A

Presented to
the faculty of the School of Engineering and Applied Science
University of Virginia

in partial fulfillment
of the requirements for the degree

by

APPROVAL SHEET

This

Dissertation

is submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Author: Renqin Cai Renqin Cai

This Dissertation has been read and approved by the examing committee:

Advisor:	Hongning Wang
Advisor:	
Committee Member:	Aidong Zhang
Committee Member:	Denis Nekipelov
Committee Member:	Yangfeng Ji
Committee Member:	Jundong Li
Committee Member:	
Committee Member:	

Accepted for the School of Engineering and Applied Science:

[illegible]

Jennifer L. West, School of Engineering and Applied Science

August 2021

Abstract

From shopping to dining, people consume items on digital service platforms as a part of their daily life routine. In the meanwhile, people leave feedback about the consumptions, e.g., browsing products or writing reviews. By leveraging user feedback, recommender systems predict the items that users would be interested in and then make recommendations. Recommender systems are important for both users and service providers. They not only save users' efforts on finding relevant items but also achieve the providers' objectives on improving user satisfaction. In this thesis, we study the problems of making contextualized recommendations and providing explanations for recommendations, since these are highly desired properties of recommender systems. Modeling the dependence among user feedback is essential to address these two problems. Various research efforts have been devoted to improving the recommendation quality and the quality of explanations for recommendations. Nevertheless, the structure among user feedback is overlooked. In the thesis, we argue that explicitly exploiting the structure (e.g., sequential structure, or graph structure) among user feedback allows us to effectively recognize the dependence among feedback, consequently improving the recommendation quality and the quality of explanations for recommendations. First, for contextualized recommendations, we developed solutions which exploit the sequential structure among user feedback in regard to the temporal information and category information of user feedback. Second, to provide explanations for recommendations, we developed a solution that exploits the graph structure among user feedback. We have demonstrated the effectiveness of our solutions through experiments on large datasets. This thesis shows that exploiting structure among feedback helps build an effective and explainable recommender system.

To everyone who makes the world better

Acknowledgments

I would like to thank all the people who helped me make this thesis happen.

First, I would like to thank my advisor Prof. Hongning Wang for his support and guidance. He is a great role model during my PhD. He is very hardworking, and very enthusiastic in pursuing innovative ideas. His insightful suggestions and keen comments have greatly inspired me. His patient discussions and encouraging messages have helped me go through a lot of difficult moments. I sincerely appreciate his devotion.

I would like to thank my thesis committee members, Prof. Aidong Zhang, Prof. Denis Nekipelov, Prof. Yangfeng Ji and Prof. Jundong Li for their feedback on my PhD research and thesis. I have benefited a lot from the discussions with them. Their suggestions and comments are insightful, which help me finish the thesis.

In addition, I sincerely thank my mentor Chong Wang in Bytedance, and my mentor Parikshit Sondhi and the colleague Zhenrui Wang in WalmartLabs. They provided me opportunities to know how industrial researchers tackle research problems. They also provided tremendous support during the internships.

Moreover, I want to thank my collaborators, Jibang Wu, Mengdi Huai, Xueying Bai, Zhendong Chu, and Peng Wang. They have devoted a lot of time and efforts to the projects we worked on. The discussions with them were always fruitful. Without them, the thesis is not possible.

Then I would like to thank my friends, Junwu Weng, Qiuqiang Kong, Ruocheng Guo, and Zhan Xu. They have provided a lot of suggestions on my research. I also want to thank all HCDM group members. They have built such a wonderful environment for research. I am very grateful for all the discussions and encouragement that I received from them.

Last but not least, I owe my gratitude to my parents and my girlfriend, Summer. They are my biggest supporters. Without them, I could not make this thesis happen.

Contents

Abstract **ii**

Acknowledgments **iv**

Contents **vi**

List of Tables . . . viii

List of Figures . . . ix

1 Introduction **1**

1.1 Motivation and Overview . . . 2

1.2 Thesis Organization . . . 7

2 Time-aware Contextualized Recommendations **9**

2.1 Introduction . . . 9

2.2 Related Work . . . 13

2.2.1 User Behavior Modeling . . . 14

2.2.2 Sequential Recommendation . . . 14

2.2.3 Temporal Recommendation . . . 15

2.3 Notations and Problem Setup . . . 17

2.4 Method 1: Long- & Short-term Hawkes Process . . . 17

2.4.1 Hawkes Process . . . 17

2.4.2 Long- and Short-Term Hawkes Process . . . 18

2.4.3 Parameter Estimation . . . 20

2.5 Method 2: Contextualized Temporal Attention . . . 23

2.5.1 Model Overview . . . 23

2.5.2 Three Stage Weighing Pipeline . . . 25

2.5.3 Parameter Learning . . . 29

2.6 Experiments . . . 30

2.6.1 Dataset . . . 31

2.6.2 Experiments on LSHP . . . 31

2.6.3 Experiments on CTA . . . 33

2.7 Conclusion . . . 41

3 Category-aware Contextualized Recommendations **43**

3.1 Introduction . . . 43

3.2 Related Work . . . 46

3.3 Notations and Problem Setup . . . 48

3.4 Method . . . 48

3.4.1 Data-Driven Statistical Analyses . . . 48

3.4.2 Category-aware collaboration Sequential Recommender . . . 49

3.4.3 Model Training & Inference . . . 54

3.5 Experiments . . . 55
3.5.1 Datasets . . . 56
3.5.2 Comparison against Baselines . . . 57
3.5.3 Detailed Analysis on Our Approach . . . 60
3.6 Conclusion . . . 63

4 Graph Based Extractive Explainer for Recommendations **64**
4.1 Introduction . . . 64
4.2 Related Work . . . 67
4.3 Problem Definition . . . 69
4.4 Graph Extractive Explainer for Recommendations . . . 69
4.4.1 Graph Structure . . . 69
4.4.2 Graph Attention Layer . . . 71
4.4.3 Relevance of Sentences . . . 72
4.4.4 Sentence Extraction . . . 73
4.5 Experiments . . . 74
4.5.1 Experiment Setup . . . 74
4.5.2 Quality of Generated Explanations . . . 76
4.5.3 Hyperparameter . . . 77
4.5.4 Ablation Analysis . . . 77
4.5.5 Case Study . . . 78
4.6 Conclusion . . . 78

5 Conclusions and Future Work **80**
5.1 Conclusions . . . 80
5.2 Future Work . . . 82
5.2.1 Short-term Future Work . . . 82
5.2.2 Long-term Future Work . . . 83

Bibliography **86**

List of Tables

2.1 Statistics of two evaluation datasets. 31
2.2 Performance comparison of LSHP against baselines on sequential recommendation. . 32
2.3 Performance comparison of CTA against different baselines on sequential recommendation. 33
2.4 Ablation analysis on two datasets under metrics of Recall@5 (left) and MRR@5 (right). The best performance is highlighted in bold face. $\downarrow$ and $\uparrow$ denote a drop/increase of performance for more than 5%. ψ, ρ, π, ω respectively denote the exponential, logarithmic, linear and constant temporal kernels. The superscript on the kernel function denotes the number of such kernel used in the model. 37

3.1 Statistics of two evaluation datasets. 54
3.2 Performance of models on Taobao dataset. 57
3.3 Performance of models on BeerAdvocate dataset. 58
3.4 Performance of CoCoRec with different number of actions in in-category encoder (T) and context encoder (L) on Taobao dataset. 62
3.5 Performance of CoCoRec with different number of retrieved neighbors on Taobao dataset. 62

4.1 Summary of the processed datasets. 75
4.2 Comparison of explanation quality by different methods. 76
4.3 Comparison of explanation quality by GREENer with different number of selected sentences k. 77
4.4 Comparison of explanation quality by different variants of GREENer 78
4.5 Example explanations produced by different models. 78

List of Figures

1.1 An illustration of contextualized recommendation. Each user's actions are indexed chronologically. The recommender system needs to predict which items to recommend to the user. 2

1.2 An illustration of explanations for recommendations. Besides recommended items, explanations composed of multiple sentences are presented to users. 4

2.1 Contextualized recommendation in online shopping scenario. 10

2.2 The architecture of our proposed Contextualized Temporal Attention Mechanism. Three stages are proposed to capture the content information at $\boldsymbol{\alpha}$ stage with self-attention, temporal information at $\boldsymbol{\beta}$ stage with multiple kernels, and local information at $\boldsymbol{\gamma}$ stage with recurrent states, for sequential recommendation. 24

2.3 Attention visualization. The blue (left) bar is the content-based importance score $\boldsymbol{\alpha}$, the orange (middle) bar is the contextualized temporal influence score $\boldsymbol{\beta}^c$, the green (right) bar is the combined importance score $\boldsymbol{\gamma}$. The figures contains three different sequences selected from the test set of the Taobao dataset. 40

3.1 An illustration of collaborative sequential recommendation. Each user's actions are indexed chronologically. The recommender system needs to predict which items to recommend to the user Lily based on her and another user Ivy's past actions. 44

3.2 Result of the statistical dependence analysis on Taobao dataset. The distribution of frequencies of 3rd-order item-to-item transition patterns in in-category subsequences are as the red points show. The distribution of the frequencies of 3rd-order item-to-item transition patterns in the original sequences are as the blue points show. 49

3.3 Overview of CoCoRec. In CoCoRec, an action sequence is decomposed into multiple subsequences with respect to the item category associated with each action. The in-category encoder encodes the category-specific action subsequences into latent vectors representing users' in-category preferences. The context encoder predicts the category of the next action to activate the corresponding in-category item-to-item transitions for the next item prediction. The context encoder infers the episodic context of the next action based on recent items. To address the sparsity issue, the collaboration module retrieves neighbors based on users' encoded in-category preferences. Based on signals from these three sources, CoCoRec predicts the next item and make recommendations to the user. 50

3.4 Performance of variants of CoCoRec for ablation analysis on two datasets. 59

3.5 Performance of CoCoRec with different number of selected categories (k) on Taobao. 61

3.6 An example of the next item prediction by CoCoRec. The target user's (user 45323) action sequence is separated into category-specific subsequences. On the top, We visualize the action subsequence containing the recent actions, and the action subsequence specific to the category of the next item. On the bottom, the most similar neighbor's (user 4461) in-category subsequence is presented. The actions in each subsequence are indexed chronologically, denoted as a tuple (item id, category id). The color indicates the attention weight in self-attention networks. 63

4.1 Example of the task of providing explanations for recommendations. Besides the recommendation, the user also cares about the reason of the recommendation. 65
4.2 Illustration of a graph constructed to extract sentences as explanations for a pair of a user and an item. The graph consists of four types of nodes: user node, item node, feature nodes, and sentence nodes. User node is connected to feature nodes, item node is connected to feature nodes, and feature nodes are connected to sentence nodes. . . 70

Chapter 1

Introduction

From e-commerce to entertainment, e.g., Amazon or Netflix, people consume items on the digital service platforms. In the meanwhile, people leave feedback about the consumptions, like browsing products or writing reviews. By leveraging user feedback, *recommender systems* are to predict the items that are relevant to users' preferences and then make recommendations.

Recommender systems have played more and more important roles in people's lives. The rise of information systems allows people to conveniently access the information. On the other hand, the huge amount of information puts burden on people to find out relevant information. What is worse, due to the lack of the knowledge about which items are available, users would miss the items. Thus, tools navigating through the information overload are demanded. Recommender systems, which suggest relevant information to people in a personalized way, consequently become one of the most critical component to the success of information systems. As disclosed in reports [1], in Netflix 75% of what people watch is from recommendations.

Not only recommender systems are important for users, but also are useful for service providers of information systems. Service providers have the growth objectives including user retention, item click-through rate in production. Helping users obtain the relevant information can improve user experience, thus retaining users and increasing the click-through rate. YouTube reports that 60 % of the clicks on the home screen are on the recommendations [1], and about 35 % of Amazon's sales are from recommendations.

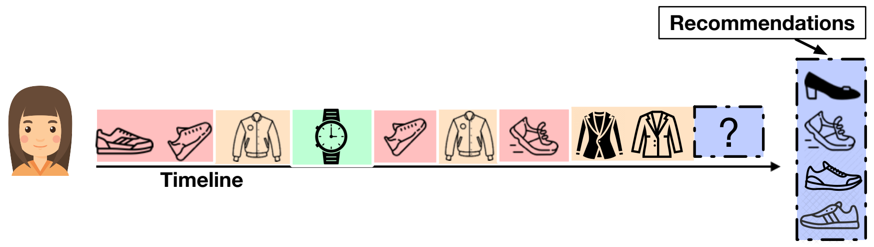

Figure 1.1: An illustration of contextualized recommendation. Each user's actions are indexed chronologically. The recommender system needs to predict which items to recommend to the user.

1.1 Motivation and Overview

Due to the great importance of recommender systems, improving the satisfaction of recommendations, including the effectiveness and the persuasiveness of recommendations, has attracted a great deal of attention [2].

To improve the effectiveness of recommendations, contextualized recommendations are highly desired. Users' preferences towards items are influenced by context. Specifically, context refers to conditions that can influence a user's perception of the relevance of an item [3]. Recommender systems face dynamically evolving user preferences under various context. For example, in Figure 1.1, when a user is looking for casual outfit (e.g., sport shoes) on e-commerce website, recommending formal outfit would annoy the user. In contrast, when the user is looking for formal outfit (e.g., high-heel shoes), it would be annoying to recommend casual outfit to the user. With recommendations impacting people's life under increasingly diverse conditions, considering the context becomes increasingly demanding.

Moreover, to improve the persuasiveness of recommendations, explanations for recommendations are demanding. People not only care about which items have been recommended but are also curious about why these items are recommended. For example, a person is recommended with several types of sport shoes by an e-commerce website. She might be curious about why the website thinks she would like these types of shoes, instead of others. Without any explanations about the recommendations, she would not trust the recommendations and thus would ignore the recommendations. Explanations like "this pair of shoes are cheap and suitable for everyday walking" could make the user realize the recommended shoes may suit her taste. Providing explanations along with recommendations enable the service provider to help users make informed decisions and build trust with users.

Because considering context and providing explanations are critical factors to the success of recommender systems, in this thesis, we study the problems of making contextualized recommendations and providing explanations about recommendations.

To make contextualized recommendations, the essence is to understand how the context affects users' preferences. It is vital to distinguish when an interaction happens (e.g. a day after the last interaction or several minutes since the last interaction) as well as differentiate a casual interaction from a serious interaction. Take the browsing actions in the e-commerce website as an example, shown in Figure 1.1. The user chooses what to browse next in several minutes after browsing the suit. Because of the short time interval, the browsed suit in the last action is a strong indicator of her next preference that she is looking for formal outfits. In contrast, the action of browsing a watch two days ago carries little signal about the next action. The importance of past actions to the prediction of the next action changes along with the changing context.

Considering the contextualized importance of past actions to the next action has driven the progress of recent development of contextualized recommendations [4,5]. The context under which the user interacted with the recommender system is not observable. To consider the context, various types of its proxies are leveraged, like the time when the feedback is generated, or the category of the item associated with the feedback. They are widely available and provide extra information about the feedback. One of the early attempts [6] models user, item and context (e.g., time) with a tensor. Similarly, matrix factorization has also been adopted to model user, item and context with matrices [7,8]. Because of the expressiveness, neural network based models have shown superior performance than matrix or tensor factorization based models [9–11]. Especially neural sequence models, such as Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM) [12], Gated Recurrent Unit (GRU) [13] and self-attention [14] models have been adopted to model user, item and context. For example, Time-LSTM [15] proposed to model the variable time intervals between actions as part of the recurrent unit in LSTM.

Furthermore, to provide explanations about recommendations, the essence is to identify how a user would perceive an item and deliver it clearly to the user. Since an item is associated with multiple features, it is vital to recognize which features of the item attract the user. In addition, making the explanation read as nature as human-generated sentences is friendly to users and easy to convince users. For example, in Figure 1.2, for a user and a pair of shoes, multiple sentences are used to explain why this pair of shoes is recommended to the user. The sentences contain features like

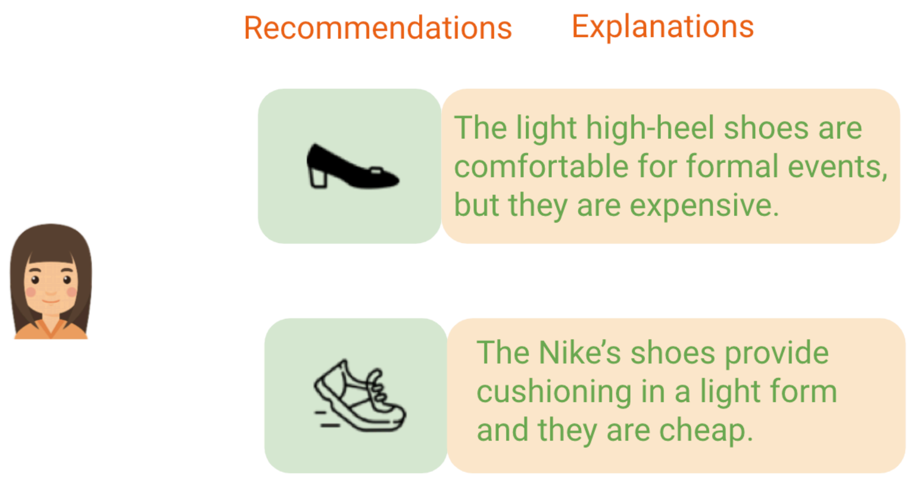
Recommendations
Explanations
The light high-heel shoes are comfortable for formal events, but they are expensive.
The Nike's shoes provide cushioning in a light form and they are cheap.

Figure 1.2: An illustration of explanations for recommendations. Besides recommended items, explanations composed of multiple sentences are presented to users.

"high-heel" and "comfortable" which this user would be interested in.

Generating sentences containing users' opinions towards items' features has been the focus of providing explanations for recommendations. There is a lack of dedicated corpus of textual sentences as explanations. Since user-generated reviews contain sentences describing users' opinions towards features, most existing solutions employed reviews as proxies of explanations. The mainstream solutions leverage text generation techniques to synthesize sentences [16–22]. Based on the encoder-decoder neural model, Attribute-to-Sequence [23] utilized users, items, and ratings as inputs and generated sentences. Similarly, based on the pre-trained language model, i.e., Bidirectional Encoder Representations from Transformers (BERT) [24], Aspect Conditional Masked Language Model has leveraged the expressiveness of pre-trained models to improve the generation quality.

Nevertheless, the *structure* among user feedback is overlooked. Users follow some logic to leave their feedback on the service platforms. The flow of the logic makes the user feedback connected with various types of structure, like sequential structure or graph structure. Take Figure 1.1 as an example of sequential structure among feedback. Browsing behaviors in the e-commerce website can be organized into sequences in regard to their occurring time. While the input data to various solutions is unstructured feedback, the structure among feedback provides extra knowledge about feedback, like which past items should be emphasized for the next item prediction in contextualized recommendations. In Figure 1.1, the order of actions in a sequence suggests the long-term and short-term influence from past actions to the next action. The distant history indicates that the user is interested in shopping sports related products. Now that she is looking for a pair of shoes, the system could have recommended some sports shoes instead of formal ones. However, the recent browsed suits suggest now she is interested in formal outfits. The system should recommend formal shoes to the user. Most existing work assume the models are sufficiently expressive to automatically and intelligently take advantage of the structure. Because of the intricacy of the dependence among feedback, the implicit utilization of structure cannot capture such fine-grained dependence.

In this thesis, we argue that explicitly exploiting the structure of user feedback allows models to effectively recognize the dependence among feedback.

For contextualized recommendations, we make use of sequential structure to improve the next item predictions. The relevance of different segments of historical actions to users' preferences is different in regard to the context. The sequential structure among actions helps models recognize the importance of different segments in suggesting the next action with respect to the context. In this thesis, we consider two types of proxies for context: occurring time of actions and item categories associated with actions. Other types of proxies are left for future work.

First, we model the time intervals among actions when considering the sequential structure. We treat the occurring time of actions as the context of actions. Time intervals among actions reveal the closeness of actions. The distant and prolonged historical actions could carry sparse yet crucial information of user preferences in general, while the more recent actions should more closely represent the user intention in near future. To capture the long-term and short-term influence suggested by the time intervals among actions, we proposed a Hawkes process based solution, Long- and Short-term Hawkes Process (LSHP), and a self-attention based solution, Contextualized Temporal Attention (CTA). In LSHP, based on observations that the importance of past actions differs in accordance with the session boundaries which are defined with respect to time intervals between two consecutive actions, we consider the sequential structure among actions in regard to their sessions. Specifically, inspired by the cognitive psychological concepts "episodic memory", we assume actions within sessions have influence to the next action. Likewise, the concept "semantic memory" motivates us to assume actions associated with the same item across sessions have influence to the next action. Since the influence from actions within sessions include actions in regardless of their associated items and actions within sessions have small time intervals to the next action, we call this type of influence as short-term mutual influence. Since the influence from actions across sessions considers actions associated with the same items and actions across sessions have large time intervals to the next action, we call this type of influence as long-term self-influence. To capture these two types of influence, we design LSHP as a mixture of a multi-variate Hawkes process and a uni-variate Hawkes process. The multi-variate Hawkes process is to capture the short-term mutual influence, while the uni-variate Hawkes process is to capture the long-term self-influence. Based on the influence from past actions, LSHP predicts the next item. In CTA, considering the LSHP's drawback that in regard to the time intervals the influence from past actions decays with the fixed rate, we proposed to utilize a mixture of decaying rates. The decaying patterns of influence is more complex than the fixed rate for all actions. Casual actions decay fast while serious examinations decay in a slow rate. To introduce the flexibility in modeling the decaying patterns concerning the time intervals, CTA adopted exponential decaying rate, logarithmic decaying rate, linear decaying rate and constant decaying rate. In addition, because LSHP utilized the one-hot vector to represent items, the item similarities are ignored with this type of representation. To mitigate this issue, CTA takes advantage of embeddings and self-attention to consider the item similarities when evaluating the importance of past actions to the next action. Thus, in CTA, the influence of past actions to the next actions is determined by their time intervals and item similarities. Based on the influence from past actions, CTA make predictions of the next item.

Second, we model the item category for contextualized recommendations. Treating item categories associated with actions as context, we proposed Category-aware Collaborative Sequential Recommender (CoCoRec) to consider the sequential structure. Based on the observation that actions of the same item category are strong indicators of the next action, CoCoRec considered the sequential structure among in-category subsequences which contain actions of the item categories. In addition, to mitigate the sparsity issue of actions by each user, CoCoRec leveraged other users' similar in-category subsequences. Fusing the signals from both users' own in-category subsequences and other users' in-category subsequences, CoCoRec predicts the next item.

When coming to the explanations for recommendations, we consider graph structure. The historical explanations written by users suggest which features are cared by users, while the historical explanations describing items suggest which item features are noticeable. The graph structure among users, items and explanations suggest which features are important for certain users when perceiving particular items. We proposed a graph based solution, GRaph Extractive ExplaiNer (GREENer), to produce explanations which depict users' perception of items. GREENer extracts sentences from the corpus of reviews as explanations. The graph structure among users, items, features and sentences indicates both the co-occurrence of user-feature and item-feature, and the correlation of feature-sentence. To utilize the graph structure to estimate the relevance of sentences to user-item pairs, GREENer appeals graph attention network to encode high-order relations among nodes in graphs into sentence representations. With encoded sentence representations, GREENer predicts the probabilities of sentences serving as explanations and then select sentences based on predicted probabilities.

To investigate the effectiveness of proposed solutions, we have conducted extensive experiments on large datasets correspondingly. The comparison against baselines and ablation analyses of proposed solutions show that exploiting the contextualized sequential structure helps proposed solutions improve the recommendation quality. In addition, exploiting the graph structure helps the proposed solution improve the quality of explanations. This thesis pushes steps toward building effective and transparent recommender systems.

1.2 Thesis Organization

According to the studied problems, we organize the proposal with two parts: 1) research on making contextualized recommendations, 2) research on providing explanations for recommendations. In particular, the overview of the subsequent chapters in this thesis is as follows.

• **Chapter 2: Temporal Contextualized Recommendations.** In this chapter, we consider two types of context, i.e., occurring time of actions and item category of actions, and study how these two types of context affect user decisions. We propose solutions which exploit the structure among actions in regard to action timestamp and item category. First, to improve the quality of recommendations, we consider the time-aware sequential structure. Inspired by psychological study about user behaviors, we separate the influence of past actions to future actions into long-term and short-term influence in regard to the time proximities among actions. We proposed a Long and Short-term Hawkes Process (LSHP) to capture these two types of influence. In addition, to consider diverse rates of influence decaying over time and improve the expressiveness of the solution, we propose a self-attention based solution, Contextualized Temporal Attention (CTA). The content is organized as, we instantiate the problem of making contextualized recommendations in regard to the action timestamp. We formulate the specific task and describe the related work about considering action timestamp for recommendations. Then we introduce our proposed solutions, including LSHP and CTA. In addition, we evaluate the performance of LSHP and CTA on making recommendations on two large datasets.

- **Chapter 3: Category-aware Contextualized Recommendations.** We consider exploiting the structure among actions in regard to item category to improve the quality of recommendations. The sequential structure among actions with the same item category is validated by our statistical analysis and captured by our model category-aware collaborative recommender (CoCoRec). To mitigate the sparsity issue, our model also leverages sequential structure collaboratively among users who have close preferences towards items of a category. The content is organized as, we specify the task with respect to the item category. Then we describe the existing work of utilizing the item category to enhance the quality of recommendations. In addition, we introduce our proposed solution CoCoRec, and show that CoCoRec can improve the recommendation quality by making use of the structure among actions regarding the item categories.

- **Chapter 4: Graph Based Extractive Explainer for Recommendations.** In this chapter, we utilize processed reviews as proxies of explanations and identify which item features attract users' attention and how users express opinions about these features in sentences. Considering the high-order and complex co-occurrence patterns among users, items, features, and sentences, we construct a graph connecting them and exploit the graph structure among them. To utilize the co-occurrence patterns to produce explanations, we propose a graph neural network based solution to select sentences as explanations, i.e., GRaph Extractive ExplaiNer (GREENer). The content is organized as, we formulate the task and describe the related work on producing natural language explanations. Then we introduce our proposed solution, GREENer. In addition, we discuss the experiments demonstrating GREENer can produce high-quality explanations.

- **Chapter 5: Conclusions and Research Frontiers.** In this chapter, we summarize the thesis. Then we discuss possible future work and research frontiers.

Chapter 2

Time-aware Contextualized Recommendations

Designed to propose a set of relevant items to its users, a recommender system faces dynamically evolving user interests over the course of time under various context. For instance, it is vital to distinguish when the history happened (e.g. a month ago or in the last few minutes). The time intervals suggests how related his/her next preferences are to the past particular event. In this chapter, we studied problem of temporal contextualized recommendation by treating the occurring time of actions as context.

2.1 Introduction

Understanding users' preferences based on their past action is essential to recommender systems. Concerning the sequential dependence within user preferences, sequential recommendations which consider the sequential structure among actions for contextualized recommendations have attracted a lot of attention recently [10,11,25–30]. Specifically, sequential recommendation is to predict the ongoing relevant items based on a sequence of the user's historical actions. Such settings have been practiced in popular industry recommender systems such as YouTube [31,32] and Taobao [33].

Take the online shopping scenario illustrated in Figure 2.1 for example: the system is given a series of user behavior records and needs to recommend the next set of items for the user to examine.

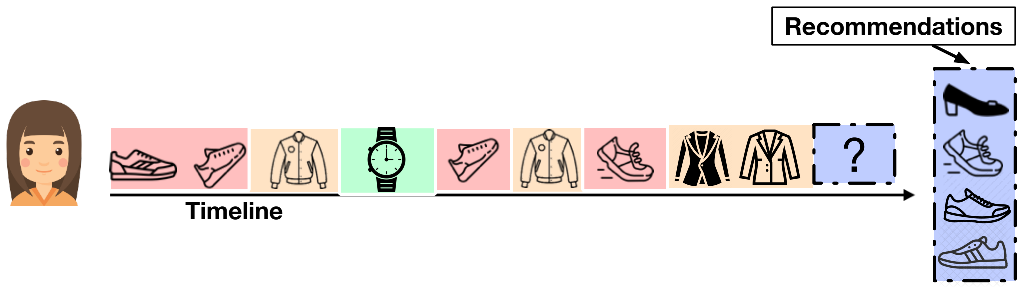

Figure 2.1: Contextualized recommendation in online shopping scenario.

We should note in this setting we do not have assumptions about how the historical actions are generated: solely from interaction between the user and the recommender system, or a mix of users' querying and browsing activities. But we do assume the actions are *not* independent from each other. This better reflects the situation where only offline data and partial records of user behaviours are accessible by a recommender system.

One major challenge to this task is that the influence patterns from different segments of history reflect user interests in different ways, as is exemplified in Figure 2.1:

- By *temporal segment*: The distant history indicates that the user is interested in shopping sports related products. Now that he or she is looking for a pair of shoes, the system could have recommended some sports shoes instead of generic ones. Essentially, the distant and prolonged user history could carry sparse yet crucial information of user preferences in general, while the more recent interactions should more closely represent the user intention in near future. Since the user closely examined several formal suits (much shorter time intervals in between than the average), these interaction events could be emphasized for estimating current user preference such that formal shoes might be preferred over sport shoes. In general, some periods of user browsing log could appear to be heterogeneous, packed with exploration insights, while at a certain point, the user would concentrate on a small subset of homogeneous items, in a repetitive or exploitative way.

Hence, the designs to capture and connect these different signals from each part of history have driven the progress of recent development of recommendation algorithms that consider the sequential structure.

There are attempts for modeling user behavior over the course of time, and mainstream approaches focus on fixed- or varying-order Markov models [34] or hidden Markov models [35], which capture transitional patterns between consecutive user actions in unit time steps. Semi-Markov models are used to model continuous time-intervals between actions [36]. However, it is very expensive to use a Markov model to capture long-term dependency between actions, since the overall state-space grows exponentially with respect to the order of dependency considered.

Some recent efforts have explored point process models, such as Hawkes processes [37,38], to capture the dependency between user actions. But the dependency is simply modeled as additive time decaying from previous actions to current ones, which cannot differentiate influence from previous actions bursting together in a short period of time versus those happening long time ago.

Neural network models have also been adopted for contextualized recommendations. RNNs have been shown to outperform Markov models [10,39]. Traditional RNN-based approaches leave little room for one to dynamically adjust the historical influence in regard to the occurring time of actions. One earlier work, known as Time-LSTM [15], proposed several variants of time gate structure to model the variable time intervals as part of the recurrent state transition. But this assumes that the temporal influence only takes effect for once during transition and is fixed regardless of context or future events. Other work [27,31,33] has borrowed the state-of-art self-attention network structure from nature language modeling [14,24]. In [31], the attention component can enhance the model capacity in determining dependence over an extensively long sequence of history. Nevertheless, Kang and McAuley [27] report that the action order in long sequence of user interaction history is lacking in boosting the empirical evaluation performance on several recommendation datasets, even though the position embedding technique is proposed for self-attention mechanism in its original paper [14]. In other words, there is no explicit ordering of input or segment of history modeled by self-attention mechanism. Therefore, this presents us the opportunity to model temporal information as a more informative and flexible order representation to complement existing attention mechanism, bridging the insights from the both sides of work in sequential recommendation. But the challenge also comes along as incorporating these information could contribute more noise than signal unless properly structured by the model.

In the thesis, we proposed a Hawkes process based solution, i.e., Long- and Short-term Hawkes Process (LSHP), and a self-attention based solution, i.e., Contextualized Temporal Attention (CTA).

The concepts of episodic memory and semantic memory in cognitive psychology [40,41] inspire the design of LSHP. These two types of memory are used to explain how people's past experience influences their next behavior differently. On the one hand, episodic memory records events and context surrounding them, so that context that colours the previous episode is experienced at the immediate moment. On the other hand, semantic memory is a structured record of facts, concepts, and skills that one has acquired in the past. It is simply memory recall and independent of context. These two types of memory are not isolated. Semantic memory is derived from accumulated episodic memory; and episodic memory can be thought of as a "map" that ties together items in semantic memory. These two concepts motivate us to model users' sequential actions over time as a mixture of stochastic point processes, which are driven by different long-term and short-term influence from past actions. To realize contextual information in episodic memory, we employ a multi-dimensional Hawkes process to model the action sequence, where the generation of an action is influenced by other actions in a close temporal proximity, e.g., within a session or task. To capture semantic memory, we employ a one dimensional Hawkes process, in which only the actions of the same item from previous periods influence the actions in the current period. These two types of influence interleave with each other and generate the observed user behavior sequence. We name our resulting stochastic process model as Long- and Short-term Hawkes process, or LSHP in short.

In LSHP, the multi-dimensional Hawkes process captures the "mutual-influence" of different actions within a period of time; and the one dimensional Hawkes process captures the "self-influence" of actions of the same items across different periods of time. Intuitively, mutual-influence reflects the transitional patterns among different actions in a close temporal proximity, and self-influence characterizes repetitive pattern of the same items of actions over a longer period. Because the mixture of these two types of behavior dynamics behind an observed action sequence is latent, we model it in a probabilistic manner and estimate model parameters via a maximum likelihood estimator (MLE). In practice, one can easily expect a large number of items in datasets, which quadratically increases the number of parameters needed to estimate the mutual influence among actions. To avoid overfitting, we assume a sparse structure in the mutual-influence matrix and impose a L1 regularization on the matrix in the objective function of MLE. We adopt alternating direction method of multipliers (ADMM) to iteratively solve resulting optimization problem.

The **C**ontextualized **T**emporal **A**ttention Mechanism (**CTA**) is an attention based sequential neural architecture that draws dependencies among the historical interactions not only through event correlation but also jointly on temporal information for sequential behavior modeling. Considering the LSHP's drawbacks that in regard to the time intervals the influence from past actions decays with the fixed rate, CTA proposed to utilize a mixture of decaying rates. The decaying patterns of influence is more complex than the fixed rate for all actions. In addition, because LSHP utilized the one-hot vector to represent items, the item similarities are ignored with this type of representation. To mitigate this issue, CTA takes advantage of embeddings and self-attention to consider the item similarities when evaluating the importance of past actions to the next action.

In this mechanism, we weigh the historical influence for each historical action at current prediction following the three design questions:

1. *What is the action?* The dependency is initially based on the action correlation through the self-attention mechanism, i.e., how such an action is co-related to the current state in the sequence.

2. *When did it happen?* The influence is also weighed by its temporal proximity to the predicted action, since the temporal dynamics should also play an important role in determining the strength of its connection to presence.

3. *How did it happen?* The temporal weighing factor is realized as a mixture of the output each from a distinct parameterized kernel function that maps the input of time gaps onto a specific context of temporal dynamics. And the proportion of such mixture is determined by local factors, inferred from the surrounding actions. In this way, the influence of a historical action would follow different temporal dynamics under different local conditions.

We apply LSHP and CTA on both XING[1] and Taobao[2] dataset each with millions of interaction events including user, item and timestamp records. The empirical results on both dataset show that our model improves recommendation performance, compared with a selection of state-of-the-art approaches. We also conducted extensive ablation studies as well as visualizations to analyze our model design to understand its advantages and limitations.

2.2 Related Work

Several lines of existing research are closely related to ours in this paper, and their insights largely inspired our model design. In this section, we briefly introduce some key work to provide the context of our work.

[1]https://github.com/recsyschallenge/2016/blob/master/TrainingDataset.md
[2]https://tianchi.aliyun.com/dataset/dataDetail?dataId=649

2.2.1 User Behavior Modeling

User behavior modeling is essential for understanding users' diverse preferences and intents, which in turn provides valuable insights for recommender systems to adaptively maximize their service utility in a per-user basis. Considerable amount of effort has been made on this direction [42–44], while most of it focuses on extracting task-specific features for improving a particular type of applications. For example, in [45], the authors studied students' learning activity patterns recorded in a Massive Online Open Courses (MOOCs) platform and developed a badge-based incentive system to improve student engagements in MOOCs. In [44], the authors proposed a probabilistic model to integrate customer-level behavior features and product-level conversion patterns in e-commerce websites for conversion prediction. Such feature engineering effort targets at specific end tasks, but it can hardly unveil the underlying dynamics of the observed user behavior. Therefore, it is of limited generality.

At a micro level, prior studies show that time intervals between users' sequential actions carry a great deal of information about their underlying intents [46,47]. At a macro level, it has been independently observed in several different application scenarios that a series of user actions burst in a short period, referred as sessions [48–50] or tasks [51,52], and a sequence of a user's interactive behavior is usually carried out over several such periods. More importantly, quantitative analysis suggests correlation of user behavior both within and across those short periods [53], and such correlation enables prediction of users' future behavior [54]. This clearly suggests that a user's sequential interactive behaviors are not a set of independent actions, but there are internal dependency and structure that reflect and characterize his/her underlying preference and intent.

2.2.2 Sequential Recommendation

Sequential recommendations which consider the sequential structure among actions for contextualized recommendations have attracted a lot of attention recently [26]. For the problem of sequential recommendation, statistical models based on Markov assumptions are the early attempts. Factorizing Personalized Markov Chains [55] is a typical model of this kind, which combines factorization machine [56] with a Markov model. Their solution assumes there are patterns in users transiting from an action to another action. The Markov chains are used to model the transition patterns. Because of considering the transition patterns, their solution can thus capture sequence effects in time and predict the next items. But due to the exponential growth of Markov state space with respect to the order of modeled dependence, these Markov models can hardly capture any long-term dependence of actions in a sequence. Recurrent Neural Network (RNN) and its variants, including Long Short-Term Memory (LSTM) [12] and Gated Recurrent Units (GRU) [13], have become a common choice for sequential recommendations [10]. Other methods based on Convolutional Neural Networks (CNN) [28], Memory Network [29] and Attention Models [57] have also been explored. The hierarchical structure generalized from RNN, Attention or CNN based models [11, 25, 26] is used to model transitions inter- and intra-sessions. The recent work [25] by You et al. showed that using Temporal Convolutional Network to encode and decode session-level information and GRU for user-level transition is the most effective hierarchical structure. Nevertheless, as many studies borrow sequence models from natural language modeling task directly, their model performance is usually limited by the relatively small size and sparse pattern of user behaviors, compared to the nature language datasets.

The attention mechanism was first coined in [58]. The original structure is constructed on the hidden states generated from RNN in order to better capture the long-term dependence and align the output for decoder in RNN. The Transformer model [14] and several follow-up work [24, 59, 60] showed that for many NLP tasks, the sequence-to-sequence network structure based on attention alone, a.k.a. self-attention mechanism, is able to outperform existing RNN structures in both accuracy and computation complexity in long sequences. Motivated by this unique advantage of self-attention, several studies introduced this mechanism to sequential recommendation. SASRec [27], based on self-attention mechanism, demonstrated promising results in modeling longer user sequences without the session assumption. Another work known as Multi-temporal range Mixture Model (M3) [31] manages to hybrid the attention and RNN models to capture the long-range dependent user sequences. The most recent work, BERT4Rec [33], adopts the bidirectional training objective via Cloze task and further improves its performance over SASRec.

2.2.3 Temporal Recommendation

Temporal recommendation specifically studies the temporal evolution of user preferences and items; and methods using matrix factorization have shown strong performance. TimeSVD++ [61] achieved strong results by splitting time into several bins of segments and modeling users and items separately in each. Bayesian Probabilistic Tensor Factorization (BPTF) [62] is proposed to include time as a special constraint on the time dimension for the tensor factorization problem. And many of these solutions [62–64] in temporal recommendation share the insight to model separately the long-term static and short-term dynamic user preference. Nevertheless, none of the models are developed specifically for sequential recommendation.

There have been various efforts to utilize temporal information in existing deep recommendation models. In [65], the authors proposed methods to learn time-dependent representation as input to RNN by contextualizing event embedding with time mask or event-time joint embeddings. In [15], authors proposed several variants of time gate structure to model the variable time interval as part of the recurrent state transition. But the empirical results of both model show limited improvement compared to their LSTM baseline without using temporal information.

Meanwhile, the time series analysis is a well established research area with broad application in real world problems [66,67]. Hawkes process [37,68] is one of the powerful tools for modeling and predicting temporal events. In [69], authors combined Hawkes process with Dirichelt process to cluster sequences of users' online video watching records. In [38], a multi-task Hawkes process is proposed to identify user clusters through learning cluster specific mutual-influence. It models the intensity of events to occur at moment t conditioned on the observation of historical events. In a standard Hawkes process, temporal dependency between actions are generally modeled as additive time decaying from previous actions to current ones. Some recent work [46,70,71] attempt to use RNN to model the intensity function of point process model and predict the time of next action. As a typical example, the Neural Hawkes Process [72] constructs a neurally self-modulating multivariate point process in LSTM, such that the values of LSTM cells decay exponentially until being updated when a new event occurs. Their model is designed to have better expressivity for complex temporal patterns and achieves better performance compared to the vanilla Hawkes process. However, these solutions cannot distinguish influence from previous actions happening together in a short period of time versus those taking place in a long period of time. Motivated by the concepts of episodic memory and semantic memory in cognitive psychology, we separate these two types of temporal dependencies into two Hakwes processes: one focuses on mutual influence across different types of actions in a close temporal proximity, and another focuses on self influence within the same type of actions in a longer period of time. This provides the model with both flexibility and constraint in modeling users' sequential actions.

2.3 Notations and Problem Setup

In this section, we introduce the notations used in this chapter and the problem setup studied in this chapter. We consider the sequential recommendation problem with temporal information. Denote the item space as $\mathcal{V}$ of size $|V|$, and the user space as $\mathcal{U}$ of size U. The model is given a set of user behavior sequences $\mathcal{C} = \{\mathcal{S}^1, \mathcal{S}^2, \dots, \mathcal{S}^U\}$ as input. Each $\mathcal{S}^u = \{(t_1^u, v_1^u), (t_2^u, v_2^u), \dots\}$ is a sequence of N actions. An action is a time-item tuple, where t_j^u is the timestamp when item v_i^u is accessed by user u. The action sequence is chronologically ordered, i.e., $t_i^u \leq t_{i+1}^u$. The interacted item is represented as a one-hot vector $v \in \mathbb{R}^{1 \times |V|}$ and the timestamp is a real valued scalar $t \in \mathbb{R}^+$. The notion of session is defined in regard to the time interval between two consecutive actions [73]. A session ends if there is a period of inactivity since the last action, like 30 minutes. Until the next action occurs, a new session starts. A sequence composed of K sessions can then be represented as $\mathcal{S}^u = \{s_1^u, ..., s_K^u\}$, where the k-th session $s_k^u = \{(t_{k1}^u, v_{k1}^u), (t_{k2}^u, v_{k2}^u), \dots\}$. The recommendation task is to select a list of items $V \subseteq \mathcal{V}$ for each user u at a given time t with respect to $\mathcal{S}^u$, such that V best matches user u's interest at the moment. When no ambiguity is incurred, we omit the subscript u to simplify our subsequent discussions.

2.4 Method 1: Long- & Short-term Hawkes Process

In this section, we first discuss some basics of Hawkes process. Based on these, we provide detailed description of our algorithm, Long- and Short-term Hawkes Process (LSHP), which differentiates short-term and long-term influence among sequential actions by separating mutual-influence among actions of different items in a close temporal proximity from self-influence among actions of the same item in a longer temporal distance. We estimate the model parameters of LSHP via alternating direction method of multipliers (ADMM).

2.4.1 Hawkes Process

Before discussing the details of our proposed model, we briefly introduce Hawkes process [37]. Hawkes Process is a type of temporal point process, which models sequences of timestamped actions. It assumes historical actions would influence the intensity of future actions over a certain period of time [68]. And it has been widely applied to modeling sequences of events over time, such as earthquake aftershocks [74]. In Hawkes process, the conditional intensity function is used to depict the generating rate of current action given historical actions and to capture the influence of historical actions on current one. For example, in one dimensional Hawkes process that models the generation of sequences with actions of a single item (e.g., the item v), the conditional intensity function at time t is defined as

$$\lambda_v^*(t) = \mu_v + \int_0^t \alpha_v \kappa(t - t') dN_v(t'),$$

where μ_v is the base intensity, representing the instantaneous generation rate of the action with the item v. The kernel function $\alpha_v \kappa(t - t')$ describes the influence of past actions $N_v(t')$ with the item v on the current action at time t in this sequence. This reflects the "self-exciting" property of Hawkes process. The parameter α represents the strength of self-excitement, and $\kappa(t - s)$ characterizes the time decaying effect. Exponential kernel or power-law kernel is typically chosen to describe this time decaying effect. And because of time decay, actions occurring temporally closer have stronger influence than those temporally further away.

A one dimensional Hawkes process only considers the influence from previous actions of the same item. Efforts have been made to extend it to multi-dimensional Hawkes process capturing dependence among actions of different items, where the conditional intensity function of an action with the item v at time t is defined as,

$$\lambda_v^*(t) = \mu_v + \sum_{v'=1}^{V} \int_0^t A_{v'v} \kappa(t - t') dN_{v'}(t'),$$

where the parameter $A_{v'v}$ represents the influence of type v' on type v, and v' denotes the type associating with previous actions occurring at time t'.

2.4.2 Long- and Short-Term Hawkes Process

Motivated by the cognitive psychology concepts of semantic memory and episodic memory, which describe how the past experience or knowledge influences people's present or future behaviors, we model users' sequential interactive behaviors as a mixture of two different stochastic processes. Specifically, we consider the actions happening in the same session of the current action as its context. As studies in cognitive psychology suggest that episodic memory would gradually lose its sensitivity in context over time, we assume the context actions only generate their influence within a session. Hence, we adopt a multi-dimension Hawkes process to realize the concept of episodic memory as the mutual influence of past actions to this current action in this session. This forms the first stochastic process in LSHP. On the other hand, it is suggested that the semantic memory is memory recall [40], independent of context. We regard the repetition of actions of the same type in a sequence as a result of semantic memory. We employ a one dimensional Hawkes process to realize the semantic memory as a self-influence driven action generation. Because semantic memory is derived from accumulated episodic memory, we assume this one dimensional Hawkes process is only influenced by actions of the same type from preceding sessions of the current action. This forms the second stochastic process in LSHP.

Using the language of Hawkes process, we incorporate these two stochastic processes into one process: for the i-th action of the item v_i appearing at time t_i in an action sequence $\mathcal{S}$, the conditional intensity specified by LSHP is:

$$\lambda^*_{v_i}(t_i) = \mu_{v_i} + \sum_{t_l < t_i} A_{v_l v_i} \kappa_m(t_i - t_l)\delta(s_l = s_i) + \sum_{t_j < t_i} B_{v_i} \kappa_s(t_i - t_j)\delta(s_j \neq s_i)\delta(v_j = v_i) \tag{2.1}$$

where s_k represents the session index of the action (t_k, v_k) and $\delta(\cdot)$ is an indicator function.

There are three key components in LSHP. First, the base intensity μ_{v_i} describes the spontaneous occurring rate of an action with the item v_i, and $\mu_{v_i} > 0$. For example, if an action with a particular item occurs at the beginning of a new session and this is its first appearance in the sequence, we will accredit this occurrence to its base intensity, since the user's episodic memory has not formed (as it is the first action in this session) and his/her semantic memory does not include this item (as it is the first time this item appears in the sequence). In this work, we assume the base intensity is static over time, and leave the dynamic base intensity for future work.

Second, to capture the dependence of the next action on its preceding actions within the same session, a mutual influence matrix $A \in \mathbb{R}_+^{V \times V}$ over different items is introduced in LSHP. Each element $A_{v_l v_i}$ specifies the influence from the item v_l to another item v_i. We do not assume the mutual influence is symmetric and leave it for the model to decide from data, as it is possible in some applications, an action of a certain item is more likely to lead to an action of another item, rather than the other way around. To realize the assumption that actions with closer temporal proximity have larger influence on each other, we use an exponential kernel $\kappa_m(t_i - t_l) = \exp\big(-\beta_m(t_i - t_l)\big)$ to scale the mutual-influence, i.e., accounting for the time decaying effect. We should notice this mutual influence is limited to actions within the same session, to realize the short-term temporal influence.

Third, LSHP models the repetitive interactions of the same item across sessions with a one dimensional

Hawkes process, in which B_{v_i} represents the item v_i's self-influence strength. Another exponential kernel function $\kappa_s(t_i - t_j) = \exp\big(-\beta_s(t_i - t_j)\big)$ is used to account for time decaying in self-influence. As mutual influence is used to characterize actions' in-session dependence and self influence is for across session dependence, the distribution of time intervals for these two types of dependence are intrinsically different. To account for the difference, we use different decaying coefficient, β_s and β_m, in the corresponding kernel functions.

Comparing with standard Hawkes process which has been used for user behavior modeling [38,69], our LSHP model defined in Eq (2.1) differentiates the long-term and short-term temporal dependency among actions. Instead of using a universal time decaying function over all historical actions, LSHP captures short-term mutual influence among actions of different types in the same session and long-term self influence between actions of the same type across sessions. These two types of dependency are integrated into one stochastic process to account for the heterogeneity of users' sequential interactive behaviors. And this design does not increase the model complexity.

2.4.3 Parameter Estimation

To apply LSHP, we need to estimate its model parameters, i.e., the base intensity vector $\mu \in R_+^{|V|}$, the mutual influence matrix $A \in R_+^{|V|\times|V|}$, and the strength vector of self-excitement $B \in R_+^{|V|}$. We treat the time decaying coefficients β_m and β_s as hyper-parameters, and appeal to the maximum likelihood estimator for parameter estimation in LSHP.

Given a corpus of sequences $\mathcal{C} = \{\mathcal{S}^1, \mathcal{S}^2, \dots, \mathcal{S}^U\}$, assuming the time span in a sequence S^u is T^u, the log likelihood of LSHP on this corpus is computed as,

$$L(\mu, A, B) = \sum_{u=1}^{|U|}\sum_{i=1}^{N} \log \lambda^*_{v_i}(t_i) - \sum_{u=1}^{|U|}\sum_{v=1}^{|V|}\int_0^{T^u} \lambda^*_v(t)dt \tag{2.2}$$

Because the size of the mutual influence matrix A increases quadratically with respect to the number of unique items in the corpus, we need to control the model complexity to avoid overfitting. We assume the mutual-influence between items is sparse in nature, and impose a $L1$ regularization on it. Consequently the optimization objective function is,

$$\min_{\mu>0, A\geq 0, B\geq 0} -L(\mu, A, B) + \eta_A||A||_1$$

where η_A is a trade-off coefficient. Because of the introduction of L1 regularizer, the objective function is not differentiable, we appeal to the alternating direction method of multipliers (ADMM) [75, 76] to solve the optimization problem.

To apply ADMM, we rewrite the objective function by introducing auxiliary variable Z and dual variable Q into the following format,

$$F(\mu, A, B, Z, Q) = \min_{\mu>0, A\geq 0, B\geq 0, Z} -L(\mu, A, B) + \eta_A ||Z||_1 + \rho Tr\big(Q^T(A-Z)\big) + \frac{\rho}{2}||A-Z||^2 \quad (2.3)$$

where $\rho > 0$ is a hyper-parameter. We solve the problem by iteratively updating μ, A, B, Z, Q with respect to the following steps.

Step 1: Update $\boldsymbol{\mu, A, B}$. The terms in Eq. (2.3) that are relevant to the update of μ, A, B include,

$$F(\mu, A, B) = \min_{\mu>0, A\geq 0, B\geq 0} -L(\mu, A, B) + \rho Tr\big(Q^T(A-Z)\big) + \frac{\rho}{2}||A-Z||^2 \quad (2.4)$$

To solve the optimization problem defined in $F(\mu, A, B)$, we adopt the majorization-minimization algorithm, which optimizes the upper bound of $F(\mu, A, B)$ by introducing a set of branching parameters p_{ii}, p_{ji} and p_{li}. One advantage of using majorization-minimization to minimize the upper bound of this objective function is that we can obtain closed form solutions for μ, A, B independently; and in the meanwhile, the non-negativity constraints are satisfied automatically. Replacing Eq. (2.2) and Eq. (2.1) into $F(\mu, A, B)$, we obtain,

$$\begin{aligned}
&F(\mu, A, B) = \\
&\min_{\mu>0, A\geq 0, B\geq 0} -\sum_{u=1}^{|U|}\sum_{i=1}^{N} \log \Big(\mu_{v_i} + \sum_{t_l<t_i} A_{v_l v_i}\kappa_m(t_i - t_l)\delta(s_l = s_i) \\
&+ \sum_{t_j<t_i} B_{v_i}\kappa_s(t_i - t_j)\delta(s_j \neq s_i)\delta(v_j = v_i)\Big) \\
&+ \sum_{u=1}^{|U|}\sum_{v=1}^{|V|}\int_0^{T^u} \lambda_v^*(t)dt + \rho Tr(Q^T(A-Z)) + \frac{\rho}{2}(||A-Z||^2) \\
\leq & -\sum_{u=1}^{|U|}\sum_{i=1}^{N}\Big(p_{ii}\log\frac{\mu_{v_i}}{p_{ii}} + \sum_{t_l<t_i} p_{li}\delta(s_l = s_i)\log\frac{A_{v_l v_i}\kappa_m(t_i - t_l)}{p_{li}} \\
&+ \sum_{t_j<t_i} p_{ji}\delta(s_j \neq s_i)\delta(v_j = v_i)\log\frac{B_{v_i}\kappa_s(t_i - t_j)}{p_{ji}}\Big) \\
&+ \sum_{u=1}^{|U|}\sum_{v=1}^{|V|}\int_0^{T^u} \lambda_v^*(t)dt + \frac{\rho}{2}\big(||A-Z+Q||^2\big)
\end{aligned}$$

The branching parameters are defined as

$$
\begin{aligned}
p_{ii} &= \frac{\mu_{v_i}}{\lambda^*_{v_i}(t_i)} \\
p_{li} &= \frac{A_{v_l v_i}\kappa_m(t_i - t_l)\delta(s_l = s_i)}{\lambda^*_{v_i}(t_i)} \\
p_{ji} &= \frac{B_{v_i}\kappa_s(t_i - t_j)\delta(s_j \neq s_i)\delta(v_j = v_i)}{\lambda^*_{v_i}(t_i)}
\end{aligned}
$$

The branching parameter p_{ii} could be considered as the probability that the i-th action is generated from the base intensity. And the branching parameter p_{li} indicates the probability that the l-th action within this session leads to the i-th action. Likewise, the branching parameter p_{ji} can be regarded as the probability that the j-th action occurring in previous sessions leads to the i-th action in this session. Setting the gradients of these parameters to zero, we obtain the updating rule of μ, A, B as follows:

$$
\mu_v = \frac{\sum\limits_{u=1}^{|U|}\sum\limits_{i}^{N} p_{ii}\delta(v_i = v)}{\sum_{u=1}^{|U|} T^c} \tag{2.5}
$$

$$
A_{vv'} = \frac{1}{2\rho}\Big(- X + \sqrt{X^2 - 4\rho Y}\Big) \tag{2.6}
$$

$$
X = \rho(U_{vv'} - Z_{vv'}) + \sum_{u=1}^{|U|}\sum_{k=1}^{K}\sum_{l=1}^{M_k} \delta(v_l = v)\int_{t_l}^{t_{M_k}} \kappa_m(t - t_l)dt
$$

$$
Y = -\sum_{u=1}^{|U|}\sum_{i=1}^{N}\sum_{t_l<t_i} p_{li}\delta(S_l = S_i)\delta(v_l = v, v_i = v')
$$

$$
B_v = \frac{\sum\limits_{u=1}^{|U|}\sum\limits_{i=1}^{N}\sum\limits_{t_j<t_i} p_{ji}\delta(v_i = v_j = v)\delta(S_j \neq S_i)}{\sum\limits_{u=1}^{|U|}\sum\limits_{k=1}^{K}\sum\limits_{j=1}^{M_k} \delta(v_j = v)\int_{t_{M_k}}^{t_{M_K}} \kappa_s(t - t_j)dt} \tag{2.7}
$$

The updating rules of A suggest that the value $A_{vv'}$, corresponding to the mutual-influence between the item v and the item v', correlates with both frequency of the item v and the item v' co-occurring in the same session, and the time interval between actions with these two items. The shorter time the actions are to each other, the stronger mutual influence they would have on each other. Besides, the updating rules for μ and B suggests that not only the frequency of items in sequences but also the relative temporal duration of actions of these items affect their estimates.

Step 2: Update $\boldsymbol{Z}$. With the updated parameters μ, A, B, we update Z through solving the following optimization problem.

$$Z =_Z \eta_A ||Z||_1 + \rho Tr(Q^T(A - Z)) + \frac{\rho}{2}(||A - Z||^2)$$

The updating rule depending on the magnitude of $A + U$ is

$$Z_{vv'} = \begin{cases} (A_{vv'} + Q_{vv'}) - \frac{\eta_A}{\rho}, & A_{vv'} + Q_{vv'} \geq \frac{\eta_A}{\rho} \\ (A_{vv'} + Q_{vv'}) + \frac{\eta_A}{\rho}, & A_{vv'} + Q_{vv'} \leq \frac{-\eta_A}{\rho} \\ 0, & |A_{vv'} + Q_{vv'}| < \frac{\eta_A}{\rho} \end{cases}$$

As the equation suggests, the auxiliary variable Z is introduced to handle the L1 regularizer on the mutual influence matrix A.

Step 3: Update $\boldsymbol{Q}$. Given the updated parameters μ, A, B and auxiliary variable Z, we update the dual variable

$$Q_{new} = Q_{old} + (A_{new} - Z_{new})$$

where A_{new}, Z_{new} represent updated mutual-influence matrix and auxiliary variable respectively.

2.5 Method 2: Contextualized Temporal Attention

In this section, we discuss the details of our proposed **C**ontextualized **T**emporal **A**ttention Mechanism (**CTA**) for sequential recommendation. We will first provide a high-level overview of the proposed model, and then zoom into each of its components for temporal and local information modeling.

2.5.1 Model Overview

In this section, we will introduce from a high level about each part of our CTA model in a bottom-up manner, from the inputs, through the three stage pipeline: content-based attention, temporal kernels and local mixture, denoted as $\boldsymbol{\alpha} \rightarrow \boldsymbol{\beta} \rightarrow \boldsymbol{\gamma}$ stages as illustrated in Figure 2.2, and finally into the output.

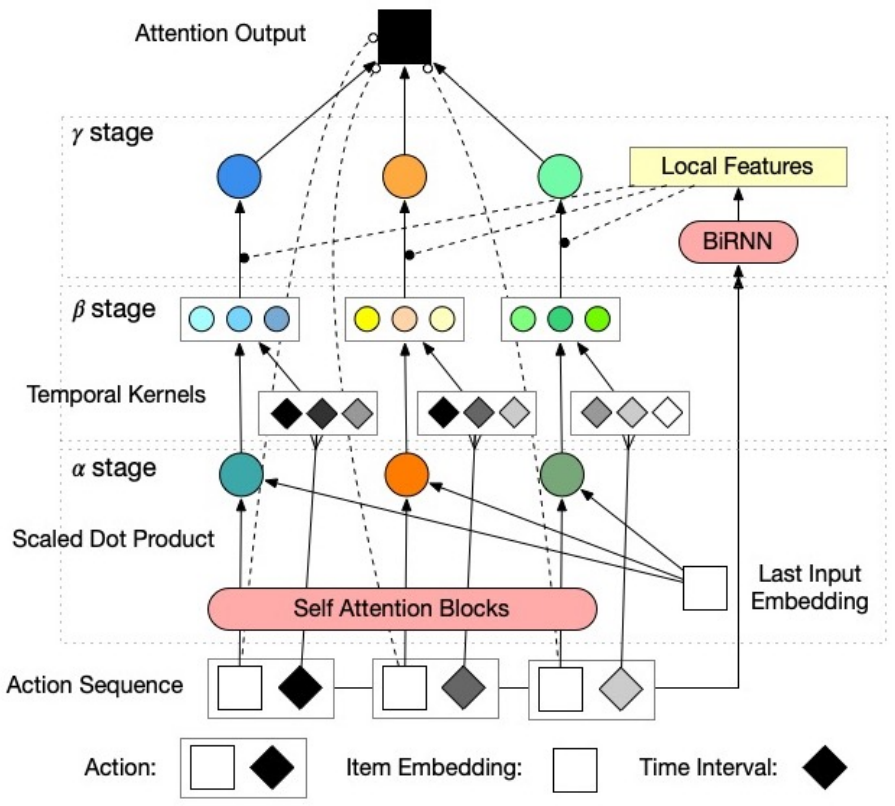

Figure 2.2: The architecture of our proposed Contextualized Temporal Attention Mechanism. Three stages are proposed to capture the content information at $\boldsymbol{\alpha}$ stage with self-attention, temporal information at $\boldsymbol{\beta}$ stage with multiple kernels, and local information at $\boldsymbol{\gamma}$ stage with recurrent states, for sequential recommendation.

The raw input consists of the user's historical events of a window size L in item and time pairs $\{(t_i, s_i)\}_{i=1}^{L}$, as well as the timestamp at the moment of recommendation t_{L+1}. The sequence of input items is mapped into embedding space with the input item embeddings $E_{\text{input}} \in \mathbb{R}^{N \times d_{\text{in}}}$: $\boldsymbol{X} = [s_1, \dots, s_L] \cdot E_{\text{input}} \in \mathbb{R}^{L \times d_{\text{in}}}$. We also transform the sequence of timestamps into the intervals between each action to current prediction time: $\boldsymbol{T} = [t_{L+1} - t_1, \dots, t_{L+1} - t_L] \in \mathbb{R}^{L \times 1}$.

Motivated by our earlier analysis, we design the three stage mechanism, namely M^{α}, M^{β} and M^{γ}, on top of the processed input $\boldsymbol{X}$ and $\boldsymbol{T}$, to model dependencies among the historical interactions respectively on their content, temporal, and local information:

$$\boldsymbol{\alpha} = M^{\alpha}(\boldsymbol{X}) \rightarrow \boldsymbol{\beta} = M^{\beta}(\boldsymbol{T}) \rightarrow \boldsymbol{\gamma} = M^{\gamma}(\boldsymbol{X}, \boldsymbol{\beta}, \boldsymbol{\alpha})$$

In essence, M^{α} weighs the influence of each input purely on content $\boldsymbol{X}$ and outputs a scalar score as importance of each events in sequence $\boldsymbol{\alpha} \in \mathbb{R}^{L \times 1}$; M^{β} transforms the temporal data $\boldsymbol{T}$ through K temporal kernels for the temporal weighing of each input $\boldsymbol{\beta} \in \mathbb{R}^{L \times K}$; M^{γ} extracts the local information from $\boldsymbol{X}$, with which it mixes the factors $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ from previous stages into the contextualized temporal importance score $\boldsymbol{\gamma} \in \mathbb{R}^{L\times 1}$. We will later explain their individual architectures in details.

In the end, our model computes the row sum of the input item sequence embedding $\boldsymbol{X}$ weighted by $\boldsymbol{\gamma}$ (through the softmax layer, the weight $\boldsymbol{\gamma}$ sums up to 1). This weighted sum design is borrowed from the attention mechanism in a sense of taking expectation on a probability distribution, $\boldsymbol{\gamma} = [P(\hat{x}_{L+1} = x_i|\boldsymbol{X}, \boldsymbol{T})]_{i=0}^{L}$. The representation is then projected to the output embedding space $\mathbb{R}^{d_{\text{out}}}$ with a feed-forward layer F^{out} :

$$\hat{x}_{L+1} = F^{\text{out}}(\boldsymbol{\gamma}^T \cdot \boldsymbol{X}) \in \mathbb{R}^{d_{\text{out}}\times 1}$$

We consider $\hat{x}_{L+1}$ as the predicted representation of recommended item. We define matrix $E_{\text{output}} \in \mathbb{R}^{N\times d_{\text{out}}}$, where its ith row vector is the item i's representation in the output embedding space. Then $\forall i \in \mathcal{I}$, the model can compute the similarity r_i between item i and the predicted representation $\hat{x}_{L+1}$ through inner-product (or any other similarity scoring function):

$$\hat{s}_{L+1} = (r_1, \ldots, r_N) = E_{\text{output}} \cdot \hat{x}_{L+1} \in \mathbb{R}^{N\times 1}$$

For a given user, item similarity scores are then normalized by a softmax layer which yields a probability distribution over the item vocabulary. After training the model, the recommendation for a user at step $L+1$ is served by retrieving a list of items with the highest scores r_v among all $v \in \mathcal{V}$.

2.5.2 Three Stage Weighing Pipeline

$\boldsymbol{\alpha}$ stage, what is the action: The goal of $\boldsymbol{\alpha}$ stage is to obtain the content-based importance score $\boldsymbol{\alpha}$ for the input sequence $\boldsymbol{X}$. Following the promising results of prior self-attentive models, we adopt the self-attention mechanism to efficiently and effectively capture the content correlation with long-term dependence. In addition, the self-attention mechanism allows us to directly define the importance score over each input, in contrast to the recurrent network structure.

We use the encoder mode of self-attention mechanism to transform the input sequence embedding $\boldsymbol{X}$, through a stack of d_l self-attentive encoder blocks with d_h heads and d_a hidden units, into representation $\boldsymbol{H}^{d_l}$, which is the hidden state of the sequence at the last layer. Due to the recursive nature of self-attention, we use the following example to explain the multi-head attention component in our solution. For example, in the ith attention head of the jth self attention block, from the input state $\boldsymbol{H}^j$, we compute one single head of the self-attended sequence representation as,

$$\boldsymbol{z}_i^j = \text{Attention}(\boldsymbol{H}^j W_i^Q, \boldsymbol{H}^j W_i^K, \boldsymbol{H}^j W_i^V)$$

where $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d_a \times d_a/d_h}$ are the learnable parameters specific to ith head of jth attention block, used to project the same matrix $\boldsymbol{H}^j$ into the query $\boldsymbol{Q}$, key $\boldsymbol{K}$, and value $\boldsymbol{V}$ representation as the input to the Scaled Dot-Product [14]:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q} \cdot \boldsymbol{K}^T}{\sqrt{d_a/d_h}}\right)\boldsymbol{V}$$

Here the scaling factor $\sqrt{d_a/d_h}$ is introduced to produce a softer attention distribution for avoiding extremely small gradients.

All the computed heads $\boldsymbol{z}_i^d$ in the jth attention block is stacked and projected as $\boldsymbol{Z}^j = [\boldsymbol{z}_1^d, \ldots, \boldsymbol{z}_{d_h}^d] \cdot W^O$, where $W^O \in \mathbb{R}^{d_a \times d_a}$. We can then employ the residue connection [77] to compute the output of this attention block as:

$$\boldsymbol{H}^{j+1} = \text{LN}\left(\boldsymbol{H}^j + F^j(\boldsymbol{Z}^j)\right)$$

where F^j is a feed-forward layer specific to the jth attention block mapping from $\mathbb{R}^{d_a}$ to $\mathbb{R}^{d_{\text{in}}}$ and LN is the Layer Normalization function [78].

Note that for the initial attention block, we use $\boldsymbol{X}$ to serve as the input $\boldsymbol{H}^0$; and in the end, we obtain $\boldsymbol{H}^{d_l}$ as the final output from self-attention blocks. In prior work [27,33], this $\boldsymbol{H}^{d_l}$ is directly used for prediction. Our usage of self-attention structure is to determine a reliable content-based importance estimate of each input, hence we compute once again the Scale Dot-Product using the last layer hidden states $\boldsymbol{H}^{j+1}$ to project as the query and the last item input embedding x_L^T to project as the key via $W_0^Q, W_0^K \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}$:

$$\boldsymbol{\alpha} = \text{softmax}\left(\frac{(\boldsymbol{H}^{j+1} W_0^Q) \cdot (x_L W_0^K)^T}{\sqrt{d_{\text{in}}}}\right)$$

Note that we can also view this operation as the general attention [79], i.e., the bi-linear product of the last layer hidden states and the last input item embedding, where $W_0^Q (W_0^K)^T$ is the learnable attention weight and $\sqrt{d_{\text{in}}}$ serves as the softmax temperature [80].

$\boldsymbol{\beta}$ stage, when did it happen: The goal of $\boldsymbol{\beta}$ stage is to determine the past events' influence based on their temporal gaps from the current moment of recommendation. The raw information of time intervals might not be as useful to indicate the actual *temporal distance* of a historical event's influence (e.g., perceived by the user), unless we transform them with some appropriate kernel functions.

Meanwhile, we incorporate the observation that each event can follow different dynamics in the variation of its temporal distance, given different local conditions. The item browsed casually should have its influence to user preference drop sharply for a near term, but it might still be an important indicator of user's general interest in the long term. In contrast, if the user is seriously examining the item, it is very likely the user would be interested to visit the same or similar ones in a short period of time. Therefore, we create multiple temporal kernels to model the various temporal dynamics and leave it for the local surrounding environment to later decide contextualized temporal influence. This design allows more flexibility in weighting the influence of each event with different temporal distances.

In this thesis, we handpicked a collection of K kernel functions $\phi(\cdot) : \mathbb{R}^L \to \mathbb{R}^L$ with different shapes including:

1. exponential decay kernel, $\phi(\boldsymbol{T}) = ae^{-\boldsymbol{T}} + b$, assumes that the user's impression of an event fades exponentially but will never fade out.

2. logarithmic decay kernel, $\phi(\boldsymbol{T}) = -a\log(1 + \boldsymbol{T}) + b$, assumes that the user's impression of an event fades slower as time goes and becomes infinitesimal eventually. Later we will introduce a softmax function that will transform negative infinity to 0.

3. linear decay kernel, $\phi(\boldsymbol{T}) = -a\boldsymbol{T} + b$, assumes that the influence drops linearly and the later softmax operation will map the influence over some time limit to 0.

4. constant kernel, $\phi(\boldsymbol{T}) = \mathbf{1}$, assumes that the influence stays static.

where $a, b \in \mathbb{R}$ are the corresponding kernel parameters. Note that the above kernels are chosen only for their stability in gradient descent and well understood property in analysis. We have no assumption of which kernel is more suitable to reflect the actual temporal dynamics, and an ablation study of different combinations is presented in the following Section 2.6.3. This mechanism should be compatible with other types of kernel function $\phi(\cdot)$ by design, and it is also possible to inject prior knowledge of the problem to set fixed parameter kernels.

Hence, given a collection of K kernel functions, $\left\{\phi(\cdot)^1, \ldots, \phi(\cdot)^K\right\}$, we transform T into K sets of temporal importance scores: $\boldsymbol{\beta} = \left[\phi^1(T), \ldots, \phi^K(T)\right]$, for next stage's use.

$\boldsymbol{\gamma}$ stage, how did it happen: The goal of $\boldsymbol{\gamma}$ stage is to fuse the content and temporal influence based on the extracted local surrounding information. The core design follows the multiple sets of proposed temporal dynamics in the $\boldsymbol{\beta}$ stage, in which it learns the probability distribution over each temporal dynamics given the local conditions.

First, we explain our design to capture local information. In our setting, we consider the local information as two parts: sensitivity and seriousness. Specifically, if one event seems to be closely related to its future actions, it means the user is likely impressed by this event and his or her ongoing preference should be sensitive to the influence of this action. In contrast, if the event appears to be different from its past actions, the user is possibly not serious about this action, since his or her preference does not support it. Such factors of sensitivity and seriousness can be valuable for the model to determine the temporal dynamics that each particular event should follow. Review the example in Figure 2.1 again, the repetitive interactions with smartphones reflect high seriousness, while the sparse and possibly a noisy click on shoes suggests low sensitivity to its related products. This observation also motivates our design to model local information as its relation from past and to future events: we choose the Bidirectional RNN structure [81] to capture the surrounding events from both directions. From the input sequence embedding $\boldsymbol{X}$, we can compute the recurrent hidden state of every action as their local feature vector:

$$\boldsymbol{C} = \text{Bi-RNN}(X) \oplus C_{\text{attr}} \in \mathbb{R}^{L \times d_r}$$

where $\oplus$ is the concatenation operation. Here, we also introduce some optional local features C_{attr} that can be the attributes of each event in the specific recommendation applications, representing the scenario when the event happened. For instance, we can infer the user's seriousness or sensitivity from the interaction types (e.g., purchase or view) or the media (e.g., mobile or desktop) associated with the action. In our experiments, we only use the hidden states of bidirectional RNN's output as the local features, and we leave the exploration of task specific local features as our future work.

Second, the model needs to learn the mapping from the local features of event i to a weight vector of length K, where each entry $p_i(k|c_i)$ is the probability of this event follows $\phi^k(\cdot)$ as the temporal dynamics. We apply the feed-forward layer F^γ to map them into the probability space $\mathbb{R}^K$ and then normalize them into probabilities that sum up to one for each action with a softmax layer:

$$\boldsymbol{P}(\cdot|\boldsymbol{C}) = \text{softmax}\left(F^\gamma(\boldsymbol{C})\right)$$

Finally, we use the probability distribution to mix the temporal influence scores from the K different kernels for the contextualized temporal influence $\boldsymbol{\beta}^c = \boldsymbol{\beta} \cdot \boldsymbol{P}(\cdot|\boldsymbol{C})$, with which we use element-wise product to reweight the content-based importance score for the contextualized temporal attention score:

$$\boldsymbol{\gamma} = \text{softmax}\left(\boldsymbol{\alpha}\boldsymbol{\beta}^c\right)$$

This design choice that uses product instead of addition to fuse the content and contextualized temporal influence score $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ is based on the consideration of their influence on the gradients of $\theta^{\boldsymbol{\alpha}}$. For example, the gradient on parameters in $\boldsymbol{\alpha}$ stage is,

$$\frac{\partial(\boldsymbol{\alpha}\boldsymbol{\beta}^c)}{\partial\theta^{\boldsymbol{\alpha}}} = \frac{\partial\boldsymbol{\alpha}}{\partial\theta^{\boldsymbol{\alpha}}}\boldsymbol{\beta}^c, \quad \frac{\partial(\boldsymbol{\alpha}+\boldsymbol{\beta}^c)}{\partial\theta^{\boldsymbol{\alpha}}} = \frac{\partial\boldsymbol{\alpha}}{\partial\theta^{\boldsymbol{\alpha}}}$$

The error gradient in the addition form is independent of the function evaluation of $\boldsymbol{\beta}^c$, while the product form has the gradients of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}^c$ depend on each other. Therefore, we choose the product form as a better fusion of the two scores.

2.5.3 Parameter Learning

Loss Functions In the previous section, we showed how the model makes recommendations by the highest similarity scores $\{r_v\}$ for all $v \in \mathcal{V}$. When training the model, we only use a subset of $\{r_v\}$. That is, since the size of the item space can be very large, we apply negative sampling [82], i.e., proportional to their popularity in the item corpus, sample a subset of items $N_S \subseteq \mathcal{V}$, that excludes the target item i, i.e., $i \notin N_S$.

We adopt negative log-likelihood (NLL) as the loss function for model estimation:

$$L_{\text{NLL}} = -\log \frac{e^{r_i}}{\sum_{j \in N_S} e^{r_j}},$$

which maximizes the likelihood of target item.

We also consider two ranking-based metrics to directly optimize the quality of recommendation list. The first metric is the Bayesian Personalized Ranking (BPR) loss [83] [3]:

$$L_{\mathrm{BPR}} = -\frac{1}{N_S} \sum_{j \in N_S} \log \sigma \left(r_i - r_j \right),$$

which is designed to maximize the log likelihood of the target similarity score r_i exceeding the other negative samples' score r_j.

The second is the TOP1 Loss [84]:

$$L_{\mathrm{TOP1}} = \frac{1}{N_S} \sum_{j \in N_S} \sigma \left(r_j - r_i \right) + \sigma \left(r_j^2 \right),$$

which heuristically puts together one part that aims to push the target similarity score r_i above the score r_j of the negative samples, and the other part that lowers the score of negative samples towards zero, acting as a regularizer that additionally penalizes high scores on the negative examples.

Regularization We introduce regularization through the dropout mechanism [85] in the neural network. In our implementation, we have dropout layer after each feed-forward layer and the output layer of the bidirection RNN with a dropout rate of 0.2. We leave as out future work to explore the effect of batch normalization as well as regularization techniques of the parameters in temporal kernels.

Hyperparameter Tuning We initialize the model parameters through the Kaiming initialization proposed by he2015delving. The temporal kernel parameters are initialized in proper range (e.g. uniform random in $[0, 1]$) in order to prevent numerical instability during training. We use the Relu function [86] by default as the activation function in the feed-forward layer.

2.6 Experiments

In this section, we perform extensive experiment evaluations of our proposed sequential recommendation solutions. We compared them with an extensive set of baselines, ranging from session-based models to temporal and sequential models, on two very large collections of online user behavior log data. We will start from the description of experiment setup and baselines, and then move onto the detailed experiment results and analysis.

[3]We use the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$.

Table 2.1: Statistics of two evaluation datasets.

Dataset	XING	Taobao
Users	64,890	68,216
Items	20,662	96,438
Actions	1,438,096	4,769,051
Actions per user	22.16±21.25	69.91±48.98
Actions per item	69.60±112.63	49.45±65.31
Time span	80 days	9 days

2.6.1 Dataset

We use two public datasets known as XING and Taobao. The statistics of the datasets are summarized in Table 3.1. The two datasets include user behaviors from two different application scenarios.

The XING dataset is extracted from the Recsys Challenge 2016 dataset [87], which contains a set of user actions on job postings from a professional social network site [4]. Each action is associated with the user ID, item ID, action timestamp and interaction type (click, bookmark, delete, etc.). Following the prior work [11,25], we removed interactions with type "delete" and did not consider the interaction types in the data. We removed items associated with less than 50 actions, and removed users with less than 10 or more than 1000 actions. We also removed the interactions of the same item and action type with less than 10 seconds dwell time.

The Taobao dataset [88] is provided by Alibaba and contains user interactions on commercial products from an e-commerce website[5]. Each action is associated with the user ID, item ID, action timestamp and interaction type (click, favor, purchase, etc.). In order to have a computationally tractable deep learning model, we randomly sub-sampled 100,000 users' sequences from each dataset for our experiment. We removed items associated with less than 20 actions, and then removed users with less than 20 or more than 300 actions. We also removed the interactions with timestamp that is outside the 9 day range that dataset specifies.

2.6.2 Experiments on LSHP

Baselines for comparison with LSHP. We compare LSHP with the following baselines on predicting the next item.

• **Global Popularity (Pop).** Rank items according to their frequency in the training dataset in a descending order.

• **Sequence Popularity (S-Pop).** Rank items according to their frequency in the target sequence.

[4]https://www.xing.com/
[5]https://www.taobao.com/

Table 2.2: Performance comparison of LSHP against baselines on sequential recommendation.

Dataset	Metric	**LSHP**	Pop	S-Pop	standardHP	sparseHP	sessionHP	RTPP
XING	Recall@5	**0.2173**	0.0118	0.2059	0.1201	0.1417	0.1338	0.1873
	MRR@5	**0.1454**	0.0062	0.1202	0.2008	0.2392	0.1454	0.2254
Taobao	Recall@5	**0.1201**	0.0026	0.1093	0.0762	0.0778	0.0769	0.1021
	MRR@5	**0.0792**	0.0013	0.0639	0.0431	0.0492	0.0489	0.0668

The frequency of items is updated as more observations in the sequence become available. We use global popularity to break the tie.

• **Standard Multi-dimension Hawkes Process (standardHP).** Following [38,69], the intensity function is defined as $\lambda^*_{v_i}(t_i) = \mu_{v_i} + \sum_{t_l<t_i} A_{v_l v_i}\kappa(t_i - t_l)$, where μ_{v_i} is the base intensity of the item v_i. This model assumes the next action depends on every historical action including actions within the same session and actions across sessions.

• **Sparse Hawkes Process (sparseHP).** This is an extension of standard Multi-dimension Hawkes Process. The matrix A capturing mutual influence is constrained to be sparse through adding a $L1$ regularization of A to the objective function.

• **Session-based Hawkes Process (sessionHP).** To verify whether considering actions across session is beneficial to predict the next item, we developed this variant of LSHP as a baseline. The intensity function is defined as $\lambda^*_{v_i}(t_i) = \mu_{v_i} + \sum_{t_l<t_i} A_{v_l v_i}\kappa(t_i - t_l)\delta(S_l = S_i)$, where the indicator function $\delta(S_l = S_i)$ excludes influence of actions which do not belong to the same session. The matrix A is to capture the mutual influence within a session as that in LSHP. As LSHP imposes sparsity structure to matrix A, we keep the same constraint in this model.

• **Recurrent Temporal Point Process (RTPP).** In [46], the authors proposed a recurrent marked temporal point process which utilized RNN to predict the item and Hawkes process to predict the occurring time. Details can be found in [46].

Experiment settings. We split all the data by user, and select 80% of the users to train the model, 10% as the validation set and the remaining 10% users to test the model. We also adopt the warm start recommendation setting, where the model is evaluated after observing at least 5 historical actions in each testing user.

Evaluation metrics. The model predicts the user action at the time of the next observed action. The result is evaluated by ranking the ground-truth action against a pool of candidate actions. For both datasets, the candidate pool is the set of all items in the dataset, though only a subset of negative items is sampled for model optimization.

Table 2.3: Performance comparison of CTA against different baselines on sequential recommendation.

Dataset	Metric	**CTA**	GRU4Rec	HRNN	LSHP	SASRec	M3R
XING	Recall@5	**0.3217**	0.2690	0.2892	0.2173	0.2530	0.2781
	MRR@5	0.1849	0.2008	0.2392	0.1454	0.2254	**0.2469**
Taobao	Recall@5	**0.1611**	0.0936	0.0940	0.1201	0.1418	0.1077
	MRR@5	**0.0925**	0.0619	0.0610	0.0792	0.0863	0.0689

We rank the candidates by their predicted probabilities and compute the following evaluation metrics:

- **Recall@K.** It reports the percentage of times that the ground-truth relevant item and ranked within the top K list of retrieved items.

- **MRR@K.** The mean reciprocal rank is used to evaluate the prediction quality from the predicted ranking of relevant items. It is defined as the average reciprocal rank for ground-truth relevant items among the top K list of retrieved items. If the rank is larger than K, the reciprocal rank is 0.

Experimental results. We report the results in Table 2.2. From the results, we can observe that LSHP outperforms all other Hawkes Process based models. Without modeling the self-influence across sessions, sessionHP could not capture the influence from past repetitive actions to the next action outside its current session. Lack of the Long-term dependence, sessionHP performs worse than LSHP. Although S-Pop considers dependence between the next action and past repetitive actions cross sessions, ignoring the dependence among actions of different items within a session leads to its worse performance. StandardHP and sparseHP do not differentiate temporal dependence within nor across sessions, and use a universal time decaying function to model the temporal influence, which leads to their less accurate modeling of dependency. Consequently, they performs worse in predicting the next item. As Pop does not consider the dependence of the next action on previous actions, its performance are much worse than all other algorithms. RTPP does not differentiate the dependence between actions of the same item and those of different items. So it does not perform as well as LSHP. In conclusion, LSHP benefits from both long-term mutual influence and short-term self-influence among actions and thus predicts the next item more accurately.

2.6.3 Experiments on CTA

Baseline methods We compare Contextualized Temporal Attention Mechanism with a variety of neural network based methods and LSHP [6]. To ensure a fair comparison of deep learning model,

[6]All implementations are open sourced at `https://github.com/Charleo85/seqrec`

we adjust the number of layers and hidden units such that all the models have similar number of trainable parameters. **Session-based Models.** We include several deep learning based models with session assumptions. We set the session cut-off threshold as 30 minutes by convention.

]**Session based Recurrent Neural network (GRU4Rec).** In [10], authors used the GRU, a variant of Recurrent Neural network, to model the user preference transition in each session. The session assumption is shown to be beneficial for a consistent transition pattern. **Hierarchical Recurrent Neural network (HRNN).** In [11], authors proposed a hierarchical structure that use one GRU to model the user preference transition in each session and another to model the transition across the sessions.

Temporal Models. Since our model additionally uses the temporal information to make the sequential recommendation, we include the following baselines that explicitly consider temporal factors and have been applied in sequential recommendation tasks.

- **Long- and Short-term Hawkes Process (LSHP).** In [30], authors proposed a Long- and Short-term Hawkes Process that uses a uni-dimension Hawkes process to model transition patterns across sessions and a multi-dimension Hawkes process to model transition patterns within a session.

Sequential Models. Similar to our proposed CTA model, we also include several deep learning based models that directly learn the transition pattern in the entire user sequence. A fixed size window is selected for better performance and more memory-efficient implementation.

- **Self-attentive Sequential Recommendation (SASRec).** In [27], the authors applied the self-attention based model on sequential recommendation. It uses the last encoder's layer hidden state for the last input to predict the next item for user. We use 4 self-attention blocks and 2 attention heads with hidden size 500 and position embedding. We set the input window size to 8.

- **Multi-temporal-range Mixture Model (M3R).** In [31], a mixture neural model is used to encode the users' actions from different temporal ranges. It uses the item co-occurrence as tiny-range encoder, RNN/CNN as short-range encoder and attention model as long-range encoder. Following the choice in its original paper, we use GRU with hidden size 500 as the short-range encoder.

Implementation Details For CTA, we use self-attention blocks $d_l = 2$ and attention heads $d_h = 2$ with hidden size $d_a = 500$. We use the same representation for input and output item embeddings $E_{\text{in}} = E_{\text{output}}$, and a combination of 5 exponential decay kernels (ψ^5). We use a bidirection RNN with hidden size $d_r = 20$ in total of both directions to extract context features. We set the learning rate as 0.001. We will present the experiments on different settings of our model in the following section.

Experiment settings. We split all the data by user, and select 80% of the users to train the model, 10% as the validation set and the remaining 10% users to test the model. We also adopt the warm start recommendation setting, where the model is evaluated after observing at least 5 historical actions in each testing user.

All the deep learning based models are trained with Adam optimizer with momentum 0.1. We also search for a reasonablely good learning rate in the set $\{0.01, 0.001, 0.0001, 0.00001\}$ and report the one that yields the best results. We set batch size to 100, and set the size of negative samples to 100. The model uses the TOP1 loss by default. The item embedding is trained along with the model, and we use the embedding size 500 for all deep learning models. The training is stopped when the validation error plateaus. For the self-attention based model, we follow the training convention [88] by warming up the model in the first few epoches with small a learning rate.

Evaluation metrics. We adopted the same metrics as those evaluating LSHP, i.e., Recall@K and MRR@K.

Overall Performance. We summarize the performance of the proposed model against all baseline models on both dataset in Table 2.3. The best solution is highlighted in bold face.

Notably, on both datasets, our proposed model CTA outperforms all baselines in Recall@5 by a large margin (11.24% on XING dataset, 14.18% on Taobao dataset). The model's MRR@5 performance is strong on Taobao dataset, but weak on XING dataset. This suggests that our model fails to learn a good ranking for the first order transition pattern, since it uses a weighted sum of input sequence for prediction. Nevertheless, such weighted sum design is powerful to capture the sequential popularity pattern. It also shows that our model outperforms the self-attentive baselines, which suggests our design of the contextual temporal influence reweighing, i.e., $\boldsymbol{\alpha}\boldsymbol{\beta}^c$, improves sequential order modeling in recommendation applications, compared to the positional embedding borrowed from natural language modeling.

Results on XING dataset. The RNN-based methods outperformed both temporal models and attention-based models. This again confirms that the recurrent model is good at capturing the first order transition pattern or the near term information. We also observe that the hierarchical RNN structure outperforms the first order baseline, while the session-based RNN performs not as well as this strong heuristic baseline. This demonstrates the advantage of hierarchical structure and reinforces our motivation to segment user history for modeling users' sequential behaviors.

Results on Taobao dataset. On the contrary to the observations on XING dataset, the temporal models and attention-based models outperformed RNN-based methods. This means the recurrent structure is weak at learning the sequential popularity pattern, while the attention-based approach is able to effectively capture such long-term dependence. Such conflicting nature of existing baselines is exactly one of the concerns this work attempts to address. This again validates our design to evaluate and capture the long- and short-term dependence through the proposed three stage weighing pipeline.

Ablation Study. We perform ablation experiments over a number of key components of our model in order to better understand their impacts. Table 2.4 shows the results of our model's default setting and its variants on both datasets, and we analyze their effect respectively:

- **Window size.** We found that the window size of 8 appears to be the best setting among other choices of input window size among $\{4, 16, 32\}$ on both datasets. The exceptions are a smaller window size on XING and a larger window size on Taobao can slightly improve MRR@5, even though Recall@5 still drops. The reason might be suggested by the previous observation that the first order transition pattern dominates XING dataset so that it favors a smaller input window, while the sequence popularity pattern is strong in Taobao dataset such that it favors a larger input window size.

- **Loss functions.** The choice of loss function also affects our model's performance. The ranking based loss function, BPR and TOP1, is consistently better than the NLL loss, which only maximizes the likelihood of target items. The TOP1 loss function with an extra regularizer on the absolute score of negative samples can effectively improve the model performance and reduce the over-fitting observed in the other two loss functions.

- **Self-Attention settings.** We compare the model performance on different d_l and d_h settings. The performance difference is minimal on XING, but relatively obvious on Taobao. This

Table 2.4: Ablation analysis on two datasets under metrics of Recall@5 (left) and MRR@5 (right). The best performance is highlighted in bold face. ↓ and ↑ denote a drop/increase of performance for more than 5%. ψ, ρ, π, ω respectively denote the exponential, logarithmic, linear and constant temporal kernels. The superscript on the kernel function denotes the number of such kernel used in the model.

Architecture		Dataset			
		XING		Taobao	
Base		0.3216	0.1847	0.1611	0.0925
Window size (L)	4	0.3115↓	0.2167↑	0.1488↓	0.0899
	16	0.3049↓	0.1733↓	0.1433↓	0.0914
	32	0.3052↓	0.1735↓	0.1401↓	0.0950
Attention blocks (d_l)	1	0.3220	0.1851	0.1631	0.0926
	4	0.3217	0.1849	0.1631	0.0924
Attention heads (d_h)	1	0.3225	0.1860	0.1622	0.0919
	4	0.3225	0.1860	0.1646	0.0940
¬ Sharing embedding		0.1263↓	0.0791↓	0.1042↓	0.0192↓
Embedding size (d_{in})	300	0.3147	0.1831	0.1622	0.0920
	1000	0.3207	0.1857	0.1628	0.0921
Loss function	NNL	0.3130	0.1806	0.1571	0.0895
	BPR	0.3163	0.1804	0.1598	0.0913
Flat attention		0.3215	0.1869	0.1588	0.0907
Global context $\boldsymbol{P}(\cdot)$		0.3207	0.1839	0.1603	0.0912
Local context $\boldsymbol{P}(\cdot\|x)$		0.3210	0.1841	0.1591	0.0912
Kernel types	ω^1	0.3191	0.1827	0.1604	0.0910
	ψ^1	0.3122	0.2141↑	0.1591	0.0907
	ψ^{10}	0.3207	0.1844	0.1627	0.0925
	π^1	0.2917↓	**0.2323**↑	0.1562	0.0976
	π^5	0.3025↓	0.2209↑	0.1670	**0.1010**↑
	π^{10}	0.3214	0.2183↑	**0.1673**	0.0997↑
	ρ^5	0.3111	0.2196↑	0.1618	0.0931
	ρ^{10}	0.3230	0.1869	0.1635	0.0932
	ψ^5, ρ^5	0.3241	0.1888	0.1635	0.0932
	ψ^5, π^5	**0.3273**	0.2146↑	**0.1673**	0.0997↑
	ψ^5, ρ^5, π^5	0.3254	0.1971↑	0.1664	0.0983↑

indicates the content-based importance score is more important in capturing the sequential popularity than first order transition pattern.

- **Item embedding.** We test the model with separate embedding space for input and output sequence representations; and the model performance drops by a large margin. Prior work, e.g., [27], in sequential recommendation found similar observations. Even though the separate embedding space is a popular choice in neural language models [**?**], but the item corpus appears to be more sparse to afford two distinct embedding spaces. The dimensionality of the embedding space, $d_{\text{in}} = d_{\text{out}}$ slightly affects the model performance on both datasets, and it at the same time increases Recall@5 and decreases MRR@5 score, and vice versa. A trade-off on ranking and coverage exists between larger and smaller embedding spaces.

Discussion on Model Architecture To further analyze the strength and weakness of our model design, we conduct experiments specifically to answer the following questions:

- **Does the model capture the content influence $\boldsymbol{\alpha}$?** To understand if our model is able to learn a meaningful α, we replace the M^α component with a flat attention module, such that it always outputs $\boldsymbol{\alpha} = \mathbf{1}$. And we list this model's performance in Table 2.4 as "Flat Attention".

  The performance stays almost the same on XING, but drops slightly on the Taobao dataset. It shows that the content-based importance score is less important for the sequential recommendation tasks when the first order transition pattern dominates, but is beneficial for the sequential popularity based patterns. It also suggests that contextualized temporal importance along is already a strong indicator of historical actions about current user preference.

- **Does the model extract the context information in $\boldsymbol{\gamma}$ stage?** As the effect of temporal influence depends on our context modeling component, we design the following experiments on the context component to understand the two follow-up questions.

  First, *whether the local context information of each event is captured.* We replace the local conditional probability vector $\boldsymbol{P}(\cdot|\boldsymbol{C})$ with a global probability vector $\boldsymbol{P}(\cdot)$, i.e., a single weight vector learnt on all contexts. This model's performance is listed in the table as 'Global Context'. We can observe a consistent drop in performance in both datasets.

  Second, *whether the local context is conditioned on its nearby events.* We replace the local conditional probability vector $\boldsymbol{P}(\cdot|\boldsymbol{C})$ with a local probability vector conditioned only on the event itself, $\boldsymbol{P}(\cdot|x)$. More specifically, instead of using the bidirectional RNN component, the model now uses a feed-forward layer to map each x_i to the probability space $\mathbb{R}^K$. This model's performance is listed in the table as 'Local Context'. We again observe a consistent drop in performance, though it is slightly better than the global context setting.

  As a conclusion, our model is able to extract the contextual information and its mapping into probability for different temporal influences on both datasets.

- **Does the model capture temporal influence $\boldsymbol{\beta}$?** We conduct multiple experiments on the number of temporal kernels and the combined effect of different kernel types.

  Firstly, we want to understand the advantages and limitations of each kernel type. We look at the model performance carried out with a single constant temporal kernel ω^1. Its performance on MRR@5 is the worst among all the other kernel settings on both datasets. At the same time, we compare the settings of 10 exponential ψ^{10}, logarithmic ρ^{10} and linear π^{10} kernels each. The 10 linear kernels setting is overall the best on both datasets, especially in improving the ranking-based metrics. It shows that it is beneficial to model the temporal influence with the actual time intervals transformed by appropriate kernel functions.

Secondly, we compare the model performance on different number of temporal kernels. The results suggested that the model performance always improves from using a single kernel to multiple kernels. This directly supports our multi-kernel design. Specifically, among the exponential kernels $\{\psi^1, \psi^5, \psi^{10}\}$, ψ^5 performs the best on XING, yet not as good as ψ^{10} on Taobao. On linear kernels $\{\pi^1, \pi^5, \pi^{10}\}$, as the kernel number increases, Recall@5 improves, but MRR@5 drops on XING. Similarly on Taobao, π^5 achieves the best ranking performance, but the π^{10} induces a better coverage. Hence, the model with more kernels does not necessarily perform better, and as a conclusion, we need to carefully tune the number of kernels for better performance on different tasks.

Thirdly, we study the combinatorial effect of different kernel types: (ψ^5, ρ^5), (ψ^5, π^5) and (ψ^5, ρ^5, π^5). We can observe that all types of kernel combinations we experimented improve the performance on both datasets, compared to the base setting ψ^5. This suggests the diversity of kernel types is beneficial to capture a better contextualized temporal influence. However, it also shows on both datasets that if mixing exponential ψ^5 with either linear π^5 or logarithmic ρ^5 kernel can improve the model performance, mixing all three of them together would only worsen the performance. We hypothesize that certain interference exists among the kernel types so that their performance improvement cannot simply add on each other. And we leave the exploration of finding the best combination of kernels as our future work.

Overall, we believe that the temporal influence can be captured by current model design, and there are opportunities left to improve the effectiveness and consistency of the current kernel based design.

Attention Visualization. To examine the model's behavior, we visualize how the importance score shifts in some actual examples in Figure 2.3. The x-axis is a series of actions with their associated items [7] and the time interval from action time to current prediction time, $(s_i, t_{L+1} - t_i)$. From left

[7] for privacy concerns, these datasets do not provide the actual item content; and we represent the items in the figure with symbols.

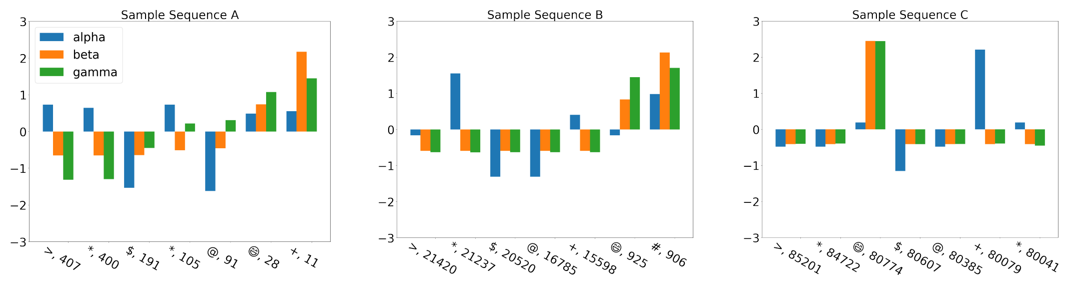

Figure 2.3: Attention visualization. The blue (left) bar is the content-based importance score $\boldsymbol{\alpha}$, the orange (middle) bar is the contextualized temporal influence score $\boldsymbol{\beta}^c$, the green (right) bar is the combined importance score $\boldsymbol{\gamma}$. The figures contains three different sequences selected from the test set of the Taobao dataset.

to right, it follows a chronological order from distant to recent history. We select the example such that the ground-truth next item is among the historical actions for the sake of simplicity, and we use smile face symbol ☺ to denote if the item of such historical action is the same as the target item. Each action on the x-axis is associated with three bars. Their values on the y-axis is presented as the computed score $\boldsymbol{\alpha}$, $\boldsymbol{\beta}^c$ and $\boldsymbol{\gamma}$ respectively of each event in the model after normalization (by z-score). The model setting uses the temporal kernel combination (ψ^5, π^5) for its best performance.

- **Orange bars.** The contextualized temporal influence score $\boldsymbol{\beta}^c$, in both sequence A and B, follows the chronological order, i.e., the score increases as time interval shortens. In addition, such variation is not linear over time: the most recent one or two actions tend to have higher scores, while the distant actions tend to have similar lower scores. The sequence C, as all actions happened long time ago from current prediction, the context factor is deciding the height of orange bar. And the model is able to extract the context condition and assign high temporal importance to this event, which is indeed the target item. These observations all suggest that the contextualized temporal influence is captured in a non-trivial way that helps our model to better determine the relative event importance.

- **Blue bars.** For the content-based importance score $\boldsymbol{\alpha}$, it shows different distribution on each of the sequences. This is expected as we want to model the importance on the event correlation that is independent of the sequence order. Only in the third example that the target, i.e., the most relevant historical action, is ranked above average according to the content-based importance score. This again shows the important role of the temporal order to improve the ranking quality for sequential recommendation.

- **Green bars.** The combined score γ largely follows the relative importance ranking in orange bar. In other words, the contextualized temporal order is the dominating factor to determine relative importance of each input in our selected examples. This corresponds to the previous observation that the model performance would only slightly drop if the self-attention component outputs flat scores. This supports our motivation to model the contextualized temporal order in sequential recommendation tasks.

Although these are only three example interaction sequences from more than $6,000$ users, we can now at least have a more intuitive understanding of the reweighing behavior of our model design – the core part that helps boost the recommendation performance over the existing baselines. However, there are also many cases where the importance scores are still hard to interpret, especially if there is no obvious correspondence between target item and the historical actions. We need to develop better techniques to visualize and analyze the importance score for interpretable neural recommender system as follow-up research.

2.7 Conclusion

In this chapter, we identify and address the critical problem in sequential recommendation, *Déjà vu*, that is the user interest based on the historical events varies over time and under different context. First, rooted in concepts of episodic memory and semantic memory in cognitive psychology, we proposed Long- and Shot-term Hawkes Process to model users' sequential interactive behavior. To capture the contextual dependence depicted in episodic memory, LSHP employs a multi-dimensional Hawkes process to model influences among actions occurring in the same session. And to realize the memory recall described in semantic memory, LSHP utilizes a one-dimensional Hawkes process to model influences among actions of the same type happening in different sessions. In this way, the long-term and short-term dependence are explicitly captured by LSHP as a mixture of stochastic processes. By adopting ADMM algorithm, we maximize the data likelihood to learn the parameters of LSHP. Extensive experiment comparisons between LSHP and several other state-of-the-art baselines prove modeling the sequential structure among users' actions in regard to occurring time of actions helps LSHP improve recommendation quality.

Second, we propose a Contextualized Temporal Attention Mechanism that learns to weigh historical actions' influence on not only what action it is, but also when and how the action took place. More specifically, to dynamically calibrate the relative input dependence from the self-attention mechanism, we deploy multiple parameterized kernel functions to learn various temporal dynamics, and then use the local information about sensitivity and seriousness to determine which of these reweighing kernels to follow for each input. Our empirical evaluations show that the proposed model, CTA, has the following advantages:

- **Efficacy & Efficiency.** Compared with the baseline work, CTA effectively improves the recommendation quality by modeling the contextualized temporal information. It also inherits the advantage of self-attention mechanism for its reduced parameters and computational efficiency, as the model can also be deployed in parallel.

- **Interpretability.** Our model, featuring the three stage weighing mechanism, shows promising traits of interpretability. From the elementary analysis demonstrated in our experiments, we can have a reasonable understanding on why an item is recommended, e.g., for its correlation with some historical actions and how much on temporal influence or under context condition.

- **Customizability.** The model design is flexible in many parts. In the $\boldsymbol{\alpha}$ stage, the model can extract the content-based importance by all means, such as the sequence popularity heuristics – customizable for recommendation applications with different sequential patterns. In the $\boldsymbol{\beta}$ stage, as we mentioned earlier, we can adapt different choices of temporal kernels to encode prior knowledge of the recommendation task. The $\boldsymbol{\gamma}$ stage is designed to incorporate extra context information from the dataset, and one can also use more sophisticated neural structures to capture the local context given the surrounding events.

Our work opens several important future directions. Our understandings are still limited in the temporal kernels including what choices are likely to be optimal for certain tasks, and how we can regularize the kernel for more consistent performance. Our current solution ignores an important factor in recommendation: the user, as we assumed everything about the user has been recorded in the historical actions preceding the recommendation. As our future work, we plan to explicitly model user in our solution, and incorporate the relation among users, e.g., collaborative learning, to further exploit the information available for sequential recommendation.

Chapter 3

Category-aware Contextualized Recommendations

Recommender systems match users under different context with items that they would be interested in. Understanding users' preferences under certain context is essential to recommender systems. Item category, a type of proxy of context, has shown to carry critical signals about users' preferences [89,90]. Take online shopping scenario as an example, the item category, like "shoes", reveals users' preferences are narrowed down to shoes. In this chapter, we study the problem of making recommendations with the item category as context.

3.1 Introduction

Modeling user preferences of the next action based on her past actions is essential to recommender systems. Concerning the sequential dependence among actions, predicting the next action for users based on their interaction history has been framed as sequential recommendation. This has been considered as a sequential prediction problem, and considering the sequential structure among actions is the key. Various sequence models borrowed from other fields have been explored [10,12,34,55,57,91, 92]. From the earliest Markov models [55] to recent neural sequence models, such as Recurrent Neural Network (RNN) or self-attention [10, 14], models with a stronger capacity in capturing complex and high-order sequential dependence among actions have shown to achieve better recommendation quality.

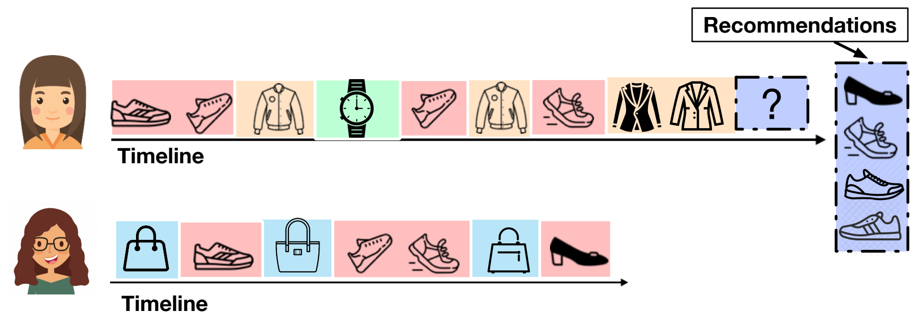

Figure 3.1: An illustration of collaborative sequential recommendation. Each user's actions are indexed chronologically. The recommender system needs to predict which items to recommend to the user Lily based on her and another user Ivy's past actions.

However, most of existing solutions treat a user's action history as a long sequence [10,29,93]. Such simplification usually ignores the fine-grained sequential structure in the action sequence. Consider the example illustrated in Figure 3.1. For the user Lily, the reoccurred transitions from clothes to shoes suggest her next action is very likely to be related to shoes, and the series of her previously browsed shoes suggest her general preference on sports shoes. But her recently browsed business suits suggest her current intent in formal outfits. As a result, it is no longer appropriate for the system to follow her general preference to recommend sports shoes; instead, recommending formal shoes becomes a better choice. Such observation informs us that sequential recommendation should be context-aware: under different contexts, the prediction of next action should depend on different subsequences of past actions.

When considering the sequential structure, another challenge is data sparsity. Observations about individual user's actions are known to be sparse [94], not to mention the transitional patterns that could be covered in a single user's action sequence. For example, in the example shown in Figure 3.1, the user Lily has never visited any formal shoes in the past. Hence, even with the knowledge that formal shoes should be recommended next, it is still clueless for the system to predict which specific type/brand to start with.

As a remedy to the data sparsity issue, collaborative learning methods have been recently introduced to sequential recommendation [6,61,95,96]. The basic idea is to exploit users with similar past action sequences for the next action prediction. As a typical solution of this type, in [97], authors modeled users' action sequences with RNNs and retrieved neighboring users based on the user latent states learnt by RNNs. Then the target user's representation is combined with the retrieved neighboring users' representations for the next item prediction. However, user similarity is still measured by the entire sequence of past actions in this type of solutions. As we argued before, neglecting the context in sequential recommendation introduces inaccurate dependency on the past actions, and therefore erroneous neighborhood for next action prediction. Consider the example in Figure 3.1 again. When looking at their entire action sequences, Lily and Ivy might not be considered as neighbors, as Lily visited mostly shoes and clothes while Ivy visited mostly shoes and handbags. But the subsequences of shoes browsed by these two users make them closer. Especially the transitions from sport shoes to formal shoes in Ivy's subsequence will be very helpful in predicting Lily's next action about formal shoes. Therefore, the neighborhood modeling in sequential recommendation should also be context-aware.

Nevertheless, the context, under which a user takes the next action, is not observable by the system [3]. We have to look for proxies of it. We believe a good proxy of context should: 1) be widely available in sequential recommendation problems, and 2) enhance the modeling of dependence on past actions. In this chapter, we consider the category of the next item, which is a type of widely available metadata about items and also provides context information [89,90]. Our statistical tests on two large public recommendation datasets prove the transitional patterns among actions in the category-specific subsequences are significantly stronger than those in the original action sequences without considering the categories. We defer the details of our statistical tests to Section 3.4 and the description of the dataset to Section 3.5.

Based on our insight discussed above, we propose a CategOry-aware COllaborative sequential Recommender (CoCoRec), which draws dependence of the next action on historical actions based on the target user's action sequence, item categories and neighboring users' action sequences. CoCoRec is composed of three key components: an in-category encoder, a context encoder, and a collaboration module. The in-category encoder utilizes self-attention to model item transition patterns in category-specific action subsequences. The context encoder infers the category context. It uses recent actions' categories to predict the category of the next action, and then based on this prediction, it uses a gating network to activate the corresponding in-category item-to-item transitions. Since the recently engaged items can suggest recent preferences, we also model them in the context encoder. It uses self-attention to model the user's most recent actions to obtain recent preferences. The collaboration module uses a memory tensor to record users' in-category preferences. For each target user, the collaboration module retrieves neighboring users with similar in-category preferences. Combining the recent preferences of target user, the in-category preferences of target user, and the neighbors' in-category preferences, CoCoRec predicts the next item for the target user.

To investigate the effectiveness of CoCoRec for sequential recommendation, we performed extensive experiments on two large public recommendation datasets. Compared with a list of state-of-the-art solutions for sequential recommendation, CoCoRec improved the recommendation quality in both recall and MRR. Our ablation analysis further demonstrated the importance of modeling the in-category user preferences and collaborative learning among users with similar in-category preferences for CoCoRec to achieve high quality recommendation performance.

3.2 Related Work

The improvement of sequence models' capacities in capturing complex and high-order sequential patterns has unleashed the development in sequential recommendation. Fixed- and varying-order Markov models are among the earliest attempts [34,55], which assume the prediction only depends on the recent several actions. Factorizing Personalized Markov Chains [55] is a typical model of this kind, which combines factorization machine [56] with a Markov model. It improves over vanilla matrix factorization by introducing sequential order among historical actions into factorization. However, the Markovian assumption also limits the performance of such models: as the state-space grows exponentially with respect to the order of dependence, this type of solution can hardly capture high order dependence in practice.

Neural sequence models, such as RNN, Long Short-Term Memory (LSTM) [12], Gated Recurrent Unit (GRU) [13] and self-attention [14] models, have been adopted to address the limitations in Markov models [10,12,29,57,91,92]. For example, in [10], the authors applied RNNs to predict the next action based on actions in a session of which the boundary is defined in regard to the idle duration between two consecutive actions. In [57], the authors used attention to model influence from the most related actions in the session. SASRec [27] and BERT4Rec [33] extended the scope of self-attention models to sequential recommendations.

Besides vanilla application of existing neural sequence models, problem-specific customizations are proposed to enhance the modeling of action sequences. In [31], the authors has developed a solution composed of a mixture of sequence models to capture both long-term and short-term action dependence. Hierarchical neural network models have been applied to model users' sequential preferences across different sessions [11,25,98].

Nevertheless, in [99], the authors found that the session assumption could be the bottleneck of these models, as the influence from past actions does not necessarily differ with respect to the manually defined session boundaries. In contrast, our fine-grained action dependency modeling is supported by statistical tests, i.e., segmenting action sequences into sub-sequences with respect to item categories enhances sequential dependency modeling.

Collaborative learning has been introduced to sequential recommendation to address the data sparsity issue. Based on the social network among users, in [100], the authors combined the transition patterns of neighbors in the social network into the next item prediction of the target user. Lifting the requirements of pre-existing social networks, in [94], the authors measured user relatedness by the degree of item overlap among action sequences. Actions from the k-nearest neighbors are introduced to the target user's prediction. As a follow-up, in [97], the authors measured user similarities based on the latent states learnt for users by RNNs; and directly combined neighboring users' latent states with target user's latent state for the next item prediction. To reduce the search space of neighbors, in [96], the authors made the initial next item predictions based on the target user's action sequence, and then utilized the initial item predictions to filter irrelevant users. All the aforementioned methods used entire action sequences to measure user similarity. However, under different context, the importance of past actions in representing users is different. Failing to characterize target user's ongoing context when retrieving neighbors prevents collaborative learning from helping the next item prediction of the target user.

Another line of work on sequential recommendation utilized item categories as proxies of action context and incorporated the context into various models to improve the modeling of action sequences. Treating the prediction of the next item category as an extra task along with the prediction of the next item, a multi-task learning based solution has been developed to predict both the next item and its category [101]. Treating the item category as a condition, a generative adversarial network based solution validates the next item prediction based on the category of the predicted item [102]. However, the in-category sequential transition patterns are ignored by these solutions.

Besides item category, other types of external knowledge about items have also been utilized for sequential recommendation, like knowledge bases [103,104] and category taxonomies [90,105]. The relations among items defined in knowledge base are utilized to capture the dependence among actions [103,104]. In [90], the authors incorporated multi-hop categories to memory network to structure the dependency. Likewise, in [105], the authors proposed to hierarchically model actions based on the hierarchy of item category. However, the limited availability of knowledge bases or category taxonomies in different recommendation scenarios directly restricts the application of these solutions in practice.

3.3 Notations and Problem Setup

In this section, we introduce the notations used in this chapter and define the problem we address in this chapter. We study sequential recommendations for a set of users $u \in U$ over a set of items $v \in V$ from a set of item categories $c \in C$. We denote an action as a tuple $a_i = (v_i, c_i)$, where i is the index of the action in a sequence, v_i is the item that the user interacts with and c_i is the item's category. Different actions may be associated with the same item and each item is associated with a unique category. A sequence of N actions from user u is denoted as $S_u = \{a_1, a_2, ..., a_N\}$, which is ordered chronologically with respect to the timestamps of actions. A subsequence of actions under category c is denoted as $S_u^c = \{a_1^c, ..., a_T^c\}$ where T represents the number of actions in this subsequence and actions are still ordered chronologically. Given S_u from user u, the goal of sequential recommendation is to rank items for this user to consider as the next item v_{N+1} in the next action a_{N+1}. When no ambiguity is incurred, we omit the subscript u to simplify our subsequent discussions.

3.4 Method

3.4.1 Data-Driven Statistical Analyses

Before introducing our proposed solution, we first describe our statistical analyses about users' sequential behaviors on two public recommendation datasets: Taobao dataset and BeerAdvocate dataset. The number of total actions on both datasets are larger than 500K, which ensures the statistical significance of our analyses. The findings in these analyses directly lead to the design of our solution. More details about these datasets are in Section 3.5.

We investigate the dependence structure introduced by item categories. We segment a sequence of actions into multiple subsequences, where each subsequence consists of actions of the same item category. We count the frequency of Mth-order item-to-item transition patterns within subsequences and original sequences respectively. Specifically, the Mth-order item-to-item transition pattern refers to M items appearing consecutively in a given sequence (or a subsequence), e.g., $\{v_i, ..., v_{i+M-2}, v_{i+M-1}\}$. Figure 3.2 (a) shows on Taobao dataset, the probability of the 3rd-order item transition patterns appearing multiple times within subsequences is significantly higher than

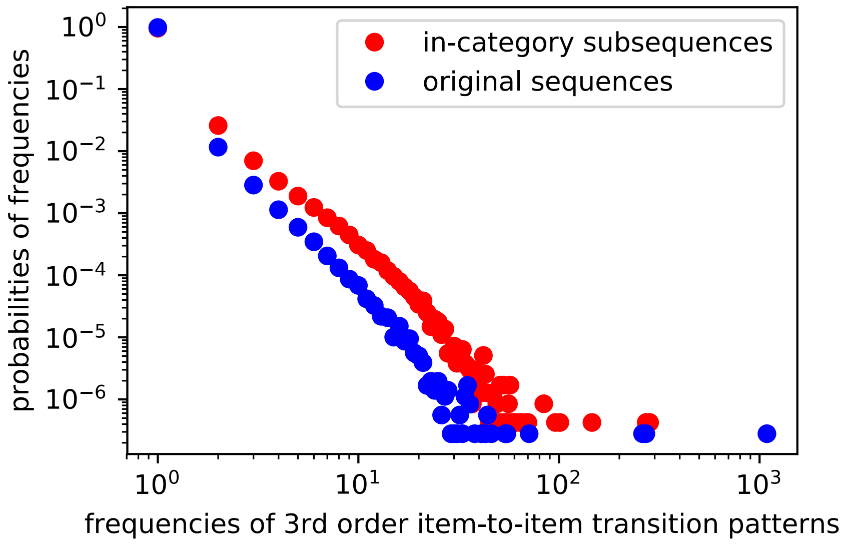

Figure 3.2: Result of the statistical dependence analysis on Taobao dataset. The distribution of frequencies of 3rd-order item-to-item transition patterns in in-category subsequences are as the red points show. The distribution of the frequencies of 3rd-order item-to-item transition patterns in the original sequences are as the blue points show.

that in the original sequences. On BeerAdvocate dataset, we obtained similar observations. By varying M from 2 to 10, we observed similar results. These findings strongly support our decision of using item category to structure actions to enhance the modeling of the action dependence.

3.4.2 Category-aware collaboration Sequential Recommender

Propelled by the findings in our statistical analyses, we propose a CategOry-aware COllaborative sequential Recommender (CoCoRec). In a nutshell, CoCoRec is composed of three modules: an in-category encoder, a context encoder, and a collaboration module. First, to model user preferences under a category, we segment an action sequence into multiple subsequences with respect to item categories and each subsequence is restricted to contain actions of the same item category. The in-category encoder utilizes self-attention to model in-category item-to-item transition patterns in the subsequences. In order to determine which in-category preference to use for the next item prediction, the context encoder predicts the next category based on the categories of recent actions with self-attention. Second, to model the episodic context, the context encoder utilizes another self-attention to model the item-to-item transition patterns among recent actions. Third, to leverage the neighboring users' in-category item-to-item transition patterns, we retrieve users with similar in-category preferences with regard to the target user's in-category preferences, based on the context encoder's next category prediction. Finally, the next item prediction is made based on the episodic context, the in-category user preferences, and neighboring users' in-category preferences.

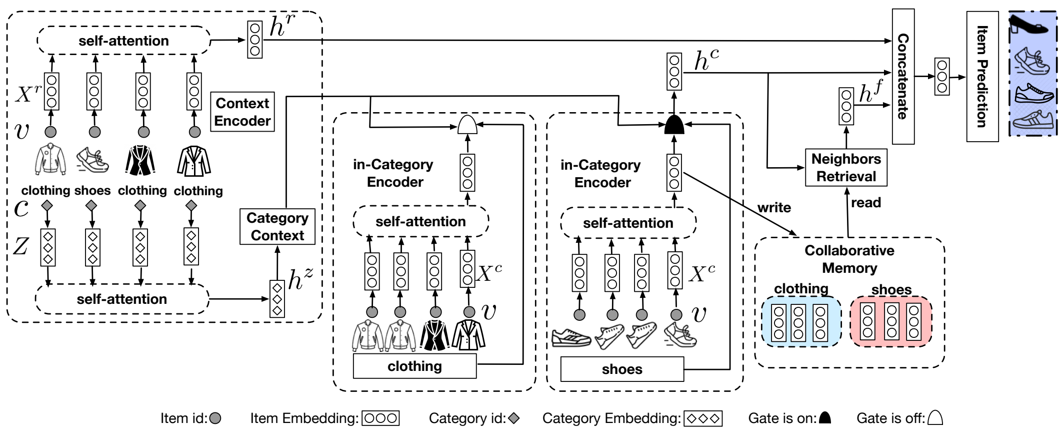

Figure 3.3: Overview of CoCoRec. In CoCoRec, an action sequence is decomposed into multiple subsequences with respect to the item category associated with each action. The in-category encoder encodes the category-specific action subsequences into latent vectors representing users' in-category preferences. The context encoder predicts the category of the next action to activate the corresponding in-category item-to-item transitions for the next item prediction. The context encoder infers the episodic context of the next action based on recent items. To address the sparsity issue, the collaboration module retrieves neighbors based on users' encoded in-category preferences. Based on signals from these three sources, CoCoRec predicts the next item and make recommendations to the user.

In the following, we dive into the details of each component of CoCoRec to discuss about their designs.

In-category Encoder. The in-category encoder is designed to obtain the category-specific user preferences. To capture high-order item-item transition patterns, we choose a self-attention network for the in-category encoder. The self-attention network parameters are shared across categories to reduce model complexity, i.e., multi-task learning via parameter sharing.

For each of the $|C|$ categories, the in-category encoder learns a hidden representation of the user preferences respectively. Without loss of generality, we take the encoding process for an action subsequence of category c as an example to illustrate our design details. The item subsequence $[v^c_1, \dots, v^c_T]$, which are associated with the action subsequence, are projected through the input item embedding layer $E_{in} \in \mathbb{R}^{|V| \times d_{in}}$ into a set of dense vectors $X^c = [e_{v^c_1}, \dots, e_{v^c_T}]$ where $X^c \in \mathbb{R}^{T \times d_{in}}$. The relative positions $[T, \dots, 1]$ of these actions to the next action are projected through the position embedding layer $P \in \mathbb{R}^{T \times d_{in}}$ into $P^c = [P^c_T, \dots, P^c_1]$. Taking the dense vectors $X^c + P^c$ as input, the self-attention network outputs the representation of the user preferences in this category, i.e., $h^c \in \mathbb{R}^{d^c_h}$ by $h^c = \text{self-attention}(X^c + P^c)$.

The self-attention network is composed of n_l layers of a multi-head attention block and a point-wise feed-forward network block [14, 106]. Due to the recursive nature of these multiple layers of blocks, we use jth layer to explain the mechanism. There are n_h heads in a multi-head attention block with

d_a hidden units. For the ith attention head, the attention block transforms the input latent states $H^j \in \mathbb{R}^{T \times d_a}$ of an action sequence into the output states as,

$$\boldsymbol{A}_i^j = \text{Attention}(\boldsymbol{H}^j W_i^Q, \boldsymbol{H}^j W_i^K, \boldsymbol{H}^j W_i^V)$$
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{\boldsymbol{Q} \cdot \boldsymbol{K}^T}{\sqrt{d_a/n_h}}\right)\boldsymbol{V}$$

where the projection matrices $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d_a \times d_a/n_h}$ are learnable parameters mapping H^j into query Q, key K and value V representations. In addition, $\sqrt{d_a/n_h}$ is the scaling factor to encourage a softer attention distribution for avoiding extremely small gradients.

We then concatenate the output states obtained by all heads and project the concatenated representation to $\boldsymbol{A}^j = [\boldsymbol{A}_1^j, \ldots, \boldsymbol{A}_{n_h}^j]W^O$, where $W^O \in \mathbb{R}^{d_a \times d_a}$ is another projection matrice. Feeding this representation into the a fully connected feed-forward network (FFN^j) and then a layer normalization network (LN), we obtain self-attended hidden vectors of this sequence as:

$$\boldsymbol{H}^{j+1} = \text{LN}\left(\boldsymbol{H}^j + FFN^j(\boldsymbol{A}^j)\right)$$

In the in-category encoder, the input to the self-attention network is $H^0 = [e_{v_1^c} + P_T^c, \ldots, e_{v_T^c} + P_1^c]$ and $d_h^c = d_a$. The self-attention network transforms H^0 into $H^{n_l} = [h_1^c, \ldots, h_T^c]$ and uses the hidden state of the last action as the output of the self-attention network to represent the user preferences within category c, i.e., $h^c = h_T^c$.

Context Encoder. The context encoder is to obtain the context and obtain the recent user preferences for the next item prediction. To decide which in-category user preferences to leverage, we predict the category of the next item. Specifically, we use a top-k gating network to obtain the category context. In addition, we use a recency encoder to capture the episodic context buried in recent items.

• ***Top-k gating network.*** We use the categories of recent items as the input to the top-k gating network. The categories of the most recent L items are projected through the input category embedding layer $E_{in}^z \in \mathbb{R}^{|C| \times d_{in}^z}$ into a set of dense vectors, $Z = [e_{c_{N-L}}^z, \ldots, e_{c_N}^z]$. The relative positions of recent actions, denoted as $[L, \ldots, 1]$, are projected through the position embedding layer $P_{cate} \in \mathbb{R}^{L \times d_{in}^z}$ into a set of dense vectors $P^z = [P_L^z, \ldots, P_1^z]$.

In the top-k gating network, a self-attention network transforms the dense category vectors $Z+P^z$ into hidden representations and utilizes the hidden representation at the last action as the representation summarizing the category information of recent actions, as $h^z = \text{self-attention}(Z+P^z) \in \mathbb{R}^{d_h^z}$. Feeding h^z into the output category embedding layer E_{out}^z and then a softmax layer, the top-k gating network generates a probability distribution over all categories:

$$p(\hat{c}_{N+1} = j) \propto \exp(\langle h^z, e_j^z \rangle), \tag{3.1}$$

where $p(\hat{c}_{N+1} = j)$ denotes the probability of category j being the category of the next item and e_j^z is the category embedding of the category j. To account for the uncertainty in next category prediction, the gating network selects top-k most probable categories according to Eq. (3.2):

$$\{c_j\}_{j=1}^k = \arg\underset{j'}{\text{topk}} \left(\{p(\hat{c}_{N+1} = j')\}_{j'=1}^{|C|} \right) \text{ where } c_j \in C. \tag{3.2}$$

With respect to these k predicted categories, we include corresponding in-category user preferences to predict the next item, i.e., the hidden representations $\{h^{c_j}\}_{j=1}^k$ from the in-category encoder are selected. Besides, we also count the probabilities $p(\hat{c}_{N+1})$ of these k categories in the prediction of the next item, which we will discuss later. Because of this design, this top-k gating network is still differentiable, i.e., the training loss can be propagated back to update the category embeddings.

• ***Recency encoder.*** The recency encoder is introduced to infer the users' recent preferences from recent actions. Due to their close proximity to the next action, the recent actions reflect the ongoing intent of the next action. For example, in Figure 3.1, Lily's recent actions suggest she is looking for formal outfits. To capture high-order dependence, we adopt another self-attention network to encode item-to-item transition patterns among recent actions.

Specifically, the input of the recency encoder is the most recent L items in the original sequence S, i.e., $[v_{N-L}, \dots, v_N]$, which are projected through the input item embedding layer E_{in} into dense vectors $X^r = [e_{v_{N-L}}, \dots, e_{v_N}]$, where $X^r \in \mathbb{R}^{L \times d_{in}}$. The relative positions of recent actions, denoted as $[L, \dots, 1]$, are projected through the position embedding layer $P_{recent} \in \mathbb{R}^{L \times d_{in}}$ into a set of dense vectors $P^r = [P_L^r, \dots, P_1^r]$. Then the self-attention network transforms vectors $X^r + P^r$ into hidden states and we use the hidden state of the last action as the representation of the inferred episodic context as $h^r = \text{self-attention}(X^r + P^r) \in \mathbb{R}^{d_h^r}$. This episodic context is utilized for the next item prediction. We should note the recency encoder examines the recent items disregarding their categories, as the episodic context refers to information shared by actions with close proximity beyond specific categories. It provides complementary view of the next action besides the category context.

Collaboration Module. The collaboration module is designed to leverage neighboring users' in-category preferences for the target user's next item prediction. Due to the sparsity of observations in individual users' actions, the item-to-item transition patterns in a single user are expected to be sparse. Collaborative learning across neighboring users with similar preferences has the potential to mitigate the sparsity issue. Because our statistical analyses demonstrate that the in-category item-to-item transition patterns strongly suggest the user's preferences of the next item, we utilize the in-category subsequences to obtain the similarities among users. Then we combine the neighboring users' information based on their similarities for the next item prediction.

The collaboration module uses a memory tensor $Mem \in \mathbb{R}^{|C| \times F \times d_h^c}$ to record users' in-category preferences, denoted as "collaborative memory" in Figure 3.3. Specifically, it records the latent states of last F users for each category c in a chronological order.

• ***Reading operation.*** Given the target user's in-category preferences h_u^c of the category c, we compute its similarities to F latent states of user preferences of this category c in Mem: $\mathrm{sim}(h_u^c, h_i^c) \propto \exp(\langle h_u^c, h_i^c \rangle)$. We choose the top-$f$ similar users as the neighbors and take a weighted sum of their representations by the corresponding similarities as the neighborhood representation for the next item prediction, as $h^f = \sum_{i'}^{\text{top-f}} \mathrm{sim}(h_u^c, h_{i'}^c) h_{i'}^c$.

• ***Writing operation.*** We randomly initialize the memory tensor Mem and update it with the latest user's in-category preference representations h^c which are outputs of the in-category encoder. The memory tensor is organized as a queue: the collaboration module pushes the most recently served user's representations of the category c to the memory tensor, while popping out the representations of users inactive for a long time.

Next Item Prediction. Based on in-category user preferences of the predicted top k categories, neighboring users' in-category preferences, and episodic context inferred from the most recent items, CoCoRec predicts the next item. Specifically, we concatenate these three representations for each of k categories and project concatenated representations into the output item embedding space with a feed-forward network layer (FFN). The mixture of obtained representations is considered as the user

Table 3.1: Statistics of two evaluation datasets.

Dataset	#Users	#Items	#Categories	#Actions	#Actions per user
Taobao	51,275	68,007	201	3,785,961	73.84±47.44
BeerAdvocate	7,313	17,373	102	563,638	55.67±69.44

representation for predicting the next item as

$$h_j = \mathrm{FFN}(h^r \oplus h^{c_j} \oplus h^{f_j}), \text{ where } j = \{1, ..., k\}$$

Matching the user representation with the item embeddings E_{out}, we obtain the ranking scores of items by,

$$\begin{aligned} \mathrm{score}(\hat{v}_{N+1}) &= \sum_{j=1}^{k} \mathrm{score}_j(\hat{v}_{N+1}) p(\hat{c}_{N+1} = c_j) \\ \mathrm{score}_j(\hat{v}_{N+1}) &= \mathrm{softmax}(\langle h_j, E_{out} \rangle) \end{aligned} \tag{3.3}$$

CoCoRec ranks the items with respect to their predicted scores $score(\hat{v}_{N+1})$ in a descending order as recommendations to the user.

3.4.3 Model Training & Inference

We train CoCoRec in an end-to-end fashion by minimizing the loss on the predictions of both the next item and its category, where the cross entropy loss is adopted.

Since the item space can be very large in practice, we apply the negative sampling trick to compute the loss of item predictions. For each positive item, we randomly sample N_s items as negative instances according to their popularities in training dataset. Thus, the loss of item prediction is,

$$\begin{aligned} L_{item} &= -\sum_{v=1}^{N_s+1} \delta(v_{N+1} = v) \log p(\hat{v}_{N+1} = v) \\ p(\hat{v}_{N+1} = v) &= \mathrm{softmax}\Big(score(\hat{v}_{N+1} = v)\Big) \end{aligned}$$

where $\delta(\cdot)$ is an indicator function, v_{N+1} is the ground-truth item, and $\hat{v}_{N+1}$ is the model's prediction. Likewise, we compute the loss against all categories,

$$L_{cate} = -\sum_{j=1}^{C} \delta(c_{N+1} = j) \log p(\hat{c}_{N+1} = j).$$

where the $p(\hat{c}_{N+1} = j)$ is computed by Eq (3.1). The joint loss is thus computed as

$$L = \lambda \times L_{item} + (1 - \lambda) \times L_{cate}.$$

where λ is a hyper-parameter controlling the weight of these two losses in the objective function.

In addition, we modify the training scheme to deal with the discrepancy between training stage and testing stage. During training, the ground-truth category of the next item is available. Thus, we can directly choose the in-category user preferences of the ground-truth category for the next item prediction. In contrast, during testing, the next category is unknown. The errors of the next category prediction will be propagated to the next item prediction. To mitigate this issue, we separate the training phase into two stages. In the early stage of training the model, we directly use the ground-truth category for the next item prediction. In the second stage, we use the top-k predicted categories. Particularly, we start the second stage model training only when the accuracy of category prediction stops increasing.

3.5 Experiments

In this section, we study the effectiveness of CoCoRec for sequential recommendation. We first describe two evaluation datasets, followed by the implementation details of our model on these two datasets. Then we compare CoCoRec against an extensive set of baselines, ranging from heuristic solutions to state-of-the-art sequential recommendation solutions. In addition, a complete ablation analyses illustrates the importance of modeling the in-category user preferences and collaborative learning among users with similar in-category preferences. We also study the influence of the hyper-parameters on the performance of CoCoRec.

3.5.1 Datasets

We performed the evaluation on Taobao dataset [1] and BeerAdvocate dataset [2], which are both publicly available. The Taobao dataset contains sequences of user actions from the online shopping website `taobao.com`. Each action is associated with a user ID, an item ID, a category ID of the item, and a timestamp of the action. Due to privacy concerns, the semantic meanings of categories are not available. We randomly sampled 100,000 sequences from November 25, 2017 to December 3, 2017 for our experiments, where we used the actions in the first 7 days as the training set, actions on the 8th day as the validation set, and actions on the 9th day as the test set. We removed items associated with fewer than 20 actions, and removed users with fewer than 20 or more than 300 actions. We merged categories which have fewer than 100 items into a special category, denoted as category "UNK". The BeerAdvocate dataset contains user reviews about beer from October 31, 2000 to January 11, 2012. The type of beer is chosen as category, and a user review is treated as an action. We use actions from October 31, 2000 to January 28, 2011 as the training set, those from January 28, 2011 to July 18, 2011 as the validation set, and the rest as the test set. We removed items with fewer than 5 actions, and removed users with fewer than 10 or more than 300 actions. Again, we merged categories with fewer than 100 items into the "UNK" category. The basic statistics of datasets are reported in Table 3.1.

Implementation Details. On both datasets, the in-category encoder of CoCoRec utilizes at most $T = 20$ actions of the same category. The context encoder utilizes the most recent $L = 20$ actions as input to both the top-k gating network and the recency encoder. The item input embedding layer shares the same parameters as the item output embedding layer. The category input embedding layer also shares the same parameters as the category output embedding layer. The self-attention networks in the in-category encoder, the top-k gating network and recency encoder stack $n_l = 2$ layers of a multi-head attention block with $n_h = 1$ head and a feed-forward network block. In the objective function, the hyper-parameter λ is set to 0.5. The dropout rates are all set to 0.2. The batch size is set to 256. We utilize Adam as the optimizer.

On Taobao dataset, the dimension d_{in} of item embeddings is chosen from $\{128, 256, 512\}$. We set $d_{in} = 256$ in our experiments, as we did not observe further improvement of performance with higher dimensions. The dimension d_h^c of the hidden representations in the self-attention network of the in-category encoder is chosen from $\{128, 256, 512\}$ and we set $d_h^c = 256$. The dimension d_{in}^z of category

[1]https://tianchi.aliyun.com/dataset/dataDetail?dataId=649
[2]https://www.beeradvocate.com/

Table 3.2: Performance of models on Taobao dataset.

Models	BeerAdvocate			
	Recall@5	MRR@5	Recall@20	MRR@20
GlobalPop	0.0078	0.0036	0.0341	0.0059
SeqPop	0.0004	0.0002	0.0016	0.0003
GRU4Rec	0.0173	0.0089	0.0492	0.0097
BERT4Rec	0.0222	0.0092	0.0533	0.0124
M3R	0.0235	0.0102	0.0615	0.0142
RNN+MTL	0.0202	0.0101	0.0573	0.0121
MFGAN	0.0233	0.0114	0.0599	0.0143
CSRM	0.0188	0.0093	0.0514	0.0112
ICM-SR	0.0214	0.011	0.0587	0.0129
CoCoRec	**0.0278**	**0.0141**	**0.0737**	**0.0192**

embeddings is chosen from $\{32, 64, 128\}$ and we set $d_{in}^z = 64$ due to its promising performance. The dimension d_h^z of the hidden representations in the self-attention network of the top-k gating network is set to be the same as d_{in}^z. The top-k gating network selects $k=5$ category-specific action subsequences for the next item prediction. Likewise, we set the dimension d_h^r in the self-attention network of the recency encoder to 256. The collaboration module records the in-category preferences of last $F = 10240$ users and retrieves $f = 256$ neighboring users. The number of negative items N_s in model training is set to 10000. We find that CoCoRec is sensitive to the learning rate, and the optimal learning rate 0.0001 is chosen from $\{0.00001, 0.0001, 0.0005, 0.001\}$. On BeerAdvocate dataset, the details of our model are as follows: $d_{in}=64, d_h^c=64, d_{in}^z=32, d_h^z=32$ and $d_h^r=64$. The hyper-parameters in collaboration module are $F=2048$ and $f=128$. The top-k gating network selects $k=5$ category-specific action subsequences for the next item prediction. The number of negative items N_s in model training is set to 1000. The learning rate is set to 0.0001. The influence of the hyper-parameters on the performance of CoCoRec is discussed later. The code has been released publicly [3].

3.5.2 Comparison against Baselines

We compare CoCoRec with an extensive set of baselines, and categorize them based on their modeling assumptions and the information they leveraged.

Heuristic solutions. We include two heuristic-based solutions, which have been shown to be strong baselines [107].

[3]code available at https://github.com/RenqinCai/CoCoRec

Table 3.3: Performance of models on BeerAdvocate dataset.

Models	Taobao			
	Recall@5	MRR@5	Recall@20	MRR@20
GlobalPop	0.0024	0.0014	0.0076	0.0019
SeqPop	0.0944	0.0533	0.1754	0.0613
GRU4Rec	0.1283	0.0811	0.1888	0.0839
BERT4Rec	0.1291	0.0813	0.2122	0.0869
M3R	0.1294	0.0818	0.2163	0.0875
RNN+MTL	0.1283	0.0801	0.1979	0.0833
MFGAN	0.1307	0.0817	0.2176	0.0852
CSRM	0.1291	0.0818	0.1923	0.0842
ICM-SR	0.1299	0.082	0.2057	0.0849
CoCoRec	**0.1557**	**0.0917**	**0.2609**	**0.1029**

• *Global Popularity (GlobalPop).* It ranks items according to their popularities in the training set in a descending order.

• *Sequence Popularity (SeqPop).* It ranks items according to their popularities in the target user's sequence in a descending order. The popularity of an item is updated sequentially when more actions are observed.

Classical sequential recommendation solutions. We include solutions which only consider sequential order of actions for dependence modeling.

• *Recurrent Neural Network (GRU4Rec).* It adopts GRU for sequential recommendations [10].

• *Bidirectional Self-attentive Sequential Recommendation (BERT4Rec).* It adopts self-attention for sequential recommendations [33], which extends the SASRec model [27].

• *Multi-temporal-range Mixture Model (M3R).* It utilizes a mixture of RNN and self-attention to capture the dependence on both distant past actions and recent past actions for sequential recommendations [31].

Category-aware sequential recommendation solutions. We include solutions leveraging the item category for recommendations.

• *Category-Based Recommender (RNN+MTL).* It incorporates category information by treating it as an additional input to the neural sequence model [101]. Particularly, this solution utilizes multi-task learning to predict both the next item and its category.

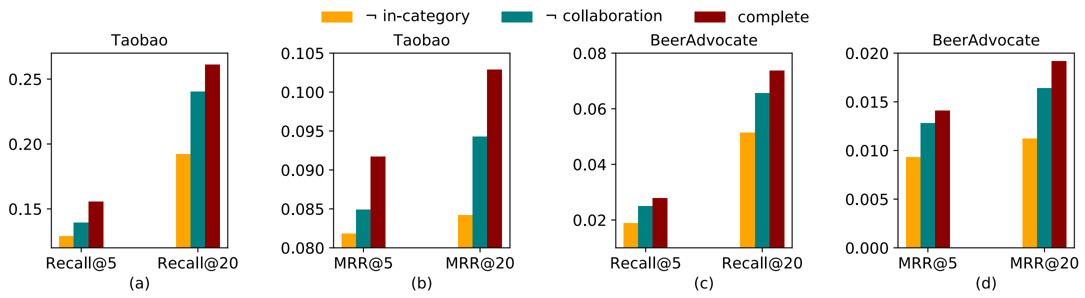

Figure 3.4: Performance of variants of CoCoRec for ablation analysis on two datasets.

• *Multi-Factor Generative Adversarial Network (MFGAN)*[4]. It utilizes adversarial training to ensure the generated action sequences still follow the distribution of the real-world action sequences [102]. It leverages the category of the next action and item popularity as conditions used by the discriminator in the adversarial learning.

Collaborative learning based sequential recommendation solutions. We include sequential recommendation solutions utilizing collaborative learning to combat data sparsity issue.

• *Collaborative Session-based Recommendation Machine (CSRM).* It utilizes the entire action sequences of users to define the users' similarities and retrieves neighboring users with similar preferences [97]. The neighboring users' preferences are combined with the target user's preferences to predict the next item.

• *Intent-guided Collaborative Machine for Session-based Recommendation (ICM-SR).* This work is an extension of CSRM, aiming at reducing the search space of neighbors [96]. It utilizes action sequences to make initial next item predictions and filters the irrelevant users based on these initial item predictions.

Evaluation Metrics. Recall@K and Mean Reciprocal Rank@K (MRR@K) are used as evaluation metrics. We rank all items for evaluation, instead of sampling a subset of items. This can avoid the bias introduced by the sampling to the evaluation, as is demonstrated in [108] .

• *Recall@K*: It counts the proportion of times when ground-truth items are ranked among the top-K predictions.

• *MRR@K*: It reports the average of reciprocal ranks of the ground-truth items among the top-k predictions. If the rank is larger than K, the reciprocal rank is set to 0.

[4]the knowledge graph in this method is omitted as it is not available on these two datasets.

Results & analysis. The results of CoCoRec and baselines on two datasets are reported in Table 3.3, where CoCoRec outperformed all baselines. Regarding *heuristic solutions*, since GlobalPop treats each action independently and SeqPop only considers the dependence on actions associated with the same item, both of them performed worse than CoCoRec.

None of the *classical sequential recommendation solutions* leverage category information when modeling the action sequences. Without considering the fine-grained dependence among actions, vanilla applications of existing neural sequence models, like GRU4Rec and BERT4Rec, performed much worse than CoCoRec. M3R considers fine-grained dependence through a mixture of these neural sequence models. But its worse performance against CoCoRec suggests that leveraging category information is more useful to calibrate the dependence modeling than blindly combining a set of distinct neural sequence models.

The *category-aware sequential recommendation solutions* leverage category information to enhance the sequential recommendations. The worse performance achieved by RNN+MTL suggests that without careful design, considering category information does not necessarily improve sequential recommendation. MFGAN utilizes the category of the next action and the popularity of items to calibrate the dependence on past actions. But it does not specifically model item-to-item transition patterns within each category. Moreover, CoCoRec also makes use of category-specific action subsequences of the neighboring users to enhance the next item prediction. Therefore, we observe that CoCoRec achieved better performance than these category-aware solutions.

The *collaborative learning based sequential recommendation solutions* take advantage of neighboring users' actions to enhance the sequential recommendations. The better performance achieved by CSRM over GRU4Rec shows explicit collaborative learning helps sequential recommendation. The comparison between CSRM and ICM-SR suggests that improving the quality of retrieved neighboring users enhances the utility of collaborative learning on sequential recommendations. Moreover, CoCoRec outperformed ICM-SR, which proves that context-aware neighborhood modeling can further increase the benefits of leveraging neighboring users' actions.

3.5.3 Detailed Analysis on Our Approach

Ablation Analysis. We conducted ablation analysis to demonstrate the importance of different components in CoCoRec. Specifically, we tested the following variants of CoCoRec:

¬ *in-category.* This variant excludes the in-category encoder from CoCoRec. It only utilizes the recency encoder and the collaboration module (ignoring item category) to predict the next item. The comparison between this variant and CoCoRec demonstrates the importance of using the in-category encoder to model the in-category user preferences.

¬ *collaboration.* This variant excludes the collaboration module from CoCoRec. It only uses the in-category encoder and the context encoder to predict the next item. The comparison between this variant and CoCoRec demonstrates the value of using the collaboration module to leverage neighboring users' in-category preferences.

Results are reported in Figure 3.4. The observation that these two variants perform worse than CoCoRec suggests that both the modeling f in-category preferences and the collaborative learning are useful for sequential recommendation.

Hyper-parameter Analysis. We changed the value of hyper-parameters in CoCoRec and investigated their influence on the recommendation quality.

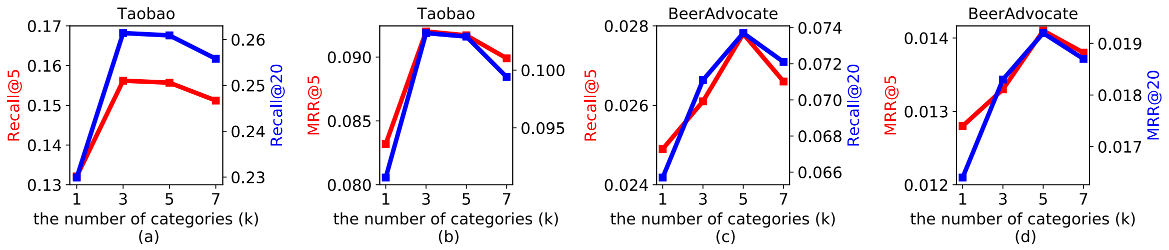

Figure 3.5: Performance of CoCoRec with different number of selected categories (k) on Taobao.

• *Number of categories in top-k gating network.* Because of the influence from the next category prediction on the next item prediction, we study the effect of the number of categories k selected by the gating network on the performance of CoCoRec. We report results in Figure 3.5. We can observe CoCoRec achieves the best performance with $k=3$; and when $k=1$, the performance drops. This is caused by imperfect category prediction, i.e., the ground-truth categories are not ranked at the first position. In particular, the performance of the category prediction is Recall@5:0.7328 and MRR@5:0.5510. The wrong predictions of the next category further impact the next item predictions. On the other hand, when k keeps increasing, the performance also drops. The reason is that larger k means more in-category user preferences are included for the next item prediction. This inevitably introduces irrelevant in-category preferences, which eventually hurts the next item prediction.

• *Number of actions in in-category encoder and context encoder.* The lengths of category-specific action subsequences affect the modeling of in-category user preferences. We study the effect of the number of actions T per category on the performance of CoCoRec. In addition, since the number of recent actions affects both the modeling of category context and episodic context, we also study the effect of the number of recent actions L on the performance of CoCoRec. The results on Taobao dataset are reported in Table 3.4. We can observe when too few actions are considered in the in-category encoder, i.e., $T = 2$, CoCoRec perform poorly. This is because limited information about the in-category user preferences can be captured. Similarly, when too few recent actions are considered in the context encoder, i.e., $L=2$, the performance drops. This suggests recent actions contain important signals for the next item prediction.

Table 3.4: Performance of CoCoRec with different number of actions in in-category encoder (T) and context encoder (L) on Taobao dataset.

Settings		Recall@5	MRR@5	Recall@20	MRR@20
T=2	L=20	0.1322	0.0821	0.2371	0.0931
T=20	L=20	0.1557	0.0917	0.2609	0.1029
T=50	L=20	0.1559	0.0918	0.2604	0.1026
T=20	L=2	0.1301	0.0811	0.2253	0.0876
T=20	L=50	0.1559	0.0919	0.2612	0.1030

• *Number of retrieved neighboring users in collaboration module.* Because the number of retrieved neighbors affects the collaborative learning, we study the effect of the number of retrieved neighboring users f on the performance of CoCoRec. The results are reported in Table 3.5. We can see retrieving either too few or too many neighboring users hurts the next item prediction. When too few neighbors are retrieved, the collaboration effect is limited. Thus, the CoCoRec cannot benefit from similar users. On the other hand, when too many neighbors are retrieved, including the action subsequences from less relevant users hurts the next item prediction.

Table 3.5: Performance of CoCoRec with different number of retrieved neighbors on Taobao dataset.

Settings	Recall@5	MRR@5	Recall@20	MRR@20
f=128	0.1502	0.0903	0.2371	0.0931
f=256	0.1557	0.0917	0.2609	0.1029
f=1024	0.1534	0.0907	0.2583	0.1017

Case Study. To examine our model's behaviors, we qualitatively study in-category action subsequences, recent action subsequences, neighors' in-category action subsequences, and the attention weights of actions in these subsequences. We select a user and one of her actions in the testing set as the target action. We retrieve the neighbors with similar in-category preferences to her, as

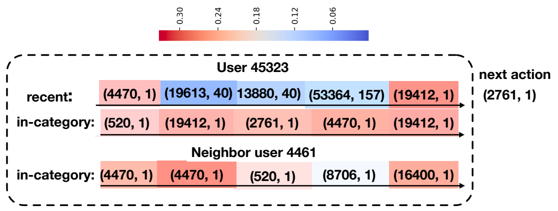

Figure 3.6: An example of the next item prediction by CoCoRec. The target user's (user 45323) action sequence is separated into category-specific subsequences. On the top, We visualize the action subsequence containing the recent actions, and the action subsequence specific to the category of the next item. On the bottom, the most similar neighbor's (user 4461) in-category subsequence is presented. The actions in each subsequence are indexed chronologically, denoted as a tuple (item id, category id). The color indicates the attention weight in self-attention networks.

Figure 3.6 shows. We can observe the in-category subsequence of the top ranked neighbor has actions in common with the in-category subsequence of the target user. The actions associated with items (e.g., item 19412) which have appeared multiple times likely have large attention weight. In addition, the item-to-item transitions (from item 19412 to item 2761) in the category-specific subsequence suggest the next item.

3.6 Conclusion

In this chapter, suggested by our statistical analyses on the dependence structure introduced by the category-specific action context, we propose CoCoRec to leverage category information to capture the context-aware sequential structure among actions for recommendations. Specifically, a sequence of actions is decomposed into multiple subsequences with respect to item categories. Within each category-specific subsequence, CoCoRec employs a self-attention network to capture item-to-item transition patterns. A top-k gating network is employed to predict the category of the next item, so as to activate the in-category preferences for the next item prediction. Besides, CoCoRec models the most recent actions as episodic context, due to their close proximity to the next action. To handle sparsity in individual user's action sequences, CoCoRec employs context-aware collaborative learning across users with similar in-category preferences. Extensive experiments and ablation analyses on two large datasets show that considering the sequential structure among actions helps CoCoRec improve its sequential recommendation quality.

Currently we only considered the sequential order of actions in users' interaction history with the system. Since the actual time of those actions also conveys important contextual information, it is important to consider how to extend CoCoRec to model such temporal information in the furture work.

Chapter 4

Graph Based Extractive Explainer for Recommendations

4.1 Introduction

Recommendations in online service platforms, from e-commerce (like Amazon) to streaming media services (like Netflix), have greatly shaped people's engagement with these platforms. However, explanations about why a particular person gets a certain recommendation are not provided. Providing explanations of recommendations to users is critical for recommender systems to improve the user experience, as users not only care about which items have been recommended, but are also interested in why these items are recommended. Take Figure 4.1, a person who drank the beer *Corona Extra* might be recommended with another beer *Modelo Especial*, by a service platform. She might not understand why the service platform thinks she would like this beer (*Modelo Especial*) and thus might reject the recommendation. However, an explanation explaining which features of an item are important for a user could make the user realize the recommended item may suit her taste. For example, suppose the user has commented the beer *Corona Extra* with "The taste is like strong corn.", presenting an explanation "Its taste is a balanced aroma of corn, with a hint of bitterness" along with the recommended beer *Modelo Especial* might give her clear information about the beer and help her make an informed decision, i.e., to purchase it or not.

Providing explanations for recommendations is attracting more and more attention for the purpose

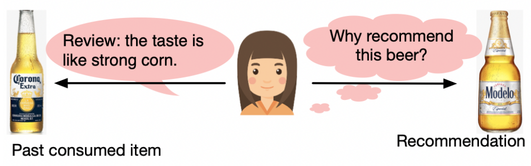

Figure 4.1: Example of the task of providing explanations for recommendations. Besides the recommendation, the user also cares about the reason of the recommendation.

of making recommendations more understandable and reliable, which helps users make decisions. Natural language based explainable methods have become one of the main stream solutions. Natural language based explainable methods are to produce a text snippet to describe which features of the item is important to the user. Due to the lack of dedicated corpus of textual sentences as explanations, user-written reviews are treated as proxies of explanations, because reviews depict noticeable features of items that users care about [16]. Most developed methods are based on the nature language generation techniques. For example, in [57], the authors proposed a Recurrent Neural Network (RNN) based solution which considers the user and item information to generate sentences word-by-word. Likewise, in [23], the authors combined attention with RNN to improve the generation quality of sentences. In [109], authors proposed to utilize the pointer network to improve the explanation quality.

Although these solutions have achieved some encouraging results, the produced explanations have issues. First, the generated sentences are generic. For different user-item pairs, the generated sentences look very similar and features in these sentences are the same. The items' noticeable features that are important to certain users are missing. These solutions assume modeling sentences, written by certain users to describe certain items, can capture which noticeable features of these items are important to users. But because feature words take up a small portion and other functional words which are not about features dominate in a sentence, these models are likely to capture the correlation between functional words and user-item pairs. Thus, these models fail in capturing items' noticeable features that are important to users. Second, the content in explanations is repetitive. As a user's perception of an item is related to the item's multiple features, an explanation is expected to cover multiple features in a long sentence or multiple sentences. Because of the limits in capturing long dependence in a long sentence or multiple sentences, the generative models have difficulties in generating long sentences or multiple sentences that cover multiple features. Consequently, the models would generate repetitive words in a long sentences or generate the same sentences multiple times.

Concerning these issues, we proposed a graph neural network based solution, GRaph Extractive ExplaiNer (GREENer). We tackle the problem by extracting sentences from reviews as an explanation, instead of generating sentences. Users care about particular features and thus tend to repeat mentioning these features, while items have some noticeable features and thus are described with these features many times. Therefore, the historical sentences written by users and historical sentences describing items provide references of new explanations. First, to select sentences containing noticeable item features which are important to the user, we explicitly model the co-occurrence of user-feature and item-feature. Modeling the co-occurrence of user-feature and item-feature is to recognize which noticeable item features are cared by the user. Based on the recognized features, we select sentences that cover them. Considering the relations of user-feature, item-feature and feature-sentence, we assume a graph structure among user, item, feature and sentence. Specifically, a graph consists of four types of nodes: user node, item node, sentence node and feature node. An edge connects a user node to a feature node if the user has used the feature to describe any item, while an edge connects an item node to a feature node if the item has been described with the feature by any user. In addition, an edge connect a feature node to a sentence node if the sentence contains the feature. To leverage the graph structure, we appeal to graph attention network to obtain sentence representations. For each sentence, based on its representation, we then predict whether a sentence is a part of an explanation. Considering the predictions of features appearing in an explanation are helpful to the predictions of sentences being included in an explanation, we propose to leverage the supervision of both features and sentences. The joint loss of predicting features and sentences is utilized to train the model, i.e., multi-task learning. Second, to reduce the redundancy in an explanation, we select sentences by using similarities among sentences as another criterion besides relevance. The advantages of GREENer include,

- GREENer uses relevance and similarity as criteria to extract sentences as explanations.
- GREENer utilizes graph structure among user, item, feature and sentence to recognize the relevance of sentences to a pair of user and item.

To investigate the effectiveness of GREENer for producing explanations, we performed extensive experiments on two large public datasets. Compared with state-of-the-art solutions for producing explanations, GREENer improved the explanation quality in both BLEU and Rouge. Our ablation analysis further demonstrated the importance of modeling these four types of nodes in the graph, and also the importance of the graph structure. Case study shows that the produced explanations contain noticeable features that are important to users.

4.2 Related Work

Due to the information explosion, it is difficult for people to digest all information and then make decisions. Recommender systems which select relevant information and present them to users have great impacts on people's decisions [56,95,110–112]. Because of its great impact, its transparency becomes a great concern to people. People not only care about the effectiveness of recommender systems but also are curious about why the recommender systems believe they like the recommended items. Explanations for recommendations are critical to the success of recommender systems, e.g., increasing user engagement [113–115].

Research on providing explanations for recommendations have attracted great attention. Several types of explanations have been studied. One type of explanation is based on user-user collaborative filtering. For example, the explanation might be "people who like *Corona Extra* also like *Modelo Especial*." Although this type of explanation can give the user a hint about how the recommendation system works in the back-end, it does not show which particular features of this item are important to the user.

Another type of explanation is natural language based explanations. These solutions exploit user-written reviews as the proxies of explanations, as reviews depict items' noticeable features that are important to users. So far, solutions which learn to generate explanations from scratch word-by-word are widely studied. The solutions are based on text generation techniques, like RNN or Attention. To make the generated content personalized to users and specific to items, these solutions introduce user and item representations as factors controlling the text generation. In [57], authors proposed a RNN based solution incorporating user and item embeddings as the initial hidden states of RNN to generate explanations. In [23], authors proposed a solution which combines RNN with attention emphasizing a portion of words to improve the quality of generated explanations. In [21], authors integrate images as extra factors to control the generation besides users and items. However, most of existing generative solutions produce generic explanations which are not distinct for different user-item pairs. They assume modeling sentences written by users to describe items can capture users' preferences towards items' features. However, the functional words dominate and feature words take up a small portion in a sentence. The generative models are likely to capture the correlations between functional words and user-item pairs, instead of the correlations between features and user-item pairs.

In addition, due to the limit of capturing long dependence in multiple sentences, the generation based solutions would produce repetitive content in explanations.

Instead of generating sentences from scratch, some work proposed to predict features and then fill these features into templates to generate explanations. In [16], authors proposed to predict features that are personalized to users and specific to items by matrix factorization. With the predicted features and the pre-defined templates, they produced explanations, like "You might be interested in [feature], on which this product performs well." In [17], authors extended the matrix factorization into tensor completion with predictions of features. In [116], authors introduced the pairwise constraints into the feature predictions. Thanks to considering the co-occurrence of user-feature and item-feature, the explanations produced by these solutions are not generic. Additionally, because of the template, explanations do not contain repetitive content. However, the strength stemmed from using templates is also the bottleneck of these solutions, that is, the manually constructed templates are not necessarily compatible with the predicted features. For example, when recommended item is the beer *Corona Extra* and the predicted feature is "corn", having an explanation "You might be interested in *corn*, on which this product performs well." sounds confusing to the user. It is not nature to state the beer performs well on corn, which can increase users' untrust on the recommendation. Different from these solutions, GREENer extracts sentences written by users as explanations, which can address these linguistic issues.

Graph attention networks (GATs) have achieved great success in many tasks, like text classification or reading comprehension [117–120]. GATs can capture the high-order and complex relations among nodes. With the message passing, GATs learn node representations by aggregating information from nodes' neighboring nodes. Based on the encoded node representations, GATs make downstream task predictions. Early applications of GATs focus on homogeneous graphs which consists of the same type of nodes , e.g., multi-class node classifications in Cora, Citeseer and Pubmed citation network datasets [117]. Recently, researchers have applied GATs to heterogeneous graphs with multiple types of nodes, e.g., text summarization [120]. They have shown that GATs can capture multiple relations among multiple types of nodes. Inspired by these work, we adopt GATs for our task to capture multiple relations among user, item, feature and sentence nodes, i.e., the relations of user-feature and item-feature, and feature-sentence.

4.3 Problem Definition

In this section, we introduce the notations used in this chapter and define the problem studied in this chapter. Denote the item space as $\mathcal{C}$ of size $|C|$, the user space as $\mathcal{U}$ of size $|U|$, and the vocabulary as $\mathcal{V}$ of size $|V|$. We assume there are G reviews written by users $\mathcal{U}$ towards items $\mathcal{C}$. Sentences in these G reviews are denoted as $\mathcal{D} = \{\mathcal{S}_{uc}\}_{u \in \mathcal{U}, c \in \mathcal{C}}$. Each $\mathcal{S}_{uc} = \{s^i_{uc}\}_{i=1}^N$ represents N sentences s^i_{uc} written by a user u towards an item c. Each sentence $s_{uc} = \{w_i\}_{i=1}^T$ consists of T words $w \in \mathcal{V}$. The set of sentences written by a user u over items is denoted as $\mathcal{S}_u = \{\mathcal{S}_{uc}\}_{c \in \mathcal{C}}$. Likewise, the set of sentences talking about an item i written by users is denoted as $\mathcal{S}_c = \{\mathcal{S}_{uc}\}_{u \in \mathcal{U}}$. The features $\mathcal{F}$ are items' popular properties mentioned in the $\mathcal{D}$, which are a subset of $|F|$ words selected from the vocabulary $\mathcal{V}$. For a pair of a user $u \in \mathcal{U}$ and an item $c \in \mathcal{C}$, the model is to select K sentences $\{S^i\}_{i=1}^K$ from the union of sentences $\mathcal{S}_u \bigcup \mathcal{S}_c$, such that these K sentences best describe how the user would perceive the item.

4.4 Graph Extractive Explainer for Recommendations

We assume the description of noticeable item features that are important to users can explain how the user would perceive the item. Thus, our solution learns to produce sentences that contain these features. Users have preferences towards particular features and tend to repeat emphasizing them, while items have noticeable features and are likely to be described with these features multiple times. Therefore, we propose to extract sentences from historical reviews as explanations. To select sentences that can help users perceive items, we use relevance of sentences to user-item pairs as an criterion. Specifically, we assume users, items, features and sentences are connected as graphs. Then we adopt graph attention network to capture the dependence among them in the graphs and estimate the relevance of sentences. Additionally, to reduce the redundancy of selected sentences, we use similarities among sentences as another criterion to select sentences. We propose to utilize a language model based strategy. It adds new sentences into an explanation, based on whether the new sentences are similar to existing sentences in the explanation. We name the proposed solution as GRaph Extractive ExplaiNer (GREENer).

4.4.1 Graph Structure

The graph structure among the user, the item, the sentences and features allow GREENer to leverage the co-occurrence of user-feature, item-feature, and feature-sentence jointly.

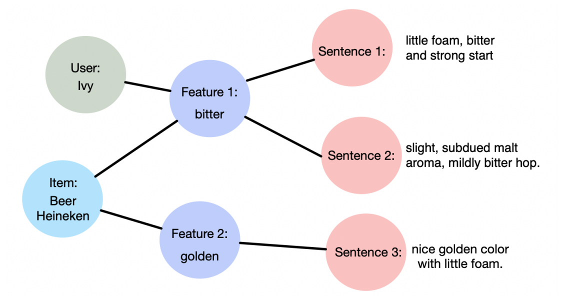
User: Ivy
Feature 1: bitter
Sentence 1:
little foam, bitter and strong start
Sentence 2:
slight, subdued malt aroma, mildly bitter hop.
Item: Beer Heineken
Feature 2: golden
Sentence 3:
nice golden color with little foam.

Figure 4.2: Illustration of a graph constructed to extract sentences as explanations for a pair of a user and an item. The graph consists of four types of nodes: user node, item node, feature nodes, and sentence nodes. User node is connected to feature nodes, item node is connected to feature nodes, and feature nodes are connected to sentence nodes.

Nodes. As Figure 4.2 shows, a graph G consists of four types of nodes: a user node u, an item node c, $|S_g|$ sentence nodes where $S_g = S_u \bigcup S_c$ represents the union of sentences written by the user u and sentences talking about the item c, and M feature nodes $\{f_g^i\}_{i=1}^M$ where $M \leq |F|$ appearing in sentences S_g. The set of sentences $\mathcal{S}_{uc} = \{s_{uc}^i\}_{i=1}^N$ written by the user u towards the item c is a subset of S_g. The input representation of the user node is a dense vector X_u, obtained by mapping the user index u through the input user embeddings $E_u \in \mathbb{R}^{|U| \times d_u}$. Likewise, the input representation of the item node is a dense vector X_c, obtained by mapping the item index i through the input item embeddings $E_c \in \mathbb{R}^{|C| \times d_c}$. In addition, to represent feature nodes, we map features $\{f_g^i\}_{i=1}^M$ into dense vectors $\{X_{f_g^i}\}_{i=1}^M$ through the input feature embeddings $E_f \in \mathbb{R}^{|F| \times d_f}$.

Because pre-trained large language models, i.e., Bidirectional Encoder Representations from Transformers (BERT) have achieved the great success in encoding sentences, we use BERT to encode sentences $\mathcal{S}_g$ as their input representations. Specifically, we fine-tune the BERT on the corpus of reviews. Then we use the hidden vectors of sentences outputted by the fine-tuned BERT $\{X_{s_g^i}\}_{i=1}^{|S_g|}$ as input representations of sentence nodes, where $X_{s_g^i} \in \mathbb{R}^{d_s}$.

Edges. An edge e_{uf} connects user node u to feature node f if the feature has been used by the user. An edge e_{cf} connects item node c to feature node f if the feature has been used to describe the item. In addition, an edge e_{fs} connects feature node f to sentence node s if the sentence contains the feature word. Notice that all these edges are nondirectional. In this chapter, the edge weights are binary and other types of edge weights (e.g., frequency based weights) are left as future work to study.

4.4.2 Graph Attention Layer

Given a graph g with nodes X_u, X_c, X_f, X_s and edges e_{uf}, e_{cf}, e_{fs}, we adopt graph attention networks (GAT) [117] to encode co-occurrence information into node representations. Specifically, we stack L graph attention layer to map input node representations into output node representations $\hat{X}_u, \hat{X}_c, \hat{X}_f, \hat{X}_s$. Due to the recursive nature of graph attention, we use a layer to illustrate the mechanism in our solution. For example, in the lth layer, the inputs to the graph attention layer are $H^l = \{H_u, H_c, H_f, H_s\}$, which correspond to hidden representations of user node, item node, feature nodes and sentence nodes. For the ith node h_i^l in the graph, we obtain attention weights α^l of its connected nodes as,

$$\begin{aligned} z_{ij}^l &= \mathrm{LeakyReLU}(W_a^l [W_q^l h_i^l || W_k^l h_j^l]) \\ \alpha_{ij}^l &= \frac{\exp(z_{ij}^l)}{\sum_{j' \in \mathcal{N}_i} \exp(z_{ij'}^l)} \end{aligned}$$

where $\mathcal{N}_i$ refers to other nodes connected to ith node, i.e., the neighboring nodes of ith node. W_a^l, W_q^l, W_k^l are trainable parameters and $||$ denotes the concatenation operation.

With the attention weights, we obtain the output hidden representation of the ith node in the lth layer as

$$h_i^{l+1} = \sigma(\sum_{j \in \mathcal{N}_i} (\alpha_{ij}^l h_i^l)) \tag{4.1}$$

With the d_h multi-head attention, we repeat the above process d_h times and merge the output hidden representations from d_h heads as the representation of the ith node as

$$h_i^{l+1} = ||_{head=1}^{d_h} h_{head,i}^{l+1}$$

where $h_{head,i}^{l+1}$ is obtained as Eq. 4.1.

Note that for the initial attention layer, we use the input node representations X_u, X_c, X_f, X_s as the input H^0, and through L attention layers, we use the output representations H^{L+1} as the output node representations $\hat{X}_u, \hat{X}_c, \hat{X}_f, \hat{X}_s$.

4.4.3 Relevance of Sentences

With output node representations by the graph attention layer, we make predictions of sentences being included in an explanation. Considering the task of predicting sentences being included in an explanation is related to the task of predicting features appearing in the explanation, we optimize these two tasks jointly, i.e., multi-task learning.

Multi-task loss function. We use pairwise loss as the loss function of sentence predictions. The ground truth pairwise order between a pair of sentences is obtained as: for each sentence, we obtain its similarity to the ground truth explanation, and then rank these two sentences based on their similarities. We use y_{s_i} to denote the similarity of the ith sentence. The ground truth pairwise order between the ith sentence and the jth sentence is denoted as,

$$\text{sign}(y_{s_i} - y_{s_j}) := \begin{cases} 1 & \text{if } y_{s_i} > y_{s_j}, \\ 0 & \text{if } y_{s_i} == y_{s_j}, \\ -1 & \text{if } y_{s_i} < y_{s_j}. \end{cases}$$

With the output node representations of sentences, we obtain the unnormalized relevance scores of sentences being selected. Specifically, we feed $\hat{x}_{s_i}$ of the ith sentence into a linear layer and obtain its unnormalized relevance score as,

$$\text{score}(s_i) = \langle W_o^s, \hat{x}_{s_i} \rangle \tag{4.2}$$

With the pairwise order and unnormalized relevance scores, we obtain the pairwise loss of the ith sentence and the jth sentence as,

$$L_s = \sum_{i \in S_g} \sum_{j \in S_g} \text{sign}(y_{s_i} - y_{s_j}) \log \ \text{sigmoid}(\text{score}(s_i) - \text{score}(s_j))$$

We use cross entropy loss as the loss function of feature predictions. We obtain probabilities of features appearing in the explanation by feeding the output node representations of features $\hat{X}_f$ into a linear layer and then a sigmoid layer. For the ith feature node, its probability is obtained as,

$$p(f_i) = \text{sigmoid}(\langle W_o^f, \hat{x}_{f_i} \rangle)$$

With the ground-truth label y_{f_i}, the loss of feature predictions is,

$$L_f = \sum_{i=1}^{M} y_{f_i} \log p(f_i)$$

Combining these two losses, we obtain the objective function as

$$L = \lambda L_s + (1 - \lambda) L_f$$

where λ is a hyper-parameter to control the weight of each loss in the objective function.

4.4.4 Sentence Extraction

Training & Testing. The sentences $S_g = S_u \bigcup S_c$ appearing in the training data are candidate sentences of an explanation. During training, the ground-truth sentences in an explanation are also a part of candidate sentences. During training, we follow Eq. 4.2 to obtain unnormalized relevance scores and then use these scores to obtain the pairwise loss to train the model.

During testing, the ground-truth sentences are excluded from the candidate sentences. we follow Eq. 4.2 to obtain unnormalized relevance scores and then we feed these scores into sigmoid function to obtain probabilities of sentences being included in an explanation. Based on the predicted probabilities, we rank sentences in descending order.

To reduce redundancy in an explanation, we propose a language model based strategy to select sentences. This strategy is to select sentences that do not have similar content and cover diverse features in an explanation. We use overlap of trigrams to evaluate the content similarity in sentences and use overlap of features to evaluate the diversity in explanations. The process of producing an explanation for a user-item pair is: we add the top-ranked sentence into the explanation and then we consider other sentences in the ranking order. We first look into whether the sentence has overlapping trigrams with other existing sentences in the explanation. If there is any overlapping trigram between the sentence and other existing sentences, we will skip this sentence and examine the next one. If there is not any overlapping trigram, we will the look into whether the sentence has overlapping features with other existing sentences in the explanation. If there is any overlapping features, we will skip this sentence and examine the next one. Otherwise, we will add this sentence into the explanation. We repeat the process until we have obtained k sentences in the explanation.

4.5 Experiments

In this section, we investigate the effectiveness of our proposed solution GREENer on producing explanations. We conduct experiments on two large datasets. We compare our model against a set of state-of-the-art baselines to illustrate its advantage. In addition, we also did ablation analysis to study the importance of components in GREENer.

4.5.1 Experiment Setup

Since GREENer focuses on producing explanations, in the experiments, we assume the recommended items are given. As GREENer is compatible with any recommendation algorithms, the recommended items can be provided by any recommendation algorithms. The datasets used for evaluation are Ratebeer dataset [121] and Yelp dataset [122]. Both of them contain reviews crawled from corresponding platforms, including user, item, text content and rating information. In the Ratebeer dataset, the rating range is $[0, 20]$. Since recommender systems generally recommend items that are attractive to users, the desired explanations are positive sentiments. Thus, we use reviews with ratings larger than 10 to construct the corpus for experiments. In the Yelp dataset, the rating range is $[1, 5]$. We use reviews with ratings larger than 3 to construct the corpus for experiments.

Data Processing. Reviews have been directly used as explanations in many previous work [18,20,22]. But as suggested in [123], a large portion of sentences in a review describes personal subjective experience, like "*I drank two bottles of this beer.*". These sentences do not provide any reason why the users like the items, which are not qualified as explanations. In contrast, sentences that serve as explanations should describe the features of items, thus helping users make informed decisions, like "*taste is of bubble gum and some banana.*". Therefore, we construct the explanation dataset by keeping informative sentences from reviews. Specifically. for the datasets, we use the Sentires toolkit [42] to extract feature words from reviews and manually filter out inappropriate ones based on domain knowledge. Then, for each review, we keep sentences that describe certain features of items as explanations.

We filter inactive users and unpopular items, i.e., we keep users who have written at least fifteen reviews and keep items which have been commented by users at least fifteen times. We keep 20,000 the most frequent words as the vocabulary. The statistics of the processed datasets are reported in Table 4.1. We split the dataset into training, validation, and testing dataset according to the ratio 70% : 15% : 15%.

Table 4.1: Summary of the processed datasets.

Dataset	# Users	# Items	# Reviews	# Sentences	# Features
Ratebeer	1,664	1,490	130,739	519,353	575
Yelp	4,604	7,837	191,227	602,572	498

Implementation Details. On both datasets, the dimension d_u of user embeddings is chosen from $\{128, 256, 512\}$. We set $d_u = 256$ in our experiments, as we did not observe further improvement of performance with higher dimensions. The dimension d_c of item embeddings is chosen from $\{128, 256, 512\}$ and we set $d_c = 256$. The dimension d_f of feature embeddings is chosen from $\{128, 256, 512\}$ and we set $d_f = 256$ due to its promising performance. The dimension d_s is 768. We have used $L = 2$ graph attention layers with $d_h = 4$ head. In the objective function, the hyper-parameter λ is set to 0.5. The dropout rates are all set to 0.2. The batch size is set to 64. We utilize Adam as the optimizer. We find that GREENer is sensitive to the learning rate. On Ratebeer dataset, the optimal learning rate is set to be 0.0001, which is chosen from $\{0.00001, 0.0001, 0.0005, 0.001\}$. On Yelp dataset, the learning rate is set to 0.0001. Our implementation is based on the pytorch geometric library [1].

Baselines. We compare our model with three baselines that can produce natural language sentences as explanations:

- **NARRE**: Neural Attentional Regression model with Review-level Explanations [20] is an extractive solution. It utilizes user and item representation to attend the reviews, and measures the relevance of the existing reviews with attention weights. It selects the review with largest attention weight as the explanation.

- **NRT**: Neural Rating and Tips Generation [57], is a generative solution. It is originally proposed for tip (a short summary) generation, but can be seamlessly adapted to generate explanations. It is based on RNN language model.

- **Attr2Seq**: Attribute-to-sequence model [57], is a generative solution. It combines attention with RNN to take the user, item and rating as input to generate an explanation. It places different weights to user, item and rating when generating every word in the explanation.

[1] https://pytorch-geometric.readthedocs.io/en/latest/

Table 4.2: Comparison of explanation quality by different methods.

Model	BLEU				Rouge		
	1	2	3	4	1	2	L
Ratebeer Dataset							
NRT	19.85	10.11	5.37	2.84	23.97	5.71	21.93
Attr2Seq	27.02	14.41	8.26	4.84	27.12	7.10	23.52
NARRE	24.23	8.95	3.45	1.38	27.20	3.37	23.85
GREENer	**33.95**	**16.89**	**9.16**	**5.28**	**35.58**	**8.12**	**32.14**
Yelp Dataset							
NRT	3.29	1.47	0.72	0.33	15.19	3.28	13.16
Attr2Seq	9.49	4.45	2.21	**1.08**	16.90	**3.77**	14.26
NARRE	17.85	5.96	2.05	0.75	24.53	2.56	20.33
GREENer	**20.86**	**7.22**	**2.58**	1.03	**27.04**	2.85	**22.64**

4.5.2 Quality of Generated Explanations

To comprehensively evaluate the quality of generated explanations, we measure the explanation quality with different types of metrics, including BLEU-1, 2, 3, 4 and Rouge score. The results are reported in Table 4.2.

GREENer outperforms baselines under every BLEU metric on the dataset. As BLEU is a precision based metric, a larger BLEU achieved by a model suggests that a larger portion of content in explanations produced by the model overlaps with the ground-truth explanations. Thus, the larger BLEU achieved by GREENer suggests that more content in explanations produced by GREENer is consistent with the users' perception of the items. The comparison of GREENer against NARRE shows though both methods are extractive methods, considering graph structure in data helps GREENer. The comparison of GREENer against NRT and Attr2Seq shows that the extractive method can produce higher quality of explanations than the generative method. Especially on Ratebeer dataset, GREENer achieves much larger BLEU-4 than others do, which further shows GREENer can produce explanations aligning with users' understanding of items.

Moreover, GREENer outperforms baselines under every Rouge metric on the dataset. As Rouge is a recall based metric, a larger Rouge achieved by a model suggests that more content in ground-truth explanations are included in the explanations produced by the model. Thus, the larger Rouge achieved by GREENer suggests that GREENer can recognize more content about users' perception of items than other methods do. This further demonstrates the effectiveness of GREENer. Especially GREENer achieves larger Rouge-L than other models do. This indicates that explanations produced by GREENer contain long phrases that match users' perceptions towards items.

Table 4.3: Comparison of explanation quality by GREENer with different number of selected sentences k.

k	BLEU				Rouge		
	1	2	3	4	1	2	L
Ratebeer Dataset							
3	21.53	11.69	6.98	4.25	23.48	6.91	21.42
4	31.24	15.84	8.76	5.13	31.71	7.53	29.06
5	**33.95**	**16.89**	**9.16**	**5.28**	**35.58**	**8.12**	**32.14**
Yelp Dataset							
2	14.02	4.41	1.52	0.60	16.48	1.46	14.63
3	21.45	7.15	2.51	0.99	22.26	2.17	19.02
4	**20.86**	**7.22**	**2.58**	**1.03**	**27.04**	**2.85**	**22.64**

4.5.3 Hyperparameter

As the number of selected sentences would influence the performance of GREENer, we vary the number of selected sentences to study the influence. On Ratebeer dataset the average number of sentences in an explanation is 3.4, while the average number of sentences on Yelp dataset is 2.5. We set the number of selected sentences k to be $3, 4, 5$ on Ratebeer dataset, while we set k to be $2, 3, 4$ on Yelp dataset. The results are reported in Table 4.3. We could observe when too few sentences are selected, i.e., $k = 3$ on Ratebeer dataset or $k = 2$ on Yelp dataset, the quality of explanations drops. This is because the extracted explanations cover less content in ground-truth explanations.

4.5.4 Ablation Analysis

We include three variants of our solution to study the contribution of each component to the performance of GREENer:

- $\neg$ *graph.* This variant excludes the graph structure. It does not consider user, item, feature and sentence are connected with graphs. We concatenate user, item, feature and sentence representations to predict whether a sentence is in an explanation. As multiple features appear in a sentence, we use the average pooling over the multiple feature representations for the sentence prediction. The comparison between this variant and GREENer demonstrates the importance of utilizing the graph structure among data to obtain relevance of sentences.

- $\neg$ *language model.* This variant excludes the language model based strategy when selecting sentences. It select sentences as explanations based on the relevance of sentences. The comparison between this variant and GREENer shows the importance of reducing redundancy.

Results are reported in Table 4.4. From the table, we can observe these variants perform worse than

Table 4.4: Comparison of explanation quality by different variants of GREENer

Model	BLEU				Rouge		
	1	2	3	4	1	2	L
Dataset	Ratebeer						
¬ graph	18.14	6.49	2.77	1.28	15.65	1.84	15.73
¬ language model	25.44	13.42	8.19	**5.34**	26.17	7.99	22.86
GREENer	**33.95**	**16.89**	**9.16**	5.28	**35.58**	**8.12**	**32.14**

Table 4.5: Example explanations produced by different models.

Model	Explanation
	Example 1
Ground-Truth	bottle 2010. dark cloudy amber colour. malty, sweet, toffee, dark fruit, licorice, a touch of alcohol, bitterness in the finish. feels rather light bodied for the abv.
NARRE	the finish is dry and ashy .
NART	flavor of chocolate , roasted malt , and light smoke .
GREENer	amber colour with a white head. aroma : malty & sweet, dried fruit, figs, caramel. bitter and roasted finish. a bit fruity, some biscuity malt, nuts, light bitterness. updated : jan 14, 2008 bottle at bishops arms in gothenburg.

GREENer. This observation suggests that utilizing the graph structure to obtain the relevance of sentences is important for GREENer. In addition, considering the similarities among sentences when selecting sentences as explanations is also important.

4.5.5 Case Study

We include example explanations produced by GREENer and other baselines in Table 4.5. The ground-truth explanations are also included for reference. From the table, we could see that the extracted sentences are close to the ground truth explanations. The features in extracted sentences match those in ground truth explanations. In the example, features “amber”, “colour”, and “malty” appear in both extracted sentences and explanations. With these features, the explanation can help the user have a clear personalized understanding of the item.

4.6 Conclusion

In this chapter, we study the problem of producing explanations for recommendations. The explanations produced by existing solutions are generic and repetitive. To address these two issues, we proposed GRaph Extractive ExplaiNer (GREENer) to extract sentences from the review corpus as explanations. GREENer utilizes relevance and similarity to select sentences as explanations. To produce explanations that describe users’ perception of items, GREENer considers the graph structure among user, item, feature and sentence to obtain the relevance of sentences. To reduce the redundancy in explanations, GREENer utilizes a language model based strategy to select relevant sentences as explanations. Experiments show that GREENer produces explanations that match users' perception of items more accurately than baselines do. We are the first one to leverage the graph structure to produce explanations for recommendations.

In this work, we use processed reviews as explanations. It would be meaningful to consider collecting large datasets of explanations by crowdsourcing. In addition, iterative communications between systems and users might further boost the utility of explanations on helping users understand the recommendations and make informed decision. Thus, it would be interesting to look into conversational explanations for recommendations.

Chapter 5

Conclusions and Future Work

Recommender systems have become indispensable for people to access information. The systems have shaped people's choices, from shopping, eating, to entertainment, etc. The great impact calls for effective recommendations and explanations for recommendations. With effective and transparent recommendations, users can make informed decisions, thus optimizing both users' goals and service providers' objectives.

In this thesis, we focus on explicitly exploiting the structure among feedback to improve the effectiveness and explainability of recommender systems. Various types of structure, like sequential structure and graph structure, are considered in proposed solutions. The utility of proposed solutions has been proved by extensive experiments on large datasets from online web service platforms. This thesis introduced new insights on developing effective and explainable recommender systems, e.g., exploiting the structural information among data. In this chapter, we will first summarize the thesis and discuss future work.

5.1 Conclusions

In this thesis, we proposed solutions to make use of the structure among feedback to improve the effectiveness and transparency of recommender systems. To improve the effectiveness of recommendations, we proposed solutions to address the problem of contextualized recommendations, i.e., making recommendations to users under different context. Treating occurring time of actions and item category of actions as context, proposed solutions consider contextualized sequential structure among actions to recommend items to users under different context. To improve the transparency of recommendations, we proposed solutions to tackle the problem of providing explanations for recommended items. Proposed solutions consider the graph structure among user-written content (e.g., sentences in reviews) to extract sentences from reviews as explanations. We have conducted extensive experiments to investigate the utility of proposed solutions on large datasets. The comparison against state-of-the-art baselines demonstrate that our proposed solutions for contextualized recommendations can enhance the recommendation quality. Additionally, the proposed solution for providing explanations for recommendations can improve the quality of explanations. In the following, we summarize each part of the thesis.

Temporal Contextualized Recommendation. In this part, treating occurring time of actions as context, the proposed solutions address the critical problem in contextualized recommendations that the user preferences based on historical actions varies over time. To predict the next item, we consider the time-aware sequential structure among actions. In the first proposed solution, Long- and Short-term Hawkes Process, the time-aware sequential structure is instantiated as, actions within the same session have short-term sequential dependence mutually, while actions associated with the same item across sessions have long-term sequential dependence. In addition, both of these two types of dependence decay over time. Experiments show that the long-term and short-term time-aware sequential structure helps LSHP improve the recommendation quality. Concerning that the decaying rates of influence from historical actions are diverse while they are fixed in LSHP, we proposed the second solution, Contextualized Temporal Attention. In CTA, the time-aware sequential structure is instantiated as the sequential dependence among actions is determined by both gaps between occurring time of actions and local sensitivity and seriousness of actions. The influence from historical actions decays with a mixture of multiple rates. Experiments show that CTA improves the recommendation quality by considering contextualized time-aware sequential structure. These solutions provide insights in effectively utilizing the temporal information to improve recommendation quality.

Category-aware Contextualized Recommendation. Treating the item category of an action as the action's context, we proposed CoCoRec considering the category-aware sequential structure for recommendations. Supported by statistical analyses, we instantiate the sequential structure as, actions in the in-category subsequence (i.e., the subsequence containing actions of the same item category) have sequential dependence. In addition, other users' in-category subsequences suggest the next action of the target user. Experiments and ablation analyses on two large datasets show that category-aware sequential structure helps CoCoRec improve its recommendation quality. This solution sheds light on utilizing item category information to enhance the effectiveness of recommendations.

Graph Extractive Explainer. We worked on the task of providing nature language explanations for recommendations. The problem we addressed in this work is nature language explanations are not personalized to a recommended item and an user who perceive the recommendation. We proposed the solution GRaph Extractive ExplaiNer (GREENer) extracting sentences from the review corpus as explanations. To model user preferences, GREENer utilizes the co-occurrence of user-feature. To model item properties, GREENer utilizes the co-occurrence of item-feature. To fuse user preferences and item properties into the sentence extraction, GREENer utilizes the co-occurrence of feature-sentence. By constructing a graph connecting the user, the item, sentences in reviews and feature words in sentences, GREENer takes advantage of these three types of co-occurrence. Experiments show that the graph structure helps GREENer produce sentences that match users' opinions towards items more accurately than state-of-the-art methods do. This solution brings new perspective into the study of improving the transparency of recommendations.

5.2 Future Work

5.2.1 Short-term Future Work

Improving the effectiveness and explainability of recommender systems is demanding. Some interesting extensions of our proposed solutions are worth considering as future work.

In the proposed solutions for temporal contextualized recommendations, we did not consider signals conveyed by different types of actions vary. When interacting with online service platforms, users generate different types of actions, like clicking items or purchasing items on e-commerce websites. Clicks may be generated casually, while purchases are generated seriously (excluding malicious behaviors). Compared with clicks, purchases can tell user preferences with high confidence. With this interpretation of action types, the influence from purchases would decay in a slower rate than clicks. On the other hand, because users would generally not continue looking for the recently purchased items while users may repeat clicking recently clicked items, the influence from purchases would decay in faster rate than clicks. Differentiating action types brings in new understanding of the influence from these actions. These signals call for new solutions considering the time-aware sequential structure. In addition, the proposed solutions did not leverage other users' actions for collaborative learning. Sparsity of actions is still a challenge temporal contextualized recommendations face. As we have shown in the work for category-aware contextualized recommendations, collaborative learning can mitigate the sparsity issue, thus improving the recommendation quality. How to do collaborative learning for temporal contextualized recommendations is worth further attention in the future.

In the proposed solution for category-aware contextualized recommendations, we did not consider the influence from past actions decays over time. We have shown that considering the decaying effect of influence over time is useful to improve the recommendation quality. Therefore, it would be worth considering the decaying effect for category-aware contextualized recommendations. In addition, other types of proxies to context are worth exploring. we have explored the occurring time of actions and the item category of actions as context. Other meta data associated with actions, like location, also provides critical information about user preferences. Moreover, considering multiple types of proxies together in a solution may improve the recommendation quality further. We have worked on solutions by considering individual type of proxy. As different types of context enable different interpretations of actions, jointly modeling multiple types of context may provide a comprehensive picture of user preferences.

In the proposed solution for graph based extractive explainer for recommendations, we did not consider edge weights in the graph. As edge weights are correlated with the importance of nodes, modeling edge weights may enable the solution to capture co-occurrence patterns more effectively than the current solution does. In addition, the sentences which appear in the same reviews are considered separately in the proposed solution. Probably sentences in the same reviews can suggest each other. Their special correlations are not modeled. It would be worth studying this in the future.

5.2.2 Long-term Future Work

Because of the pervasiveness of recommender systems on people's lives, it is demanding to care about the social impact of recommendations. This involves effective, explainability, fairness, and privacy of recommender systems. In this thesis, we have studied the explainability of recommender systems. Both the fairness and the privacy of recommender systems call for further attentions.

Fairness. Recommender systems are trained to capture patterns in data. The bias in data would lead to biased recommender systems, like female users are associated with low-paid jobs. Consequently, the recommender systems provide biased recommendations, e.g., recommending low-paid jobs to female users. The definitions of fairness are various with different concerns and not standardized. Since recommender systems involve at least two parties, i.e., users and items, the definitions of fairness in recommender systems become more diverse. Concerning users' fairness in recommender systems, some commonly accepted definitions include: the probabilities of recommending items to users should not dependent on users' sensitive attributes, like race or gender, i.e., demographic parity; the probabilities of recommending relevant items to users should not dependent on users' sensitive attributes, i.e., equal opportunity. Likewise, there are corresponding definitions emphasizing items' fairness. Adopting which definition depends on specific applications. Moreover, the bias issues in recommender systems are not tolerable. Mitigating the bias is of great urgency. Conventionally, when training recommender systems, we develop objective functions encouraging recommender systems to fit the data. Concerning the bias in data, it is necessary to include fairness related losses into objective functions. Thus, the recommender systems are trained to care about both the effectiveness and the fairness of recommendations.

To make the recommender systems fair, a large amount of research problems need to be addressed. Some interesting problems include how to mitigate the popularity bias of items and how to mitigate the activity bias of users. Popularity bias of items refer to the observation that popular items tend to be recommended more accurately than unpopular items. This type of bias prevents unpopular items from being recommended accurately. Consequently items which are new to the recommender systems would hardly be accessed by others. Activity bias of users refer to the observation that active users tend to have higher quality recommendations than inactive users do. This type of bias discourages inactive users to have wonderful experience in accessing information.

The insight about the structure among user feedback in this thesis can help address the fairness problems in recommender systems. For example, for the popularity bias of items, we could utilize the sequential structure among feedback to infer the correlation between popular items and unpopular items. Therefore, we can improve the recommendation quality of popular items.

Privacy. Data used to train recommender systems is composed of user interactions. The user interactions can be used to infer user preferences. On the other hand, they can be used to infer users' identities. This would cause privacy issues, e.g., private information leakage. What is worse, service providers keep the data on their side, which allows malicious third parties to misuse the data. Privacy preserving recommender systems are demanding. Differential Privacy (DP) aims at solving the problem in recommender systems that other users' information is leaked to improve the target user's recommendation quality. For example, in Netflix, to recommend movies to a user, both this user's historical ratings and other users' historical ratings are used. To mitigate the leakage of other users' information, DP based solutions add noise to the data [124]. Although these solutions achieve encouraging results in preserving privacy, the recommendation quality drops a lot. In addition, to avoid keeping all user data on the service providers' side, federated learning has been studied for recommender systems. However, so far the existing solutions sacrifice the recommendation quality dramatically. This thesis may shed some light on addressing the problem. Effectively utilizing the structural information among a user's own feedback can save the model from relying on utilizing other user's private information to improve the recommendation quality.

Developing effective, explainable, fair and privacy preserving recommender systems requires long-term and interdisciplinary efforts. Ultimately, we would like the recommender systems make the world better.

Bibliography

[1] Dietmar Jannach and Michael Jugovac. Measuring the business value of recommender systems. *ACM Transactions on Management Information Systems (TMIS)*, 10(4):1–23, 2019.

[2] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38, 2019.

[3] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer, 2011.

[4] Gediminas Adomavicius, Ramesh Sankaranarayanan, Shahana Sen, and Alexander Tuzhilin. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Transactions on Information systems (TOIS)*, 23(1):103–145, 2005.

[5] Quan Yuan, Gao Cong, Zongyang Ma, Aixin Sun, and Nadia Magnenat Thalmann. Time-aware point-of-interest recommendation. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 363–372, 2013.

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

[7] Steffen Rendle, Zeno Gantner, Christoph Freudenthaler, and Lars Schmidt-Thieme. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 635–644, 2011.

[8] Linas Baltrunas, Bernd Ludwig, and Francesco Ricci. Matrix factorization techniques for context aware recommendation. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 301–304, 2011.

[9] Yifei Ma, Balakrishnan Narayanaswamy, Haibin Lin, and Hao Ding. Temporal-contextual recommendation in real-time. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2291–2299, 2020.

[10] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[11] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 130–137. ACM, 2017.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[13] Ajay Shrestha and Ausif Mahmood. Review of deep learning algorithms and architectures. *IEEE Access*, 7:53040–53065, 2019.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[15] Yu Zhu, Hao Li, Yikang Liao, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai. What to do next: Modeling user behaviors by time-lstm. In *IJCAI*, pages 3602–3608, 2017.

[16] Yongfeng Zhang, Guokun Lai, Min Zhang, Yi Zhang, Yiqun Liu, and Shaoping Ma. Explicit factor models for explainable recommendation based on phrase-level sentiment analysis. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 83–92, 2014.

[17] Nan Wang, Hongning Wang, Yiling Jia, and Yue Yin. Explainable recommendation via multi-task learning in opinionated text data. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 165–174, 2018.

[18] Xiting Wang, Yiru Chen, Jie Yang, Le Wu, Zhengtao Wu, and Xing Xie. A reinforcement learning framework for explainable recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 587–596. IEEE, 2018.

[19] Yiyi Tao, Yiling Jia, Nan Wang, and Hongning Wang. The fact: Taming latent factor models for explainability with factorization trees. In *Proceedings of the 42Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 295–304, New York, NY, USA, 2019. ACM.

[20] Chong Chen, Min Zhang, Yiqun Liu, and Shaoping Ma. Neural attentional rating regression with review-level explanations. In *Proceedings of the 2018 World Wide Web Conference*, pages 1583–1592, 2018.

[21] Quoc-Tuan Truong and Hady Lauw. Multimodal review generation for recommender systems. In *The World Wide Web Conference*, pages 1864–1874, 2019.

[22] Peijie Sun, Le Wu, Kun Zhang, Yanjie Fu, Richang Hong, and Meng Wang. Dual learning for explainable recommendation: Towards unifying user preference prediction and review generation. In *Proceedings of The Web Conference 2020*, pages 837–847, 2020.

[23] Li Dong, Shaohan Huang, Furu Wei, Mirella Lapata, Ming Zhou, and Ke Xu. Learning to generate product reviews from attributes. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 623–632, 2017.

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

[25] Jiaxuan You, Yichen Wang, Aditya Pal, Pong Eksombatchai, Chuck Rosenberg, and Jure Leskovec. Hierarchical temporal convolutional networks for dynamic recommender systems. *arXiv preprint arXiv:1904.04381*, 2019.

[26] Qiang Cui, Shu Wu, Yan Huang, and Liang Wang. A hierarchical contextual attention-based gru network for sequential recommendation. *arXiv preprint arXiv:1711.05114*, 2017.

[27] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 197–206. IEEE, 2018.

[28] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, pages 565–573, New York, NY, USA, 2018. ACM.

[29] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. Sequential recommendation with user memory networks. In *Proceedings of the eleventh ACM international conference on web search and data mining*, pages 108–116. ACM, 2018.

[30] Renqin Cai, Xueying Bai, Zhenrui Wang, Yuling Shi, Parikshit Sondhi, and Hongning Wang. Modeling sequential online interactive behaviors with temporal point process. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 873–882. ACM, 2018.

[31] Jiaxi Tang, Francois Belletti, Sagar Jain, Minmin Chen, Alex Beutel, Can Xu, and Ed H Chi. Towards neural mixture recommender for long range dependent user sequences. *arXiv preprint arXiv:1902.08588*, 2019.

[32] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. Latent cross: Making use of context in recurrent recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 46–54, 2018.

[33] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. *arXiv preprint arXiv:1904.06690*, 2019.

[34] Ron Begleiter, Ran El-Yaniv, and Golan Yona. On prediction using variable order markov models. *Journal of Artificial Intelligence Research*, 22:385–421, 2004.

[35] Yuling Shi, Zhiyong Peng, and Hongning Wang. Modeling student learning styles in moocs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 979–988. ACM, 2017.

[36] Jacques Janssen and Nikolaos Limnios. *Semi-Markov models and applications*. Springer Science & Business Media, 2013.

[37] Alan G Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971.

[38] Dixin Luo, Hongteng Xu, Yi Zhen, Xia Ning, Hongyuan Zha, Xiaokang Yang, and Wenjun Zhang. Multi-task multi-dimensional hawkes processes for modeling event sequences. In *IJCAI*, pages 3685–3691, 2015.

[39] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. Recurrent recommender networks. In *Proceedings of the tenth ACM international conference on web search and data mining*, pages 495–503, 2017.

[40] Lee Ryan, Christine Cox, Scott M Hayes, and Lynn Nadel. Hippocampal activation during episodic and semantic memory retrieval: Comparing category production and category cued recall. *Neuropsychologia*, 46(8):2109–2121, 2008.

[41] Endel Tulving et al. Episodic and semantic memory. *Organization of memory*, 1:381–403, 1972.

[42] Mengting Wan, Di Wang, Matt Goldman, Matt Taddy, Justin Rao, Jie Liu, Dimitrios Lymberopoulos, and Julian McAuley. Modeling consumer preferences and price sensitivities from large-scale grocery shopping transaction logs. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1103–1112. International World Wide Web Conferences Steering Committee, 2017.

[43] Caroline Lo, Dan Frankowski, and Jure Leskovec. Understanding behaviors that lead to purchasing: A case study of pinterest. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 531–540. ACM, 2016.

[44] Jinyoung Yeo, Sungchul Kim, Eunyee Koh, Seung-won Hwang, and Nedim Lipka. Predicting online purchase conversion for retargeting. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 591–600. ACM, 2017.

[45] Ashton Anderson, Daniel Huttenlocher, Jon Kleinberg, and Jure Leskovec. Engaging with massive online courses. In *Proceedings of the 23rd international conference on World wide web*, pages 687–698. ACM, 2014.

[46] Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. Recurrent marked temporal point processes: Embedding event history to vector. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1555–1564. ACM, 2016.

[47] Hongning Wang, Yang Song, Ming-Wei Chang, Xiaodong He, Ahmed Hassan, and Ryen W White. Modeling action-level satisfaction for search task satisfaction prediction. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 123–132. ACM, 2014.

[48] Amanda Spink, Minsoo Park, Bernard J Jansen, and Jan Pedersen. Multitasking during web search sessions. *Information Processing & Management*, 42(1):264–275, 2006.

[49] Claudio Lucchese, Salvatore Orlando, Raffaele Perego, Fabrizio Silvestri, and Gabriele Tolomei. Identifying task-based sessions in search engine query logs. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 277–286. ACM, 2011.

[50] Wendy W Moe. Buying, searching, or browsing: Differentiating between online shoppers using in-store navigational clickstream. *Journal of consumer psychology*, 13(1-2):29–39, 2003.

[51] Hongning Wang, Yang Song, Ming-Wei Chang, Xiaodong He, Ryen W White, and Wei Chu. Learning to extract cross-session search tasks. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1353–1364. ACM, 2013.

[52] Alexander Kotov, Paul N Bennett, Ryen W White, Susan T Dumais, and Jaime Teevan. Modeling and analysis of cross-session search tasks. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 5–14. ACM, 2011.

[53] Rosie Jones and Kristina Lisa Klinkner. Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 699–708. ACM, 2008.

[54] Eugene Agichtein, Ryen W White, Susan T Dumais, and Paul N Bennet. Search, interrupted: understanding and predicting search task continuation. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 315–324. ACM, 2012.

[55] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 811–820. ACM, 2010.

[56] Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2010.

[57] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1419–1428. ACM, 2017.

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

[59] Zihang Dai, Zhilin Yang, Yiming Yang, William W Cohen, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[60] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report.

[61] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 447–456. ACM, 2009.

[62] Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff Schneider, and Jaime G Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the 2010 SIAM international conference on data mining*, pages 211–222. SIAM, 2010.

[63] Yang Song, Ali Mamdouh Elkahky, and Xiaodong He. Multi-rate deep learning for temporal recommendation. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 909–912. ACM, 2016.

[64] Lei Li, Li Zheng, Fan Yang, and Tao Li. Modeling and broadening temporal user interest in personalized news recommendation. *Expert Systems with Applications*, 41(7):3168–3177, 2014.

[65] Yang Li, Nan Du, and Samy Bengio. Time-dependent representation for neural event sequence prediction. *arXiv preprint arXiv:1708.00065*, 2017.

[66] Samarjit Das. *Time series analysis*. Princeton University Press, Princeton, NJ, 1994.

[67] Louis L Scharf and Cédric Demeure. *Statistical signal processing: detection, estimation, and time series analysis*, volume 63. Addison-Wesley Reading, MA, 1991.

[68] Patrick J Laub, Thomas Taimre, and Philip K Pollett. Hawkes processes. *arXiv preprint arXiv:1507.02822*, 2015.

[69] Hongteng Xu and Hongyuan Zha. A dirichlet mixture model of hawkes processes for event sequence clustering. In *Advances in Neural Information Processing Systems*, pages 1354–1363, 2017.

[70] Bjørnar Vassøy, Massimiliano Ruocco, Eliezer de Souza da Silva, and Erlend Aune. Time is of the essence: a joint hierarchical rnn and point process model for time and item predictions. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 591–599. ACM, 2019.

[71] Shuai Xiao, Junchi Yan, Xiaokang Yang, Hongyuan Zha, and Stephen M Chu. Modeling the intensity function of point process via recurrent neural networks. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[72] Hongyuan Mei and Jason M Eisner. The neural hawkes process: A neurally self-modulating multivariate point process. In *Advances in Neural Information Processing Systems*, pages 6754–6764, 2017.

[73] Martin Arlitt. Characterizing web user sessions. *ACM SIGMETRICS Performance Evaluation Review*, 28(2):50–63, 2000.

[74] Yosihiko Ogata. Statistical models for earthquake occurrences and residual analysis for point processes. *Journal of the American Statistical association*, 83(401):9–27, 1988.

[75] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[76] Hua Ouyang, Niao He, Long Tran, and Alexander Gray. Stochastic alternating direction method of multipliers. In *International Conference on Machine Learning*, pages 80–88, 2013.

[77] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

[78] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[79] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[80] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[81] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[82] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[83] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 452–461, Arlington, Virginia, United States, 2009. AUAI Press.

[84] Balázs Hidasi and Alexandros Karatzoglou. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 843–852. ACM, 2018.

[85] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[86] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[87] Andrzej Pacuk, Piotr Sankowski, Karol Wegrzycki, Adam Witkowski, and Piotr Wygocki. Recsys challenge 2016: Job recommendations based on preselection of offers and gradient boosting. In *Proceedings of the Recommender Systems Challenge*, RecSys Challenge '16, pages 10:1–10:4, New York, NY, USA, 2016. ACM.

[88] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. Learning tree-based deep model for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1079–1088. ACM, 2018.

[89] Jing He, Xin Li, and Lejian Liao. Category-aware next point-of-interest recommendation via listwise bayesian personalized ranking. In *IJCAI*, volume 17, pages 1837–1843, 2017.

[90] Jin Huang, Zhaochun Ren, Wayne Xin Zhao, Gaole He, Ji-Rong Wen, and Daxiang Dong. Taxonomy-aware multi-hop reasoning networks for sequential recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 573–581. ACM, 2019.

[91] Qinyong Wang, Hongzhi Yin, Zhiting Hu, Defu Lian, Hao Wang, and Zi Huang. Neural memory streaming recommender networks with adversarial training. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2467–2475. ACM, 2018.

[92] Jianling Wang and James Caverlee. Recurrent recommendation with local coherence. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 564–572. ACM, 2019.

[93] Mingxiao An, Fangzhao Wu, Chuhan Wu, Kun Zhang, Zheng Liu, and Xing Xie. Neural news recommendation with long-and short-term user representations. In *Proceedings of the 57th Conference of the Association for Computational Linguistics*, pages 336–345, 2019.

[94] Dietmar Jannach and Malte Ludewig. When recurrent neural networks meet the neighborhood for session-based recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 306–310, 2017.

[95] Badrul Munir Sarwar, George Karypis, Joseph A Konstan, John Riedl, et al. Item-based collaborative filtering recommendation algorithms. *Www*, 1:285–295, 2001.

[96] Zhiqiang Pan, Fei Cai, Yanxiang Ling, and Maarten de Rijke. An intent-guided collaborative machine for session-based recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1833–1836, 2020.

[97] Meirui Wang, Pengjie Ren, Lei Mei, Zhumin Chen, Jun Ma, and Maarten de Rijke. A collaborative session-based recommendation approach with parallel memory modules. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 345–354, 2019.

[98] Nengjun Zhu, Jian Cao, Yanchi Liu, Yang Yang, Haochao Ying, and Hui Xiong. Sequential modeling of hierarchical user intention and preference for next-item recommendation. In *Proceedings of the Thirteenth ACM International Conference on Web Search and Data Mining.* ACM, 2020.

[99] Jibang Wu, Renqin Cai, and Hongning Wang. Déjà vu: A contextualized temporal attention mechanism for sequential recommendation. In *Proceedings of The Web Conference 2020*, pages 2199–2209, 2020.

[100] Weiping Song, Zhiping Xiao, Yifan Wang, Laurent Charlin, Ming Zhang, and Jian Tang. Session-based social recommendation via dynamic graph attention networks. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 555–563, 2019.

[101] Qian Zhao, Jilin Chen, Minmin Chen, Sagar Jain, Alex Beutel, Francois Belletti, and Ed Chi. Categorical-attributes-based multi-level classification for recommender systems. 2018.

[102] Ruiyang Ren, Zhaoyang Liu, Yaliang Li, Wayne Xin Zhao, Hui Wang, Bolin Ding, and Ji-Rong Wen. Sequential recommendation with self-attentive multi-adversarial network. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 89–98, 2020.

[103] Hao Hou and Chongyang Shi. Explainable sequential recommendation using knowledge graphs. In *Proceedings of the 5th International Conference on Frontiers of Educational Technologies*, pages 53–57, 2019.

[104] Chenyang Wang, Min Zhang, Weizhi Ma, Yiqun Liu, and Shaoping Ma. Make it a chorus: knowledge-and time-aware item modeling for sequential recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 109–118, 2020.

[105] Yulong Gu, Zhuoye Ding, Shuaiqiang Wang, and Dawei Yin. Hierarchical user profiling for e-commerce recommender systems. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 223–231, 2020.

[106] Jiacheng Li, Yujie Wang, and Julian McAuley. Time interval aware self-attention for sequential recommendation. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 322–330, 2020.

[107] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 565–573. ACM, 2018.

[108] Walid Krichene and Steffen Rendle. On sampled metrics for item recommendation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1748–1757, 2020.

[109] Aobo Yang, Nan Wang, Hongbo Deng, and Hongning Wang. Explanation as a defense of recommendation. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 1029–1037, 2021.

[110] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[111] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[112] Charu C Aggarwal et al. *Recommender systems*, volume 1. Springer, 2016.

[113] Mustafa Bilgic and Raymond J Mooney. Explaining recommendations: Satisfaction vs. promotion. In *Beyond Personalization Workshop, IUI*, volume 5, 2005.

[114] Jonathan L Herlocker, Joseph A Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 241–250. ACM, 2000.

[115] Rashmi Sinha and Kirsten Swearingen. The role of transparency in recommender systems. In *CHI'02 extended abstracts on Human factors in computing systems*, pages 830–831. ACM, 2002.

[116] Trung-Hoang Le and Hady W Lauw. Explainable recommendation with comparative constraints on product aspects. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 967–975, 2021.

[117] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[118] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, pages 2022–2032, 2019.

[119] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, pages 2704–2710, 2020.

[120] Danqing Wang, Pengfei Liu, Yining Zheng, Xipeng Qiu, and Xuanjing Huang. Heterogeneous graph neural networks for extractive document summarization. *arXiv preprint arXiv:2004.12393*, 2020.

[121] Julian McAuley, Jure Leskovec, and Dan Jurafsky. Learning attitudes and attributes from multi-aspect reviews. In *Proceedings of the 2012 IEEE 12th International Conference on Data Mining*, ICDM '12, page 1020–1025, USA, 2012. IEEE Computer Society.

[122] Hongning Wang, Yue Lu, and Chengxiang Zhai. Latent aspect rating analysis on review text data: a rating regression approach. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 783–792, 2010.

[123] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 188–197, 2019.

[124] Frank McSherry and Ilya Mironov. Differentially private recommender systems: Building privacy into the netflix prize contenders. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 627–636, 2009.

[125] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[126] Hongning Wang, Yue Lu, and ChengXiang Zhai. Latent aspect rating analysis without aspect keyword supervision. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 618–626, 2011.

[127] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[128] Xiangnan He, Tao Chen, Min-Yen Kan, and Xiao Chen. Trirank: Review-aware explainable recommendation by modeling aspects. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1661–1670, 2015.

[129] Qingyao Ai, Vahid Azizi, Xu Chen, and Yongfeng Zhang. Learning heterogeneous knowledge base embeddings for explainable recommendation. *Algorithms*, 11(9):137, 2018.

[130] Chenliang Li, Cong Quan, Li Peng, Yunwei Qi, Yuming Deng, and Libing Wu. A capsule network for recommendation and explaining what you like and dislike. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 275–284, 2019.

[131] Yongfeng Zhang and Xu Chen. Explainable recommendation: A survey and new perspectives. *arXiv preprint arXiv:1804.11192*, 2018.

[132] Junjie Li, Xuepeng Wang, Dawei Yin, and Chengqing Zong. Attribute-aware sequence network for review summarization. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2991–3001, 2019.

[133] Trung-Hoang Le, Hady W Lauw, and C Bessiere. Synthesizing aspect-driven recommendation explanations from reviews. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 2427–2434, 2020.

[134] Yi Xie and Shun-Zheng Yu. A large-scale hidden semi-markov model for anomaly detection on user browsing behaviors. *IEEE/ACM Transactions on Networking (TON)*, 17(1):54–65, 2009.

[135] Yuyu Zhang, Hanjun Dai, Chang Xu, Jun Feng, Taifeng Wang, Jiang Bian, Bin Wang, and Tie-Yan Liu. Sequential click prediction for sponsored search with recurrent neural networks. In *AAAI*, volume 14, pages 1369–1375, 2014.

[136] Mehrdad Farajtabar, Yichen Wang, Manuel Gomez Rodriguez, Shuang Li, Hongyuan Zha, and Le Song. Coevolve: A joint point process model for information diffusion and network co-evolution. In *Advances in Neural Information Processing Systems*, pages 1954–1962, 2015.

[137] Ke Zhou, Hongyuan Zha, and Le Song. Learning social infectivity in sparse low-rank networks using multi-dimensional hawkes processes. In *Artificial Intelligence and Statistics*, pages 641–649, 2013.

[138] Marian-Andrei Rizoiu, Swapnil Mishra, Quyu Kong, Mark Carman, and Lexing Xie. Sir-hawkes: Linking epidemic models and hawkes processes to model diffusions in finite populations. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 419–428. International World Wide Web Conferences Steering Committee, 2018.