over STAMP are 9.03%, 14.27%, 18.19% and 32.51% under the fifth ($H \in [400, 500)$), sixth ($H \in [500, 600)$), seventh ($H \in [600, 700]$), and eighth ($H > 700$) groups in terms of

Figure 4.6: Co-attention visualization. The depth of color indicates the importance of an event. The red number above the bar is the category of the corresponding item.

Recall@10. This shows the effectiveness of using personalization strategies, i.e., users' long-term preferences, to improve the recommendation performance.

4.6.5 Co-attention visualization

To illustrate the role of the co-attention mechanism (RQ3.5), we present examples of two users in Figure 4.6. For each, we randomly choose two sessions from the test set on the Tianchi dataset, as the Tianchi dataset contains category information for items, which helps us assess the association between interactions. In Figure 4.6, the depth of the color indicates the importance of an event, the darker the color the more important an event is. The red numbers above the bar are the categories of the corresponding items.

DCN-SR is capable of highlighting a number of factors in predicting a user's next interaction as shown in Figure 4.6. First, although the two sessions of a single user share the same historical interactions, the weights of these historical interactions differ. For example, for User A, the first event in the historical interactions plays a more important role in Session1 than in Session2. Also, items that have the same category as the target item have larger attention weights than others. The category of an item can partially reflect the interest of the user, thus it indicates that the co-attention mechanism captures the user's dynamic interests to some extent.

Second, the interactions in a session also have different weights for predicting a user's preference, which proves that DCN-SR can select important events and ignore unintended interactions. In addition, interactions close to the end of the session often have larger importance, which is especially clear in Session1 for User B. This confirms our intuition that incorporating a user's last interaction in the co-attention mechanism can help to improve the performance.

Third, there are some important interactions in a session that are not near the user's last click. For example, in Session2 for User A, the sixth event is more important than the last event. This may be due to the user's interests drift. However, DCN-SR can also pick them up and give them high weights.

Therefore, based on the visualization results, we claim that the co-attention mechanism is able to capture important events both in users' historical interactions as well

as their current interactions.

4.7 Conclusion

In this capter, we propose a dynamic co-attention network for session-based recommendation, DCN-SR. DCN-SR applies a co-attention network to capture the dynamic relations between a user's long-term and short-term interactions and generate co-dependent representations of the user's long-term and short-term preferences. It not only exploits the combination of long-term and short-term knowledge, but also considers dynamic aspects of the relation between a user's long-term and short-term preferences. For modeling a user's short-term interests, we design a Contextual GRU network to take a user's actions into account, as different types of action, e.g., "click," "collect" and "buy," can help to reflect users' next consumption motivations.

Our experimental results confirm the effectiveness and robustness of DCN-SR with different session lengths and varying numbers of users' historical interactions. DCN-SR outperforms the best performing state-of-the-art model STAMP across different session lengths, especially for short sessions. As to users with different numbers of historical interactions, DCN-SR shows more competitive recommendation performance on all users than the state-of-the-art baseline model STAMP. In addition, the improvements of DCN-SR are higher on users with more historical interactions. To conclude and answer **RQ3**, our proposed DCN-SR model that applies a co-attention network and a CGRU network can help to capture the dynamic interactions between a user's long- and short-term behavior for session-based recommendation.

As to future work, on the one hand, we plan to investigate the use of information contained in different action sequences, e.g., click-click-buy, and click-click-collect, as sequential actions can provide more context information than single actions [10, 37, 129]. On the other hand, we plan to extend the DCN-SR model with more auxiliary information, such as content information, to generate more informative representations of items [44, 121, 151].

Next in Chapter 5, we consider users' multiple intents revealed in their sequential behavior instead of only one main intent, and provide a recommendation list containing accurate as well as diverse items.

5 Intent-aware Diversified Sequential Recommendation

In previous chapters, we have studied how to recommend personalized queries as well as items, and we have mainly focused on improving the recommendation accuracy. In this chapter, we study how to learn users' multiple interests from their sequential behavior and recommend a list containing accurate as well as diverse items. To do so, we propose an intent-aware end-to-end neural approach for diversified sequential recommendation, which answers the following research question asked in Chapter 1: **RQ4**: How can we address the challenge of diversified sequential recommendation in an end-to-end framework?

5.1 Introduction

Conventional recommendation methods, e.g., Collaborative Filtering (CF) based methods [108] or Matrix Factorization (MF) based models [68], assume that user intents are static. They ignore the dynamic and evolving characteristics of user behavior [86]. Sequential Recommendations (SRs) have been introduced to address these characteristics with the aim of predicting the next item(s) by modeling the sequence of a user's previous behavior [99].

Early studies on SRs are mainly based on Markov chains (MCs) [104], which cannot handle long sequences [51, 58]. Recurrent Neural Network (RNN) and Transformer based neural models have attracted a lot of attentions [50, 62]. Over the years, many factors have been considered that influence the performance of SR performance, e.g., personalization [98], repeat consumption [102], context [100], and collaboration [132]. Previous work that focuses on these factors usually aims to improve recommendation accuracy only. However, it has been shown that diversity is also an important metric to consider in recommender systems, as users prefer more diverse lists of recommended items [147].

This is especially true in SR as users may have multiple intents, e.g., different topics or categories of items. For example, as shown in Figure 5.1, although the user shows most interest in cartoon movies from her historic watching behavior, occasionally she

This chapter was published as [27].

Figure 5.1: An example showing sequential recommendations with (bottom) and without (top) diversification.

also watches family and action movies. A better recommendation strategy should provide a diverse list of recommended items so as to satisfy all these intents. Concretely, in the case of Figure 5.1, we would like to recommend a list of cartoons and action as well as family movies simultaneously instead of cartoons only. In addition, user intents are occasionally exploratory which means that they do not have a specific goal in mind. A homogeneous list of recommendations cannot satisfy such users, leading to a boring user experience [111].

Diversification has been well studied in some conventional recommendation scenarios [139] as well as in web search [1, 78, 87]. Current approaches to diversified recommendation mainly focus on how to re-rank the items in a list of recommendations based on a given diversity metric with general recommendation models. Such approaches do not constitute an optimal solution for SRs. First, some assume that user intents are static and they require that user intents are prepared beforehand, which is unrealistic in most SR scenarios [5, 20]. Second, most belong to the post-processing paradigm and achieve recommendation accuracy and diversity in two separate steps, i.e., (1) scoring items and generating a candidate item set with a recommendation model; and (2) selecting a diverse list of recommendations based on both the item scores and some implicit/explicit diversity metrics [70, 139]. Because the recommendation models are not aware of diversity during training and it is hard to design ideal diversity strategies for different recommendation models, their performance is unsatisfactory.

In this chapter, we address the task of SR by taking into account both recommendation accuracy and diversity. Previous methods focusing on accuracy adopt a strategy where items are ranked by a score, which cannot capture the relationship among the recommended items. Instead, we reformulate SR as a list generation task so as to model the relationship among recommended items and propose an end-to-end intent-aware diversified sequential recommendation (IDSR) model. IDSR employs an *implicit intent mining* (IIM) module to automatically capture multiple latent user intents reflected in sequences of user behavior, and an *intent-aware diversity promoting* (IDP) decoder to directly generate accurate and diverse lists of recommendations for the latent user intents. In order to supervise the learning of the implicit intent mining (IIM) module and force the model to take recommendation diversity into account during training, we design an intent-aware diversity promoting (IDP) loss function that evaluates recom-

mendation accuracy and diversity based on the generated lists of recommended items.

More specifically, a sequence encoder is first used to encode user behavior into representations. Then, the IIM module employs multiple attention areas to mine user's multiple intents with each attention area capturing a particular latent user intent. Finally, an intent-aware recommendation decoder is used to generate a recommendation list by selecting one item at a time. When selecting the next item, IDSR also takes the items already selected as input so that it can track to what extent each latent user intent is satisfied. During training, we fuse the IDP loss function to learn to mine and track user intents, and recommend diversified items. In order to supervise the learning of diversity, ideally we have a ground truth diverse list of recommended items. However, in practice, we only have the next one ground truth item, which is not enough to define diversity supervision. To address this, we devise a self-critic strategy for the IDP loss. The idea is that, under the premise that the ground truth item can be recommended correctly, we reward our list generation strategy whenever it generates a more diverse recommendation list than the baseline strategy (i.e., the conventional rank-by-score strategy) evaluated by some diversity metrics. All parameters are learned in an end-to-end back-propagation training paradigm within a unified framework.

We conduct extensive experiments on four benchmark datasets. IDSR outperforms the state-of-the-art baselines on those datasets in terms of both accuracy metrics, i.e., Recall and MRR, and a diversity metric, i.e., intra-list distance (ILD).

Our contributions in this chapter can be summarized as follows:

- We propose an intent-aware diversified sequential recommendation (IDSR) method. To the best of our knowledge, this is the first end-to-end list generation based neural framework that considers diversification for SRs.
- We devise an implicit intent mining (IIM) module to automatically mine latent user intents from user behavior and an intent-aware recommendation decoder to generate diverse lists of recommendations.
- We present an IDP loss function to supervise IDSR in terms of both accuracy and diversity.
- We carry out extensive experiments and analyses on four publicly available benchmark datasets to verify the effectiveness of the proposed IDSR.

5.2 Related Work

We discuss two types of work that is closely related to ours: sequential recommendation and diversified recommendation.

5.2.1 Sequential recommendation

Traditional methods for SRs are often based on Markov chains (MCs) [153]. Previous work introducing such methods investigates how to extract sequential patterns to learn users' next preferences with probabilistic decision-tree models. Following this idea, He and McAuley [42] fuse similarity models with MCs to address the problem

of sparse recommendations. MC-based methods only model local sequential patterns with adjacent interactions, which fails to take the whole sequence into account.

Hidasi et al. [51] introduce an RNN-based model for SRs that consists of Gated Recurrent Units (GRUs) and uses a session-parallel mini-batch training process. Quadrana et al. [98] develop a hierarchical RNN structure that takes users' profiles into account by considering cross-session information. Attention mechanisms have been applied to SRs to help models explore users' preferences [48]. Li et al. [73] propose a neural attentive session-based recommendation machine that takes the last hidden state from the session-based RNN as the sequential behavior, and uses the other hidden states for computing attention to capture users' current preferences in a given session. Xu et al. [141] propose a recurrent convolutional neural network to capture both long-term and short-term dependencies for SR. Kang and McAuley [62] apply a two-layer Transformer model [127] to SRs to capture users' sequential behavior. Sun et al. [117] use a bidirectional encoder representations from Transformers for SRs. Chen et al. [28] propose to apply a user memory network with attention mechanism to store and update a user's historical records for SRs.

Previous studies on SRs, e.g., [7, 18, 23, 143, 144], mostly focus on improving the recommendation accuracy. The studies mentioned above ignore the fact that users might have multiple intents reflected in their sequential behavior. Wang et al. [136] have proposed a mixture-channel purpose routing networks (MCPRNs) to capture users' different intents in a given session. MCPRN first applies a purpose routing network to detect multiple purpose of a user and then models the sequential items with a mixture-channel RNN, where each channel RNN models the item dependencies for a specific purpose. Finally, MCPRN integrates all channel embeddings to predict the next item. During training, MCPRN only applies the cross-entropy loss to supervise the model in terms of recommendation accuracy, which means there is no supervision for the model to learn to distinguish multiple intents or generate diversified recommendations.

Unlike the studies listed above, we propose to address recommendation accuracy and diversification in a unified framework, where we propose an implicit intent mining (IIM) module to capture multiple intents and an intent-aware diversity promoting (IDP) decoder to generate the list of recommended items to satisfy those intents gradually. We devise an IDP loss function to supervise the model to learn different intents and generate diversified recommendations.

5.2.2 Diversified recommendation

Promoting diversity of recommendation or search results has long been an important research topic. A lot of works have been proposed to tackle the task of diversified recommendation, mainly including determinantal point process (DPP) [69] and submodular optimization [97]. The most representative implicit approach is maximal marginal relevance (MMR) [17]. MMR represents relevance and diversity by independent metrics and uses the notion of marginal relevance to combine the two metrics with a trade-off parameter. Qin and Zhu [97] propose an entropy regularizer to promote recommendation diversity. It satisfies monotonicity and submodularity so that the objective function can be maximized approximately by greedy algorithm. Chen et al. [20]

propose to improve recommendation diversification through a DPP [69] with a greedy maximum a posterior inference algorithm. Sha et al. [111] introduce a submodular objective function to combine relevance, coverage of user's intents, and the diversity between items. Learning to rank (LTR) has also been exploited to address diversification. Cheng et al. [30] first label each user by a set of diverse as well as relevant items with a heuristic method and then propose a diversified collaborative filtering algorithm to learn to optimize the performance of accuracy and diversity for recommendation. The main issue of LTR based methods is that they all need diversified ranked lists as ground truth for learning [139]; these are usually unavailable in recommendations.

The methods listed above achieve accuracy and diversity of recommendation in two separate steps, i.e., training an offline recommendation model to score items in terms of accuracy and then re-ranking items by taking diversity into account. We show through experiments that our end-to-end model can achieve significantly better performance. Besides, none of the methods listed is suitable for SRs, where users' sequential behavior needs to be considered. In contrast, we consider users' temporal preferences and optimize for accuracy and diversity in one go.

5.3 Approach

5.3.1 Overview

Given a user u and her/his behavior sequence $S_u = \{x_1, x_2, \ldots, x_T\}$ where every x_i is an item that u interacted with, e.g., watched movie, the goal of SRs is to provide u with a list of recommended items R_L for predicting her/his next interaction; the items are expected to be both relevant and diverse.

Unlike existing SR methods, we assume there are M latent intents behind each behavior sequence, i.e., $A = \{a_1, \ldots, a_M\}$. Then, we seek to generate a list of recommended items R_L by maximizing the degree of satisfaction for all intents:

$$P(R_L \mid u, S_u) = \sum_{m=1}^{M} P(a_m \mid u) P(R_L \mid a_m, u, S_u), \tag{5.1}$$

where $P(a_m \mid u)$ denotes the importance of intent a_m to user u; $P(R_L \mid a_m, u, S_u)$ is the probability of satisfaction of R_L to a_m.

It is hard to directly optimize $P(R_L \mid u, S_u)$ due to the huge search space. Therefore, we propose to generate R_L greedily, i.e., selecting one item at a time with the maximum score $S(v)$:

$$v_t \leftarrow \underset{v \in V \setminus R_{t-1}}{\operatorname{argmax}} \; S(v), \tag{5.2}$$

where v_t is the item to be selected at step t; V is the set of all items; R_{t-1} is the list of recommended items generated until step t-1; $V \setminus R_{t-1}$ guarantees that the selected item is different from previous generated recommendations in R_{t-1} at step t; and $S(v)$ returns the score of item v by

$$S(v) \leftarrow \lambda P(v \mid u, S_u) + (1 - \lambda) \sum_{m=1}^{M} P(v \mid a_m) W(\overline{R_{t-1}}, a_m). \tag{5.3}$$

The score $S(v)$ is a combination of the relevance score and the diversification score, balanced by a hyper-parameter λ; $P(v \mid u, S_u)$ is the relevance score reflecting the importance of v for u; $P(v \mid a_m)$ is the degree of satisfaction of v to a_m; $W(\overline{R_{t-1}}, a_m)$ denotes the likelihood that the already generated recommendation list R_{t-1} does not satisfy a_m.

Then, we propose an end-to-end intent-aware diversified sequential recommendation (IDSR) model to directly generate a diversified list of recommended items according to Eq. (5.3). The main framework of IDSR is shown in Figure 5.2.

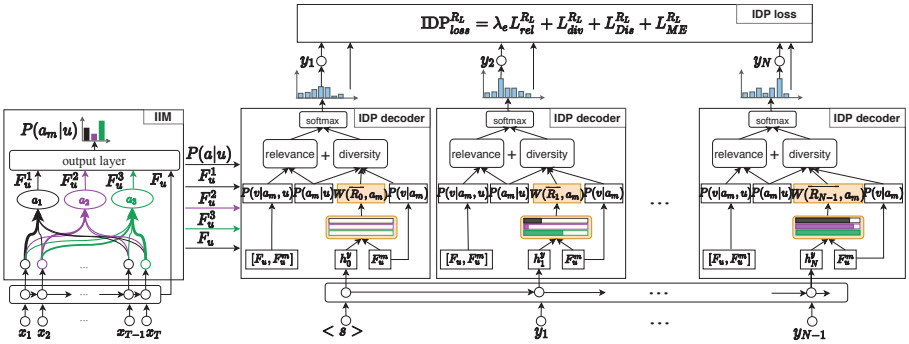


Figure 5.2: Overview of IDSR. The blue, purple and green colors denote different user intents.

As shown in Figure 5.2, IDSR consists of three modules: a *sequence encoder*, an *implicit intent mining* (IIM) *module*, and an *intent-aware diversity promoting* (IDP) *decoder*. First, the sequence encoder projects users' sequential behavior into latent representations. Then, the IIM module is used to capture users' multiple latent intents reflected in their sequential behavior. Finally, the IDP decoder is employed to generate a list of recommended items according to Eq. (5.3). We devise an IDP loss to train IDSR; it evaluates the whole list of recommended items in terms of both accuracy and diversity. Note that there is no re-ranking involved in IDSR. That is, both recommendation accuracy and diversity are jointly learned in an end-to-end way. Next, we introduce the separate modules.

5.3.2 Sequence encoder

Since the encoder module is not the focus of this chapter, we simply adapt the commonly used GRUs to verify the validity of our proposed method [51]:

$$\begin{aligned} z_t &= \sigma\big(\boldsymbol{W_z}[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}]\big) \\ r_t &= \sigma\big(\boldsymbol{W_r}[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}]\big) \\ \hat{\boldsymbol{h}}_t &= \tanh(\boldsymbol{W}_h[\boldsymbol{x}_t, r_t \odot \boldsymbol{h}_{t-1}]) \\ \boldsymbol{h}_t &= (1 - z_t) \odot \boldsymbol{h}_{t-1} + z_t \odot \hat{\boldsymbol{h}}_t, \end{aligned} \tag{5.4}$$

where $\boldsymbol{x}_t$ denotes the embedding of item x_t; $\boldsymbol{W}_z$, $\boldsymbol{W}_r$ and $\boldsymbol{W}_h$ are weight parameters; σ denotes the sigmoid function. The input of the encoder is the behavior sequence

$S_u = \{x_1, x_2, \ldots, x_T\}$ and the outputs are hidden representations $\{\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_T\}$, where $\boldsymbol{h}_i \in \mathbb{R}^{d_e}$. We stack those representations into a matrix $\boldsymbol{H}_S \in \mathbb{R}^{T \times d_e}$. Like [73], we consider the last representation $\boldsymbol{h}_T$ to be the user's global representation, which summarizes the whole sequence:

$$F_u = \boldsymbol{h}_T. \tag{5.5}$$

5.3.3 IIM module

The IIM module is meant to mine users' multiple intents behind the sequence. Intuitively, a user's multiple intents can be reflected by different interactions in their sequential behavior. Some interactions are more representative for a particular intent than others, e.g., the last two actions in Figure 5.1 reflect the user's intent of watching cartoon movies. Motivated by this, we fuse a multi-intent attention mechanism where each attention captures one particular intent. Specifically, IIM first projects $\boldsymbol{H}_S$ and F_u into M spaces w.r.t. the latent intents, respectively. Then, M attention functions are employed in parallel to produce user's intent-specific representations $\{S_u^1, S_u^2, \ldots, S_u^M\}$:

$$S_u^i = \text{Attention}\big(F_u \boldsymbol{W}_i^Q, \boldsymbol{H}_S \boldsymbol{W}_i^K, \boldsymbol{H}_S \boldsymbol{W}_i^V\big), \tag{5.6}$$

where the projection matrices for intent i, i.e., $\boldsymbol{W}_i^Q \in \mathbb{R}^{d_e \times d}$, $\boldsymbol{W}_i^K \in \mathbb{R}^{d_e \times d}$ and $\boldsymbol{W}_i^V \in \mathbb{R}^{d_e \times d}$, are learnable parameters. We use the scaled dot-product attention in this work [127] as:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \boldsymbol{A}\boldsymbol{V} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}}\right)\boldsymbol{V}, \tag{5.7}$$

where $\boldsymbol{A}$ denotes the attention distribution produced by each intent. We finally apply a two-layer feed-forward network to each S_u^i to introduce nonlinearity:

$$F_u^i = \text{FFN}(S_u^i) = \text{ReLU}\big(S_u^i \boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)}\big)\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)}, \tag{5.8}$$

where $\boldsymbol{W}^{(1)} \in \mathbb{R}^{d \times d}$, $\boldsymbol{W}^{(2)} \in \mathbb{R}^{d \times d}$, $\boldsymbol{b}^{(1)} \in \mathbb{R}^d$, and $\boldsymbol{b}^{(2)} \in \mathbb{R}^d$ are trainable parameters.

5.3.4 IDP decoder

The IDP decoder is used to generate R_L based on the intents mined with the IIM module. To begin with, we model the relevance score of v to user u (i.e., $P(v \mid u, S_u)$ in Eq. (5.3)) with a bilinear decoding scheme as follows:

$$\begin{aligned} P(v_n \mid u, S_u) &= \frac{\boldsymbol{S}_{\boldsymbol{v_n}}}{\sum_{j=1}^{|V|} \boldsymbol{S}_{\boldsymbol{v_j}}} \\ S_{v_n} &= \sum_{m=1}^{M} S_{v_n}^m \\ S_{v_n}^m &= P(a_m \mid u) P(v_n \mid a_m, u) \\ P(v_n \mid a_m, u) &= \text{softmax}(\boldsymbol{v_n}^\top \boldsymbol{B}[F_u, F_u^m]), \end{aligned} \tag{5.9}$$

where $\boldsymbol{B}$ is a bilinear parameter; $\boldsymbol{v_n}$ is the item embedding which can be trained within the network; and $S^m_{v_n}$ means the relevance score of item v_n to intent a_m, weighted by the importance of intent a_m, i.e., $P(a_m \mid u)$. We can calculate $P(a_m \mid u)$ by:

$$P(a_m \mid u) = \frac{\exp(F_u \boldsymbol{W}^w {F_u^m}^\top)}{\sum_{j=1}^{M} \exp(F_u \boldsymbol{W}^w {F_u^j}^\top)}, \tag{5.10}$$

where $\boldsymbol{W}^w \in \mathbb{R}^{d_e \times d}$ is used to transform the intent-specific representations back to the same space with F_u, so that we can generate the weight of each intent.

To track the already selected items to date, we use another GRU to encode $R_{t-1} = \{y_1, y_2, \ldots, y_{t-1}\}$ into $\{\boldsymbol{h}^y_1, \boldsymbol{h}^y_2, \ldots, \boldsymbol{h}^y_{t-1}\}$. Then we estimate the degree of "unsatisfactoriness" of R_{t-1} to each intent (i.e., $W(\overline{R_{t-1}}, a_m)$ in Eq. (5.3)) by calculating the matching between $\boldsymbol{h}^y_{t-1}$ and F^m_u as:

$$\begin{aligned} W(\overline{R_{t-1}}, a_m) &= 1 - \frac{P(a_m \mid u) \exp(w^m_{t-1})}{\sum_{j=1}^{M} P(a_j \mid u) \exp(w^j_{t-1})} \\ w^i_{t-1} &= \boldsymbol{W_y}^\top \sigma(\boldsymbol{W_A} F^i_u + \boldsymbol{W_B} \boldsymbol{h}^y_{t-1}), \end{aligned} \tag{5.11}$$

where w^i_{t-1} denotes the matching between already generated recommendations and F^i_u. Thus $W(\overline{R_{t-1}}, a_m)$ indicates to what extent intent a_m is unsatisfied and should be paid more attention to when generating the next recommendation. Here, we also incorporate the initial weight of each intent $P(a \mid u)$. We calculate $P(v \mid a_m)$ in Eq. (5.3) with:

$$P(v_n \mid a_m) = \text{softmax}(\boldsymbol{v_n}^\top F^m_u). \tag{5.12}$$

Finally, we can calculate the score $S(v)$ of each item (Eq. (5.3)), select the item with the highest probability, and append it to the list of recommended items.

5.3.5 IDP loss

Since our goal is to generate a list of recommended items that is both relevant and diverse, we design our loss function to evaluate the whole generated list R_L based on the accuracy as well as the diversity of R_L:

$$\text{Loss}^{R_L} = \lambda_e \mathcal{L}^{R_L}_{rel} + \mathcal{L}^{R_L}_{div}, \tag{5.13}$$

where λ_e is a weight parameter to balance the relative contributions of accuracy and diversification.

Given the output list of recommended items from IDSR, i.e., $R_L = \{y_1, y_2, \ldots, y_N\}$ and the ground truth item y^* (i.e., the next consumed item), $\mathcal{L}^{R_L}_{rel}$ is defined as:

$$\mathcal{L}^{R_L}_{rel} = -\sum_{i=1}^{|V|} p_i \log\big(q^0_i\big), \tag{5.14}$$

where p_i indicates the ground truth probability distribution and q^0_i is the prediction probability of the first item in R_L. When generating the first item, IDSR only considers the relevance score without diversification, thus we use this part to optimize the

prediction accuracy of IDSR. With this relevance loss, we can also take the position of the ground truth item y^* in the ranked list into consideration.

To promote diversity, we apply a self-critic strategy. Specifically, at each step, we select an item based on $S(v)$ and output a list of recommended items R_L. Meanwhile, we also select an item only based on the maximum relevance score $P(v_i \mid u, S_u)$ and output a list of recommended items R_L^{rel}. Thus we propose a pair-wise diversity loss:

$$\begin{aligned} \mathcal{L}_{div}^{R_L} &= \boldsymbol{w} \log \frac{1}{1+\exp(Pr(R_L^{rel}) - Pr(R_L))} \\ Pr(R_L) &= \sum_{v_i \in R_L} \log S(v_i) \\ Pr(R_L^{rel}) &= \sum_{v_i \in R_L^{rel}} \log S(v_i) \\ \boldsymbol{w} &= \boldsymbol{M}(R_L^{rel}) - \boldsymbol{M}(R_L), \end{aligned} \tag{5.15}$$

where $S(v_i)$ is the final score of item v_i calculated by Eq. (5.3); $Pr(R_L)$ indicates the log likelihood of generating recommendation list R_L, so as R_L^{rel}; $\boldsymbol{w}$ is the diversity evaluation metric score gap of the two recommendation list R_L^{rel} and R_L, e.g., ILD in this chapter. We use R_L^{rel} as a baseline to compare with, so that we can evaluate the diversity of the generated list of recommended items R_L. If the diversity of R_L^{rel} is larger than R_L, we would punish the decoder to decrease the probability for generating R_L with the weight of $\boldsymbol{w}$. Otherwise, we would reward the decoder to increase probability of R_L, which is larger than the probability of generating R_L^{rel}.

Besides the relevance and diversity losses, we also add two regularization terms to our loss function. One is a disagreement regularization, which is meant to enlarge the distance among multiple intents. Specifically, the differences among multiple intent representations are reflected by different attention distributions produced by each intent, thus we apply a strategy to disperse the attended positions predicted by each intent. We use an alignment disagreement regularization [74] as:

$$\mathcal{L}_{Dis}^{R_L} = \frac{1}{M^2} \sum_{i=1}^{M} \sum_{j=1}^{M} \left| \boldsymbol{A}^i \odot \boldsymbol{A}^j \right|, \tag{5.16}$$

where $\boldsymbol{A}^i$ denotes the attention distribution produced by intent i in Eq. (5.7). We employ the sum of element-wise multiplication of vector cells.

The other regularization term that we add is the maximum entropy regularization, which helps to avoid the situation that one of the intents dominates [142, 150]:

$$\mathcal{L}_{ME}^{R_L} = \sum_{m=1}^{M} P(a_m \mid u) \log P(a_m \mid u). \tag{5.17}$$

Thus, our final IDP loss is:

$$\text{IDP}_{loss}^{R_L} = \lambda_e \mathcal{L}_{rel}^{R_L} + \mathcal{L}_{div}^{R_L} + \mathcal{L}_{Dis}^{R_L} + \mathcal{L}_{ME}^{R_L}. \tag{5.18}$$

All parameters of IDSR as well as the item embeddings can be learned in an end-to-end back-propagation training paradigm.

Table 5.1: Dataset statistics.

Dataset	ML100K	ML1M	Tafeng	Tmall
Number of users	943	6,022	1,703	25,958
Number of items	1,349	3,043	2,461	57,677
Number of interactions	93,629	959,022	42,921	623,124
Number of item categories	19	18	469	70
Avg. number of genres per item	1.7	1.6	1.0	1.0

5.4 Experiments

We design experiments to answer the following questions, both of which refine **RQ4**:

(RQ4.1) What is the performance of IDSR compared with state-of-the-art baselines in terms of accuracy?

(RQ4.2) Does IDSR outperform state-of-the-art baselines in terms of diversity?

5.4.1 Datasets

We use four public benchmark datasets for our experiments, two of them are based on movies and the others are e-commerce datasets. Table 5.1 lists the statistics of these four datasets:

- **ML100K**[1] is collected from the MovieLens web site. It contains 100,000 ratings from 943 users on 1,682 movies.
- **ML1M**[1] is a larger and sparser version of ML100K, which contains 1,000,209 ratings for movies.
- **Tafeng**[2] is collected from a grocery store and released by Kaggle, which contains one month log data.
- **Tmall**[3] is released by a competition that records user online shopping behavior on an e-commerce platform Tmall.

Note that each item/movie from both ML1M and ML100K belongs to multiple movie genres at the same time. Each item from Tafeng and Tmall only belongs to a single category.

We follow Li et al. [73] to process the data. First, we filter out users who have less than 5 interactions and items that are rated less than 5 times in ML100K. For the other datasets, we only keep users as well as items with more than 20 interactions. Then, we sort the interactions according to the "timestamp" field to get a behavioral sequence for each user. Finally, we prepare each data sample using a sliding-window approach by

[1] https://grouplens.org/datasets/movielens/
[2] https://www.kaggle.com/chiranjivdas09/ta-feng-grocery-dataset
[3] https://tianchi.aliyun.com/dataset/dataDetail?dataId=42

regarding the previous 9 actions as input and the next action as output. We use the first 90% interactions for model training, the last 10% for model testing. The validation set is split from the training set in the same way as the test set.

Since we do not target cold-start items so that we make sure that all items in the test set have been rated by at least one user in the training set and the test set contains the most recent actions which happened later than those in the training and validation sets.

5.4.2 Methods used for comparison

There have been a number of SR methods proposed in the last few years. Our model focus on combining recommendation accuracy and diversity in a unified framework, thus we do not make comparisons with those works aiming to improve recommendation accuracy, e.g., BERT4Rec [117] and SASRec [62], as they can be incorporated into our encoder part to help improve the accuracy performance of our model. There is another work, i.e., S-DIV [65], which proposes a sequential and diverse recommendation model. Since in S-DIV the term "diverse" means to incorporate more rare or tail items which is different from our work, we do not compare with it in this paper. For a fair comparison, we select state-of-the-art neural SR methods that adapt a similar architecture as ours as baselines:

GRU4Rec An RNN-based model for SR. GRU4Rec utilizes session-parallel mini-batches as well as a ranking-based loss function in the training process [51].

NARM An RNN-based model that applies an attention mechanism to capture users' main purposes from the hidden states and combines it with sequential behavior as final representations of users' current preferences [73], which shares a similar spirits as IDSR when calculating the relevance scores for items.

MCPRN The most recently proposed method that models users' multiple purposes in a session. They claim that they can improve the performance over the state-of-the-art methods in terms of both accuracy and diversity [136]. Thus, we consider it as a state-of-the-art baseline model.

We also report results of a popularity based method, **POP**, which ranks items based on the number of interactions, because the performance of **POP** can reflect some characteristics of the datasets and is quite effective in some scenarios [3].

Because there is no previous work specific for diversified SR, we construct a baseline, **NARM+MMR**, ourselves. With carefully tuned hyperparameters, NARM can achieve state-of-the-art performance most of the time. MMR is a simple yet effective approach, which is still commonly used in web search and recommendation. Specifically, we first get the relevance scores $S(v)$ for each item with NARM. Then, we rerank the items using the MMR criteria:

$$v \leftarrow \text{argmax}_{v_i \in R_c \setminus R_L} \theta S(v_i) + (1-\theta) \min_{v_k \in R_L} d_{ki},$$

where R_c is a candidate item set and $\theta \in [0,1]$ is a trade-off parameter to balance the relevance and the minimal dissimilarity d_{ki} between item v_k and item v_i. MMR

first initializes $R_L = \emptyset$ and then iteratively selects the item into R_L, until $|R_L| = N$. When $\theta = 0$, MMR returns diversified recommendations without considering relevance; when $\theta = 1$, it returns the same results as the original baseline models. Unless specified otherwise, for all the results that we presented in this chapter, the number of recommendations (N) equals 10.

5.4.3 Evaluation metrics

For accuracy evaluation, we use Recall and MRR as most previous studies [73, 82]; for diversity evaluation, we choose ILD [147], which is commonly used to evaluate the recommendation diversity.

Recall Whether the test item is contained in the list of recommendations.

MRR Whether the test item is ranked at the top of the list.

ILD Measures the diversity of a list of recommendations as the average distance between pairs of recommended items:

$$\text{ILD} = \frac{2}{|R_L|(|R_L|-1)} \sum_{(i,j)\in R_L} d_{ij}. \tag{5.19}$$

We calculate the dissimilarity d_{ij} between two items based on the Euclidean distance between the item genre vectors [5].

5.4.4 Implementation details

We set the item embedding size and GRU hidden state sizes to 128. We use dropout with drop ratio $p = 0.5$. We initialize the model parameters randomly using the Xavier method [38]. We optimize the model using Adam [66] with the initial learning rate $\alpha = 0.001$, two momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. The mini-batch size is set to 512. We set the parameter $\lambda_e = 1.0$ for the ML100K, ML1M and Tmall datasets and $\lambda_e = 0.1$ for Tafeng after fine-tuning the parameter on the validation set. We test the model performance on the validation sets for every epoch and select the best model to report results on the test sets accordingly. The code used to run our experiments is available online[4].

5.5 Results and Discussion

5.5.1 Performance in terms of accuracy

To answer RQ4.1, we compare IDSR with the baselines in terms of Recall and MRR; see Table 5.2.

First, note that Neural Attentive Recommendation Machine (NARM) has a similar encoding architecture as IDSR, thus we can see that NARM and IDSR are comparable

[4] https://bitbucket.org/WanyuChen/idsr/

Table 5.2: Performance of recommendation models. The results from the best baseline and the best performer in each row are underlined and boldfaced, respectively. Statistical significance of pairwise differences of IDSR vs. the best baseline is determined by a paired t-test ($^{\triangle}$ for p-value $\leq$ .05).

Dataset	Metric	POP	GRU4Rec	NARM	MCPRN	NARM+MMR	IDSR
ML100K	Recall (%)	4.02	6.23	9.68	9.27	9.53	**9.79**
	MRR (%)	1.21	2.09	3.18	2.99	2.77	**3.22**
	ILD	1.501	1.527	1.518	1.561	1.583	**1.666**$^{\triangle}$
ML1M	Recall (%)	9.11	11.67	**15.02**	14.89	14.72	14.89
	MRR (%)	2.02	4.02	**5.39**	5.26	4.89	5.30
	ILD	1.233	1.307	1.289	1.301	1.325	**1.383**$^{\triangle}$
Tafeng	Recall (%)	2.01	4.11	4.71	4.57	4.33	**4.97**$^{\triangle}$
	MRR (%)	1.09	1.42	1.69	1.60	1.41	**1.96**$^{\triangle}$
	ILD	1.233	1.267	1.214	1.248	1.263	**1.318**$^{\triangle}$
Tmall	Recall (%)	9.56	12.11	**14.41**	14.19	14.00	14.32
	MRR (%)	4.11	5.41	**7.51**	7.27	6.28	7.43
	ILD	.8817	.8789	.8343	.8864	.8917	**.9468**$^{\triangle}$

in terms of recommendation accuracy (Recall and MRR). However, IDSR can help to improve the diversity (see Section 5.5.2) of recommendation list without much sacrifice of accuracy, i.e., a 0.87% and 1.70% decrease in terms of Recall and MRR on the ML1M dataset, and of 0.65% and 1.01% on the Tmall dataset, respectively, none of which are significant. That is because although IDSR tries to diversify the recommendations, IDSR still assigns high probability to those most relevant items without considering much of the diversification in the first few decoding steps. In addition, the IDP loss also considers recommendation accuracy, which can help the model to capture users' main intents. When users have multiple intents, NARM shows bias towards the main intent, which will lead to unsatisfactory recommendations. For example, IDSR shows better performance than NARM on the Tafeng dataset. The improvements of IDSR over NARM in terms of Recall and MRR are 5.61% and 16.23% on the Tafeng dataset, respectively. We believe that this is due to the fact that Tafeng records users' behavior in a grocery store, where users tend to have multiple intents, and buy items with different categories when they are shopping. Compared with MCPRN, we can see that IDSR shows better performance in terms of both Recall and MRR on all datasets than MCPRN. The IIM module considers not only users' multiple intents but the importance of each intent, which can help improve the recommendation accuracy.

Second, we note that after re-ranking with MMR, the accuracy of NARM drops dramatically, especially in terms of MRR. This indicates that although post-processing with MMR can improve the diversity of recommendation list, it hurts the accuracy a lot. Because most of the candidate items generated by NARM have similar genres/characteristics. When the diversity scores for the relevant items are lower than the irrelevant ones, the irrelevant items will get higher final scores than the relevant items, which results in a worse performance in terms of accuracy. Besides, we found that the

re-ranking process is time-consuming, while our model is much more efficient.

In summary, IDSR can achieve comparable or superior performance compared with state-of-the-art methods in terms of recommendation accuracy. It is also worth to note that we can incorporate any other effective mechanisms into our framework to further improve the recommendation accuracy such as SASRec [62]. However, this is beyond the scope of this work.

5.5.2 Performance in terms of diversity

To answer RQ4.2, we report the diversity scores, i.e., ILD, on all datasets in Table 5.2. We can see that IDSR consistently outperforms all baselines. The improvements of IDSR over MCPRN are 6.71% and 6.33% in terms of ILD on ML100K and ML1M, respectively. As to the e-commerce datasets, the improvements are 5.58% and 6.81% on the Tafeng and Tmall datasets, respectively. Although MCPRN models users' multiple intents, there is no supervision signal for the model to learn to distinguish different intents in order to generate diverse recommendations. However, in IDSR, we have the diversity loss and disagreement regularization term in our designed IDP loss, which can help the model to learn to distinguish different intents and satisfy each of them during the recommendation list generation process.

Clearly, IDSR significantly outperforms NARM+MMR. For example, the improvements of IDSR over NARM+MMR are 4.34% and 6.18% on Tafeng and Tmall, respectively. Since MMR is heuristically defined, we find that MMR relies heavily on the performance of NARM. When the candidate items from NARM all have similar genres, the performance of MMR method is limited. In contrast, IDSR avoids this issue by learning to diversify the recommendation list through optimizing the IDP loss in Eq. (5.18).

5.6 Analysis

In this section, we perform a number of analyses of the factors that impact the performance of IDSR:

- What is the impact of the number of latent intents on IDSR, i.e, IDSR with single head or multiple heads?
- How does the trade-off parameter λ affect the performance of IDSR?
- What is the effect of the disagreement regularization loss $\mathcal{L}_{Dis}^{R_L}$ in Eq. (5.18)?
- Does the IIM module in IDSR capture users' multiple intents?

5.6.1 Impact of the number of latent intents

We examine the performance of IDSR with different numbers of latent intents/attention heads in Table 5.3. We can see that when the number of heads is set to one, the performance is inferior in terms of diversity on all datasets. The reason is that the model will only focus on the main intent when generating recommendations.

Table 5.3: Performance of IDSR with different numbers of intents.

Dataset	Metric	1-head	2-head	3-head	4-head
ML100K	Recall (%)	9.99	9.83	9.79	9.41
	MRR (%)	3.29	3.19	3.22	2.99
	ILD	1.57	1.62	1.67	1.67
ML1M	Recall (%)	15.26	14.93	14.89	14.01
	MRR (%)	5.55	5.36	5.30	5.02
	ILD	1.29	1.29	1.38	1.40
Tafeng	Recall (%)	5.35	5.16	4.97	4.97
	MRR (%)	1.79	2.02	1.96	1.84
	ILD	1.26	1.26	1.32	1.34
Tmall	Recall (%)	14.51	14.36	14.32	14.21
	MRR (%)	7.49	7.38	7.43	7.23
	ILD	0.82	0.90	0.95	0.95

As for accuracy, we can see that with the number of heads increasing, the performance in terms of MRR and Recall is getting worse in general. On the e-commerce dataset, i.e., Tmall, the differences in terms of Recall and MRR when we change our model from single head to multiple heads are smaller than those on the MovieLens dataset, e.g., ML1M. The improvement of IDSR with 4-heads over 1-head in terms of ILD on Tmall is larger than that on ML1M. This may be because users are more likely to have multiple intents when they do online shopping than when choosing movies to watch next. Another reason is that the time gap between adjacent interactions in the MovieLens datasets is larger than that in the e-commerce datasets, so historical behavior and multiple intents do not have much impact on users' current behavior.

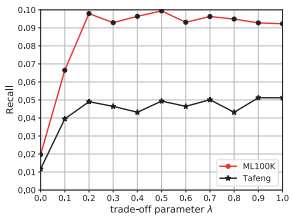
Table 5.3 shows that adding more heads will hurt the accuracy much and also increases the number of parameters for training, thus we choose to use three heads in our experiments which are tuned on the validation set.

5.6.2 Influence of the trade-off parameter λ

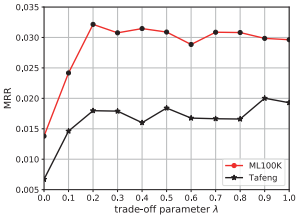
In order to investigate the impact of the trade-off parameter λ on IDSR, we test the performance of IDSR on all datasets by ranging it from 0 to 1 with a step size of 0.1. The results are shown in Figure 5.3.

The accuracy metrics, i.e., Recall and MRR, show upward trends when λ increases from 0 to 1. When $\lambda = 0$, IDSR shows the worst performance. However, a noticeable increase is observed when λ changes from 0 to 0.1: the setting with $\lambda = 0$ means that we only consider diversity without accuracy, thus the model cannot be trained well to recommend relevant items. IDSR shows its best performance in terms of accuracy metrics with λ at around 0.2 and 0.5 on the ML100K and ML1M datasets. Similar trends can be found on e-commerce datasets in terms of MRR and Recall.

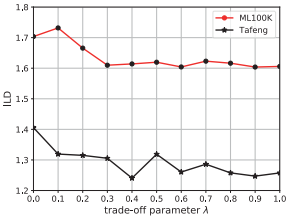
Regarding recommendation diversity, IDSR achieves the best performance in terms of ILD when $\lambda = 0.0$ on all datasets since we maximize diversity only in this case.



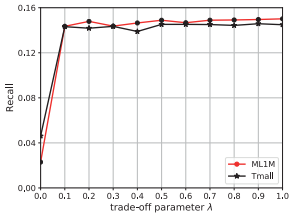
(a) Performance in terms of Recall.



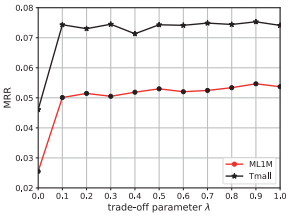
(b) Performance in terms of MRR.



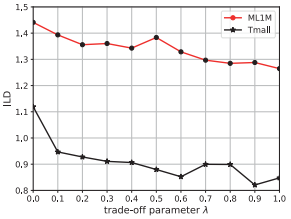
(c) Performance in terms of ILD.



(d) Performance in terms of Recall.



(e) Performance in terms of MRR.



(f) Performance in terms of ILD.

Figure 5.3: Performance of IDSR on four datasets with the parameter λ in Eq. (5.3) changing from 0 to 1.

When λ changes from 0 to 1, ILD naturally decreases on all datasets. On the e-commerce datasets, there are more fluctuations than on the MovieLens datasets, especially on Tmall. The performance of IDSR in terms of ILD decreases sharply from 0 to 0.1.

5.6.3 Effect of disagreement regularization

In order to look into the effect of the disagreement regularization loss $\mathcal{L}_{Dis}^{R_L}$ is IDSR, we modify the IDP loss as:

$$\text{IDP}_{loss}^{R_L} = \lambda_e \mathcal{L}_{rel}^{R_L} + \mathcal{L}_{div}^{R_L} + \lambda_{Dis} \mathcal{L}_{Dis}^{R_L} + \mathcal{L}_{ME}^{R_L}, \tag{5.20}$$

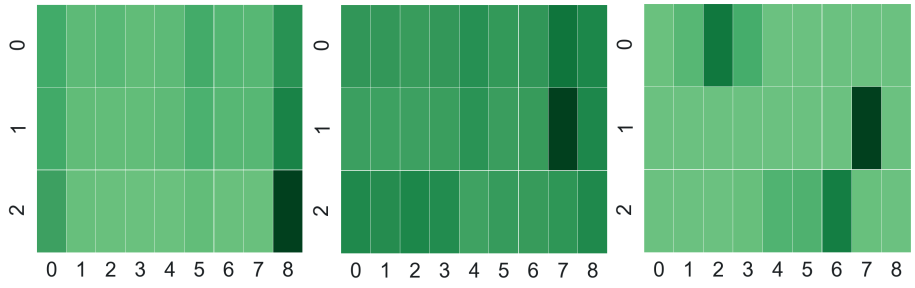
where $\mathcal{L}_{Dis}^{R_L}$ is weighted by the parameter λ_{Dis}. We test the performance of IDSR with $\lambda_{Dis} = 0.0$, 0.5 and 1.0, respectively. The results are shown in Table 5.4.

We can see that $\mathcal{L}_{Dis}^{R_L}$ can help to boost the performance of IDSR in terms of diversity when λ_{Dis} changes from 0.0 to 1.0. This indicates that the IIM module can effectively capture different latent intents by applying $\mathcal{L}_{Dis}^{R_L}$. To further show the effect of the IIM module with different weights of $\mathcal{L}_{Dis}^{R_L}$, we randomly select one sequence from the test set of ML100K and visualize the attention weights of different positions with multiple intents when $\lambda_{Dis} = 0.0$, 0.5 and 1.0 in Figure 5.4.

From Figure 5.4, it is obvious when $\lambda_{Dis} = 0.0$, the three intents share similar attention weights distributions, which fails to extract this user's different intents and thus leads to worse performance in terms of diversity than that when $\lambda_{Dis} = 0.5$. As

Table 5.4: Performance of IDSR with different weights of disagreement regularization.

Dataset	Metric	$\lambda_{Dis} = 0.0$	$\lambda_{Dis} = 0.5$	$\lambda_{Dis} = 1.0$
ML100K	Recall (%)	9.95	9.78	9.79
	MRR (%)	3.23	3.13	3.21
	ILD	1.58	1.61	1.67
ML1M	Recall (%)	15.14	14.97	14.89
	MRR (%)	5.48	5.32	5.30
	ILD	1.29	1.30	1.38
Tafeng	Recall (%)	5.71	5.19	4.97
	MRR (%)	2.15	1.93	1.96
	ILD	1.24	1.28	1.32
Tmall	Recall (%)	14.57	14.43	14.32
	MRR (%)	7.50	7.45	7.43
	ILD	0.84	0.91	0.95



(a) $\lambda_{Dis} = 0.0$. **(b)** $\lambda_{Dis} = 0.5$ **(c)** $\lambda_{Dis} = 1.0$

Figure 5.4: Weight distributions of multiple intents with different values of λ_{Dis}.

λ_{Dis} changes from 0.0 to 1.0, the differences between the three intents become more distinct. To sum up, the IIM module can effectively capture different latent intents with a disagreement regularization loss, as indicated by various weights for items in a sequence.

5.6.4 Case study

In this subsection, we show an example from the test set of ML100K to illustrate the different recommendation results by IDSR and NARM in Figure 5.5.

Figure 5.5 (top) shows 7 movies that the user watched recently and the top 5 recommendations generated by IDSR and NARM, respectively. The ground truth item is marked with a red box. According to the user's historical views, we see that the user likes Children and Comedy recently. But the user also shows interest in Adventure, Animation, Action, Crime, Drama, Romance and Thriller. The list of items recommended by NARM is mainly about the Children genre, e.g., cartoon movies, which

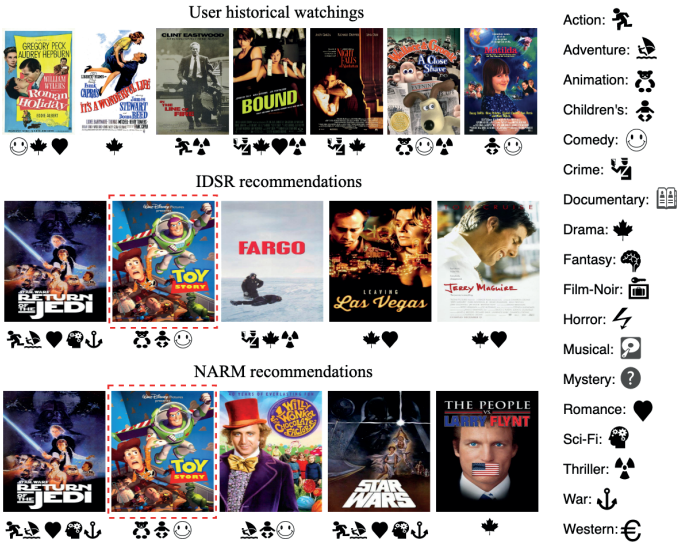


Figure 5.5: An example of recommendation results generated by IDSR and NARM.

is close to the recent intents of this user. Differently, IDSR accommodates multiple intents and diversifies the list of recommended movies with Drama, Crime, Romance and Thriller. IDSR also recognizes the most important intent and gives a high rank to the ground truth movie. This confirms that IDSR cannot only mine users' multiple intents, but generate a diversified list of recommended items to cover those intents.

5.7 Conclusion

In this chapter, we propose the IDSR model to improve diversification for sequential recommendation (SR). We devise an implicit intent mining (IIM) module to capture users' multiple intents and an intent-aware diversity promoting (IDP) decoder to generate a diverse recommendation list covering those intents. We also design an IDP loss to supervise the model to simultaneously consider accuracy and diversification during training. We have conducted experiments on four datasets and have found that IDSR significantly outperforms the state-of-the-art baselines in terms of recommendation diversity while maintaining competitive accuracy scores. In addition, we discuss the impact of the trade-off parameter and the number of intents as well as the disagreement regularization in our model, and include a case study to compare the items recommended by IDSR vs. those recommended by the baseline model. In summary, we answer **RQ4** by reformulating SR as a list generation task and proposing IDSR to capture users' multiple intents as well as recommend diversified items.

As to future work, we plan to apply IDSR to other recommendation scenarios, e.g., shared-account recommendations, where the observed behavior may be generated

by multiple users with more distinct intents [59, 86]. We also hope to improve the recommendation accuracy by incorporating other useful SR models into IDSR [119, 141]. In IDSR, there is a trade-off parameter controlling the balance between accuracy and diversity, i.e., λ, which needs to be pre-defined. This is a one-size-fits-all method that provides recommendations to all users with a constant accuracy-diversity balance. However, individuals have different needs for diversity, thus it is important to provide recommendations with an adaptive degree of diversity [34, 145]. We aim to investigate how to learn the trade-off parameter from users' behavior so as to address this need.

6 Conclusions

In previous chapters, we have described how we addressed the research questions raised in Chapter 1 as well as the answers that we have obtained. In this chapter, we first look back to our research questions and summarize the main findings and implications of our work in Section 6.1. We then describe some future research directions that follow the work in this thesis in Section 6.2.

6.1 Main Findings

6.1.1 Learning users' search intent and recommending personalized queries

We first considerd the recommendation task in search engines, i.e., recommending queries, and raised the following question:

(RQ1) How to capture users' search intent by learning from their historical submitted queries?

To answer this question, we have proposed an attention-based hierarchical neural query suggestion model (Attention-based Hierarchical Neural Query Suggestion (AHNQS)) that combines a hierarchical user-session Recurrent Neural Network (RNN) with an attention mechanism. The hierarchical structure, which incorporates a session-level and a user-level RNN, can model both the user's short-term and long-term search behavior effectively. The attention mechanism aims to capture a user's preference towards certain queries over others. For the session-level RNN, a combined session state is applied to capture both of the user's sequential behavior and his main purpose in the current session, which is then used as the input for the user-level RNN. For the user-level RNN, we used its final hidden state to initialize the next session-level RNN, which can automatically transport the user information within the network.

We evaluated the effectiveness of our proposed model by extensive experiments. The experimental results show that: (1) the proposed AHNQS model helps to boost query suggestion performance in terms of MRR and Recall across sessions with various lengths; (2) using the combined session state in the AHNQS model achieves better performance than only using the local session state; and (3) the AHNQS model yields better performance than the best baseline for inactive, active, as well has highly active

users. The theoretical implication of this research is that a hierarchical model that is combined with an attention mechanism for query suggestion can capture the dynamic search intents of a user. The practical implication of our research is that the improvements of AHNQS over the best baseline model are significant; they are especially prominent for short sessions and for inactive users with few search sessions, which is a realistic setup that online services are always confronted with [101]. Compared to the state-of-the-art, AHNQS achieves improvements of 9.66% and 12.51% in terms of Recall@10 and MRR@10, respectively, on average over all users, and of 10.22% and 13.01% for inactive users.

6.1.2 Learning users' general preferences and recommending personalized items

Next, we focused on recommender system and investigated how to learn users' general preferences and asked:

RQ2 Can we learn users general preference by modeling non-linear user-item relationships as well as characteristics based on their interactions?

We have proposed a joint neural collaborative filtering model, Joint Neural Collaborative Filtering (J-NCF), to answer this research question. J-NCF uses a unified deep neural network to tightly couple two important parts in a recommender system, i.e., deep feature learning of users and items, and deep modeling of user-item interactions. For the user and item feature extraction, we used a deep neural network with matrix factorization and a combination of explicit and implicit feedback as inputs. Then we adopted another neural network for modeling user-item interactions using the feature vectors as inputs. Thus, J-NCF enables the two parts to be optimized with each other through a joint training process. In order to make J-NCF fit the top-N recommendation task, we designed a new loss function that incorporates information from both pair-wise and point-wise loss.

The experimental results confirm the effectiveness of J-NCF. In addition, we have also experimentally investigated the performance of J-NCF under various settings, e.g., with different loss functions, with varying numbers of layers in the networks, and with different types of feedback as inputs. The results confirm the effectiveness of our hybrid loss function and demonstrate that J-NCF performs better with more layers in the networks and using the combination of implicit and explicit feedback as inputs. In addition, we have investigated the robustness of J-NCF with different degrees of data sparsity and different numbers of user ratings. J-NCF outperforms the best baseline model Deep Matrix Factorization (DMF) for users across all activity levels, especially for "inactive users" who constitute the majority of users in the datasets. As for datasets with different levels of sparsity, in general, J-NCF shows a more competitive recommendation performance on all datasets than the state-of-the-art baseline model DMF. Moreover, we have also tested J-NCF model with a large and sparse dataset, i.e., AEle, and the results show that J-NCF also outperforms state-of-the-art baseline models on the dataset.

6.1.3 Learning users' dynamic preferences for sequential recommendation

For recommender systems, we further investigated another task, namely learning users' dynamic preferences for sequential recommendation and asked:

RQ3 How can we incorporate users' long- and short-term interaction behavior for session-based recommendation?

To address this research question, we proposed a dynamic co-attention network for session-based recommendation, Dynamic Co-attention Network for Session-based Recommendation (DCN-SR). DCN-SR applies a co-attention network to capture the dynamic relations between a user's long-term and short-term interactions and generate co-dependent representations of the user's long-term and short-term preferences. It not only exploits the combination of long-term and short-term knowledge, but also considers dynamic aspects of the relations between a user's long-term and short-term preferences. For modeling a user's short-term interests, we designed a Contextual GRU network to take a user's actions into account, as different types of actions, e.g., "click," "collect" and "buy," can help to reflect the user's next consumption motivation.

Our experimental results confirm the effectiveness and robustness of DCN-SR with different session lengths and varying numbers of users' historical interactions. DCN-SR outperforms the best performing state-of-the-art model Short-Term Attention/Memory Priority Model (STAMP) across different session lengths, especially for short sessions. As to users with different numbers of historical interactions, DCN-SR shows more competitive recommendation performance on all users than the state-of-the-art baseline model STAMP. In addition, the improvements of DCN-SR are higher on users with more historical interactions.

6.1.4 Learning users' multiple intents for diversified sequential recommendation

Finally, we took a step towards diversified sequential recommendation and answered the following research question:

RQ4 How can we address the challenge of diversified sequential recommendation in an end-to-end framework?

We proposed an intent-aware diversified sequential recommendation (IDSR) model to improve diversification for sequential recommendation (SR). We devised an implicit intent mining (IIM) module to capture users' multiple intents and an intent-aware diversity promoting (IDP) decoder to generate a diverse recommendation list covering those intents. We also designed an intent-aware diversity promoting (IDP) loss to supervise the model to simultaneously consider accuracy and diversification during training. We have conducted experiments on four datasets and have found that IDSR significantly outperforms the state-of-the-art baselines in terms of recommendation diversity while maintaining competitive accuracy scores. In addition, we discussed the impact of the trade-off parameter and the number of intents as well as the disagreement regularization in our model. We also included a case study to compare the items recommended by IDSR vs. those recommended by the baseline model.

6.2 Future Work

In this section, we list several possible directions for future work according to our main research questions:

6.2.1 Incorporating semantic information

We have answered **RQ1** by proposing an Attention-based Hierarchical Neural Query Suggestion (AHNQS) model. This model mainly considers the sequential relations among queries while ignoring semantic similarity within the hierarchical structure, as we only use the one-hot embeddings when encoding the input queries. Semantic information can help to improve the model performance as well as ensure the generalization of AHNQS. In future, we can optimize AHNQS by combining semantic similarity within the hierarchical structure [89]. One possible way is to use different encoding methods for input queries [63, 71, 72], e.g., concatenating with word embeddings. Besides, we can also leverage topic modeling [130] to incorporate semantic analysis for sequential queries.

6.2.2 Incorporating content and context information

As collaborative filtering usually suffers from limited performance due to the sparsity of user-item interactions [113], auxiliary information could be used to boost the performance. In our proposed model J-NCF, we only use the id information of items as well as users, which can be extended with more auxiliary information [11, 12, 134, 151].

On the one hand, content information of items and reviews can help to get a more informed expression of users as well as items. It would also be interesting to explore heterogeneous information in a knowledge base to improve the quality of recommender systems with deep learning [146].

On the other hand, we also plan to explore context information to provide some background/situation information when doing recommendation. The difference between user/item content and context is that content information, such as attributes, is only attached to either an item or user, whereas context is attached to the interaction event itself, e.g., the time when a user buys an item. A direct way might be using pre-filtering or post filtering to filter out irrelevant items according to explicit context information. Besides this, we can also use latent variable models to automatically characterize implicit context as well as their relationships to items or users. In addition, contextual cues can also be derived from interaction data and used to predict user preferences.

6.2.3 Heterogeneous behavior modeling

We answered **RQ3** by proposing a Dynamic Co-attention Network for Session-based Recommendation (DCN-SR), which models sequential items accompanied with user actions towards items. In real scenarios, there is an abundance of implicit feedback such as “click” and “add to cart” actions, especially in e-commerce platforms. Although these actions are heterogeneous in terms of both representation and data distri-

butions, they can be aligned on sequential behaviors. For example, an action sequence could be click-add to cart-buy-review. As to future work, we plan to investigate the use of information contained in different action sequences, e.g., click-click-buy, and click-click-collect, as sequential actions can provide more context information than single actions [10, 37, 129].

6.2.4 Adaptive diversification for sequential recommendation

In intent-aware diversified sequential recommendation (IDSR) model, there is a trade-off parameter controlling the balance between accuracy and diversity, i.e., λ, which needs to be pre-defined. This is a one-size-fits-all method that provides recommendations to all users with a constant accuracy-diversity balance. However, different individuals have different needs for diversity, especially in sequential recommendation. For example, some users may have focused interests in the current session, thus providing a highly diversified recommendation list may hurt the accuracy and dissatisfy the user. Meanwhile, there are also users who do not have clear preferences in mind at the beginning of the session, and providing diversified recommendations can help such users to do more explorations and avoid a boring user experience. Hence, it is important to provide recommendations with an adaptive degree of diversity [34, 145], which means to learn a personalized trade-off parameter when doing sequential recommendation. One possible way is to understand a user's current diversity needs from his or her past sequential behavior, since if a user always clicks items with the same genre in the current session, she may not prefer a diversified recommendation list.

Bibliography

[1] A. Abid, N. Hussain, K. Abid, F. Ahmad, M. S. Farooq, U. Farooq, S. A. Khan, Y. D. Khan, M. A. Naeem, and N. Sabir. A survey on search results diversification techniques. *Neural Computing and Applications*, 27(5):1207–1229, 2016. (Cited on page 86.)

[2] D. A. Adeniyi, Z. Wei, and Y. Yang. Automated web usage data mining and recommendation system using k-nearest neighbor (knn) classification method. *Applied Computing and Informatics*, 12(1): 90–108, 2016. (Cited on page 36.)

[3] G. Adomavicius and A. 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. (Cited on pages 4, 33, 35, 45, 46, 76, and 95.)

[4] W. U. Ahmad, K.-W. Chang, and H. Wang. Context attentive document ranking and query suggestion. In *SIGIR '19*, pages 385–394, 2019. (Cited on page 3.)

[5] A. Ashkan, B. Kveton, S. Berkovsky, and Z. Wen. Optimal greedy diversity for recommendation. In *IJCAI '15*, pages 1742–1748, 2015. (Cited on pages 2, 86, and 96.)

[6] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR'15*, pages 1–15, 2015. (Cited on page 12.)

[7] B. Basiliyos, Tilahun, O. Charles, Awono, and B. Bernabe. Deep learning methods on recommender system: A survey of state-of-the-art. *International Journal of Computer Applications*, 162(10):17–22, 2017. (Cited on pages 37 and 88.)

[8] A. Bellogin, P. Castells, and I. Cantador. Precision-oriented evaluation of recommender systems: An algorithmic comparison. In *RecSys '11*, pages 333–336, 2011. (Cited on page 48.)

[9] A. Borisov, I. Markov, M. de Rijke, and P. Serdyukov. A neural click model for web search. In *WWW '16*, pages 531–541, 2016. (Cited on page 14.)

[10] A. Borisov, M. Wardenaar, I. Markov, and M. de Rijke. A click sequence model for web search. In *SIGIR '18*, pages 45–54, 2018. (Cited on pages 13, 84, and 109.)

[11] F. Cai and M. de Rijke. Learning from homologous queries and semantically related terms for query auto completion. *Information Processing and Management*, 52(4):628 – 643, 2016. (Cited on pages 3, 13, 64, and 108.)

[12] F. Cai and M. de Rijke. A survey of query auto completion in information retrieval. *Foundations and Trends in Information Retrieval*, 10(4):273–363, 2016. (Cited on pages 3, 13, 64, and 108.)

[13] F. Cai and M. de Rijke. Selectively personalizing query auto-completion. In *SIGIR '16*, pages 993–996, 2016. (Cited on pages 1 and 13.)

[14] F. Cai, S. Liang, and M. de Rijke. Prefix-adaptive and time-sensitive personalized query auto completion. *IEEE Transactions on Knowledge and Data Engineering*, 28(9):2452–2466, 2016. (Cited on page 64.)

[15] F. Cai, R. Reinanda, and M. de Rijke. Diversifying query auto-completion. *ACM Transactions on Information Systems*, 34(4):25:1–25:33, 2016. (Cited on pages 3, 13, and 64.)

[16] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *KDD '08*, pages 875–883, 2008. (Cited on pages 1 and 13.)

[17] J. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR '98*, pages 335–336, 1998. (Cited on page 88.)

[18] S. P. Chatzis, P. Christodoulou, and A. S. Andreou. Recurrent latent variable networks for session-based recommendation. In *DLRS '17*, pages 38–45, 2017. (Cited on pages 64 and 88.)

[19] J. Chen, H. Zhang, X. He, L. Nie, W. Liu, and T.-S. Chua. Attentive collaborative filtering: Multimedia recommendation with item- and component-level attention. In *SIGIR '17*, pages 335–344, 2017. (Cited on page 64.)

[20] L. Chen, G. Zhang, and H. Zhou. Fast greedy map inference for determinantal point process to improve recommendation diversity. In *NIPS '18*, pages 5627–5638, 2018. (Cited on pages 2, 86, and 88.)

[21] W. Chen, F. Cai, H. Chen, and M. de Rijke. Personalized query suggestion diversification. In *SIGIR '17*, pages 817–820, 2017. (Cited on pages 1, 3, 9, 11, and 13.)

[22] W. Chen, F. Cai, H. Chen, and M. de Rijke. Attention-based hierarchical neural query suggestion. In *SIGIR '18*, pages 1093–1096, 2018. (Cited on page 9.)

[23] W. Chen, F. Cai, H. Chen, and M. de Rijke. A dynamic co-attention network for session-based recommendation. In *CIKM '19*, pages 1461–1470, 2019. (Cited on pages 9, 65, and 88.)

[24] W. Chen, F. Cai, H. Chen, and M. de Rijke. Hierarchical neural query suggestion with an attention mechanism. *Information Processing and Management*, page 102040, 2019. (Cited on pages 9 and 11.)

[25] W. Chen, F. Cai, H. Chen, and M. de Rijke. Joint neural collaborative filtering for recommender systems. *ACM Transactions on Information Systems*, 37(4), 2019. (Cited on pages 9 and 33.)
[26] W. Chen, F. Cai, H. Chen, and M. de Rijke. Personalized query suggestion diversification in information retrieval. *Frontiers of Computer Science*, 14(3):143602, 2020. (Cited on page 9.)
[27] W. Chen, P. Ren, F. Cai, F. Sun, and M. de Rijke. Improving end-to-end sequential recommendations with intent-aware diversification. In *CIKM '20*, page 175–184, 2020. (Cited on pages 9 and 85.)
[28] X. Chen, H. Xu, Y. Zhang, J. Tang, Y. Cao, Z. Qin, and H. Zha. Sequential recommendation with user memory networks. In *WSDM '18*, pages 108–116, 2018. (Cited on pages 68 and 88.)
[29] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah. Wide & deep learning for recommender systems. In *DLRS '2016*, pages 7–10, 2016. (Cited on page 38.)
[30] P. Cheng, S. Wang, J. Ma, J. Sun, and H. Xiong. Learning to recommend accurate and diverse items. In *WWW '17*, pages 183–192, 2017. (Cited on page 89.)
[31] K. Cho, B. van Merrienboer, Ç. Gülçehre, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP '14*, pages 1724–1734, 2014. (Cited on page 15.)
[32] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *RecSys '10*, pages 39–46, 2010. (Cited on pages 41 and 49.)
[33] J. Cui, H. Liu, J. Yan, L. Ji, R. Jin, J. He, Y. Gu, Z. Chen, and X. Du. Multi-view random walk framework for search task discovery from click-through log. In *CIKM '11*, pages 135–140, 2011. (Cited on page 13.)
[34] T. Di Noia, J. Rosati, P. Tomeo, and E. Di Sciascio. Adaptive multi-attribute diversity for recommender systems. *Information Sciences*, 382-383:234–253, 2017. (Cited on pages 103 and 109.)
[35] S. Feng, X. Li, Y. Zeng, G. Cong, Y. M. Chee, and Q. Yuan. Personalized ranking metric embedding for next new poi recommendation. In *IJCAI '15*, pages 2069–2075, 2015. (Cited on page 67.)
[36] N. Fiorini and Z. Lu. Personalized neural language models for real-world query auto completion. In *NAACL '18*, pages 208–215, 2018. (Cited on page 14.)
[37] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional sequence to sequence learning. *CoRR*, abs/1705.03122, 2017. (Cited on pages 84 and 109.)
[38] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AI&Statistics '10*, pages 249–256, 2010. (Cited on page 96.)
[39] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He. Deepfm: A factorization-machine based neural network for ctr prediction. In *IJCAI '17*, pages 1725–1731, 2017. (Cited on page 38.)
[40] J. Guo, X. Cheng, G. Xu, and X. Zhu. Intent-aware query similarity. In *CIKM '11*, pages 259–268, 2011. (Cited on page 21.)
[41] Q. He, D. Jiang, Z. Liao, S. C. H. Hoi, K. Chang, E. Lim, and H. Li. Web query recommendation via sequential query prediction. In *ICDE '09*, pages 1443–1454, 2009. (Cited on pages 1 and 13.)
[42] R. He and J. McAuley. Fusing similarity models with markov chains for sparse sequential recommendation. In *ICDM '17*, pages 191–200, 2016. (Cited on pages 65, 67, and 87.)
[43] X. He and T.-S. Chua. Neural factorization machines for sparse predictive analytics. In *SIGIR '17*, pages 355–364, 2017. (Cited on page 37.)
[44] X. He, T. Chen, M.-Y. Kan, and X. Chen. Trirank: Review-aware explainable recommendation by modeling aspects. In *CIKM '15*, pages 1661–1670, 2015. (Cited on page 84.)
[45] X. He, H. Zhang, M.-Y. Kan, and T.-S. Chua. Fast matrix factorization for online recommendation with implicit feedback. In *SIGIR '16*, pages 549–558, 2016. (Cited on pages 42, 48, and 53.)
[46] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural collaborative filtering. In *WWW '17*, pages 173–182, 2017. (Cited on pages 34, 37, 40, 41, 42, 43, 45, 46, 47, 48, 49, 51, 71, and 77.)
[47] X. He, Z. He, X. Du, and T.-S. Chua. Adversarial personalized ranking for recommendation. In *SIGIR '18*, pages 355–364, 2018. (Cited on pages 2, 4, and 65.)
[48] X. He, Z. He, J. Song, Z. Liu, Y. Jiang, and T. Chua. Nais: Neural attentive item similarity model for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2354–2366, 2018. (Cited on pages 68 and 88.)
[49] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, 2004. (Cited on page 49.)
[50] B. Hidasi and A. Karatzoglou. Recurrent neural networks with top-k gains for session-based recommendations. In *CIKM '18*, pages 843–852, 2018. (Cited on pages 42, 43, 48, 51, and 85.)
[51] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. Session-based recommendations with recurrent

neural networks. In *ICLR '16*, pages 1–10, 2016. (Cited on pages 16, 21, 23, 37, 67, 76, 85, 88, 90, and 95.)

[52] B. Hidasi, M. Quadrana, A. Karatzoglou, and D. Tikk. Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *RecSys '16*, pages 241–248, 2016. (Cited on page 64.)

[53] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. (Cited on page 17.)

[54] K. Hofmann, A. Schuth, S. Whiteson, and M. de Rijke. Reusing historical interaction data for faster online learning to rank for information retrieval. In *WSDM '13*, pages 183–192, 2013. (Cited on page 67.)

[55] X. Hong-Jian, D. Xinyu, Z. Jianbing, H. Shujian, and C. Jiajun. Deep matrix factorization models for recommender systems. In *IJCAI '17*, pages 3203–3209, 2017. (Cited on pages 34, 38, 40, 41, 42, 43, 46, 47, 48, 49, 55, 71, 76, and 77.)

[56] C.-K. Huang, L.-F. Chien, and Y.-J. Oyang. Relevant term suggestion in interactive web search based on contextual information in query session logs. *Journal of the American Society for Information Science and Technology*, 54(7):638–649, 2003. (Cited on pages 20 and 21.)

[57] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In *CIKM '13*, pages 2333–2338, 2013. (Cited on pages 14 and 37.)

[58] D. Jannach. Keynote: Session-based recommendation – challenges and recent advances. In *KI 2018: Advances in Artificial Intelligence*, pages 3–7, 2018. (Cited on pages 66 and 85.)

[59] J. Jiang, C. Li, Y. Chen, and W. Wang. Identifying users behind shared accounts in online streaming services. In *SIGIR '18*, pages 65–74, 2018. (Cited on page 103.)

[60] J.-Y. Jiang, Y.-Y. Ke, P.-Y. Chien, and P.-J. Cheng. Learning user reformulation behavior for query auto-completion. In *SIGIR '14*, pages 445–454, 2014. (Cited on pages 1 and 13.)

[61] S. Kabbur, X. Ning, and G. Karypis. Fism: Factored item similarity models for top-n recommender systems. In *KDD '13*, pages 659–667, 2013. (Cited on pages 7, 35, 42, and 48.)

[62] W. Kang and J. J. McAuley. Self-attentive sequential recommendation. In *ICDM '18*, pages 197–206, 2018. (Cited on pages 85, 88, 95, and 98.)

[63] P. Karisani, M. Rahgozar, and F. Oroumchian. A query term re-weighting approach using document similarity. *Information Processing and Management*, 52(3):478 – 489, 2016. (Cited on pages 31 and 108.)

[64] D. Kim, C. Park, J. Oh, S. Lee, and H. Yu. Convolutional matrix factorization for document context-aware recommendation. In *RecSys '16*, pages 233–240, 2016. (Cited on pages 33 and 37.)

[65] Y. Kim, K. Kim, C. Park, and H. Yu. Sequential and diverse recommendation with long tail. In *IJCAI '19*, pages 2740–2746, 2019. (Cited on page 95.)

[66] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR '15*, pages 1–13, 2015. (Cited on pages 49, 77, and 96.)

[67] Y. Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *KDD '08*, pages 426–434, 2008. (Cited on pages 4, 36, and 65.)

[68] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. (Cited on pages 4, 33, 34, 35, 36, and 85.)

[69] A. Kulesza and B. Taskar. *Determinantal Point Processes for Machine Learning*. Now Publishers Inc., Hanover, MA, USA, 2012. (Cited on pages 88 and 89.)

[70] M. Kunaver and T. Porl. Diversity in recommender systems a survey. *Knowledge-Based Systems*, 123 (C):154–162, 2017. (Cited on page 86.)

[71] C. Li, J. Xing, A. Sun, and Z. ma. Effective document labeling with very few seed words: A topic model approach. In *CIKM '16*, pages 85–94, 2016. (Cited on pages 31 and 108.)

[72] C. Li, Y. Duan, H. Wang, Z. Zhang, A. Sun, and Z. Ma. Enhancing topic modeling for short texts with auxiliary word embeddings. *ACM Transactions on Information Systems*, 36(2):11:1–11:30, 2017. (Cited on pages 31 and 108.)

[73] J. Li, P. Ren, Z. Chen, Z. Ren, T. Lian, and J. Ma. Neural attentive session-based recommendation. In *CIKM '17*, pages 1419–1428, 2017. (Cited on pages 68, 72, 77, 88, 91, 94, 95, and 96.)

[74] J. Li, Z. Tu, B. Yang, M. R. Lyu, and T. Zhang. Multi-head attention with disagreement regularization. In *EMNLP '18*, pages 2897—-2903, 2018. (Cited on page 93.)

[75] L. Li, Z. Yang, L. Liu, and M. Kitsuregawa. Query-url bipartite based approach to personalized query recommendation. In *AAAI '08*, pages 1189–1194, 2008. (Cited on page 13.)

[76] S. Li, J. Kawale, and Y. Fu. Deep collaborative filtering via marginalized denoising auto-encoder. In

CIKM '15, pages 811–820, 2015. (Cited on pages 2 and 37.)

[77] J. Lian, F. Zhang, X. Xie, and G. Sun. Cccfnet: A content-boosted collaborative filtering neural network for cross domain recommender systems. In *WWW '17*, pages 817–818, 2017. (Cited on page 38.)

[78] S. Liang, E. Yilmaz, H. Shen, M. de Rijke, and W. B. Croft. Search result diversification in short text streams. *ACM Transactions on Information Systems*, 36(1):8, 2017. (Cited on page 86.)

[79] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003. (Cited on page 35.)

[80] J. Liu and C. Wu. Deep learning based recommendation: A survey. In *ICISA '17*, pages 451–458, 2017. (Cited on page 37.)

[81] Q. Liu, S. Wu, and L. Wang. Multi-behavioral sequential prediction with recurrent log-bilinear model. *IEEE Transactions on Knowledge and Data Engineering*, 29(6):1254–1267, 2017. (Cited on page 77.)

[82] Q. Liu, Y. Zeng, R. Mokhosi, and H. Zhang. Stamp: Short-term attention/memory priority model for session-based recommendation. In *KDD '18*, pages 1831–1839, 2018. (Cited on pages 68, 72, 76, 77, and 96.)

[83] X. Liu, Y. Ouyang, W. Rong, and Z. Xiong. Item category aware conditional restricted boltzmann machine based recommendation. In *ICONIP '15*, pages 609–616, 2015. (Cited on pages 37 and 51.)

[84] C. Ma, Y. Zhang, Q. Wang, and X. Liu. Point-of-interest recommendation: Exploiting self-attentive autoencoders with neighbor-aware influence. In *CIKM '18*, pages 697–706, 2018. (Cited on page 68.)

[85] H. Ma, H. Yang, I. King, and M. R. Lyu. Learning latent semantic relations from clickthrough data for query suggestion. In *CIKM '08*, pages 709–718, 2008. (Cited on page 13.)

[86] M. Ma, P. Ren, Y. Lin, Z. Chen, J. Ma, and M. de Rijke. π-net: A parallel information-sharing network for cross-domain shared-account sequential recommendations. In *SIGIR '19*, pages 685–694, 2019. (Cited on pages 85 and 103.)

[87] R. Mccreadie, R. L. T. Santos, C. Macdonald, and I. Ounis. Explicit diversification of event aspects for temporal summarization. *ACM Transactions on Information Systems*, 36(3):25, 2018. (Cited on page 86.)

[88] Q. Mei, D. Zhou, and K. Church. Query suggestion using hitting time. In *CIKM '08*, pages 469–478, 2008. (Cited on page 13.)

[89] E. Meij, M. Bron, L. Hollink, B. Huurnink, and M. de Rijke. Mapping queries to the linking open data cloud: A case study using dbpedia. *Journal of Web Semantics*, 9(4):418–433, 2011. (Cited on pages 31 and 108.)

[90] B. Mitra and N. Craswell. Query auto-completion for rare prefixes. In *CIKM '15*, pages 1755–1758, 2015. (Cited on page 14.)

[91] K. D. Onal, Y. Zhang, I. S. Altingovde, M. M. Rahman, P. Karagoz, A. Braylan, B. Dang, H.-L. Chang, H. Kim, Q. McNamara, A. Angert, E. Banner, V. Khetan, T. McDonnell, A. T. Nguyen, D. Xu, B. C. Wallace, M. de Rijke, and M. Lease. Neural information retrieval: At the end of the early years. *Information Retrieval Journal*, 21(2–3):111–182, 2018. (Cited on pages 3, 11, 13, 14, and 37.)

[92] U. Ozertem, O. Chapelle, P. Donmez, and E. Velipasaoglu. Learning to suggest: A machine learning framework for ranking query suggestions. In *SIGIR '12*, pages 25–34. ACM, 2012. (Cited on page 13.)

[93] D. H. Park and R. Chiba. A neural language model for query auto-completion. In *SIGIR '17*, pages 1189–1192, 2017. (Cited on page 14.)

[94] G. Pass, A. Chowdhury, and C. Torgeson. A picture of search. In *InfoScale '06*, pages 1–7, 2006. (Cited on pages 3 and 12.)

[95] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *KDD '07*, pages 5–8, 2007. (Cited on page 36.)

[96] G. Prem, J. M. Hofman, and D. M. Blei. Scalable recommendation with hierarchical poisson factorization. In *UAI '15*, pages 326–335, 2015. (Cited on pages 28 and 57.)

[97] L. Qin and X. Zhu. Promoting diversity in recommendation by entropy regularizer. In *IJCAI '13*, pages 2698–2704, 2013. (Cited on page 88.)

[98] M. Quadrana, A. Karatzoglou, B. Hidasi, and P. Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *RecSys '17*, pages 130–137, 2017. (Cited on pages 14, 21, 65, 67, 76, 85, and 88.)

[99] M. Quadrana, P. Cremonesi, and D. Jannach. Sequence-aware recommender systems. *ACM Computing Surveys*, 51(4):66:1–66:36, 2018. (Cited on page 85.)

[100] L. Rakkappan and V. Rajan. Context-aware sequential recommendations with stacked recurrent neural networks. In *WWW '19*, pages 3172–3178, 2019. (Cited on page 85.)
[101] A. M. Rashid, G. Karypis, and J. Riedl. Learning preferences of new users in recommender systems. *ACM SIGKDD Explorations Newsletter*, 10(2):90, 2008. (Cited on pages 30 and 106.)
[102] P. Ren, Z. Chen, J. Li, Z. Ren, J. Ma, and M. de Rijke. Repeatnet: A repeat aware neural recommendation machine for session-based recommendation. In *AAAI '19*, pages 4806–4813, 2019. (Cited on pages 66 and 85.)
[103] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI '09*, pages 452–461, 2009. (Cited on pages 7, 42, 45, 46, and 53.)
[104] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *WWW '10*, pages 811–820, 2010. (Cited on pages 65, 67, 76, and 85.)
[105] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *NIPS '07*, pages 1257–1264, 2007. (Cited on page 42.)
[106] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In *ICML '07*, pages 791–798, 2007. (Cited on pages 33 and 37.)
[107] R. L. T. Santos, C. Macdonald, and I. Ounis. Learning to rank query suggestions for adhoc and diversity search. *Information Retrieval*, 16(4):429–451, 2013. (Cited on page 13.)
[108] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW '01*, pages 285–295, 2001. (Cited on pages 35, 36, 37, and 85.)
[109] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl. Application of dimensionality reduction in recommender system–a case study. In *KDD '00*, pages 1–12, 2000. (Cited on page 36.)
[110] S. Sedhain, A. Menon, S. Sanner, and L. Xie. Autorec: Autoencoders meet collaborative filtering. In *WWW '15*, pages 111–112, 2015. (Cited on pages 2 and 37.)
[111] C. Sha, X. Wu, and J. Niu. A framework for recommending relevant and diverse items. In *IJCAI '16*, pages 3868–3874, 2016. (Cited on pages 2, 5, 86, and 89.)
[112] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *CIKM '14*, pages 101–110, 2014. (Cited on page 14.)
[113] L. Shi, W. X. Zhao, and Y.-D. Shen. Local representative-based matrix factorization for cold-start recommendation. *ACM Transaction on Information Systems*, 36(2):22:1–22:28, 2017. (Cited on pages 64 and 108.)
[114] C. L. Smith, J. Gwizdka, and H. Feild. The use of query auto-completion over the course of search sessions with multifaceted information needs. *Information Processing and Management*, 53(5):1139 – 1155, 2017. (Cited on pages 3 and 13.)
[115] A. Sordoni, Y. Bengio, and H. Vahabi. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In *CIKM '15*, pages 553–562, 2015. (Cited on pages 14, 15, and 20.)
[116] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 1(1):1–4, 2009. (Cited on pages 35 and 37.)
[117] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *CIKM '19*, pages 1441–1450, 2019. (Cited on pages 88 and 95.)
[118] Y. K. Tan, X. Xu, and Y. Liu. Improved recurrent neural networks for session-based recommendations. In *DLRS '16*, pages 17–22, 2016. (Cited on page 68.)
[119] J. Tang and K. Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *WSDM '18*, pages 565–573, 2018. (Cited on pages 2, 4, 65, and 103.)
[120] Y. Tay, L. Anh Tuan, and S. C. Hui. Latent relational metric learning via memory-based attention for collaborative ranking. In *WWW '18*, pages 729–739, 2018. (Cited on page 68.)
[121] Y. Tay, L. A. Tuan, and S. C. Hui. Multi-pointer co-attention networks for recommendation. *CoRR*, abs/1801.09251, 2018. (Cited on page 84.)
[122] S. D. Torres, D. Hiemstra, I. Weber, and P. Serdyukov. Query recommendation in the information domain of children. *Journal of the Association for Information Science and Technology*, 65(7):1368–1384, 2014. (Cited on page 13.)
[123] B. Trapit, B. David, and M. Andrew. Ask the gru: Multi-task learning for deep text recommendations. In *RecSys '16*, pages 107–114, 2016. (Cited on page 37.)
[124] T. T. Truyen, D. Q. Phung, and S. Venkatesh. Ordinal boltzmann machines for collaborative filtering. In *UAI '09*, pages 548–556., 2009. (Cited on page 37.)
[125] A. van den Oord, S. Dieleman, and B. Schrauwen. Deep content-based music recommendation. In *NIPS '13*, pages 2643–2651, 2013. (Cited on page 37.)

[126] C. Van Gysel, M. de Rijke, and E. Kanoulas. Mix 'n match: Integrating text matching and product substitutability within product search. In *CIKM '18*, pages 1373–1382, 2018. (Cited on page 66.)
[127] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, L. u. Gomez, Aidan Nand Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS '17*, pages 5998–6008, 2017. (Cited on pages 88 and 91.)
[128] I. B. Vidinli and R. Ozcan. New query suggestion framework and algorithms: A case study for an educational search engine. *Information Processing and Management*, 52(5):733 – 752, 2016. (Cited on pages 3 and 13.)
[129] M. Wan and J. McAuley. Item recommendation on monotonic behavior chains. In *RecSys '18*, pages 86–94, 2018. (Cited on pages 84 and 109.)
[130] C. Wang and D. M. Blei. Collaborative topic modeling for recommending scientific articles. In *KDD '11*, pages 448–456, 2011. (Cited on pages 33, 37, and 108.)
[131] H. Wang, N. Wang, and D.-Y. Yeung. Collaborative deep learning for recommender systems. In *KDD '15*, pages 1235–1244, 2015. (Cited on pages 33 and 38.)
[132] M. Wang, P. Ren, L. Mei, Z. Chen, M. Jun, and M. de Rijke. A collaborative session-based recommendation approach with parallel memory modules. In *SIGIR '19*, pages 345–354, 2019. (Cited on page 85.)
[133] P. Wang, J. Guo, Y. Lan, J. Xu, S. Wan, and X. Cheng. Learning hierarchical representation model for nextbasket recommendation. In *SIGIR '15*, pages 403–412, 2015. (Cited on pages 65 and 67.)
[134] S. Wang, Y. Wang, J. Tang, K. Shu, S. Ranganath, and H. Liu. What your images reveal: Exploiting visual contents for point-of-interest recommendation. In *WWW '17*, pages 391–400, 2017. (Cited on pages 64 and 108.)
[135] S. Wang, L. Cao, and Y. Wang. A survey on session-based recommender systems. *CoRR*, abs/1902.04864, 2019. (Cited on page 66.)
[136] S. Wang, L. Hu, Y. Wang, Q. Z. Sheng, M. Orgun, and L. Cao. Modeling multi-purpose sessions for next-item recommendations via mixture-channel purpose routing networks. In *IJCAI '19*, pages 3771–3777, 2019. (Cited on pages 88 and 95.)
[137] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua. Neural graph collaborative filtering. In *SIGIR '19*, pages 165–174, 2019. (Cited on page 37.)
[138] B. Wu, W. H. Cheng, Y. Zhang, H. Qiushi, L. Jintao, and T. Mei. Sequential prediction of social media popularity with deep temporal context networks. In *IJCAI '17*, pages 3062–3068, 2017. (Cited on pages 5 and 65.)
[139] Q. Wu, Y. Liu, C. Miao, Y. Zhao, L. Guan, and H. Tang. Recent advances in diversified recommendation. *arXiv preprint arXiv:1905.06589*, 2019. (Cited on pages 5, 86, and 89.)
[140] Y. Wu, C. DuBois, A. X. Zheng, and M. Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *WSDM '16*, pages 153–162, 2016. (Cited on pages 2, 34, and 37.)
[141] C. Xu, P. Zhao, Y. Liu, J. Xu, V. S. S.Sheng, Z. Cui, X. Zhou, and H. Xiong. Recurrent convolutional neural network for sequential recommendation. In *WWW '19*, pages 3398–3404, 2019. (Cited on pages 88 and 103.)
[142] Y. Yao, F. Shen, J. Zhang, L. Liu, Z. Tang, and L. Shao. Extracting privileged information for enhancing classifier learning. *IEEE Transactions on Image Processing*, 28(1):436–450, 2019. (Cited on page 93.)
[143] H. Ying, F. Zhuang, F. Zhang, Y. Liu, G. Xu, X. Xie, H. Xiong, and J. Wu. Sequential recommender system based on hierarchical attention networks. In *IJCAI '18*, pages 3926–3932, 2018. (Cited on pages 68, 71, 76, 77, and 88.)
[144] F. Yu, Q. Liu, S. Wu, L. Wang, and T. Tan. A dynamic recurrent model for next basket recommendation. In *SIGIR '16*, pages 729–732, 2016. (Cited on pages 65, 67, 77, and 88.)
[145] T. Yu, J. Guo, W. Li, H. J. Wang, and L. Fan. Recommendation with diversity: An adaptive trust-aware model. *Decision Support Systems*, 123:113073, 2019. (Cited on pages 103 and 109.)
[146] F. Zhang, N. J. Yuan, D. Lian, X. Xie, and W.-Y. Ma. Collaborative knowledge base embedding for recommender systems. In *KDD '16*, pages 353–362, 2016. (Cited on pages 64 and 108.)
[147] M. Zhang and N. Hurley. Avoiding monotony: Improving the diversity of recommendation lists. In *RecSys '08*, pages 123–130, 2008. (Cited on pages 2, 5, 85, and 96.)
[148] S. Zhang, L. Yao, and A. Sun. Deep learning based recommender system: A survey and new perspectives. *arXiv preprint arXiv:1707.07435*, 2017. (Cited on pages 1, 4, 33, 34, 37, and 47.)
[149] S. Zhang, L. Yao, A. Sun, S. Wang, G. Long, and M. Dong. Neurec: On nonlinear transformation for personalized ranking. *arXiv preprint arXiv:1805.03002*, 2018. (Cited on pages 2, 4, and 65.)
[150] X. Zhang, C. Mei, D. Chen, and J. Li. Feature selection in mixed data: A method using a novel fuzzy

rough set-based information entropy. *Pattern Recognition*, 56:1–15, 2016. (Cited on page 93.)
[151] L. Zheng, V. Noroozi, and P. S. Yu. Joint deep modeling of users and items using reviews for recommendation. In *WSDM '17*, pages 425–434, 2017. (Cited on pages 38, 64, 84, and 108.)
[152] Y. Zheng, B. Tang, W. Ding, and H. Zhou. A neural autoregressive approach to collaborative filtering. In *ICML '16*, pages 764–773, 2016. (Cited on page 37.)
[153] A. Zimdars, D. M. Chickering, and C. Meek. Using temporal data for making recommendations. In *UAI '01*, pages 580–588, 2001. (Cited on page 87.)

Summary

How to learn from users' interactions is a topic that has been widely investigated by search engines as well as recommender systems. The main purpose is to predict users' search intents or preferences and to produce satisfactory recommendations accordingly. However, this can be a challenging task since (1) the relation between users' interactions are complex; (2) users' preferences are dynamic thus recommendation should be up-to-date; and (3) users may have multiple interests in a short session, thus recommendations should not only be personalized but also diverse.

In this thesis, we focus on recommending queries and items in search engines and recommender systems based on learning from user interactions, respectively. We start from analyzing user interactions in search engines and providing personalized query suggestions. We propose a model that applies a hierarchical structure to incorporate users' historical as well as current preferences. An attention mechanism inside the hierarchical structure is meant to capture the varying importance of queries in a session for a certain user. We show the effectiveness of the proposed model in Chapter 2.

Next we investigate user behavior in recommender systems and propose a joint neural collaborative filtering method to capture complex relations in user-item interactions in Chapter 3. This model enables two processes: feature extraction and user-item interaction modeling, to be trained jointly in a unified framework. Interaction modeling can optimize the feature learning process and more accurate feature representations can, in turn, improve the user-item interaction prediction. We also conduct several experiments and show the scalability and sensitivity of our proposed model with different degrees of data sparsity and varying numbers of users' interactions.

We then continue our research on learning users' dynamic preferences for sequential recommendation and mainly consider two aspects: personalization and diversification. In Chapter 4, we focus on personalization and design a dynamic co-attention network model for session-based recommendation that is able to integrate users' long-term and short-term preferences. To further improve the recommendation accuracy, we design a contextual gated recurrent unit to incorporate different types of user actions so as to better estimate users' next consumption motivations. We show that this model can obtain better performance than state-of-the-art methods.

We further take diversification into consideration and propose an intent-aware end-to-end neural approach for diversified sequential recommendation in Chapter 5. We reformulate sequential recommendation as a list generation task in order to model the relationship among recommended items. This model can learn users' multiple interests from their sequential behavior and recommend a preferred as well as diverse set of items. We conduct extensive experiments and the results show that our model outperforms the state-of-the-art baselines in terms of both accuracy and diversity metrics.

Finally, in Chapter 6, we conclude the thesis by summarizing the aforementioned methods and additionally list several possible directions for future work based on the research in this thesis, i.e., incorporating semantic information, content and context information, heterogeneous behavior modeling, and adaptive diversification for sequential recommendation.

Samenvatting

Het leren van gebruikersinteracties is een onderwerp dat zowel door zoekmachines als door aanbevelingssystemen op grote schaal is onderzocht. Het belangrijkste doel hiervan is om de zoekintenties of voorkeuren van gebruikers te voorspellen en daarmee passende aanbevelingen te doen.

Dit kan echter een uitdagende taak zijn, aangezien (1) de relaties tussen de interacties van gebruikers complex zijn; (2) de voorkeuren van gebruikers zijn dynamisch, dus de aanbeveling moet up-to-date blijven; en (3) gebruikers kunnen in een korte sessie meerdere interesses hebben, dus aanbevelingen moeten niet alleen persoonlijk maar ook divers zijn.

In dit proefschrift richten we ons op het leren aanbevelen van zoekopdrachten in zoekmachines en items in aanbevelingssystemen op basis van gebruikersinteracties. We beginnen met het analyseren van gebruikersinteracties in zoekmachines en het geven van gepersonaliseerde zoeksuggesties. We stellen een model voor dat een hiërarchische structuur toepast om de historische en huidige voorkeuren van gebruikers op te nemen. Een attention-mechanisme binnen de hiërarchische structuur is bedoeld om de verschillende maten van belangrijkheid van zoekopdrachten in een sessie voor een bepaalde gebruiker vast te leggen. In Hoofdstuk 2 laten we de effectiviteit van het voorgestelde model zien.

Vervolgens onderzoeken we gebruikersgedrag in aanbevelingssystemen en stellen we een gezamenlijke, neurale, collaboratieve filtermethode voor om complexe relaties vast te leggen in user-item interacties in Hoofdstuk 3. Dit model maakt twee processen mogelijk: feature extractie en user-item interactiemodellering, om gezamenlijk getraind te worden in een verenigd kader. Interactiemodellering kan het leerproces van functies optimaliseren en nauwkeurigere representaties van functies kunnen op hun beurt de voorspelling van interactie tussen gebruikersitems verbeteren. We voeren ook verschillende experimenten uit en tonen hiermee de schaalbaarheid en gevoeligheid van ons voorgestelde model met verschillende mate van data-sparsity en een wisselend aantal interacties van gebruikers.

Vervolgens zetten we ons onderzoek voort naar het leren van dynamische voorkeuren van gebruikers voor opeenvolgende aanbevelingen en houden we voornamelijk rekening met twee aspecten: personalisatie en diversificatie. In Hoofdstuk 4 richten we ons op personalisatie en ontwerpen we een dynamisch co-attention netwerkmodel voor sessiegebaseerde aanbevelingen die de voorkeuren van gebruikers op lange en korte termijn kunnen integreren. Om de nauwkeurigheid van de aanbevelingen verder te verbeteren, ontwerpen we een contextuele gated recurrent unit om verschillende soorten gebruikersacties op te nemen om de volgende consumptiemotivaties van gebruikers beter in te schatten. We laten zien dat dit model betere prestaties kan behalen dan state-of-the-art methoden.

We houden verder rekening met diversificatie en stellen een intentiebewuste end-to-end neurale benadering voor voor gediversifieerde sequentiële aanbeveling in Hoofdstuk 5. We herformuleren sequentiële aanbeveling als een lijstgenererende taak om de relatie tussen aanbevolen items te modelleren. Dit model kan de verschillende interesses van gebruikers leren van hun sequentiële gedrag en een set items aanbevelen die zowel divers is en bij de gebruikersvoorkeuren passen. We voeren uitgebreide exper-

imenten uit en de resultaten tonen aan dat ons model beter presteert dan de state-of-the-art baselines in termen van zowel nauwkeurigheidsstatistieken als diversiteitsmetrieken. Tenslotte, in Hoofdstuk 6, sluiten we het proefschrift af door de bovengenoemde methoden samen te vatten en daarnaast een aantal mogelijke richtingen op te sommen voor toekomstig werk gebaseerd op het onderzoek in dit proefschrift: semantische informatie, inhouds- en contextinformatie, heterogene gedragsmodellering en adaptieve diversificatie voor opeenvolgende aanbevelingen.