# Towards Recommendation with User Action Sequences

by

**Jiaxi Tang**

B.Eng., Wuhan University, 2015

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Science


**SIMON FRASER UNIVERSITY**
**Fall 2019**

# Approval

**Name:** **Jiaxi Tang**

**Degree:** **Doctor of Philosophy (Computing Science)**

**Title:** **Towards Recommendation with User Action Sequences**

**Examining Committee:**

**Chair:** Qianping Gu
Professor
School of Computing Science

**Ke Wang**
Senior Supervisor
Professor
School of Computing Science

**Martin Ester**
Supervisor
Professor
School of Computing Science

**Greg Mori**
Internal Examiner
Professor
School of Computing Science

**James Caverlee**
External Examiner
Professor
Department of Computer Science and Engineering
Texas A&M University

**Date Defended:** **December 18, 2019**

# Abstract

Across the web and mobile applications, recommender systems are relied upon to surface the right items to users at the right time. This implies user preferences are usually dynamic in real-world recommender systems, and a user's historical action records are not equally important when predicting her/his future preferences. Most existing recommendation algorithms, including both shallow and deep approaches, usually treat all user's historical actions equally, which may have lost order information between actions.

In this thesis, we study the problem of modeling user action sequences for recommendation (a.k.a sequential recommendation). Motivated by the distinct challenges when modeling user sequences, we focus on building sequential recommendation models to capture various types of dependencies (sequential patterns). In particular, the dependencies can be in different forms. Also, they can either from the local part or the long-tail of user sequences. Though usually neglected in existing approaches, these dependencies are informative for accurate prediction of user preference.

In our first work, we discover the dependencies in real user sequences can have two different forms: point-level and union-level. We propose a unified model to jointly capture both forms of sequential patterns.

In our next work, we analyze the property of dependency from different temporal ranges of long user sequences. Based on our observation, we propose a neural mixture model as a tailored solution to deal with dependencies from all temporal ranges.

Finally, inference efficiency is critical for each model since recommendation is an online service. It is particularly important for sequential recommendation as user's sequence frequently changes and inference is needed with the new sequence. We provide a knowledge transfer framework to satisfy the efficiency requirement for recommendation models. We show this framework can be used to learn a compact recommendation model with better inference efficiency but with the similar efficacy of a large model. Our proposed solution can be also used for other ranking problems.

**Keywords:** Recommender System; User Modeling; Sequential Prediction; Neural Networks

# Dedication

*Dedicate to my parents and my love.*

# Acknowledgements

First, I would like to thank my senior advisor Dr. Ke Wang, for his supervision throughout my study at SFU. I am grateful not only for the trust he gave me on leaving me enough independence, but also for consistent support and pushing me for being a better researcher. Besides, I would like to thank Dr. Martin Ester, Dr. Greg Mori, Dr. James Caverlee for serving my thesis committee and providing constructive suggestions on my thesis. Thanks Dr. Qianping Gu for charing my thesis defence.

Thanks to my host at Google Research, Sagar Jain, for choosing me to be your intern for two summers. Without you I couldn't imagine I have the capability to work on cutting-edge problems from industry. Also, I want to thank Francois Belletti, Rakesh Shivanna, Zhe Zhao, Ed Chi and all other people from the same team, I feel really enjoyable to work with you.

Thanks to all my collaborators, labmates, and friends in SFU. Especially thanks to for their support and suggestions on my research and my daily life.

Finally, I would like to express my special thanks to my parents for their continuous and unconditional encouragement and love.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

This dissertation focuses on exploiting the sequence of actions that each user left in the system, to provide better personalized experiences using machine learning. The goal is to pinpoint the opportunities when using user action sequences for recommendation which have been widely recognized by academia and industry recently. We analyze the unique challenges for recommendation with user action sequences (a.k.a sequential recommendation) and introduce techniques based on machine learning to overcome such challenges. The work presented in this dissertation take several essential steps to transit static user modeling approach to a new generation of more dynamic personalization methodology.

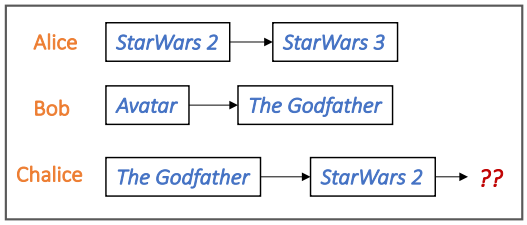
**Why recommendation with user action sequences?** To show our motivation for sequential recommendation, we'd like to discuss the importance of recommendation research in general and the specific opportunities for the recommendation research with user action sequences.

First of all, we care about the problem of recommendation in this dissertation since it enables better content discovery and browsing experience: Users can enjoy accurate and personalized recommendations without the need of effort to actively retrieve information from the web. In industry, recommender system has become a core technology and even a main user interface in large e-commerce, streaming and social media platforms [33, 62, 23]. As a result, it contributed a large proportion of the traffic and revenue to these online services. One could think that this either is a solved problem or merely need attention from industry, but this is far from the truth. In fact, the evolution of recommendation largely depend on research contribution from academia [38, 57, 77, 82]. There are many facets of the recommendation which have received little or no attention. Thus, there is a large untapped potential for improving user experience, that this thesis attempts to shed light on.

Conventional recommendation techniques depend on static user-item feedback, which is usually represented as a user-item interaction matrix as shown in Figure 1.1a. In this formulation, recommendation can be viewed as a matrix completion problem to match user long-term preference. Sequential recommendation takes a step forward to consider order

| | Avatar | The Godfather | StarWars 2 | StarWars 3 |
|---|---|---|---|---|
| Alice | | | 4 | 4 |
| Bob | 5 | 4 | | |
| Chalice | | 4 | 5 | |

(a) An example user-item interaction matrix.




(b) An example user action sequences.

Figure 1.1: Example showing the differences between interaction matrix and user action sequence. In this example, if we want to make a recommendation for Chalice, only knowing the interaction matrix cannot differentiate the choice between *Avatar* and *StarWar 3*. However, once knowing the sequential information, we know *StarWar 3* is better choice over *Avatar*.

sensitivity in user actions. It views recommendation as a sequence modeling problem as shown in Figure 1.1b. The order of actions is commonly logged by real systems. In real-world scenarios, drifts of individual interest can happen within a short time period (e.g., a user is interested in phone accessories soon after buying an iPhone). On the other hand, items can hold certain relationship and are usually consumed in the certain order (e.g., most users prefer to watch *StarWars 3* after finishing *StarWars 2*). That's the reason why sequential recommendation could make a huge difference to better understand user need in the "near future".

**Why Using Machine Learning based Approaches?** Recommendation methods start from neighborhood-based approaches [82]. We calculate item-item similarities and recommend a user with items similar to the items she/he has consumed. However, a good similarity measurement with heavy feature engineering still cannot properly model complex user-item relations. Thus, people turned to find more effective approaches – model-based approaches powered by machine learning.

Since the 2009 Netflix Prize competition, latent factor models [57] shown to have much better performance than neighborhood-based methods and it 'automatically' learns user/item representations from low-dimensional latent space. Recently, with the great impact of deep learning on computer vision [58, 51], a new type of model using deep neural networks has shown strong performances. Better harnessing the power of rich data, neural networks have an incredible ability to automatically capture high-level features and learn feature interactions. Recent works [38, 97] show the success for deep neural networks for recommendation with static user feedback.

Back to sequential recommendation, using sequence model allows us to easily incorporate user dynamics when making recommendations. Moreover, promising results in natural language processing with neural networks [67, 53] suggest deep learning's potential benefits for better modeling dependencies (patterns) within user action sequences.

## 1.1 Research Questions and Contributions

From the above introduction, one could think merely applying deep neural networks on user sequences can achieve considerable performance. However, this naive assumption largely overlooked the challenges existing in sequential recommendation. In what follows, we pinpoint these unique challenges under the context of machine learning (especially deep learning), which can be categorized as challenges for modeling user sequences and challenges for serving such models. Besides, we present our contributions.

### 1.1.1 Capturing Different Forms of Sequential Patterns

*How can we model multiple forms of dependencies when they co-exist in the recent part of user action sequence?*

*Challenge:* From data mining point of view, the prediction on the item a user will likely to buy usually depend on certain patterns from her/his action sequence in the past. Inspired by sequential association rule mining [2], the sequential pattern can be in *point-level* or *union-level*. The former means previous actions influence the target action individually. An example pattern may look like: $i_a \rightarrow i_t$, here $i_a$ is the item previously interacted by a user and $i_t$ is the target item. Union-level sequential pattern is an extension of point-level pattern, where multiple previous actions jointly influence the target action. An example may be: $(i_a, i_b, i_c) \rightarrow i_t$, here $(i_a, i_b, i_c)$ are the items in an episode of of a user sequence (with order preserved). If these two forms of sequential dependencies co-exist in the recent part of user sequence, leveraging them may help further improve the recommendation accuracy. From the past literature, however, we found seldom works have been done to explicitly model both of these two forms of sequential patterns.

*Our Contributions:* In Chapter 3, we confirm the existence of sequential dependencies with multiple forms and study how they can influence user action in the future, by using a commonly used public dataset. Inspired from the recent success of convolution filters of Convolutional Neural Network (CNN) to capture local invariant features [58, 51, 53], we propose a *ConvolutionAl Sequence Embedding Recommendation Model*, or Caser for short. Compared to existing methods, Caser offers several distinct advantages. (1) Caser uses 1-D convolutional filters with various shapes to capture sequential patterns at point-level, union-level. (2) Caser is very flexible that it generalizes several existing state-of-the-art methods in a single unified framework. (3) Caser outperforms state-of-the-art methods for top-$N$ sequential recommendation on real life datasets.

### 1.1.2 Utilizing Long-range Dependent User Sequences

*How can we model dependencies from everywhere of long user action sequence where different parts of the sequence show different properties?*

*Challenge:* In the last problem, we only focus on modeling user's recent actions (short-range part) in his/her sequence. While user sequence can be very long, modeling user's actions from far past (long-range part) also enables us to learn user interests from a longer time span. For example, if a movie recommender can 'remember' a user's interest (*e.g.*, love watching Jackie Chan's movies) from long time ago, it may recommend relevant movies (Jackie Chan's new released movies) that fit this user's taste, based on the fact that users' tastes for movie don't change too much over time. However, capturing sequential patterns from extremely long user sequence could be very challenging. Firstly, dependencies (patterns) from different part of the sequence may show different properties. Moreover, dependencies from long-range part of the user sequence, which is also called the long-range-dependent patterns, are usually hard to capture by the existing models in the literature, even though sometimes they are informative and crucial to accurate predictions. Existing sequential recommenders with factorized Markov chain methods [37] or deep neural networks [39] arguably provide reliable sequential recommendation strategies. Unfortunately, they are all limited by a short window of significant temporal dependence when leveraging sequential data to make a recommendation prediction.

*Our Contributions:* In Chapter 4, we dive into a large-scale YouTube dataset and demonstrate that in real-world recommendation tasks there are significant long-range temporal dependencies in user sequence data. Besides, the dependencies have certain interesting properties. First, the dependency between two events decreases in a hyperbolic manner, as the time step separating their consumption grows. Second, the dependencies from events recently are very sensitive to order but are order insensitive from the events in the far past. Based on our observations, we realize the limitation of using a monolithic model and propose the *multi-temporal-range mixture model* (M3). From its data-driven design, M3 is a mixture model consisting of three sub-models (each with a distinct manually designed architecture) that specialize in capturing dependencies from different temporal ranges. It can also learn how to dynamically choose to focus on different temporal dynamics and ranges depending on the application context.

### 1.1.3 Mitigating Model Serving Cost Overhead

*How can we have an efficient model that works as effectively as possible?*

*Challenge:* After a recommendation model is trained, the next step is to serve it, making it respond to online user requests. Specifically, the routine serving pipeline for machine learning models consists of two main parts as shown in Figure 1.2. First of all, we have to train the designed model *offline* with massive logged data, which are processed through a pipeline including data cleaning, feature selection, etc. Then the trained model is used as a prediction service to make responses to various user requests. The later process is also called the inference phase and is performed in an *online* manner. Usually speaking, an online

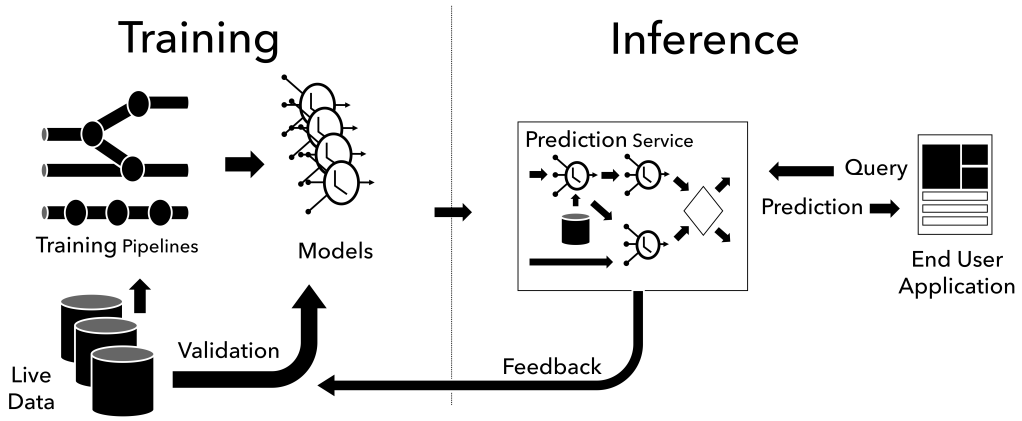



Figure 1.2: Serving pipeline of a developed machine learning model [24]. When training recommendation models, we use logged feedback data to learn a model offline. Then the learned model is composed as a prediction service and respond to user requests in an online manner.

service needs to control the response time strictly less than 0.2 seconds (200 ms), that's why enhancing the efficiency of model inference is a critical research problem.

The inference time cost for recommendation models is extremely high, since in order to sort the items, we need to compute the relevance scores for all items given a single user request. And the time cost for computation is $O(|\mathcal{I}| \times \Delta_c)$, where $|\mathcal{I}|$ is the number of items and $\Delta_c$ is the time cost needed to compute relevance score for single user-item pair. For sequential recommenders, since users' representations are changing all the time when interacting with the platform, we need to do the inference much more frequently, thus we suffer more from efficiency issue. What's worse, as neural networks becoming the main stream for recommendation due to their excellent performances, these models incur even larger latency[1] at online inference phase due to the larger model size.

*Our Contributions:* As we all know, $\Delta_c$ is highly depend on model size. Therefore, to answer the research question, a proper solution is to reduce the model size, and at the same time, keep its accuracy as much as possible. Put differently, the ideal way is increasing a model's performance while having its size fixed. Also, the proposed solution is better to be model-agnostic, not an ad-hoc approach designed for certain models specifically. In Chapter 5, we present our solution called *ranking distillation* (RD) to learn a compact ranking model that remains effective. The idea is an adapted solution for knowledge distillation (KD) for top-$N$ ranking problems. To be specific, a small student model is trained to learn to rank from two sources of information, i.e., the training set and the top $N$ recommendations for each user generated by a large well-trained teacher ranking model. With the large

[1]As described in Chapter 2, the computation cost for making recommendations with a commonly used two-tower neural framework is highly depend on the neural model size.

model size, the teacher model captures more complex user-item interaction patterns from the training set and provides top-$N$ ranked items as an extra training data for the student model. The student model benefits from the extra supervisions from the teacher (in addition to the data from usual training set), thus, improves its recommendation accuracy, but is more efficient for online inferences thanks to its small model size. It is also worth noting that, besides sequential recommendation, our proposed method can be also generalized to other recommendation (and ranking) tasks.

To summarize, this dissertation focuses on improving the performance of the sequential recommender system, while making the resulting system more efficient to use. Our first two works focus on how to gain a better ranking performance for sequential recommender by modeling more complex user-item interactions that potentially exist in the data. However, it is often the case that these complex models incur a larger latency at online inference phase due to the much larger model size. In order to make the resulting methods efficient to use in real-world scenarios, we propose a model independent approach that is able to reduce the model size and at the same time keep the accuracy as much as possible.

## 1.2 Thesis Organization

The remainder of the thesis is structured as follows.

- In Chapter 2, we present some background knowledge on conventional recommendation and sequential recommendation, including the basic concepts, problem definition, evaluation measurements and commonly used techniques.

- In Chapter 3, we empirically investigate the sequential association rules containing the multiple forms of patterns (left hand side of the rule) in the real-world dataset and study how they influence the user's actions (right hand side of the rule) in the near future. We propose the *Convolutional Sequence Embedding Recommendation Model* (Caser) as a unified approach to capture these patterns. We show how we represent a sequence of recent items into an "image" in the time step and latent spaces and learn sequential patterns as local features of the image using 1-D convolutional filters. Our proposed approach is flexible to generalize other existing methods. Extensive experiments conducted on public datasets demonstrate the effectiveness of our method.

- In Chapter 4, we examine how to build a model that can make use of different temporal ranges and dynamics depending on the request context. We begin with the analysis of an anonymized Youtube dataset comprising millions of user sequences. We quantify the degree of long-range dependence in these sequences and demonstrate that both short-term and long-term dependent behavioral patterns coexist. We then propose a neural *Multi-temporal-range Mixture Model* (M3) as a tailored solution to deal with both short-term and long-term dependencies. Our approach employs a mixture of

models, each with a different temporal range. These models are combined by a learned gating mechanism capable of exerting different model combinations given different contextual information. In empirical evaluations on a public dataset and our own anonymized YouTube dataset, M3 consistently outperforms state-of-the-art sequential recommendation methods.

- In Chapter 5, we present our work *Ranking Distillation (RD)* as a novel way to train ranking models that are both effective and efficient. Based on a similar idea of knowledge distillation (KD), we achieve this goal by improving model performances while keeping the model size fixed. Specifically, we employ the student-teacher learning paradigm and train a smaller student model to learn to rank documents/items from both the training data and the supervision of a larger teacher model. The resulting student model achieves a similar ranking performance to that of the large teacher model, but its smaller model size makes the online inference more efficient. From empirical studies, we show our solution can learn a compact student model with size less than half of the teacher model while achieving a ranking performance similar to the teacher model and much better than the model learnt without RD.

- Finally, we conclude the thesis with a summary of our contributions and point out some potential future directions in Chapter 6.

- A list of our publications about the three proposed works is included in Appendix A.

# Chapter 2

# Preliminaries and Background

We begin with an overview of the background concepts used throughout this dissertation. We then present the problem definition and necessary background knowledge. In discussing such backgrounds, we cover some related works, but we will go into more depth in the related chapters of this thesis.

## 2.1 Basic Concepts

**Recommendation task.** In a recommender system, there is a set of users $\mathcal{U}$, a set of items $\mathcal{I}$ (e.g., products, video, venues, etc.), and feedback/interaction data (e.g., user purchased a product, watched a video, checked-in at a venue). The logged data from system represents the feedback users from $\mathcal{U}$ have given on items from $\mathcal{I}$. Usually, a timestamp is also recorded along with the feedback data. Besides, additional data might be available, such as contextual feature data, describing more properties of users (e.g., age, sex, device used, etc.) and/ or items (e.g., category, text description, associated images, etc). We will not consider the contextual features in most chapters of this thesis. The only exception is in Chapter 4, when we evaluate our method on YouTube dataset which has rich contextual annotations. The ultimate goal of recommendation is to build a model on users' historical data that can be used to predict top $N$ ranked items that each user will have the greatest chance to *interact* with in the future.

**Types of feedback.** While interacting with an online service provider, the feedback that users left in the system can have different types. The feedback can be *explicit*. That is, the user explicitly shows how she/he likes an item. For example, a user may give five stars or one star to a certain movie. Feedback can also be *implicit* such that the user's behavior implicitly reflect her/his preferences. For example, a user may click a link, watch a video or check-in at a venue. Such feedback usually implies user's positive preference on a given item and is much more prevalent than the explicit feedback in real systems.

In this thesis, we focus on implicit feedback as it is more common in real-world scenarios.

**Data sparsity and feedback missing-not-at-random assumption.** For recommendation problem specifically, although we have a lot of signals from interaction data, we only observe a few items that have feedback from a given user. This cause the feedback data extremely sparse. Besides, those unobserved data are not treated equally and are usually regarded as negative feedback. This is based on a well-known assumption that feedback is not missing (completely) at random [66, 88]. In particular, it suggests each item is not equally likely to be clicked or viewed by a user and the unobserved feedback has a higher propensity to be a negative feedback.

**Offline model learning vs. Online model inference.** Similar to other machine learning based methods, there are two phases for building a recommendation model: learning and inference. In most cases, the efficiency requirements for the two phases are different: the learning phase can be done in the offline so we don't have a limitation on its efficiency; the inference phase for recommendation models has a tight constraint for efficiency if it is required to perform in real-time.

For learning phase in recommendation, people tend to learn (train) a model $M$ (parameterized $\theta$) offline with logged feedback data. In particular, we define a loss function over a real preference label (e.g., positive or negative) $y_i^{(u)}$ and a model predicted relevance score $\hat{y}_i^{(u)}$, for a given user-item pair[1] $(u, i)$. We learn the recommendation model $M(u, i; \theta) = \hat{y}_i^{(u)}$ via optimizing the loss function using some variants of Stochastic Gradient Descent. Specific to implicit feedback considered in this thesis, data may only contain positive preference label. However, according to the missing-not-at-random assumption, we can randomly sample unobserved user feedback on items then label it as negative.

For inference phase in recommendation, whenever we need to make recommendations for a certain user, we need to calculate the predicted relevance scores for all possible items. And then we can make recommendations for this user with the top $N$ items that have highest relevance scores. Note that the inference phase is not needed to be online for conventional recommendation task, as the recommended items for every user can be pre-computed offline. However, for the sequential recommendation, since users' action sequences are dynamically changing while interacting with the system, the inference usually has to be done in the real-time with the new user sequences as model inputs.

## 2.2 Sequential Recommendation Problem

Now we define the *top-N sequential recommendation* problem considered in this dissertation. Similar to conventional recommendation task, we recommends $N$ items that a user likely

[1]In some cases, preference label not only depend on user and item but also need to consider the context. For simplicity and to facilitate presentation, we omit the contextual features here.

interacts with in a near future. This problem assumes a set of users $\mathcal{U} = \{u_1, u_2, \cdots, u_{|\mathcal{U}|}\}$ and a universe of items $\mathcal{I} = \{i_1, i_2, \cdots, i_{|\mathcal{I}|}\}$. Each user $u$ is associated with a sequence of some items from $\mathcal{I}$, $\mathcal{S}^u = (\mathcal{S}^u_1, \cdots, \mathcal{S}^u_{|\mathcal{S}^u|})$, where $\mathcal{S}^u_i \in \mathcal{I}$. The index $t$ for $\mathcal{S}^u_t$ denotes the order in which an action occurs in the sequence $\mathcal{S}^u$, not the absolute timestamp as in temporal recommendation like [100, 108, 56]. Given all users' sequences $\mathcal{S}^u$, the goal is to recommend each user a list of items that maximize her/his future needs. Unlike conventional top-$N$ recommendation, top-$N$ sequential recommendation models the user behavior as a sequence of items, instead of a set of items.

## 2.3 Recommendation Model

As mentioned above, a model is needed to take user sequence into consideration and make prediction on the user's feedback to items. In the following, we will cover several popular choices and analyze their efficiency for making recommendations. We categorize them as shallow latent factor based models and deep neural network based models. As we will elaborate in the following chapters, these models have certain limitations when modeling user sequences and will serve as our baselines.

**Shallow latent factor based model.** Latent factor or matrix factorization (MF) based approaches are very popular in recommendation for both implicit [77] and explicit feedback [57]. They predict a user's preference on an item on the basis of low dimensional latent factors, by optimizing an objective function. Let $\boldsymbol{P} \in \mathbb{R}^{|\mathcal{U}| \times d}$ be the latent factor matrix corresponding to users, where $d$ is the number of latent factors and the $u$-th row $P_u \in \mathbb{R}^d$ is the latent factors for user $u$. Similarly, we have the $\boldsymbol{Q} \in \mathbb{R}^{|\mathcal{I}| \times d}$ to represent item latent factors and $Q_i \in \mathbb{R}^d$ is the latent factors for item $i$. The predicted relevance score $\hat{y}^{(u)}_i = P_u \cdot Q_i$ is a inner-product of user and item's latent factors. These latent factors are semantically similar to the 'embeddings' of neural network. This is a dimensionality reduction approach, projecting both users and items in the same lower dimensional space, where the nearby items/users are similar, and items that are close to a user are a good fit for this user. The nature of latent factor based approaches is factorizing the *user-item interaction matrix* as showed in Figure 1.1a. Then to incorporate user sequence, one can use the similar idea and factorize the *item-to-item transition matrix*, where each entry $(i, j)$ records the probability of interacting with item $j$ right after a user has interacted with item $i$. When predicting an item's relevance score, we can incorporate the predicted transit probability from the last item this user interacted with. This approach is named as factorizing personalized Markov chain (FPMC) [78]. We will discuss more about this approach in the Chapter 3.

The time cost for making recommendations with such shallow models is $O(|\mathcal{I}| \times \Delta_c)$. Here $\Delta_c$ is the time required to compute relevance score for each user-item pair, it grows nearly linearly with the number of latent factors $d$ to use.

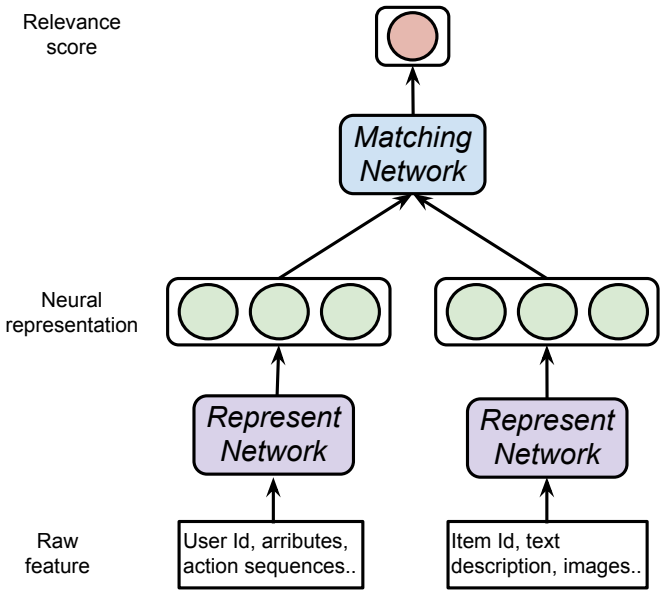



Figure 2.1: A common two-tower neural framework for recommendation.

**Deep neural network based framework.** As for neural network based methods, here we present a very general two-tower framework that adopted by many existing works. Figure 2.1 shows this neural architecture. As can be seen, this framework has a two-tower structure and consists of two important components. *Represent network* projects users' and items' raw features to a lower dimensional dense representations. *Matching network* approximates the matching function, it takes the representations as inputs and output the relevance score. There are various choices for represent network and matching network and many existing works can be categorized into this framework. For example, the latent factor model discussed above can be generalized into this framework by having embedding layer as represent network to map each user (id) and item (id). Then using the inner-product function as matching network. Neural collaborative filtering [38] further shows a more effective model when having several additional fully-connected layers, that inject non-linearities, in matching network. To incorporate user action sequences, one can use recurrent neural nets (RNN) as part of the user represent network. However, simply applying RNN to model user action sequences has several limitations, as we shall discuss in Chapter 3 and Chapter 4.

It is easily to see that the time cost for making recommendation with this two-tower neural networks is $O(|\mathcal{I}| \times (\Delta_c + \Delta_i) + \Delta_u)$, where $\Delta_u$ and $\Delta_i$ are the time cost needed to compute for user and item representations, respectively. $\Delta_c$ is the time cost for matching network. It is worth noting that all of ($\Delta_u$, $\Delta_i$, $\Delta_c$) are highly depend on network complexity. As a result, higher network complexity provides more expressive power but will incur larger latency for inference. In Chapter 5, we present a way to mitigate this.

## 2.4 Evaluation Metrics

Ideally, the best way to evaluate the recommender systems is to carry out online A/B testing experiments on real systems with a decent amount of user, measuring the business metrics (i.e., metrics to evaluate user satisfactory with the recommended item) after deploying different methods then running statistical tests. However, online A/B testing is usually very expensive and might hurt user experience (and also profit) if the tested models are not good enough. Thus, offline evaluation is a necessary step to compare candidate methods before doing large scale online tests. And it may be the best way for researchers to evaluate a recommendation model without the need of a real environment.

For offline evaluation, two basic metrics to evaluate the effectiveness of a ranking model are Precision@$N$, Recall@$N$. Given a list of top $N$ predicted items for a user, denoted $\hat{R}_{1:N}$, and the ground-truth items of this user in test set, denoted as $R$, Precision@$N$ and Recall@$N$ are computed by

$$\begin{aligned} \text{Prec@}N &= \frac{|R \bigcap \hat{R}_{1:N}|}{N}, \\ \text{Recall@}N &= \frac{|R \bigcap \hat{R}_{1:N}|}{|R|}. \end{aligned} \tag{2.1}$$

The precision and recall look similar in the numerator but have different denominators. When the number of ground-truth items of each user is limited, the precision@$N$ will become smaller as $N$ gets higher while the recall@$N$ will become larger. Note that these metrics are averaged by all users and we usually set $N$ to be small (*e.g.,* $N \leq 10$), as recommendations at top positions of ranked lists are more important.

Precision and Recall treat the prediction in top $N$ positions equally, while some works [102, 38] also use metrics that assign different weights for different positions. The weights are usually discounted as the position goes higher, so that it emphasizes more on the correctness of the top predictions. To this extent, mean Average Precision (mAP) and normalized Discounted Cumulative Gain (nDCG) are often used. The Average Precision (AP) is defined by

$$\text{AP} = \frac{\sum_{i=1}^{|\hat{R}|} \text{Prec@}i \times \text{rel}(i)}{|\hat{R}|}, \tag{2.2}$$

where $rel(i) = 1$ if the $i$-th item in the predicted ranking $\hat{R}$ is in the ground-truth items $R$. The Mean Average Precision (mAP) is the average of AP for all users. The Discounted Cumulative Gain (DCG) is defined by

$$\text{DCG} = \sum_{i=1}^{|\hat{R}|} \frac{rel(i)}{\log_2(i+1)}. \tag{2.3}$$

The nDCG@n is the ratio of DCG@n to the optimal DCG@n for that user, where the optimal DCG@n is computed by using the ideal ranking.

## 2.5 Notations

In the rest of thesis, we use $\mathcal{U} = \{u_1, u_2, \cdots, u_{|\mathcal{U}|}\}$ to denote the set of users, and $\mathcal{I} = \{i_1, i_2, \cdots, i_{|\mathcal{I}|}\}$ to denote the items. $|\cdot|$ stands for the cardinality of a set. We use $u$ to index a user, and $i$ and $j$ to index items. User $u$'s action sequence is denoted by $\mathcal{S}^{(u)}$. User and item latent factor matrix (embedding matrix) are denoted by $\boldsymbol{P} \in \mathbb{R}^{|\mathcal{U}| \times d}$ and $\boldsymbol{Q} \in \mathbb{R}^{|\mathcal{U}| \times d}$, respectively. $d$ is used to represent the number of latent factors to use.

Besides, vectors and matrices are denoted by bold symbols, where symbols in lower case (e.g., $\boldsymbol{x}$) represent vectors and symbols in upper case (e.g., $\boldsymbol{X}$) represent matrices. Unless stated differently, $x_i$ represents the $i$-th element of vector $\boldsymbol{x}$. We denote the i-th row of matrix $\boldsymbol{X}$ by $X_i$ and its $(i, j)$-th entry by $X_{ij}$.

# Chapter 3

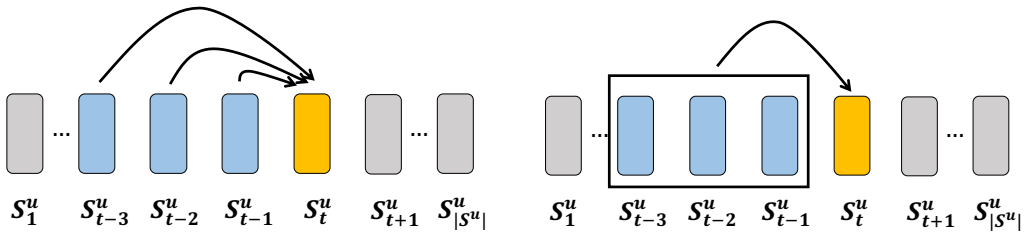
# On Capturing Different Forms of Sequential Dependencies

From sequential association rule mining perspective, sequential dependencies (patterns) may have various forms. Such dependencies if modeled explicitly and properly, can be informative to predict user needs in the future. As discussed in Chapter 1.1, the sequential pattern can be in *point-level* where previous actions influence the target action individually. Or the pattern can be in *union-level* where multiple previous actions jointly influence the target action. However, existing methods fail to best utilize all forms of dependencies. *Can we build a unified and flexible model for capturing all kinds of sequential patterns informed by sequential association rules?*

## 3.1 Background and Motivations

The Markov chain based model [78, 37, 19, 98] is an early approach to top-$N$ sequential recommendation, where an $L$-order Markov chain makes recommendations based on $L$ previous actions. The first-order Markov chain is an item-to-item transition matrix learnt using maximum likelihood estimation. Factorized personalized Markov chains (FPMC) proposed by Rendle et al. [78] and its variant [19] improved this method by factorizing this transition matrix into two latent and low-rank sub-matrices. Factorized Sequential Prediction with Item Similarity ModeLs (Fossil) proposed by He and McAuley [37] generalizes this method to high-order Markov chains using a weighted sum aggregation over previous items' latent representations. However, existing approaches suffered from two major limitations:

**Fail to model union-Level sequential patterns.** As shown in Figure 3.1a, the Markov chain models only capture **point-level** sequential patterns where each of the previous actions (blue) influences the target action (yellow) individually, instead of collectively. FPMC and Fossil fall into this taxonomy. Although Fossil [37] considers a high-order Markov chain, the overall influence is a weighted sum of previous items' latent representations factorized from first-order Markov transition matrices. Such aggregation of point-level influences is not




(a) point-level (b) union-level

Figure 3.1: An example of point and union level dynamic pattern influences, the order of Markov chain $L = 3$

sufficient to model the **union-level** influences shown in Figure 3.1b where several previous actions, in that order, jointly influence the target action. For example, buying both milk and butter together leads to a higher probability of buying flour than buying milk or butter individually; buying both RAM and Hard Drive is a better indication of buying Operating System next than buying only one of the components.

**Fail to allow skip behaviors.** Existing models don't consider **skip behaviors** of sequential patterns, where the impact from past behaviors may skip a few steps and still have strength. For example, a tourist has check-ins sequentially at airport, hotel, restaurant, bar, and attraction. While the check-ins at the airport and hotel do not immediately precede the check-in of the attraction, they are strongly associated with the latter. On the other hand, the check-in at the restaurant or bar has little influence on the check-in of the attraction (because they do not necessarily occur). A $L$-order Markov chain does not explicitly model such skip behaviors because it assumes that the $L$ previous steps have an influence on the immediate next step.

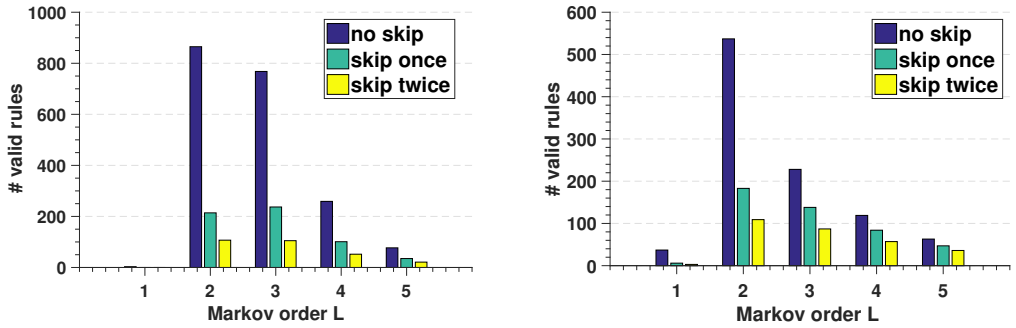
### 3.1.1 Observation from Data

To provide evidences of union-level influences and skip behaviors, we mine sequential association rules [2, 34] of the following form from two real life datasets, MovieLens and Gowalla (see the details of these data sets in Section 3.4)

$$(\mathcal{S}^u_{t-L}, \cdots, \mathcal{S}^u_{t-2}, \mathcal{S}^u_{t-1}) \rightarrow \mathcal{S}^u_t. \tag{3.1}$$

For a rule $X \rightarrow Y$ of the above form, the support count $sup(XY)$ is the number of sequences in which $X$ and $Y$ occur in order as in the rule, and the confidence, $\frac{sup(XY)}{sup(X)}$, is the percentage of the sequences in which $Y$ follows $X$ among those in which $X$ occurs. This rule represents the joint influence of all the items in $X$ on $Y$. By changing the right hand side to $\mathcal{S}^u_{t+1}$ or $\mathcal{S}^u_{t+2}$, the rule also captures the influences with one or two step skips. Figure 3.2 summarizes the number of rules found versus the Markov order $L$ and skip steps with the minimum support

count = 5 and the minimum confidence = 50% (we also tried the minimum confidence of 10%, 20%, and 30%, these trends are similar). Most rules have the orders $L = 2$ and $L = 3$ and the confidence of rules gets higher for larger $L$. The figure also tells that a sizable number of rules have skip steps 1 or 2. These findings support the existence of union-level influences and skip behaviors.




(a) MovieLens (b) Gowalla

Figure 3.2: The number of association rules vs $L$ and skip steps. The minimum support count = 5 and the minimum confidence = 50%.

### 3.1.2 Contributions

To address these above limitations of existing works, we propose a *ConvolutionAl Sequence Embedding Recommendation Model*, or *Caser* for short, as a solution to top-$N$ sequential recommendation. This model leverages the recent success of convolution filters of Convolutional Neural Network (CNN) to capture local features for image recognition [58, 51] and natural language processing [53]. The novelty of Caser is to represent the previous $L$ items as an $L \times d$ matrix $\boldsymbol{E}$, where $d$ is the number of latent dimensions and the rows preserve the order of the items. Similar to [53], we regard this embedding matrix as the "image" of the $L$ items in the latent space and search for sequential patterns as local features of this "image" using various convolutional filters. Unlike image recognition, however, this "image" is not given in the input and must be learnt simultaneously with all filters.

Compared to existing methods, Caser offers several distinct advantages:

- Caser uses horizontal and vertical convolutional filters to capture sequential patterns at point-level, union-level, and of skip behaviors.
- Caser models both users' general preferences and sequential patterns, and generalizes several existing state-of-the-art methods in a single unified framework.
- Caser outperforms state-of-the-art methods for top-$N$ sequential recommendation on real life datasets.

## 3.2 Related Work

Besides the pre-existing methods for sequential recommendation that discussed above, we introduce some further related works as follow.

Conventional recommendation methods, e.g., collaborative filtering [82], matrix factorization [57, 80], and top-$N$ recommendation [43][70], are not suitable for capturing sequential patterns because they do not model the order of actions. Early works on sequential pattern mining [2, 34] find explicit sequential association rules based on statistical co-occurrences [64]. This approach depends on the explicit representation of patterns, thus, could miss patterns in unobserved states. Also, it suffers from a potentially large search space, sensitivity to threshold settings, and a large number of rules, most being redundant.

Restricted Bolzmann Machine (RBM) [81] is the first successful 2-layers neural network that is applied to recommendation problems. Auto-encoder framework [83, 97] and its variant denoising auto-encoder [102] also produce a good recommendation performance. Convolutional neural network (CNN) [112] has been used to extract users' preferences from their reviews. None of these works is for sequential recommendation.

Recurrent neural networks (RNN) was used for session-based recommendation [39, 47]. While RNN has shown to have an impressive capability in modeling sequences [67], its sequentially connected network structure may not work well under sequential recommendation setting. Because in sequential recommendation problem, not all adjacent actions have dependency relationships (e.g., a user bought $i_2$ after $i_1$ only because she loves $i_2$). Our experimental results in Section 3.4 verify this point: RNN-based method performs better when datasets contains considerable sequential patterns. While our proposed method doesn't model sequential pattern as adjacent actions, it adopts convolutional filters from CNN and model sequential patterns as local features of the embeddings of previous items. This approach offers the flexibility of modeling sequential patterns at both point level and union level, and skip behaviors in a single unified framework. In fact, we will show that Caser generalizes several state-of-the-art methods.

A related but different problem is temporal recommendation [108, 100, 86]. For example, temporal recommendation recommends coffee in the morning, instead of evening, whereas our top-$N$ sequential recommendation would recommend phone accessories soon after a user bought an iPhone, independently of the time. Clearly, the two problems are different and require different solutions.

## 3.3 Proposed Methodology

The proposed model, ConvolutionAl Sequence Embedding Recommendation (Caser), incorporates the Convolutional Neural Network (CNN) to learn sequential features, and Latent Factor Model (LFM) to learn user specific features. The goal of Caser's network design is multi-fold: capture both user's general preferences and sequential patterns, at both union-

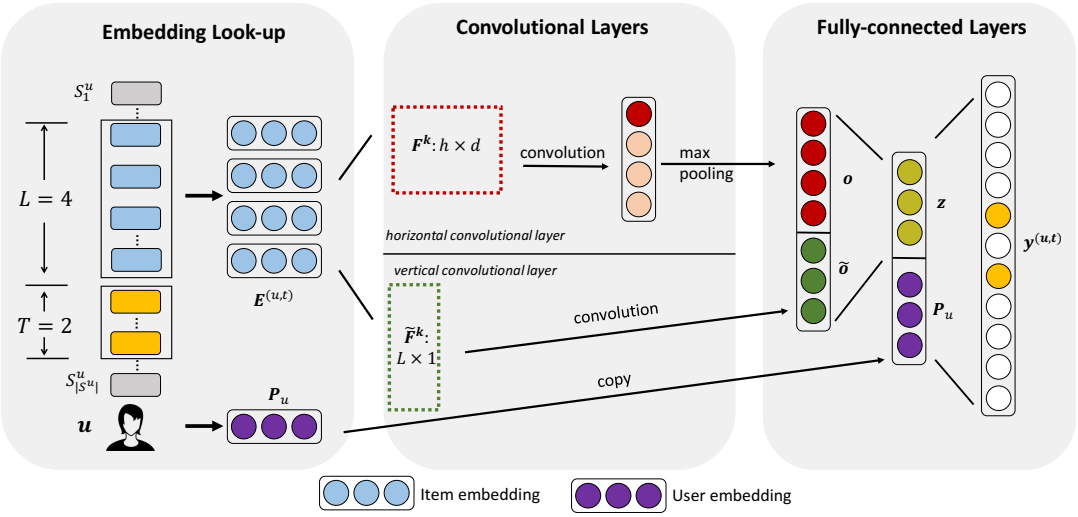



Figure 3.3: The network architecture of *Caser*. The rectangular boxes represent items $\mathcal{S}_1^u, \cdots, \mathcal{S}_{|\mathcal{S}^u|}^u$ in user sequence, whereas a rectangular box with circles inside stands for a certain vector *e.g.,* user embedding $\boldsymbol{P}_u$. The dash rectangular boxes are convolutional filters with different sizes. The red circles in convolutional layers stand for the max values in each of the convolution results. Here we are using previous 4 actions ($L = 4$) to predict which items this user will interact with in next 2 steps ($T = 2$).

level and point-level, and capture skip behaviors, all in unobserved spaces. Shown in Figure 3.3 Caser consists of three components: Embedding Look-up, Convolutional Layers, and Fully-connected Layers. To train the CNN, for each user $u$, we extract every $L$ successive items as input and their next $T$ items as the targets from the user's sequence $\mathcal{S}^u$, shown on the left side of Figure 3.3. This is done by sliding a window of size $L + T$ over the user's sequence, and each window generates a training instance for $u$, denoted by a triplet ($u$, previous $L$ items, next $T$ items).

### 3.3.1 Model Formulation

#### Embedding Look-up

*Caser* captures sequence features in the latent space by feeding the embeddings of previous $L$ items into the neural network. The embedding $Q_i \in \mathbb{R}^d$ for item $i$ is a similar concept to its latent factors. Here $d$ is the number of latent dimensions. The embedding look-up operation retrieves the previous $L$ items' embeddings and stacks them together, resulting in a matrix $\boldsymbol{E}^{(u,t)} \in \mathbb{R}^{L \times d}$ for user $u$ at time step $t$:

$$\boldsymbol{E}^{(u,t)} = \begin{bmatrix} Q_{\mathcal{S}_{t-L}^u} \\ \vdots \\ Q_{\mathcal{S}_{t-2}^u} \\ Q_{\mathcal{S}_{t-1}^u} \end{bmatrix}. \tag{3.2}$$

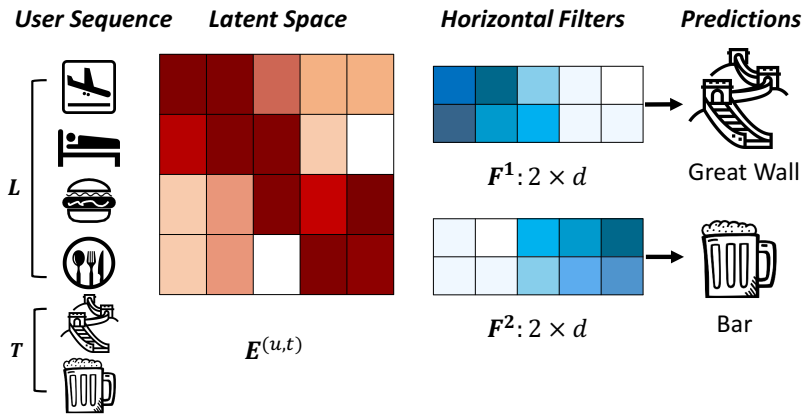



Figure 3.4: Darker colors mean larger values. The first filter captures "(Airport, Hotel) → Great Wall" by interacting with the embedding of airport and hotel and skipping that of fast food and restaurant. The second filter captures "(Fast Food, Restaurant) → Bar".

Along with the item embeddings, we also have an embedding $P_u \in \mathbb{R}^d$ for a user $u$, representing user features in latent space. These embeddings are represented by blue and purple circles in the box of Embedding Look-up in Figure 3.3.

### Convolutional Layers

Our approach leverages the recent success of convolution filters of CNN in capturing local features for image recognition [58, 51] and natural language processing [53]. Borrows the idea of using CNN in text classification [53], our approach regards the $L \times d$ matrix $\boldsymbol{E}$ as the "image" of the previous $L$ items in the latent space and regard sequential patterns as local features of this "image". This approach enables the use of convolution filters to search for sequential patterns. Figure 3.4 shows two "horizontal filters" that capture two union-level sequential patterns. These filters, represented as $h \times d$ matrices, have the height $h = 2$ and the full width equal to $d$. They pick up signals for sequential patterns by sliding over the rows of $\boldsymbol{E}$. For example, the first filter picks up the sequential pattern "(Airport, Hotel) → Great Wall" by having larger values in the latent dimensions where Airport and Hotel have larger values. Similarly, a "vertical filter" is a $L \times 1$ matrix and will slide over the columns of $\boldsymbol{E}$. More details are explained below. Unlike image recognition, the "image" $\boldsymbol{E}$ is not given because the embedding $Q_i$ for all items $i$ must be learnt simultaneously with all filters.

**Horizontal Convolutional Layer.** This layer, shown in the upper part of the second component in Figure 3.3, has $n$ horizontal filters $\boldsymbol{F}^k \in \mathbb{R}^{h \times d}$, $1 \leq k \leq n$. $h \in \{1, \cdots, L\}$ is the height of a filter. For example, if $L = 4$, one may choose to have $n = 8$ filters, two for each $h$ in $\{1, 2, 3, 4\}$. $\boldsymbol{F}^k$ will slide from top to bottom on $\boldsymbol{E}$ and interact with all horizontal dimensions of $\boldsymbol{E}$ of the items $i$, $1 \leq i \leq L - h + 1$. The result of the interaction is the $i$-th

convolution value given by

$$\boldsymbol{c}_i^k = \phi_c(\boldsymbol{E}_{i:i+h-1} \odot \boldsymbol{F}^k). \tag{3.3}$$

where the symbol $\odot$ denotes the inner product operator and $\phi_c(\cdot)$ is the activation function for convolutional layers. This value is the inner product between $\boldsymbol{F}^k$ and the sub-matrix formed by the row $i$ to row $i-h+1$ of $\boldsymbol{E}$, denoted by $\boldsymbol{E}_{i:i+h-1}$. The final convolution result of $\boldsymbol{F}^k$ is the vector

$$\boldsymbol{c}^k = \left[\boldsymbol{c}_1^k\ \boldsymbol{c}_2^k \cdots \boldsymbol{c}_{L-h+1}^k\right]. \tag{3.4}$$

We then apply a max pooling operation to $\boldsymbol{c}^k$ to extract the maximum value from all values produced by this particular filter. The maximum value captures the most significant feature extracted by the filter. Therefore, for the $n$ filters in this layer, the output value $\boldsymbol{o} \in \mathbb{R}^n$ is

$$\boldsymbol{o} = \{max(\boldsymbol{c}^1), max(\boldsymbol{c}^2), \cdots, max(\boldsymbol{c}^n)\}. \tag{3.5}$$

Horizontal filters interact with every successive $h$ items through their embeddings $\boldsymbol{E}$. Both the embeddings and the filters are learnt to minimize an objective function that encodes the prediction error of target items (more in Section 3.3.2). By sliding filters of various heights, a significant signal will be picked up regardless of location. Therefore, horizontal filters can be trained to capture **union-level patterns** with multiple union sizes.

**Vertical Convolutional Layer.** This layer is shown in the lower part of the second component in Figure 3.3. We use tilde ($\sim$) for the symbols of this layer. Suppose that there are $\tilde{n}$ vertical filters $\tilde{\boldsymbol{F}}^k \in \mathbb{R}^{L\times 1}$ , $1 \leq k \leq \tilde{n}$. Each filter $\tilde{\boldsymbol{F}}^k$ interacts with the columns of $\boldsymbol{E}$ by sliding $d$ times from left to right on $\boldsymbol{E}$, yielding the vertical convolution result $\tilde{\boldsymbol{c}}^k$:

$$\tilde{\boldsymbol{c}}^k = \left[\tilde{\boldsymbol{c}}_1^k\ \tilde{\boldsymbol{c}}_2^k \cdots \tilde{\boldsymbol{c}}_d^k\right]. \tag{3.6}$$

For the inner product interaction, it is easy to verify that this result is equal to the weighted sum over the $L$ rows of $\boldsymbol{E}$ with $\tilde{F}^k$ as the weights:

$$\tilde{\boldsymbol{c}}^k = \sum_{l=1}^{L} \tilde{\boldsymbol{F}}_l^k \cdot \boldsymbol{E}_l, \tag{3.7}$$

where $E_l$ is the $l$-th row of $\boldsymbol{E}$. Therefore, with vertical filters we can learn to *aggregate* the embeddings of the $L$ previous items, similar to Fossil's [37] weighted sum to aggregate the $L$ previous items' latent representations. The difference is that each filter $\tilde{\boldsymbol{F}}^k$ is acting like a different aggregator. Thus, similar to Fossil, these vertical filters are capturing **point-level sequential patterns** through weighted sums over previous items' latent representations. While Fossil uses a single weighted sum for each user, we can use $\tilde{n}$ global vertical filters to

produce $\tilde{n}$ weighted sums $\tilde{\boldsymbol{o}} \in \mathbb{R}^{d\tilde{n}}$ for all users:

$$\tilde{\boldsymbol{o}} = \left[\tilde{\boldsymbol{c}}^1 \ \tilde{\boldsymbol{c}}^2 \cdots \tilde{\boldsymbol{c}}^{\tilde{n}}\right]. \tag{3.8}$$

Since their usage is aggregation, vertical filters have some differences from horizontal ones: (1) The size of each vertical filter is fixed to be $L \times 1$. This is because each column of $\boldsymbol{E}$ is latent for us, it is meaningless to interact with multiple successive columns at one time. (2) There is no need to apply max pooling operation over the vertical convolution results, as we want to keep the aggregation for every latent dimension. Thus, the output of this layer is $\tilde{\boldsymbol{o}}$.

**Fully-connected Layers**

We concatenate the outputs of the two convolutional layers and feed them into a fully-connected neural network layer to get more high-level and abstract features:

$$\boldsymbol{z} = \phi_a(\boldsymbol{W}\begin{bmatrix}\boldsymbol{o} \\ \tilde{\boldsymbol{o}}\end{bmatrix} + \boldsymbol{b}), \tag{3.9}$$

where $\boldsymbol{W} \in \mathbb{R}^{d\times(n+d\tilde{n})}$ is the weight matrix that projects the concatenation layer to a $d$-dimensional hidden layer, $\boldsymbol{b} \in \mathbb{R}^d$ is the corresponding bias term and $\phi_a(\cdot)$ is the activation function for fully-connected layer. $\boldsymbol{z} \in \mathbb{R}^d$ is what we called *convolutional sequence embedding*, which encodes all kinds of sequential features of the $L$ previous items.

To capture user's general preferences, we also look-up the user embedding $P_u$ and concatenate the two $d$-dimensional vectors, $\boldsymbol{z}$ and $P_u$, together and project them to an output layer with $|\mathcal{I}|$ nodes, written as

$$\boldsymbol{y}^{(u,t)} = \boldsymbol{W'}\begin{bmatrix}\boldsymbol{z} \\ \boldsymbol{P}_u\end{bmatrix} + \boldsymbol{b'}, \tag{3.10}$$

where $\boldsymbol{b'} \in \mathbb{R}^{|\mathcal{I}|}$ and $\boldsymbol{W'} \in \mathbb{R}^{|\mathcal{I}|\times 2d}$ are the bias term and weight matrix for output layer, respectively. As explained in Section 3.3.2, the value $\boldsymbol{y}_i^{(u,t)}$ in the output layer is associated with the probability of how likely user $u$ will interact with item $i$ at time step $t$. $\boldsymbol{z}$ intends to capture short term sequential patterns, whereas the user embedding $P_u$ captures user's long-term general preferences. Here we put the user embedding $P_u$ in the last hidden layer for several reasons: (1) As we shall see in Section 3.3.3, it can have the ability to generalize other models. (2) we can pre-train our model's parameters with other generalized models' parameters. As stated in [38], such pre-training is critical to model performance

### 3.3.2 Model Learning and Inference

#### Model Parameters Learning

To train the network, we transform the values of the output layer, $\boldsymbol{y}^{(u,t)}$, to probabilities by:

$$p(\mathcal{S}_t^u \mid \mathcal{S}_{t-1}^u, \mathcal{S}_{t-2}^u, \cdots, \mathcal{S}_{t-L}^u) = \sigma(\boldsymbol{y}_{\mathcal{S}_t^u}^{(u,t)}), \tag{3.11}$$

where $\sigma(x) = 1/(1+e^{-x})$ is the *sigmoid* function. Let $\mathcal{C}^u = \{L+1, L+2, ..., |\mathcal{S}^u|\}$ be the collection of time steps for which we would like to make predictions for user $u$. The likelihood of all sequences in the dataset is:

$$p(\mathcal{S}|\Theta) = \prod_u \prod_{t \in \mathcal{C}^u} \sigma(\boldsymbol{y}_{\mathcal{S}_t^u}^{(u,t)}) \prod_{j \neq \mathcal{S}_t^u} (1 - \sigma(\boldsymbol{y}_j^{(u,t)})). \tag{3.12}$$

To further capture **skip behaviors**, we could consider the next $T$ target items, $\mathcal{D}_t^u = \{\mathcal{S}_t^u, \mathcal{S}_{t+1}^u, ..., \mathcal{S}_{t+T}^u\}$, at once by replacing the immediate next item $\mathcal{S}_t^u$ in the above equation with $\mathcal{D}_t^u$. Taking the negative logarithm of likelihood, we get the objective function, also known as *binary cross-entropy* loss:

$$\ell = \sum_u \sum_{t \in \mathcal{C}^u} \sum_{i \in \mathcal{D}_t^u} -\log(\sigma(\boldsymbol{y}_i^{(u,t)})) + \sum_{j \neq i} -\log(1 - \sigma(\boldsymbol{y}_j^{(u,t)})). \tag{3.13}$$

Following previous works [78, 37, 102], for each target item $i$, we randomly sample several (3 in our experiments) negative instances $j$ in the second term.

The *model parameters* $\Theta = \{\boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{F}, \tilde{\boldsymbol{F}}, \boldsymbol{W}, \boldsymbol{W'}, \boldsymbol{b}, \boldsymbol{b'}\}$ are learned by minimizing the objective function in Eqn equation 3.13 on the training set, whereas the *hyperparameters* (e.g., $d, n, \tilde{n}, L, T$) are tuned on the validation set via grid search. We adopt an variant of Stochastic Gradient Descent (SGD) called Adaptive Moment Estimation (Adam) [55] for faster convergence, with a batch size of 100. To control model complexity and avoid overfitting, we use two kinds of regularization methods: the $l_2$ Norm (weight decay) is applied for all model parameters and *Dropout* [87] technique with 50% drop ratio is used on fully-connected layers. We implemented Caser with MatConvNet [96]. The whole training time is proportional to the number of training instances. For example, it took around 1 hour for MovieLens data and 2 hours for Gowalla data, 2 hours for Foursquare and 1 hour for Tmall on a 4-cores i7 CPU and 32GB RAM machine. These times are comparable to Fossil's [37] running time and can be further reduced by using GPU.

#### Model Inference for Recommendations

After obtaining the trained neural network, to make recommendations for a user $u$ at time step $t$, we take $u$'s latent embedding $P_u$ and extract his last $L$ items' embeddings given by Eqn equation 3.2 as the neural network input. We recommend the $N$ items that have the

highest values in the output layer $\boldsymbol{y}$. The complexity for making recommendations to all users is $O(|\mathcal{U}||\mathcal{I}|d)$, where the complexity of convolution operations is ignored. Note that the number of target items $T$ is a hyperparameter used during the model training, whereas $N$ is the number of items recommended after the model is trained.

### 3.3.3 Connection to Existing Models

We show that Caser is a generalization of several previous models.

**Caser vs. MF.** By discarding all convolutional layers and all bias terms, our model becomes a vanilla LFM with user embeddings as user latent factors and its associated weights as item latent factors. MF usually contains bias terms[1], which is $\boldsymbol{b'}$ in our model. After discarding all convolutional layers, the resulting model is the same as MF:

$$\boldsymbol{y}_i^u = \boldsymbol{W}_i' \begin{bmatrix} \mathbf{0} \\ \boldsymbol{P}_u \end{bmatrix} + \boldsymbol{b}_i'. \tag{3.14}$$

**Caser vs. FPMC.** FPMC fuses factorized first-order Markov chain with LFM and is optimized by Bayesian personalized ranking (BPR). Although Caser uses a different optimization criterion, *i.e.,* the cross-entropy, it is able to generalize FPMC by copying the previous item's embedding to the hidden layer $\boldsymbol{z}$ and not using any bias terms:

$$\boldsymbol{y}_i^{(u,t)} = \boldsymbol{W}_i' \begin{bmatrix} \boldsymbol{Q}_{\mathcal{S}_{t-1}^u} \\ \boldsymbol{P}_u \end{bmatrix}. \tag{3.15}$$

As FPMC uses BPR as the criterion, our model is not exactly the same as FPMC. However, BPR is limited to have only 1 target and negative sample at each time step. Our cross-entropy loss does not have these limitations.

**Caser vs. Fossil.** By omitting the horizontal convolutional layer and using one vertical filter and copying the vertical convolution result $\tilde{\boldsymbol{c}}$ to the hidden layer $\boldsymbol{z}$, we get

$$\boldsymbol{y}_i^{(u,t)} = \boldsymbol{W}_i' \begin{bmatrix} \tilde{\boldsymbol{c}} \\ \boldsymbol{P}_u \end{bmatrix} + \boldsymbol{b}_i'. \tag{3.16}$$

As discussed for Eqn equation 3.7, this vertical filter serves as the weighted sum of the embeddings of the $L$ previous items, like in Fossil, though Fossil uses *Similarity Model* instead of LFM and factorizes it in the same latent space as Markov model. Another difference is

[1]Top-$N$ recommendation ranks the items for each user individually, which is invariant to user bias and global bias.

Table 3.1: Statistics of the datasets

| Datasets | Sequential Intensity | #users | #items | avg. actions per user | Sparsity |
|---|---|---|---|---|---|
| MovieLens | 0.3265 | 6.0k | 3.4k | 165.50 | 95.16% |
| Gowalla | 0.0748 | 13.1k | 14.0k | 40.74 | 99.71% |
| Foursquare | 0.0378 | 10.1k | 23.4k | 30.16 | 99.87% |
| Tmall | 0.0104 | 23.8k | 12.2k | 13.93 | 99.89% |

that Fossil uses one local weighting for each user while we use a number of global weighting through vertical filters.

## 3.4 Experimental Studies

We compare Caser with state-of-the-art methods. The source code of Caser and processed datasets are available online[2].

### 3.4.1 Experimental Setup

**Datasets**. Sequential recommendation makes sense only when the dataset contains sequential patterns. To identify such datasets, we applied sequential association rule mining to several public datasets and computed their sequential intensity defined by:

$$\text{Sequential Intensity } (SI) = \frac{\#\text{rules}}{\#\text{users}}. \tag{3.17}$$

The numerator is the total number of rules in the form of Eqn equation 3.1 found using a minimum threshold on support (i.e., 5) and confidence(i.e., 50%) with Markov order $L$ range from 1 to 5. The denominator is the total number of users. We use $SI$ to estimate the intensity of sequential signals in a dataset.

The four datasets with their $SI$ are described in Table 3.1. MovieLens[3] is the widely used movie rating data. Gowalla[4] constructed by [20] and Foursquare obtained from [107] contain implicit feedback through user-venue check-ins. Tmall, the largest B2C platform in China, is a user-purchase data obtained from IJCAI 2015 competition[5], which aims to forecast repeated buyers. Following previous works [37, 77, 102], we converted all numeric

[2]https://github.com/graytowne/caser

[3]https://grouplens.org/datasets/movielens/1m/

[4]https://snap.stanford.edu/data/loc-gowalla.html

[5]https://ijcai-15.org/index.php/repeat-buyers-prediction-competition

ratings to implicit feedback of 1. We also removed cold-start users and items of having less than $n$ feedbacks, as dealing with cold-start recommendation is usually treated as a separate issue in the literature [102, 38, 37, 78]. $n$ is [5,15,10,10] for MovieLens, Gowalla, Foursquare, and Tmall. The Amazon data previously used in [37, 36] was not used due to its $SI$ (0.0026 for 'Office Products' category, 0.0019 for 'Clothing, Shoes, Jewelry' and 'Video Games' category), in other words, its sequential signals are much weaker than the above datasets.

Following [64, 107], we hold the first 70% of actions in each user's sequence as the *training set* and use the next 10% of actions as the *validation set* to search the optimal hyperparameter settings for all models. The remaining 20% actions in each user's sequence are used as the *test set* for evaluating a model's performance.

**Evaluation Metrics**. As in [70, 78, 97, 102], we evaluate a model by Precision@$N$, Recall@$N$, and Mean Average Precision (mAP). We report the average of these values of all users. $N \in \{1, 5, 10\}$.

### 3.4.2 Performance Comparison

We compare our method, Caser, proposed in Section 4.2 with the following baselines.

- **POP**. All items are ranked by their popularity in all users' sequences, and the popularity is determined by the number of interactions.
- **BPR**. Combined with Matrix Factorization model, Bayesian personalized ranking [77] is the state-of-the-art method for non-sequential item recommendation on implicit feedback data.
- **FMC and FPMC**. As introduced in [78], FMC factorizes the first-order Markov transition matrix into two low-dimensional sub-matrices, and FPMC is a fusion of FMC and LFM. These are the state-of-the-art sequential recommendation methods. FPMC allows a basket of several items at each step. For our sequential recommendation problem, each basket has a single item.
- **Fossil**. Fossil [37] models high-order Markov chains and uses Similarity Model instead of LFM for modeling general user preferences.
- **GRU4Rec**. This is the session-based recommendation proposed by [39]. This model uses RNN to capture sequential dependencies and make predictions.

For each method, the grid search is applied to find the optimal settings of hyperparameters using the validation set. These include latent dimensions $d$ from $\{5, 10, 20, 30, 50, 100\}$, regularization hyperparameters, and the learning rate from $\{1, 10^{-1}, ..., 10^{-4}\}$. For Fossil, Caser and GRU4Rec, the Markov order $L$ is from $\{1, \cdots, 9\}$. For Caser itself, the height

$h$ of horizontal filters is from $\{1, \cdots, L\}$, the target number $T$ is from $\{1, 2, 3\}$, the activation functions $\phi_a$ and $\phi_c$ are from $\{identity, sigmoid, tanh, relu\}$. For each height $h$, the number of horizontal filters is from $\{4, 8, 16, 32, 64\}$. The number of vertical filters is from $\{1, 2, 4, 8, 16\}$. We report the result of each method under its optimal hyperparameter settings.

The best results of the six baselines and Caser are summarized in Table 3.2. The best performer on each row is highlighted in bold face. The last column is the improvement of Caser relative to the best baseline, defined as $\frac{Caser-baseline}{baseline}$. Except for MovieLens, Caser improved the best baseline on all $N$ tested by a large margin w.r.t. the three metrics. Among the baseline methods, the sequential recommenders (e.g., FPMC and Fossil) usually outperform non-sequential recommenders (i.e., BPR) on all datasets, suggesting the importance of considering sequential information. FPMC and Fossil outperform FMC on all datasets, suggesting the effectiveness of personalization. On MovieLens, GRU4Rec achieved a performance close to Caser's, but got a much worse performance on the other three datasets. In fact, MovieLens has more sequential signals than the other three data sets, thus, the RNN-based GRU4Rec could perform well on MovieLens but can easily get biased on training sets of the other three datasets despite the use of regularization and dropout as described in [39]. In addition, GRU4Rec's recommendation is session-based, instead of personalized, which enlarge the generalization error to some extent.

In the following studies, we examine the impact of the hyperparameters $\{d, L, T\}$ one at a time by holding the remaining hyperparameters at their optimal settings. We focus on MAP as it is an overall performance indicator and consistent with other metrics.

**Influence of Latent Dimensionality $d$.** Figure 3.5 shows mAP for various $d$ while keeping the other optimal hyperparameters unchanged. On the denser MovieLens, a larger $d$ does not always lead to a better model performance. A model achieves its best performance when $d$ is chosen properly and gets worse for a larger $d$ because of over-fitting. But for the other three sparser datasets, each model requires more latent dimensions to achieve their best results. For all datasets, Caser beats the strongest baseline performance by using a relatively small number of latent dimensions.

**Influence of Markov Order $L$ and Target Number $T$.** We vary $L$ to explore how much of Fossil, GRU4Rec and Caser can gain from high-order information while keeping other optimal hyperparameters unchanged. Caser-1, Caser-2, and Caser-3 denote Caser with the target number $T$ at 1, 2, 3 to study the effect of skip behaviors. The results are shown in Figure 3.6. On the dense MovieLens, Caser best utilizes the extra information provided by a larger $L$ and Caser-3 performs the best, suggesting the benefits of skip steps. However, for the sparser datasets, all models do not consistently benefit from a larger $L$. This is reasonable, because for a sparse dataset, a higher order Markov chain tends to introduce

Table 3.2: Performance comparison on the four datasets.

| Dataset | Metric | POP | BPR | FMC | FPMC | Fossil | GRU4Rec | Caser | Improv. |
|---|---|---|---|---|---|---|---|---|---|
| *MovieLens* | Prec@1 | 0.1280 | 0.1478 | 0.1748 | 0.2022 | 0.2306 | **0.2515** | 0.2502 | -0.5% |
| | Prec@5 | 0.1113 | 0.1288 | 0.1505 | 0.1659 | 0.2000 | 0.2146 | **0.2175** | 1.4% |
| | Prec@10 | 0.1011 | 0.1193 | 0.1317 | 0.1460 | 0.1806 | 0.1916 | **0.1991** | 4.0% |
| | Recall@1 | 0.0050 | 0.0070 | 0.0104 | 0.0118 | 0.0144 | **0.0153** | 0.0148 | -3.3% |
| | Recall@5 | 0.0213 | 0.0312 | 0.0432 | 0.0468 | 0.0602 | 0.0629 | **0.0632** | 0.5% |
| | Recall@10 | 0.0375 | 0.0560 | 0.0722 | 0.0777 | 0.1061 | 0.1093 | **0.1121** | 2.6% |
| | mAP | 0.0687 | 0.0913 | 0.0949 | 0.1053 | 0.1354 | 0.1440 | **0.1507** | 4.7% |
| *Gowalla* | Prec@1 | 0.0517 | 0.1640 | 0.1532 | 0.1555 | 0.1736 | 0.1050 | **0.1961** | 13.0% |
| | Prec@5 | 0.0362 | 0.0983 | 0.0876 | 0.0936 | 0.1045 | 0.0721 | **0.1129** | 8.0% |
| | Prec@10 | 0.0281 | 0.0726 | 0.0657 | 0.0698 | 0.0782 | 0.0571 | **0.0833** | 6.5% |
| | Recall@1 | 0.0064 | 0.0250 | 0.0234 | 0.0256 | 0.0277 | 0.0155 | **0.0310** | 11.9% |
| | Recall@5 | 0.0257 | 0.0743 | 0.0648 | 0.0722 | 0.0793 | 0.0529 | **0.0845** | 6.6% |
| | Recall@10 | 0.0402 | 0.1077 | 0.0950 | 0.1059 | 0.1166 | 0.0826 | **0.1223** | 4.9% |
| | mAP | 0.0229 | 0.0767 | 0.0711 | 0.0764 | 0.0848 | 0.0580 | **0.0928** | 9.4% |
| *Foursquare* | Prec@1 | 0.1090 | 0.1233 | 0.0875 | 0.1081 | 0.1191 | 0.1018 | **0.1351** | 13.4% |
| | Prec@5 | 0.0477 | 0.0543 | 0.0445 | 0.0555 | 0.0580 | 0.0475 | **0.0619** | 6.7% |
| | Prec@10 | 0.0304 | 0.0348 | 0.0309 | 0.0385 | 0.0399 | 0.0331 | **0.0425** | 6.5% |
| | Recall@1 | 0.0376 | 0.0445 | 0.0305 | 0.0440 | 0.0497 | 0.0369 | **0.0565** | 13.7% |
| | Recall@5 | 0.0800 | 0.0888 | 0.0689 | 0.0959 | 0.0948 | 0.0770 | **0.1035** | 7.9% |
| | Recall@10 | 0.0954 | 0.1061 | 0.0911 | 0.1200 | 0.1187 | 0.1011 | **0.1291** | 7.6% |
| | mAP | 0.0636 | 0.0719 | 0.0571 | 0.0782 | 0.0823 | 0.0643 | **0.0909** | 10.4% |
| *Tmall* | Prec@1 | 0.0010 | 0.0111 | 0.0197 | 0.0210 | 0.0280 | 0.0139 | **0.0312** | 11.4% |
| | Prec@5 | 0.0009 | 0.0081 | 0.0114 | 0.0120 | 0.0149 | 0.0090 | **0.0179** | 20.1% |
| | Prec@10 | 0.0007 | 0.0063 | 0.0084 | 0.0090 | 0.0104 | 0.0070 | **0.0132** | 26.9% |
| | Recall@1 | 0.0004 | 0.0046 | 0.0079 | 0.0082 | 0.0117 | 0.0056 | **0.0130** | 11.1% |
| | Recall@5 | 0.0019 | 0.0169 | 0.0226 | 0.0245 | 0.0306 | 0.0180 | **0.0366** | 19.6% |
| | Recall@10 | 0.0026 | 0.0260 | 0.0333 | 0.0364 | 0.0425 | 0.0278 | **0.0534** | 25.6% |
| | mAP | 0.0030 | 0.0145 | 0.0197 | 0.0212 | 0.0256 | 0.0164 | **0.0310** | 21.1% |

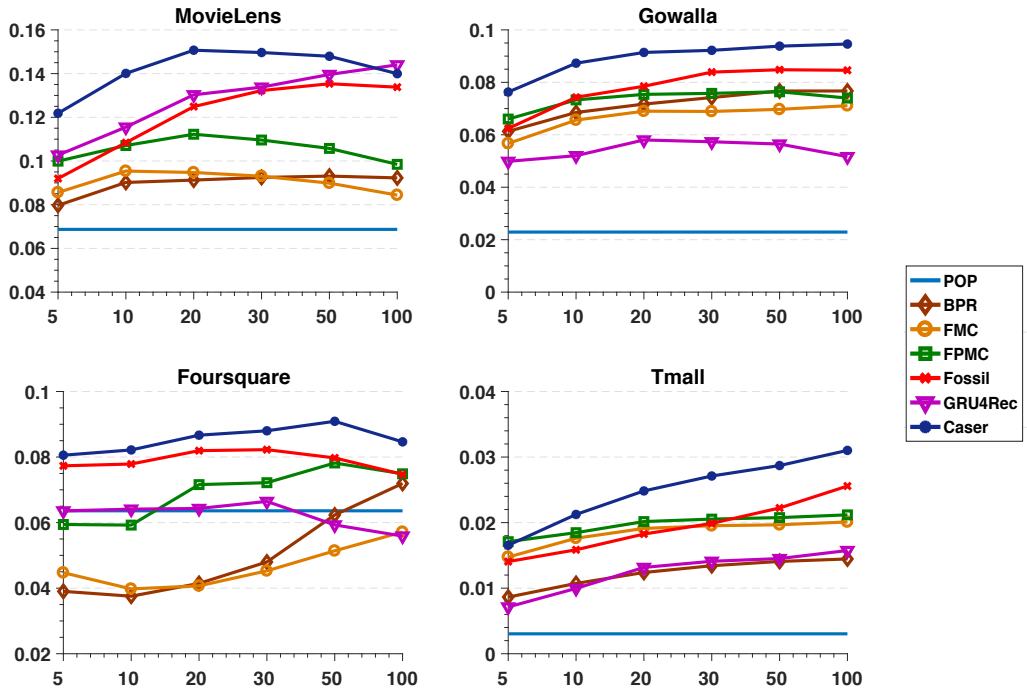



Figure 3.5: mAP (y-axis) vs. the number of latent dimensions $d$ (x-axis).

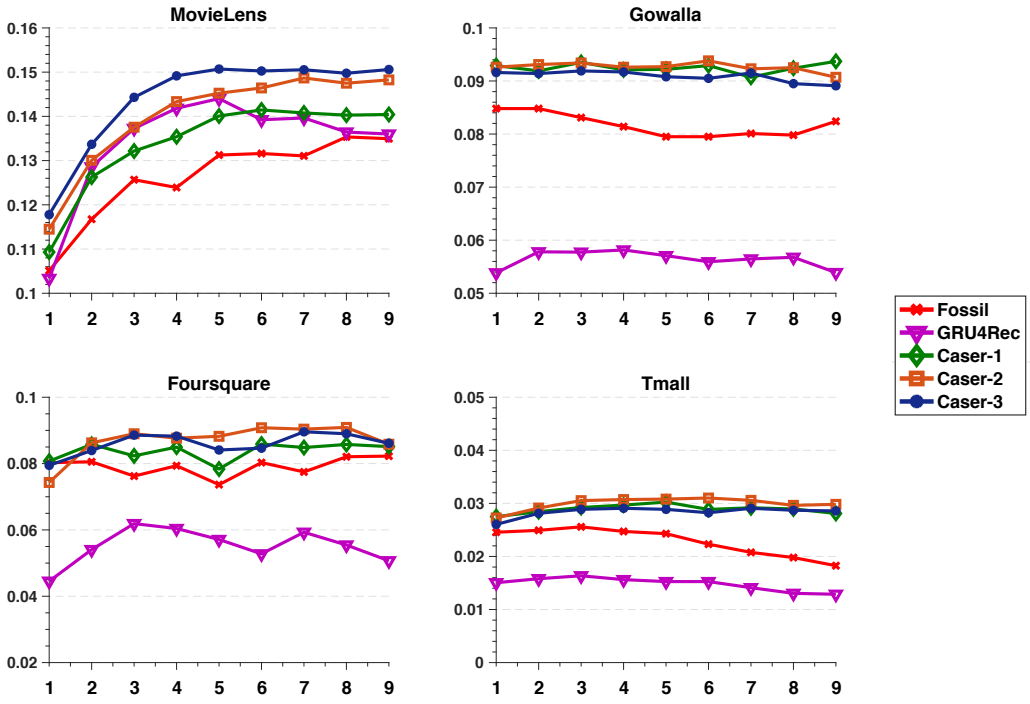



Figure 3.6: mAP (y-axis) vs. the Markov order $L$ (x-axis). Caser-1, Caser-2, and Caser-3 denote Caser with the number of targets $T$ set to $1, 2, 3$.

both extra information and more noises. In most cases, Caser-2 slightly outperforms the other models on these three datasets.

Table 3.3: mAP vs. Caser Components

| | **MovieLens** | **Gowalla** |
|---|---|---|
| Caser-p | 0.0935 | 0.0777 |
| Caser-h | 0.1304 | 0.0805 |
| Caser-v | 0.1403 | 0.0841 |
| Caser-vh | 0.1448 | 0.0856 |
| Caser-ph | 0.1372 | 0.0911 |
| Caser-pv | 0.1494 | 0.0921 |
| Caser-pvh | 0.1507 | 0.0928 |

**Analysis of Caser Components.** Finally, we evaluate the contribution of each of Caser's components, the horizontal convolutional layer (i.e., $o$), the vertical convolutional layer (i.e., $\tilde{o}$), and personalization (i.e., $P_u$), to the overall performance while keeping all hyperparameters at their optimal settings. The result is shown in Table 3.3 for MovieLens and Gowalla; the results of the other two datasets are similar. For $x \in \{p, h, v, vh, ph, pv, pvh\}$, Caser-$x$ denotes Caser with the components $x$ enabled. h denotes horizontal convolutional layer; v denotes vertical convolutional layer; p denotes personalization, which is similar to BPR and uses LFM only. Any missing component is represented by setting its corresponding $o$, $\tilde{o}$, and $P_u$ to zero. For example, vh denotes both vertical convolutional layer and horizontal convolutional layer by setting $P_u$ to all zeros, and pv denotes vertical convolutional layer and personalization by setting $o$ to all zeros. Caser-p performs the worst whereas Caser-h, Caser-v, and Caser-vh improve the performance significantly, suggesting that treating top-$N$ sequential recommendation as the conventional top-$N$ recommendation will lose useful information, and that modeling both sequential patterns at the union-level and point-level is useful for improving the prediction. For both datasets, the best performance is achieved by jointly using all parts of Caser, i.e., Caser-pvh.

**Network Visualization.** We have a closer look at some trained networks and prediction. Figure 3.7 shows the values of four vertical convolutional filters after training Caser on MovieLens with $L = 9$. In the micro perspective, the four filters are trained to be diverse, but in the macro perspective, they follow an ascending trend from past positions to recent positions. With each vertical filter serving as a way of weighting the embeddings of previous actions (see the related discussion in Section 3.3), this trend indicates that Caser puts more

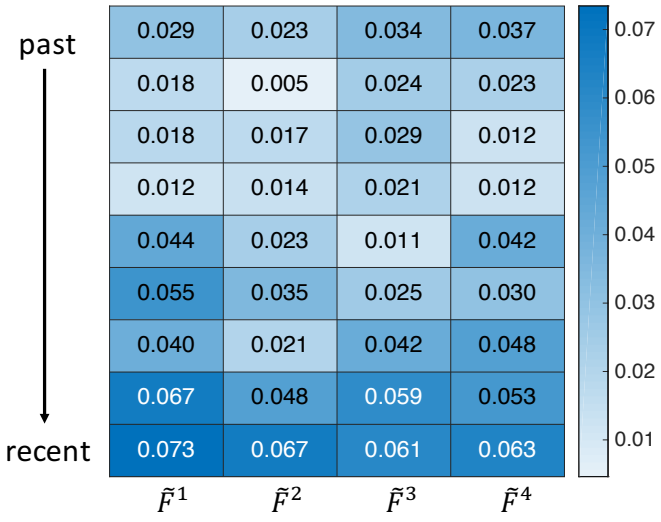
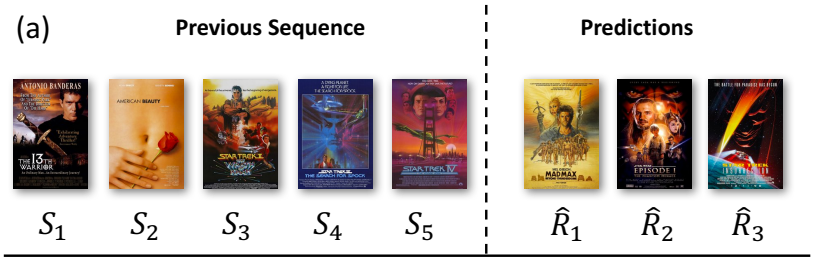



Figure 3.7: Visualization for four vertical convolutional filters of a trained model on MovieLens data when $L = 9$.




(b)

| Masking Items | New Rank of $\hat{R}_3$ after masking |
|---|---|
| $S_1, S_2$ | 2 |
| $S_3$ | 32 |
| $S_4$ | 117 |
| $S_5$ | 77 |
| $S_3, S_4, S_5$ | 513 |

Figure 3.8: Horizontal convolutional filters's effectiveness of capturing union-level sequential patterns on MovieLens data.

emphasis on recent actions, demonstrating a major difference from the conventional top-$N$ recommendation.

To see the effectiveness of horizontal filters, Figure 3.8(a) shows top $N = 3$ ranked movies recommended by Caser, i.e., $\hat{R}_1$ (Mad Max), $\hat{R}_2$ (Star War), $\hat{R}_3$ (Star Trek) in that order, for a user with $L = 5$ previous movies, i.e., $S_1$ (13th Warrior), $S_2$ (American Beauty), $S_3$ (Star Trek), $S_4$ (Star Trek III), and $S_5$ (Star Trek IV). $\hat{R}_3$ is the ground truth (i.e., the next movie in the user sequence). Note that $\hat{R}_1$ and $\hat{R}_2$ are quite similar to $\hat{R}_3$, i.e., all

being action and science fiction movies, so are also recommended to the user. Figure 3.8(b) shows the new rank of $\hat{R}_3$ after masking some of the $L$ previous movies by setting their item embeddings to zeros in the trained network. Masking $S_1$ and $S_2$ actually increases the rank of $\hat{R}_3$ to 2 (from 3); in fact, $S_1$ and $S_2$ are history or romance movies and act like noises for recommending $\hat{R}_3$. Masking each of $S_3$, $S_4$ and $S_5$ decreases the rank of $\hat{R}_3$ because these movies are in the same category as $\hat{R}_3$. The most decrease occurs after masking $S_3$, $S_4$ and $S_5$ all together. This study clearly indicates that our model correctly captures the dependence of $\hat{R}_3$ on the related $\{S_3, S_4, S_5\}$ as a union-level sequential feature for recommending $\hat{R}_3$.

## 3.5 Conclusion

Caser is a novel solution to top-$N$ sequential recommendation by modeling recent actions as an "image" among time and latent dimensions and learning sequential patterns using convolutional filters. This approach provides a unified and flexible network structure for capturing many important features of sequential recommendation, i.e., point-level and union-level sequential patterns, skip behaviors, and long term user preferences. Our experiments and case studies on public real life datasets suggested that Caser outperforms the state-of-the-art methods for top-$N$ sequential recommendation.

# Chapter 4

# On Exploiting Long-range Dependent User Sequences

In this chapter, we observe and study an open challenge for sequential recommender systems. That is, *how can we best utilize extremely-long user action sequences?* As described in Chapter 1.1, using a longer user sequence gives a model more flexibility to learn user interests from a longer time span. However, modeling long user sequence is challenging. Specifically, different part of a long user sequence may have different properties and cannot be treated equally. Also, modeling the dependencies from the long tail of user sequence is difficult. *How can we design a model that works, simultaneously, across all of temporal ranges?*

## 4.1 Background and Motivations

In the previous chapter, we assumed each user has long-term preferences, which don't change much with the time. For example, users' tastes for movie don't change too much over time, thus we should recommend Jackie Chan's new released movies if this user love watching Jackie Chan's movies long time ago. In Caser, we learned an embedding for each user to model her/his long-term preference. However, as we will elaborate later in this section, learning user embedding explicitly is impractical for large-scale recommender systems in industry. Fortunately, we can directly learn this long-term preference from the long user sequences once these sequences are available. By doing this, one can discard the user embeddings and can greatly reduce the model size when the number of users is huge. This raises two natural questions: (1) Do users really have long-term preferences? and (2) What's the challenges for modeling these preferences with long user sequences? In the following, We first present our findings on YouTube dataset which reveal the existence of users' long-term preference. We also uncover properties of behavioral patterns in different temporal ranges of user sequences. Finally, we show some limitations of existing methods which motivate us to design a better adapted solution.

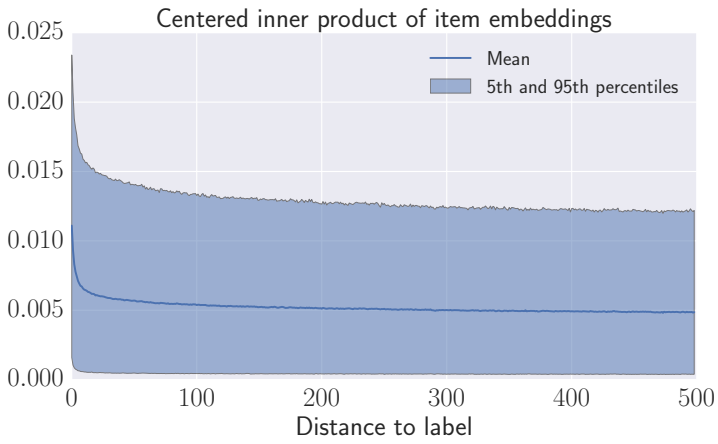



Figure 4.1: Trace of covariance (i.e. centered inner product similarity) of item embeddings between the last item in user sequence and the item located $L$ steps before (100K samples).

### 4.1.1 Observation from Data

We now describe how we developed a better understanding of long user sequences in YouTube dataset through quantitative data exploration. To quantify how past events in a user sequence[1] $\mathcal{S}$ can influence a user's current behavior in our dataset, i.e. measure the range of temporal dependency within a sequence of actions, one can examine the covariance matrix of two events $L$-step apart [8, 74], where step denotes the relative order of events within sequence. In particular, we look at the trace of the covariance matrix as a measurement of dependency:

$$\text{Dep}_L = \text{tr}\left(\text{Cov}\left(Q_{\mathcal{S}_M}, Q_{\mathcal{S}_{M-L}}\right)\right)$$

where $\mathcal{S}_M$ is the item in last event in a logged user action sequence $\mathcal{S}$ and $\mathcal{S}_{M-L}$ is the item corresponding to the interaction that occurred $L$ time steps before the last event. We focus on the trace of the covariance matrix as it equals the sum of the eigenvalues of the covariance matrix and its rate of decay is therefore informative of the rate of decay of these eigenvalues as a whole.

We utilize the item embeddings $\boldsymbol{Q}$ that have been learned by a pre-existing model—in our case an RNN-based sequential recommender which we describe later as one of M3's sub-models. $\text{Dep}_L$ here measures the similarity between the current event and the event $L$ steps back from it. To estimate $\text{Dep}_L$ for a particular value of $L$ we employ a classic empirical averaging across user sequences in our dataset. From Figure 4.1, we can extract multiple findings:

[1]For simplicity, we omit the superscripts related to users (i.e., $\mathcal{S}^u$ will be denoted $\mathcal{S}$) and focus on one single user sequence to illustrate our observation and our proposed method.

- The dependency between two events decreases as the time separating their consumption grows. This suggests that recent events bear most of the influence of past user behavior on a user's future behavior.

- The dependency slowly approaches zero even as the temporal distance becomes very large (i.e., $L > 100$). The clear hyperbolic-decay of the level of temporal dependencies indicates the presence of long-range-dependent patterns existing in user sequences [74]. In other words, a user's past interactions, though far from the current time step, still cumulatively influence their current behavior significantly.

These findings suggest that users do have long-term preferences and better capturing such long-range-dependent pattern could help predicting their future interests. In further work, we plan to use off-policy correction methods such as [32, 15] to remove presentation bias when estimating correlations.

### 4.1.2 Limitations of Previous Work

**Limitations of Existing Sequential Models.** The previous section has demonstrated the informational value of long-range temporal patterns in user sequences. Unfortunately, it is still generally challenging for existing sequential predictive models to fully utilize information located far into the past.

*Most prior models have difficulties when learning to account for sequential patterns involving long-range dependence.* Existing sequential recommenders with factorized Markov chain methods [37] or CNNs (i.e., Caser in Chapter 3) arguably provide reliable sequential recommendation strategies. Unfortunately they are all limited by a short window of significant temporal dependence when leveraging sequential data to make a recommendation prediction. And the performance of these model may drop when having a larger window. RNNs [39, 47, 11] and their variants [75, 26] are widely used in sequential recommendation. RNN-based models, though effective for short user sequences (e.g., short event sequences within a session), are challenged by long-range dependent patterns in long user sequences. Because of the way they iterate over sequential items [67] and their use of saturating nonlinear functions such as tanh to propagate information through time, RNNs tend to have difficulties leveraging the information contained in states located far into the past due to gradient propagation issues [73, 9]. Even recent architectures designed to facilitate gradient propagation such as Gated Recurrent Unit [21] and Long-short Term Memory [41, 89] have also been shown to suffer from the same problem of not being able to provably account for long-range dependent patterns in sequences [9].

*A second challenge in sequential recommendations is learning user latent factors $P_u$ explicitly from data*, which has been observed to create many difficulties [37, 17, 78]. In the corresponding works, users' long-term preferences have been modeled through learning a set of latent factors $P_u$ for each user. However, learning $P_u$ explicitly is difficult in large-scale

production systems. As the number of users is usually several magnitudes higher than the number of items, building such a large user vocabulary and storing the latent factors in a persistent manner is challenging. Also, the long-tail users (a.k.a cold users) and visitor users (i.e., users who are not logged in) could have much worse recommendations than engaged users [10].

**Limitations of Single Monolithic Models.** Figure 4.1 clearly indicates that although the influence of past user events on future interactions follows a significant decaying trend, significant predictive power can still be carried by events located arbitrarily far in the past. Very recent events (i.e., $1 \leq L \leq 10$) have large magnitude similarities with the current user behavior and this similarity depends strongly on the sequential order of related events. As the distance $L$ grows larger, the informative power of previously consumed items on future user behavior is affected by more uncertainty (e.g., variance) and is less sensitive to relative sequential position. That is, the events from 100 steps ago and from 110 steps ago may have a generally similar influence on future user decisions regardless of their relative temporal location. Therefore, *for the kind of sequential signals we intend to leverage, in which different scales of temporal dependencies co-exist, it may be better to no longer consider a single model*. While simple monolithic models such as Deep Neural Network (DNN) with pooling and dropout [101, 23] are provably robust to noise, they are unfortunately not sensitive to sequential order (without substantial modifications). On the other hand, RNNs [39, 11] provide cutting-edge sequential modeling capabilities but they are heavily sensitive to noise in sequential patterns. Therefore, it is natural to choose a mixture of diverse models which would then complement each other to provide better overall predictive power.

### 4.1.3 Contributions

We address the issue of providing a single model adapted to the diversity of contexts and scales of temporal dependencies in sequential recommendations through data analysis and the design of a Multi-temporal-range Mixture Model, or *M3* for short. We make the following contributions to this problem:

- **Data-driven design:** We demonstrate that in real world recommendation tasks there are significant long-range temporal dependencies in user sequence data, and that previous approaches are limited in their ability to capture those dynamics. M3's design is informed by this quantitative analysis.
- **Multi-range Model:** We offer a single model, M3, which is a mixture model consisting of three sub-models (each with a distinct manually designed architecture) that specialize in capturing different ranges of temporal dependencies. M3 can learn how to dynamically choose to focus on different temporal dynamics and ranges depending on the application context.

- **Empirical Benefits and Interpretability:** We show on both public academic and private data that our approach provides significantly better recommendations. Further, using its interpretable design, we analyze how M3 dynamically switches between patterns present at different temporal ranges for different contexts, thus showing the value in enabling context-specific multi-range modeling. Our private dataset consists in anonymized user sequences from YouTube. To the best of our knowledge this paper is the first to focus on sequential patterns in such a setting.

## 4.2 Proposed Methodology

Motivated by our earlier analyses in Section 4.1.2, we now introduce a novel method aimed at addressing the shortcoming of pre-existing approaches for long user/item interaction sequences: Multi-temporal-range Mixture Model (M3) and its two variants (M3R/M3C). Put everything in a nutshell, M3 is a tailored solution with the following design to overcome the limitations we discussed above:

- Instead of using monolithic model, we adopt a Mixture-of-Experts (MOE) deign to cater the distinct properties from different parts of user sequence.
- To overcome the challenge of modeling long-range dependencies, we use Attention Model, which is insensitive to long temporal range, as one of the experts in complement to RNN/CNN.
- The sequence embedding from M3 encodes both user's short-term and long-term preferences, without the need of explicitly using user embeddings.

### 4.2.1 Overview

Figure 4.2 gives a general schematic depiction of M3. We will now introduce each part of the model separately in a bottom-up manner, starting from the inputs and progressively abstracting their representation which finally determines the model's output.

**Process sequence inputs.** When predicting the next item $\mathcal{S}_{t+1}$ a user is going to interact in her/his logged sequence, we employ item embeddings and context features (optional) from past events as inputs:

$$x_t = [Q_t \oplus c_t^{\text{in}}], \tag{4.1}$$

where $\oplus$ denotes the concatenation operator. To map the raw context features and item embeddings to the same high-dimensional space for future use, a feed-forward layer $F^{\text{in}}$ is used:

$$Z_t^{\text{in}} = \{z_i^{\text{in}}\}_{i=1\cdots t} \text{ where } z_t^{\text{in}} = F^{\text{in}}(x_t) \tag{4.2}$$

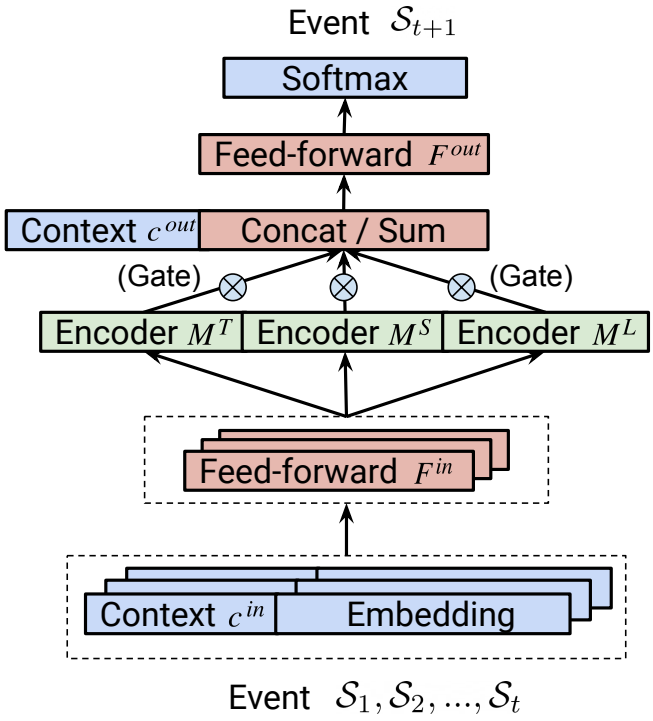



Figure 4.2: An overview of the proposed M3 model. From bottom to top, we first process the sequence inputs with a feed-forward layer $F^{\text{in}}$. Next, we apply three different sequence encoders on the processed sequence and aggregate their results with a gate network. Here $M^T$ is the tiny-range encoder that make prediction based on user's last action; $M^S$ is the short-range encoder using RNN/CNN to encode user's recent actions and $M^L$ is the long-range encoder that can utilize the long-tail of user sequence.

here $z_t^{\text{in}} \in \mathbb{R}^{1\times d_{\text{in}}}$ represents the input processed at step $t$ and $Z_t^{\text{in}} \in \mathbb{R}^{t\times d_{\text{in}}}$ stands for the collection of all processed inputs before step $t$ (included). Either the identity function or a ReLU [68] can be used to instantiate the feed-forward layer $F^{\text{in}}$.

**Mixture of multiple sequence models.** In the previous section, we assessed the limitations of using a single model on long user sequence. To circumvent the issues we highlighted, we employ in M3 three different sequence models (encoders) in conjunction, namely $M^T$, $M^S$ and $M^L$, on top of the processed input $Z_t^{\text{in}}$. We will later explain their individual architectures in details. The general insight is that we want *each of these sub-models to focus on different ranges of temporal dependencies in user sequences* to provide a better representation (i.e., embedding) of the sequence. We want the sub-models to be *architecturally diverse and address each other's shortcomings*. Hence

$$SE_t^T = M^T(Z_t^{\text{in}}),\ SE_t^S = M^S(Z_t^{\text{in}}),\ SE_t^L = M^L(Z_t^{\text{in}}), \tag{4.3}$$

which yields three different representations, one produced by each of the three sequence encoders. The three different sub-model encoders are expected to produce outputs—denoted by $d_{\text{enc}}$—of identical dimension. By construction, each sequential encoder produces its own abstract representation of a given user's logged sequence, providing diverse latent semantics for the same input data.

**Aggregate representations with MOE-like design.** Our approach builds upon the success of Mixture-of-Experts (MOE) model [46]. One key difference is that our 'experts' are constructed to work with different ranges of temporal dependencies, instead of letting the cohort of 'experts' specialize by learning from data. As shown in [84], heavy regularization is needed to learn different experts sharing the same architecture in order to induce specialization and prevent starvation when learning (only one expert performs well because it is the only one to learn which creates a self-reinforcing loop when learning with back-propagation).

Informed by the insights underlying the architecture of MOE models, we aggregate all sequence encoders' results by weighted-concatenate or weighted-sum, with weights $G_t$ computed by a small gating network. In fact, we concatenate the outputs with

$$SE_t = (G_t^T \times SE_t^T) \oplus (G_t^S \times SE_t^S) \oplus (G_t^L \times SE_t^L), \tag{4.4}$$

where $G_t \in \mathbb{R}^3$ corresponds to the outputs of our gating network. We can also aggregate outputs with a weighted-sum:

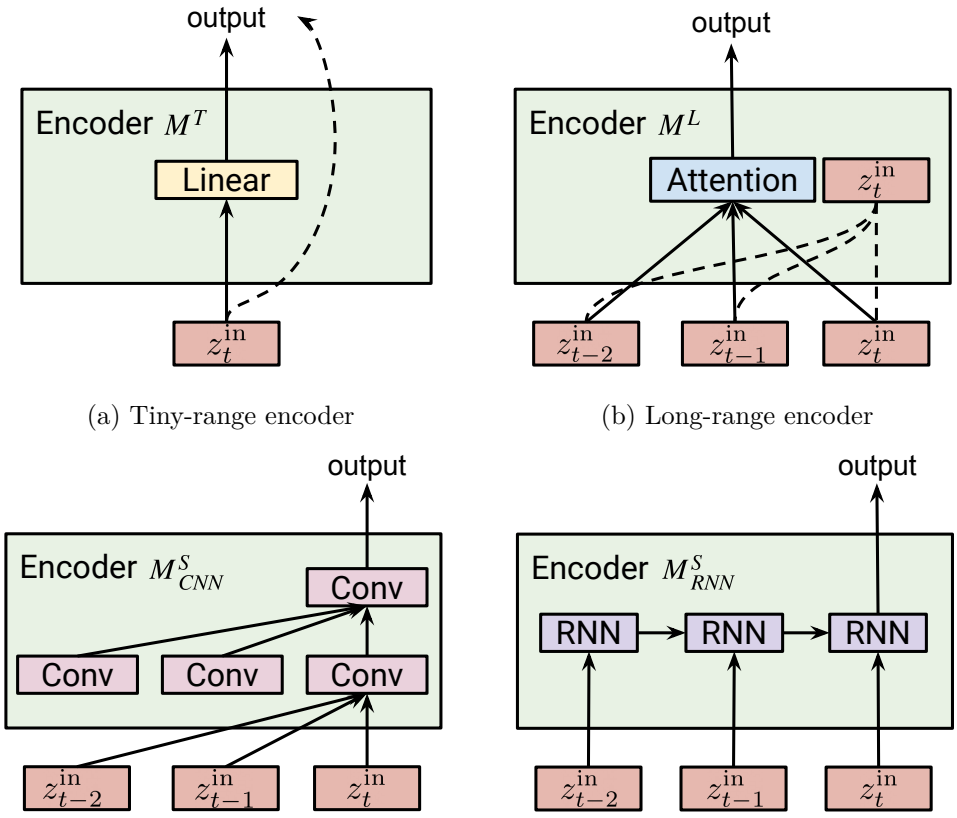
$$SE_t = (G_t^T \times SE_t^T) + (G_t^S \times SE_t^S) + (G_t^L \times SE_t^L). \tag{4.5}$$

Note that there is no theoretical guarantee whether concatenation is better than summation or not. The choice of aggregation, as well as the choice of activation functions, is determined by observing a given model's performance from a validation set extracted from different datasets. Such a procedure is usual in machine learning and will help practitioners determine which variant of the model we propose is best suited to their particular application.

Because of its MOE-like structure, our model can adapt to different recommendation scenarios and provide insightful interpretability (as we shall see in Section 4.3). In many recommendation applications, some features annotate each event and represent the context in which the recommendation query is produced. Such features are for instance indicative of the page or device on which a user is being served a recommendation.

**Output layer.** After obtaining a sequence representation at step $t$ (i.e., $SE_t$), we fuse it with the annotation's context features (optional) at prediction-time and project them to the same latent space with another hidden feed-forward layer $F^{\text{out}}$:

$$z_t^{\text{out}} = F^{\text{out}}([SE_t \oplus c_t^{\text{out}}]) \tag{4.6}$$




(a) Tiny-range encoder

(b) Long-range encoder

(c) Short-range encoder (CNN)

(d) Short-range encoder (RNN)

Figure 4.3: The sequence encoders of *M3*. The solid lines are used to denote the data flow. The dotted line in (a) means an identity copy whereas in (b) it means the interaction of attention queries and keys.

where $c^{\text{out}}$ is a vector encoding contextual information to use after the sequence has been encoded. Here the $z_t^{\text{out}} \in \mathbb{R}^{1 \times d_{\text{out}}}$ is what we name *user representation*, it is computed based on the user's history as it has been gathered in logs and *encodes both user's short-term and long-term preferences*. Finally, a user similarity (relevance) score $r_v$ is predicted for each item via an inner-product (which can be changed to another similarity scoring function):

$$r_v = z_t^{\text{out}} \cdot Q'_i \tag{4.7}$$

where $Q'_i$ is a set of latent factors (a embedding) representing the item $i$. For a given user, item similarity scores are then normalized by a softmax layer which yields a recommendation distribution over the item vocabulary. After training M3, the recommendations for a user at step $t$ are served by sorting the similarity scores $r_i$ obtained for all $i \in \mathcal{I}$ and retrieving the items associated with the highest scores.

### 4.2.2 Different Encoders for Dependencies from Different Ranges

#### Item Co-occurrence as a Tiny-range Encoder

The Tiny-range encoder $M^T$ only focuses on the user's last event $e_t$, ignoring all previous events. In other words, given the processed inputs from past events $Z_t^{\text{in}}$, this encoder will only consider $z_t^{\text{in}}$. As in factorizing Markov chain (FMC) models [78], $M^T$ makes predictions based on item range-1 co-occurrence within observed sequences. For example, if most of users buy iPhone cases after purchasing an iPhone, then $M^T$ should learn this item-to-item co-occurrence pattern. As shown in Figure 4.3a, we compute $M^T$'s output as:

$$M^T(Z_t^{\text{in}}) = \phi(z_t^{\text{in}}), \text{ where } \phi(x) = \begin{cases} xW^{(T)} + b^{(T)}, & \text{if } d_{\text{in}} \neq d_{\text{enc}}, \\ x, & \text{otherwise.} \end{cases} \tag{4.8}$$

That is, when the dimensionality of processed input and encoder output are the same, the tiny-range encoder performs a role of residual for the other encoders in mixture. If $d_{\text{in}} \neq d_{\text{enc}}$ , it is possible to down-sample (if $d_{\text{in}} > d_{\text{enc}}$) or up-sample (if $d_{\text{in}} < d_{\text{enc}}$) from $z_t^{\text{in}}$ by learned parameters $W^{(T)} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{enc}}}$ and $b^{(T)} \in \mathbb{R}^{d_{\text{enc}}}$.

In summary, the tiny-range encoder $M^T$ can only focus on the last event by construction, meaning *it has a temporal range of 1 by design*. If we only use the output of $M^T$ to make predictions, we obtain recommendations results based on item co-occurrence.

#### RNN/CNN as Short-range Encoder

As discussed in Section 4.1, the recent behavior of a user has substantial predictive power on current and future interactions. Therefore, to leverage the corresponding signals entailed in observations, we consider instantiating a short-range sequence encoder that puts more emphasis on recent past events. Given the processed input from past events $Z_t^{\text{in}}$, this encoder, represented as $M^S$, focuses by design on a recent subset of logged events. Based on our quantitative data exploration, we believe it is suitable for $M^S$ to be highly sensitive to sequence order. For instance, we expect this encoder to capture the purchasing pattern iPhone $\rightarrow$ iPhone case $\rightarrow$ iPhone charger if it appears frequently in user sequences. As a result, we believe the Recurrent Neural Network (RNN [67]) and the Temporal Convolutional Network ([94, 7, 106]) are fitting potential architectural choices. Such neural architectures have shown superb performances when modeling high-order causalities. Beyond accuracy, these two encoders are also order sensitive, unlike early sequence modeling method (i.e., Bag-of-Word [50]). As a result we develop two interchangeable variants of M3: *M3R* and *M3C* using an RNN and a CNN respectively.

To further describe each of these options, let us introduce our RNN encoder $M^S_{\text{RNN}}$. As shown in Figure 4.3d we obtain the output of $M^S_{\text{RNN}}$ by first computing the *hidden state* of

RNN at step $t$:

$$h_t = \text{RNN}(z_t^{\text{in}}, h_{t-1}), \tag{4.9}$$

where RNN$(\cdot)$ is a *recurrent cell* that updates the hidden state at each step based on the previous hidden state $h_{t-1} \in \mathbb{R}^{1\times d_{\text{in}}}$ and the current RNN input $z_t^{\text{in}}$. Several choices such as Gated Recurrent Unit (GRU) [21] or Long Short Term Memory (LSTM) [41] can be used. The output is then computed as follows:

$$M^S_{\text{RNN}}(Z_t^{\text{in}}) = h_t W^{(R)}, \text{ where } W^{(R)} \in \mathbb{R}^{d_{\text{in}}\times d_{\text{enc}}} \tag{4.10}$$

where $W^{(R)}$ maps the hidden state to the encoder output space. We design our CNN encoder $M^S_{\text{CNN}}$ as a Temporal Convolutional Networks which has provided state-of-art sequential modeling performance [94, 31, 7]. As shown in Figure 4.3c, this encoder consists of several stacked layers. Each layer computes

$$h_t^{(1)} = \text{Conv}(Z_t^{\text{in}}), \ldots, h_t^{(k)} = \text{Conv}(h_t^{(k-1)}), \tag{4.11}$$

where $k$ indicates the layer number. The Conv$(\cdot)$ is a 1-D convolutional operator (combined with non-linear activations, see [7] for more details), which contains $d_{\text{enc}}$ convolutional filters and operates on the convolutional inputs. With $K$ layers in our CNN encoder, the final output will be:

$$M^S_{\text{CNN}}(Z_t^{\text{in}}) = h_t^{(K)}. \tag{4.12}$$

As highly valuable signals exist in the short-range part of user sequence, we propose two types of encoders to capture them. Our model can be instantiated in its first variant, *M3R*, if we use RNN encoder or *M3C* if a CNN is employed. Here M3C and M3R are totally interchangeable with each other and they show comparable results in our experiments (see Section 4.3.2). We believe such flexibility will help practitioners adapt their model to the hardware they intend to use, i.e., typically using GPU for faster CNN training or CPU for which RNNs are better suited. In terms of temporal range, the CNN only considers a limited finite window of inputs when producing any output. The RNN, although it does not have a finite receptive field, is hampered by difficulties when learning to leverage events located further back into the past (to leverage an event located $L$ observations ago the RNN needs $L-1$ steps). Regardless of the choice of a CNN or an RNN, our short-range encoder $M^S$ *has a temporal range greater than 1, although it is challenging for this sub-model to capture signals too far away from current step.* This second encoder is specifically designed to capture sequence patterns that concern recent events.

**Attention Model as Long-range Encoder**

The choice of an attention model is also influenced by our preliminary quantitative analysis. As discussed in Section 4.1, as the temporal distance grows larger, the uncertainties affecting

the influence of item consumption on future events get larger as well. Moreover, as opposed to the recent part of a given user's interaction sequence, relative position does not matter as much when it comes to capturing the influence of temporally distant events. As we take these properties into account, we choose to employ *Attention Model* [6, 95] as our long-range sequence encoder. Usually, an attention model consists of three parts: attention *queries*, attention *keys* and attention *values*. One can simply regard an attention model as weighted-sum over attention values with weights resulting from the interaction between attention queries and attention keys. In our setting, we use (1) the last event's processed input $z_t^{\text{in}}$ as attention queries, (2) all past events' processed inputs $Z_t^{\text{in}}$ as keys and values and (3) scaled dot-product [95] as the similarity metric in the attention softmax. For instance, if a user last purchased a pair of shoes, the attention mechanism will focus on footwear related previous purchases.

So that all encoders have the same output dimensionality, we need to transform[2] our processed input first as follows:

$$\tilde{Z}_t^{\text{in}} = Z_t^{\text{in}} W^{(A)}, \tag{4.13}$$

where $W^{(A)} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{enc}}}$ is a learned matrix of parameters. Then for each position $i \in [1, t]$, we obtain its raw attention weights, with respect to the processed input $\tilde{z}_i^{\text{in}}$, as follows:

$$\omega_{t,i} = \frac{\tilde{z}_t^{\text{in}} \cdot \tilde{z}_i^{\text{in}}}{\sqrt{d_{\text{enc}}}}, \tag{4.14}$$

where $\omega_{t,i}$ is the raw weight at position $i$. Similarly, we compute the raw attention weights $\boldsymbol{\omega}_t \in \mathbb{R}^{1 \times t}$ for all positions $\boldsymbol{\omega}_t = \{\omega_i\}_{i=1..t}$ and normalize them with a softmax$(\cdot)$ function. Finally, we acquire the output of our long-range encoder as follows:

$$M^L(Z_t^{\text{in}}) = \text{softmax}(\boldsymbol{\omega}_t) Z_t^{\text{in}}. \tag{4.15}$$

Our long-range encoder borrows several advantages from the attention model. First, it is not limited by a certain finite temporal range. That is, *it has an unlimited temporal range and can 'attend' to anywhere in user's sequence with O(1) steps.* Second,because it computes its outputs as a weighted sum of inputs, the attention-based encoder is not as sensitive to sequential order as an RNN or a CNN as each event from the past has an equal chance of influencing the prediction. Third, the attention model is robust to noisy inputs due to its normalized attention weights and weighted-sum aggregation.

### Gating Network

Borrowing the idea from from Mixture-of-Experts model [46, 65], we build a gating network to aggregate our encoders' results. The gate is also helpful to better understand our

[2]It is unnecessary if $d_{\text{in}}$ is same as $d_{\text{enc}}$.

model (see Section 4.3). To produce a simpler gating network, we use a feed-forward layer $F^g$ on the gating network's inputs:

$$G_t = [G_t^T, G_t^S, G_t^L] = \text{sigmoid}(F^g(G_t^{\text{in}})), \tag{4.16}$$

where $G_t^{\text{in}}$ is the input we feed into our gating network. We will discuss how the model performs overall with different choices of gate inputs in Section 4.3.4. The resulting $G_t \in \mathbb{R}^3$ contains the gate value modulating each encoder. More importantly, an element-wise sigmoid function is applied to the gate values which allows encoders to 'corporate' with each other [9]. Note that a few previous works [65, 49, 84] also normalize the gate values, but we found this choice led to the degeneracy of our mixture model as it would learn to only use $M^S$ which in turn hampers model performance.

**Summary**

M3 is able to address limitations of pre-existing models as shown in Table 4.1: (1) M3 has a mixture of three encoders with different temporal ranges which can capture sequential patterns located anywhere in user sequences. (2) Instead of learning a set of latent factor $P_u$ for each user, M3 represents the long-term user preferences by using a long-range sequence encoder that provides a representation of the entire history of a user. Furthermore, M3 is efficient in both model size and computational cost. In particular M3 does not introduce any extra parameters under certain settings (i.e., $d_{\text{in}} = d_{\text{enc}}$), and the computation of $M^T$ and $M^L$ are very efficient when using specialized hardware such as a GPU. With its simple gate design, M3 also provides good interpretability and adaptability.

- **Effectiveness.** Given our analysis on user sequences, we assume M3 to be effective. As compared to past works, M3 is capable to capture signals from the whole sequence, it also satisfies the properties we found in different parts of sequence. Moreover, our three encoders constitute a diverse set of sequential encoder and, if well-trained, can model user sequence in a multi-scale manner, which is a key to success in past literature [94, 105].

- **Efficiency.** In terms of model size, M3 is efficient. As compared to existing works which use short-range encoder only, though uses two other encoders, our M3 model doesn't introduce any extra parameters (if $d_{\text{in}} = d_{\text{enc}}$). In terms of computational efficiency, our M3 is good as well, as both $M^T$ and $M^L$ are nothing other than matrix multiplication, which is cheap when computed with optimized hardwares like Graphics Processing Unit (GPU).

- **Interpretability.** Model's interpretability is critical for diagnosing purpose. As we shall see later, with the gate network, we are able to visualize our network transparently by observing the gate values.

Table 4.1: A summary of relationships and differences between sequence encoders in *M3*.

| | **Base model** | **Temporal range** | **Model size** | **Sensitive to order** | **Robustness** |
|---|---|---|---|---|---|
| $M^T$ | Item Co-occurrence | 1 | small (or 0) | very high | no |
| $M^S_{\text{RNN}}$ | Recurrent Neural Nets | unknown | large | high | no |
| $M^S_{\text{CNN}}$ | Temporal Convolution Nets | limited | large | high | no |
| $M^L$ | Attention Model | unlimited | small (or 0) | no | high |

- **Adaptability.** One issue in production recommender system is modeling users for different recommendation scenarios, as people may behave very differently. Two typical scenarios are *HomePage* recommendation and product *DetailPage* recommendation. However, as we shall introduce in later section, M3 is able to adapt to these scenarios if we use the scenario information as our gate input.

## 4.3 Experimental Studies

In this section, we study the two variants of M3 against several baseline state-of-the-art methods on both a publicly available dataset and our large-scale Youtube dataset.

**Datasets.** We use MovieLens 20M[3], which is a publicly available dataset, along with a large-scale anonymized dataset from YouTube, which is private, anonymized and at production-scale (much larger than any publicly available datasets).

### 4.3.1 Experiments on MovieLens Dataset

#### Experimental Setup

As in previous works [37], we process the MovieLens data by first converting numeric ratings to 1 values, turning them into implicit logged item consumption feedback. We remove the items with less than 20 ratings. Such items, because of how little user feedback is available for them, represent another research challenge — cold start — which is outside the scope of the present paper.

To focus on long user sequences, we filtered out users who had a sequence length of less than $\delta_{\text{min}} = 20$ item consumed, while we didn't filter items specifically. The maximum sequence length in the dataset being 7450, we follow the method proposed in [37] and employ a sliding window of length $\delta_{\text{win}} = 300$ to generate similarly long sequences of user/item interactions in which we aim to capture long range dependent patterns. Some statistics can be found in the first row of Table 4.3.

We do not use contextual annotations for the MovieLens data.

**Evaluation protocol.** We split the dataset into training and test set by randomly choosing 80% of users for training and the remaining 20% for validation (10%) and testing (10%). As with the training data, a sliding window is used on the validation and test sets to generate sequences. We measure the mean average precision (mAP) as an indicator for models' performances [10]. We only focus on the top positions of our predictions, so we choose to use mAP@n with $n \in \{5, 10, 20\}$. There is only one target per instance here and therefore

[3]https://grouplens.org/datasets/movielens/20m/

the mAP@n is expected to increase with $n$ which is consistent with [9] but differs from the performances we presented in Chapter 3.

**Model details.** We keep architectural parameters consistent across all experiments on MovieLens. In particular, we use identical representation dimensions: $d_{\text{in}} = d_{\text{enc}} = d_{\text{out}} = 32$. Such a choice decreases the number of free parameters as the sub-models $M^T$ and $M^L$ will not have learned parameters. A GRU cell is employed for the RNN while 2 stacked temporal convolution layers [7] of width 5 are used in the CNN. A ReLU activation function is employed in the feed-forward layers $F^{\text{in}}$ and $F^{\text{out}}$. Item embeddings of dimension 64 are learned with different weights on the input side (i.e., $Q$ in Eq. 4.1) and output side (i.e., $Q'$ in Eq. 4.7). Although previous work [37] has constrained such embeddings to be identical on the input and output side of the model, we found that increasing the number of degrees of freedom led to better results.

**Baselines.** We compare our two variants, i.e., M3R and M3C, with the following baselines:

- **FMC**: The Factorizing model for the first-order Markov chain (FMC) [78] is a simple but strong baseline in sequential recommendation task [11, 85]. As discussed in Section 1, we do not want to use explicit user representations. Therefore, we do not compare the personalized version of this model (FPMC).

- **DeepBoW**: The Deep Bag-of-word model represent user by averaging item embeddings from all past events. The model then makes predictions through a feed-forward layer. In our experiments, we use a single hidden layer with size of 32 and ReLU as activation function.

- **GRU4Rec**: Originally presented in [39], this method uses a GRU RNN over user sequences and is a state-of-the-art model for sequential recommendation with anonymized data.

- **Caser**: The Convolutional Sequence Embeddings model (proposed in Chapter 3) applying horizontal and vertical convolutional filters over the embedding matrix and achieves state-of-the-art sequential recommendation performance. We try $\{2, 4, 8\}$ vertical filters and $\{16, 32, 64\}$ horizontal filters of size $(3, 5, 7)$. In order to focus on the sequential encoding task, we discard the user embedding and only use the sequence embedding of this model to make predictions.

In the models above, due to the large number of items in input and output dictionaries, the learned embeddings comprise most of the free parameters. Therefore, having set the embedding dimension to 64 in all the baselines as well as in M3R and M3C, we consider models with similar numbers of learned parameters. The other hyperparameters mentioned above are tuned by looking at the mAP@20 on validation set. The training time

of M3R/M3C is comparable with others and can be further improved with techniques like model compression [91], quantization [44], etc.

### Overall Performances

We report each model's performance in Table 4.2. Each metric is averaged across all user sequences in test set. The best performer is highlighted in bold face. The results show that both M3C and M3R outperform other baselines by a large margin. Among the baselines, GRU4Rec achieves the best performance and DeepBoW worst one, suggesting the sequence order plays a very important predictive role. FMC performs surprisingly well, suggesting we could get considerable results with a simple model only taking the last event into account. The poor results of Caser may be caused by its design which relies on vertical filters of fixed size. Caser performs better in the next subsection which considers sequences whose lengths vary less within the training data.

Table 4.2: Performance comparison on MovieLens 20M. M3C and M3R outperform the baselines significantly.

| | **mAP@5** | **mAP@10** | **mAP@20** |
|---|---|---|---|
| **FMC** | 0.0256 | 0.0291 | 0.0317 |
| **DeepBoW** | 0.0065 | 0.0079 | 0.0093 |
| **GRU4Rec** | 0.0256 | 0.0304 | 0.0343 |
| **Caser** | 0.0225 | 0.0269 | 0.0304 |
| **M3C** | 0.0295 | 0.0342 | 0.0379 |
| **M3R** | **0.0315** | **0.0367** | **0.0421** |
| Improv. | +23.4% | +20.7% | +22.7% |

### Investigating the influence of sequence length through variants of MovieLens

The previous results have shown strong performance gains achieved by the models we intruced: M3C and M3R. We now investigate the origin of such improvements. The design of these models was inspired by an attempt to capture sequential patterns with different characteristic temporal extents. To check whether the models we introduced achieve this aim we construct multiple variants of MovieLens with different sequence lengths.

We vary the sequence length by having a maximum cutoff threshold $\delta_{\max}$ which complements the minimal sequence length threshold $\delta_{\min}$. A sequence with more than $\delta_{\max}$ only has its latest $\delta_{\max}$ observations remained. We vary the values of $\delta_{\min}$, $\delta_{\max}$ and the sequence generation window size. Table 4.3 summarizes the properties of the four variants of the MovieLens dataset we construct. It is noteworthy that such settings make Caser perform better as the sequence length is more consistent within each dataset variant.

Table 4.3: Statistics of the variants of the MovieLens dataset.

| | Min. length | Max. length | Window size | Avg. length | Num. sequences | Num. items |
|---|---|---|---|---|---|---|
| **ML20M** | 20 | $\infty$ | 300 | 144.1 | 138.4K | 13.1K |
| **ML20M-S** | 20 | 50 | 20 | 42.8 | 138.4K | 13.1K |
| **ML20M-M** | 50 | 150 | 50 | 113.6 | 85.2K | 13.1K |
| **ML20M-L** | 150 | 300 | 150 | 250.7 | 35.8K | 12.9K |
| **ML20M-XL** | 300 | $\infty$ | 300 | 605.5 | 16.3K | 12.5K |

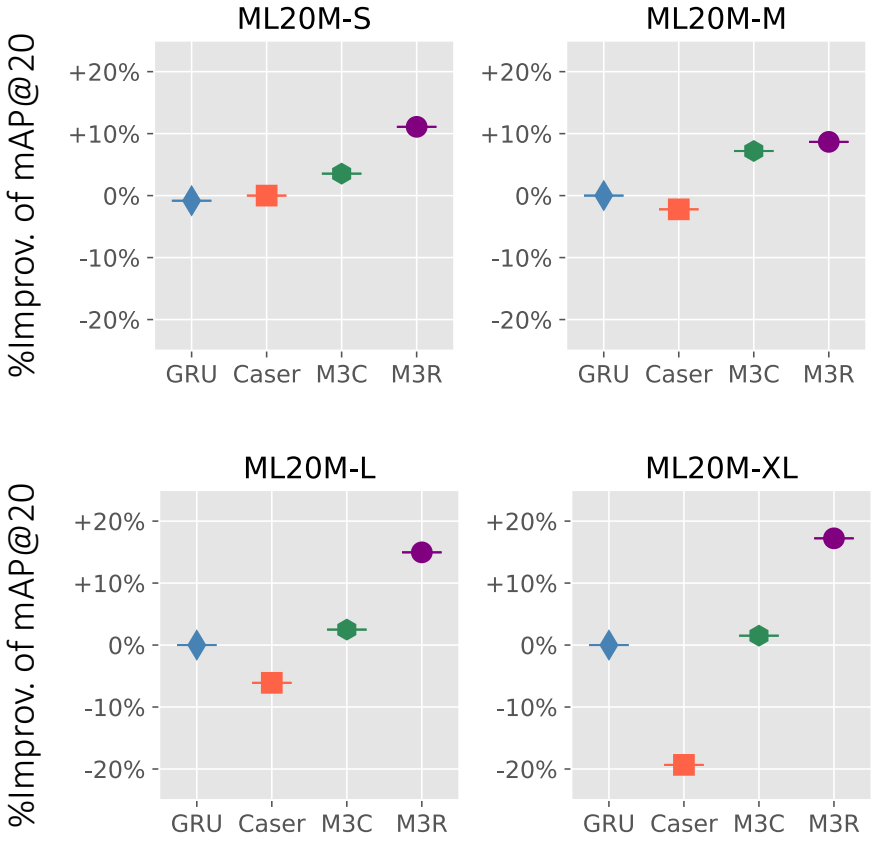



Figure 4.4: Uplifts with respect to the best baselines on four MovieLens variants. The improvement percentage of each model is computed by its relative mAP@20 gain against the best baseline. For all variants, M3R significantly outperforms the two baselines we consider according to a one-tail paired t-test at level 0.01, while M3C outperforms the other two significantly only on ML20M-M. Note that the standard error of all uplifts gets higher as we use a MovieLens variant with longer sequences. The standard error reaches 2.3% on ML20M-XL.

GRU4Rec and Caser outperform the other baselines in the present setting and therefore we only report their performance. Figure 4.4 shows the improvements of M3C and M3R over

the best baselines on four MovieLens variants. The improvement of each model is computed by its mAP@20 against the best baseline. In most cases, M3C and M3R can outperform the highest performing baseline. Specifically, on ML20M-S and ML20M-M, Caser performs similarly to GRU4Rec while both M3C and M3R have good performance. This is probably due to the contribution of the tiny-range encoder.

### 4.3.2 Experiments on Anonymized YouTube Dataset

#### Experimental Setup

For the YouTube dataset, we filtered out users whose logged sequence length was less than 150 ($\delta_{\min} = 150$) and keep each user's last 300 events ($\delta_{\max} = 300$) in their item consumption sequence. In the following experiments, we exploit contextual annotations such as user device (e.g., from web browser or mobile App), time-based features (e.g., dwelling time), etc. User sequences are all anonymized and precautions have been taken to guarantee that users cannot be re-identified. In particular, only public videos with enough views have been retained.

Neural recommender systems attempt at foreseeing the interest of users under extreme constraints of latency and scale. We define the task as predicting the next item the user will consume given a recorded history of items already consumed. Such a problem setting is indeed common in collaborative filtering [82, 62] recommendations. We present here results obtained on a dataset where only about 2 million items are present that correspond to most popular items. While the user history can span over months, only watches from the last 7 days are used for labels in training and watches in the last 2 days are used for testing. The train/test split is 90/10%. The test set does not overlap with the train set and corresponds to the last temporal slice of the dataset. In all, we have more than 200 million training sequences and more than 1 million test sequences, and with overall average sequence length approximately being 200.

The neural network predicts, for a sample of negatives, the probability that they are chosen and classically a negative sampling loss is employed in order to leverage observations belonging to a very large vocabulary [48]. The loss being minimized is

$$\sum_{l \in \text{Labels}} w_l \times \text{CrossEntropy}(\text{SampledSoftmax}(\xi(t+1)))$$

where the SampledSoftmax [48] uses 20000 randomly sampled negatives and $w_l$ is the weight of each label.

**Evaluation protocol.** To test the models' performances, still we measure mAP@$n$ with $n \in \{5, 10, 20\}$. The mAP is computed with the entire dictionary of candidate items as opposed to the training loss which samples negatives.

Table 4.4: Performance comparison on the anonymized YouTube dataset. M3C and M3R outperform the baselines significantly.

| | **mAP@5** | **mAP@10** | **mAP@20** |
|---|---|---|---|
| **Context-FMC** | 0.1103 | 0.119 | 0.1240 |
| **DeepYouTube** | 0.1295 | 0.1399 | 0.1455 |
| **Context-GRU** | 0.1319 | 0.1438 | 0.1503 |
| **M3C** | 0.1469 | 0.1591 | 0.1654 |
| **M3R** | **0.1541** | **01670** | **0.1743** |
| Improv. | +16.8% | +16.1% | +16.0% |

**Baselines.** In order to make fair comparisons with all previous baselines, we used their contextual counterparts if they are proposed or compared in literature.

- **Context-FMC**: The Context-FMC condition the last event's embedding on last event's context features by concatenating them and having a feed-forward layer over them.
- **DeepYouTube**: Proposed by [23], the DeepYoutube model is a state-of-the-art neural model for recommendation. It concatenates: (1) item embedding from users' last event, (2) item embeddings averaged by all past events and (3) context features. The model then makes predictions through a feedforward layer composed of several ReLU layers.
- **Context-GRU**: We used the contextual version of GRU proposed in [85]. Among the three conditioning paradigms on context, we used the concatenation as it gives us better performances.

All models are implemented by TensorFlow [1] and by Adagrad [28] over a parameter server [61] with many workers.

**Model details.** In the following experiments, we keep the dimensions of processed input $d_{\text{in}}$ and encoder outputs $d_{\text{enc}}$ identical for all experiments conducted on the same dataset. Once more, we also want to share some of our architectural parameters so that they are consistent across the two datasets. Again, by doing this, we make the parametrization of our models more parsimonious, because the sub-models $M^T$ and $M^L$ will be parameter-free. For the RNN cell, we use a GRU on both datasets for its effectiveness as well as efficiency. For the CNN version, we stacked 3 layers of temporal convolution [7], with no dilation and width of 5. For the feed-forward layers $F^{\text{in}}$ and $F^{\text{out}}$, we used ReLU as their activation functions, whereas they contains different number of sub-layers. For item embeddings on the input side (i.e., $Q$ in Eq. 4.1) and on the output side (i.e., $Q'$ in Eq. 4.7), we learn them separately which improves all results.

Table 4.5: mAP@20 vs. different components of M3R on both datasets, where T,S,L stands for $M^T$, $M^S$ and $M^L$ respectively.

| | **MovieLens 20M** | **YouTube Dataset** |
|---|---|---|
| M3R-T | 0.0269 | 0.1406 |
| M3R-S | 0.0363 | 0.1673 |
| M3R-L | 0.0266 | 0.1359 |
| M3R-TS | 0.0412 | 0.1700 |
| M3R-TL | 0.0293 | 0.1485 |
| M3R-SL | 0.0403 | 0.1702 |
| M3R-TSL | 0.0421 | 0.1743 |

**Overall Performances**

We report each model's performance on the private dataset in Table 4.4. The best performer is highlighted in bold face. As can be seen from this table, on our anonymized YouTube dataset, the Context-FMC performs worse followed by DeepYoutube while Context-GRU performs best among all baselines. The DeepYouTube and Context-GRU perform better than Context-FMC possibly because they have longer temporal range, which again shows that the temporal range matters significantly in long user sequences. One can therefore improve the performance of a sequential recommender if the model is able to leverage distant (long-range dependent) informantion in user sequences.

On both datasets, we observed our proposed two model variants M3R and M3C significantly outperform all other baselines. Within these two variants, the M3R preforms marginally better than the M3C, and it improves upon the best baselines by a large margin (more than 20% on MovieLens data and 16.0% on YouTube data).

### 4.3.3 Ablation Study of Mixture of Models

To demonstrate how each encoder contributes to the overall performance, we now present an ablation test on our M3R model (results from M3C are similar) on our proprietary data. We use T, S, L to denote $M^T$, $M^S$ and $M^L$ respectively. The results are described in Table 4.5. When we only enable single encoder for M3R, the best performer is M3R-T on MovieLens data and M3R-S on the YouTube data. This result is consistent with the results in Section 4.3.2. With more encoders involved in M3R the model performs better. In particular, when all encoders are incorporated, our M3R-TSL performs best on both datasets, indicating all three encoders matter for performance.

Table 4.6: mAP@20 vs. different types of gating network on the two datasets for M3R. 'Fixed' indicates we fix gate values to 1.0, 'Contextual-switch' means that we use context features $c^{\text{in}}$ and $c^{\text{out}}$ as gate input and 'Bottom-switch' corresponds to the use of $z_t^{\text{in}}$ as gate input.

| | **MovieLens** | **YouTube** |
|---|---|---|
| Fixed | 0.0413 | 0.1715 |
| Bottom-switch | 0.0421 | 0.1734 |
| Contextual-switch | / | 0.1743 |

### 4.3.4 Role of Gating Network

We now begin to study our gating network in order to answer the following questions:

1. Is the gating network beneficial to the overall model performance?
2. How do different gating network inputs influence the model performance?
3. How can the gating network make our model more adaptable and interpretable?

**Fixed gates versus learned gates.** First of all, we examine the impact of our gating network by comparing it with a set of fixed gate values. More precisely, we fixed the gate values to be all equal to 1.0 during the model training: $G_t = \mathbf{1}$, here $\mathbf{1} \in \mathbb{R}^3$ is a vector. The first row of Table 4.6 shows the result of this fixed-gate model. We found that the fixed models are weaker than the best performing version of M3R (i.e., mAP@20 of 0.1743) and M3C (i.e., mAP@20 of 0.1654). This reveals that the gating network consistently improves M3-based models' performances.

**Influence of different gate inputs.** In this paragraph we investigate the potential choices of inputs for the gating network, and how they result in different performance scores. In the existing Mixture-of-Experts (MOE) literature, the input for the gating network $G_t^{\text{in}}$ can be categorized into Contextual-switch and Bottom-switch. The Contextual-switch, used in [9], uses context information as gate input:

$$G_t^{\text{in}} = [c_t^{\text{in}} \oplus c_t^{\text{out}}], \tag{4.17}$$

where $c_t^{\text{in}}$ and $c_t^{\text{out}}$ are context features from input and output side. Intuitively, this suggests how context may influence the choices of different encoders. If no context information is available, we can still use the output of a shared layer operating before the MOE layer [65, 84] as gate input, i.e., Bottom-switch:

$$G_t^{\text{in}} = z_t^{\text{in}}. \tag{4.18}$$

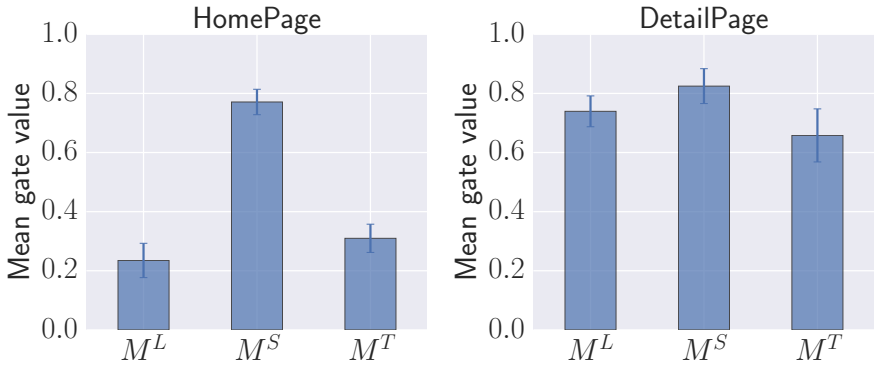



Figure 4.5: Average gate values of M3R in different scenarios. The model learns to use different combination of encoders in different recommendation scenarios.

The shared layer contains high-level semantic knowledge from the last event, which can also enable gate switching.

On the MovieLens data, we used Bottom-switched gate for all the results above because of the absence of contextual annotations. On the YouTube dataset, the last two rows from Table 4.6 provide the comparison results between Contextual-switched gate and Bottom-switched gate. We observe that context information is more useful to the gates than a shared layer. In other words, the decision of whether to focus more on recent part (i.e., large gate values for $M^T$ and $M^S$) or on the distant part (i.e., large values for $M^L$) from user sequence is easier to make based on contextual annotations.

**Discussion on model interpretability and adaptability.** The model architecture we design is based on quantitative findings and has two primary goals: capturing co-existing short-range and long-range behavioral patterns as well as serving recommendations given in different contexts with a single model.

We know for recommender systems in most applications (*e.g.* e-commerce like Amazon, streaming services like Netflix) that recommendations commonly occur in at least two different contexts: either a *HomePage* or a *DetailPage*. The Homepage is the page shown when users open the website or open the mobile App, while DetailPage is the page shown when users click on a certain item. User behaviors are different depending on which of these two pages they are browsing. Users are more likely to be satisfied by a recommendation related to recent events, especially the last event, when they are on a DetailPage. A straightforward solution to deal with these changing dynamics is to train two different models.

We now demonstrate that with the multi-temporal-range encoders architecture and gating mechanism in M3, we can have a single adaptive end-to-end model that provides good performance in a multi-faceted recommendation problem. To that end we analyze the

behavior of our gating network and show the adaptability of the model as we gain a better understanding of its behavior.

What we observe in Figure 4.5 is that when contextual information is available to infer the recommendation scenario, the gating network can effectively automatically decide how to combine the results from different encoders in a dynamic manner to further improve performance. Figure 4.5 shows how gate values of M3R change w.r.t across different recommendation scenarios. It is clear that M3R puts more emphasis on $M^S$ when users are on the HomePage, while it encourages all three encoders involved when users are on DetailPage. This result shows that the gating network uses different combinations of encoders for different recommendation scenarios.

As a result, we can argue that our architectural design choices do meet the expectations we set in our preliminary analysis. It is noteworthy that the gating mechanism we added on top of the three sub-models is helpful to improve predictive performance and ease model diagnosis. We have indeed been able to analyze recommendation patterns seamlessly.

## 4.4 Conclusion

M3 is an effective solution to provide better recommendations based on long user sequences. M3 is a neural model that avoids most of the limitations faced by pre-existing approaches and is well adapted to cases in which short term and long term temporal dependencies coexist. Other than effectiveness, this approach also provides several advantages such as the absence of a need extra parameters and interpretability. Our experiments on large public dataset as well as a large-scale production dataset suggest that M3 outperforms the state-of-the-art methods by a large margin for sequential recommendation with long user sequences. One shortcoming of the architecture we propose is that all sub-models are computed at serving time. As a next step, we plan to train a sparse context dependent gating network to address this shortcoming.

# Chapter 5

# On Learning Compact Model for Efficient Recommendation

The previous chapters showed how to design model to capture different types of sequential patterns in different parts of user sequence. However, our proposed neural network based approaches may incur a large inference time, resulting an issue when responding user requests in the real-time. Recall that in Chapter 2, we analyzed the different efficiency requirements for different phases of recommendation models. For sequential recommendation task, since users' action sequences are dynamically changing while interacting with the system, the inference phase usually has to be done more frequently and in the real-time. Therefore, besides the effectiveness of every model, we also have tight efficiency constraints. It is known that more model complexity (especially larger model size) will benefit the effectiveness but will hurt the efficiency for neural network based model. *How can we have a model-agnostic approach to learn a compact model that maintians similar performance?*

## 5.1 Background and Motivations

Balancing effectiveness and efficiency has been a line of recent recommendation research [109, 113, 111, 93, 60]. Discrete hashing techniques [109, 110] and binary coding of model parameters [113] are suggested to speed up the calculation of the relevance score for a given user-item pair. Other works focus on database-related methods, such as pruning and indexing to speed-up retrieval of related items [93, 60], using fast models for candidate generation and applying time-consuming models to the candidates for online inferences [23, 63]. These methods either lose much of effectiveness, due to the introduced model constraints, or cannot be easily extended to other models in most cases, due to the model-dependency nature. In the following, we first review the learning to rank problem in general (we will regard sequential recommendation as an application), then revisit the issues of effectiveness and efficiency in the problem, which serves to motivate our ranking distillation.

### 5.1.1 Ranking from scratch

Without loss of generality, we use the IR terms “query" $q$ and “document" $d$ in our discussion, but these terms can be replaced with “user profile" and “item" when applied to recommender systems.

The learning to rank problem can be summarized as follows: Given a set of queries $\mathcal{Q}=\{q_1,\cdots,q_{|\mathcal{Q}|}\}$ and a set of documents $\mathcal{D}=\{d_1,\cdots,d_{|\mathcal{D}|}\}$, we want to retrieve documents that are most relevant to a certain query. The degree of relevance for a query-document pair $(q,d)$ is determined by a relevance score. Sometimes, for a single $(q,d)$ pair, a relevance score $y_d^{(q)}$ is labeled by human (or statistical results) as ground-truth, but the number of labeled $(q,d)$ pairs is much smaller compared to the pairs with unknown labels. Such labels can be binary (i.e., relevant/non-relevant) or ordinal (i.e., very relevant/relevant/non-relevant). In order to rank documents for future queries with unknown relevance scores, we need a ranking model to predict their relevance scores. A ranking model $M(q,d;\boldsymbol{\theta})=\hat{y}_d^{(q)}$ is defined by a set of model parameters $\boldsymbol{\theta}$ and computes a relevance score $\hat{y}_d^{(q)}$ given the query $q$ and document $d$. The model predicted document ranking is supervised by the human-labeled ground truth ranking. The optimal model parameter set $\boldsymbol{\theta}^*$ is obtained by minimizing a ranking-based loss function:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{q\in\mathcal{Q}} \mathcal{L}^R(\boldsymbol{y}^{(q)}, \hat{\boldsymbol{y}}^{(q)}). \tag{5.1}$$

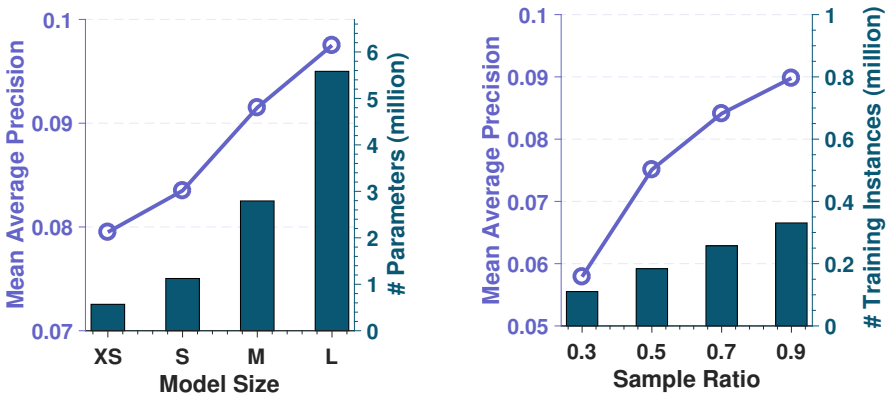
For simplicity, we focus on a single query $q$ and omit the superscripts related to queries (i.e., $y_d^{(q)}$ will becomes $y_d$).

The ranking-based loss could be categorized as point-wise, pair-wise, and list-wise. Since the first two are more widely adopted, we don't discuss list-wise loss in this work. The point-wise loss is widely used when relevance labels are binary [38]. One typical point-wise loss is taking the negative logarithmic of the likelihood function:

$$\begin{aligned}\mathcal{L}^R(\boldsymbol{y},\hat{\boldsymbol{y}}) = -(&\sum_{d\in\boldsymbol{y}_{d+}} \log(P(rel=1|\hat{y}_d)) \\ &+ \sum_{d\in\boldsymbol{y}_{d-}} \log(1-P(rel=1|\hat{y}_d))),\end{aligned} \tag{5.2}$$

where $\boldsymbol{y}_{d+}$ and $\boldsymbol{y}_{d-}$ are the sets of relevant and non-relevant documents, respectively. We could use the *logistic* function $\sigma(x)=1/(1+e^{-x})$ and $P(rel=1|\hat{y}_d)=\sigma(\hat{y}_d)$ to transform a real-valued relevance score to the probability of a document being relevant ($rel=1$). For ordinal relevance labels, pair-wise loss better models the partial order information:

$$\mathcal{L}^R(\boldsymbol{y},\hat{\boldsymbol{y}}) = -\sum_{d_i,d_j\in\boldsymbol{C}} \log(P(d_i \succ d_j|\hat{y}_i,\hat{y}_j)), \tag{5.3}$$




(a) MAP vs. model size (b) MAP vs. training instances

Figure 5.1: Two ways of boosting mean average precision (MAP) on Gowalla data for recommendation. (a) shows that a larger model size in number of parameters, indicated by the bars, leads to a higher MAP. (b) shows that a larger sample size of training instances leads to a higher MAP.

where $\boldsymbol{C}$ is the set of document pairs $\{(d_i, d_j) : y_i \succ y_j\}$ and the probability $P(d_i \succ d_j)$ can be modeled using the *logistic* function $P(d_i \succ d_j | y_i, y_j) = \sigma(y_i - y_j)$.

### 5.1.2 Rethinking Effectiveness and Efficiency

We consider ranking models with latent factors or neural networks (a.k.a neural ranking models) instead of traditional models (e.g., SVM, tree-based models) for the following reasons. First, these models are well-studied recently for its capability to capture features from a latent space and are shown to be highly effective; indeed, neural ranking models are powerful for capturing rich semantics for queries and documents, which eliminates the tedious and ad-hoc feature extraction and engineering normally required in traditional models. Second, these models usually require many parameters and suffer from efficiency issue when making online inferences. Third, traditional models like SVM usually has convex guarantees and are trained through convex optimization. The objectives of latent factor models and neural networks are usually non-convex [22, 52], which means that their training processes are more challenging and need more attentions.

The goal of ranking models is predicting the rank of documents as accurately as possible near the top positions, through learning from human-labeled ground-truth document ranking. Typically, there are two ways to make a ranking model perform better at top positions:

1. By having a large model size, as long as it doesn't overfit the data, the model could better capture complex query-document interaction patterns and has more predictive capability. Figure 5.1a shows that, when a ranking model has more parameters, it

acquires more flexibility to fit the data and has a higher MAP, where the mean average precision (MAP) is more sensitive to the precision at top positions.

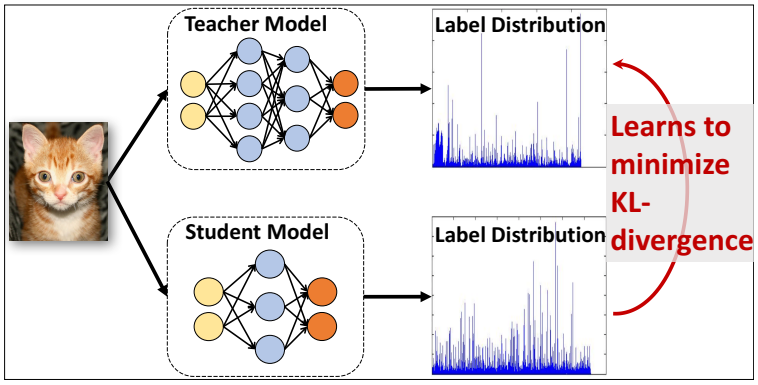
2. By having more training data, side information, human-defined rules etc., the model can be trained with more guidance and has less variance in gradients [40]. Figure 5.1b shows that, when more training instances are sampled from the underlying data distribution, a ranking model could achieve a better performance.

However, each method has its limitations: method (1) surrenders efficiency for effectiveness whereas method (2) requires additional informative data, which is not always available or is expensive to obtain in practice.

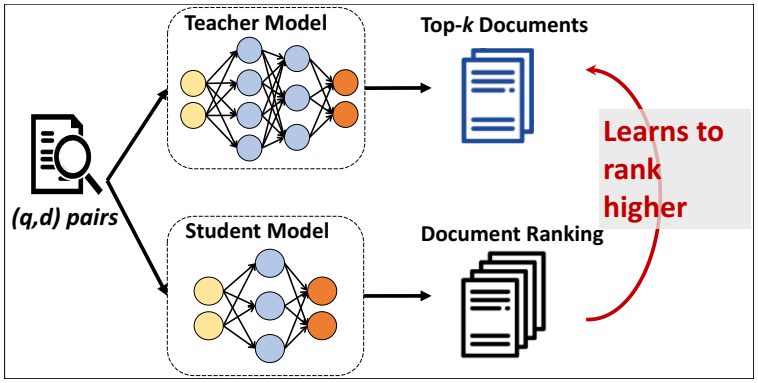
### 5.1.3 Knowledge Distillation

*Knowledge distillation* (KD), is a model-independent knowledge transfer strategy for improving model performance [40, 5, 3] while having its size fixed. Therefore, it can be used for generating a compact model for better inference efficiency while retaining the model effectiveness. The idea of KD is shown in Figure 5.2a for image recognition. During the offline training phase, a larger *teacher model* is first trained from the training set, and a smaller *student model* is then trained by minimizing two deviations: the deviation from the training set's ground-truth label distribution, and the deviation from the label distribution generated by the teacher model. Then the student model is used for making online inferences. Intuitively, the larger teacher model helps capture more information of the label distribution (for example, outputting a high probability for "tiger" images for a "cat" image query due to correlation), which is used as an additional supervision to the training of the student model. The student model trained with KD has an effectiveness comparable to that of the teacher model [40, 54, 3] and can make more efficient online inference due to its small model size.

Despite of this breakthrough in image recognition, it is not straightforward to apply KD to ranking models and ranking problems (e.g., recommendation). First, the existing KD is designed for classification problems, not for ranking problems. In ranking problems, the focus is on predicting the relative order of documents or items, instead of predicting a label or class as in classification. Second, KD requires computing the label distribution of documents for each query using both teacher and student models, which is feasible for image classification where there are a small number of labels, for example, no more than 1000 for the ImageNet data set; however, for ranking and recommendation problems, the total number of documents or items could be several orders of magnitudes larger, say millions, and computing the distribution for all documents or items for each training instance makes little sense, especially only the highly ranked documents or items near the top of the ranking will matter. We also want to point out that, for context sensitive recommendation, such as sequential recommendation, the items to be recommended usually depend on the user




(a) Knowledge Distillation: given an input, student model learns to minimize the KL Divergence of its label distribution and teacher model's.




(b) Ranking Distillation: given an query (or user in the context of recommendation), student model learns to give higher rank for it's teacher model's top-$K$ ranking of documents (or items in the context of recommendation).

Figure 5.2: The relationship between (a) Knowledge distillation and (b) Ranking distillation.

behaviors prior to the recommendation point (e.g., what she has viewed or purchased), and the set of contexts of a user is only known at the recommendation time. This feature requires recommender system to be strictly responsive and makes online inference efficiency particularly important.

### 5.1.4 Contributions

In this work, we study knowledge distillation for the learning to rank problem that is the core in recommender systems and many other IR systems. Our objective is achieving the ranking performance of a large model with the online inference efficiency of a small model. In particular, by fusing the idea of knowledge transfer and learning to rank, we propose a technique called *ranking distillation* (RD) to learn a compact ranking model that remains effective. The idea is shown in Figure 5.2b where a small student model is trained to learn to rank from two sources of information, i.e., the training set and the top-$K$ documents for each

query generated by a large well-trained teacher ranking model. With the large model size, the teacher model captures more ranking patterns from the training set and provides top-$K$ ranked unlabeled documents as an extra training data for the student model. This makes RD differ from KD, as teacher model in KD only generate additional labels on existing data, while RD generate additional training data and labels from unlabeled data set. The student model benefits from the extra training data generated from the teacher, in addition to the data from usual training set, thus, inherits the strong ranking performance of the teacher, but is more efficient for online inferences thanks to its small model size.

We will examine several key issues of RD, i.e., the problem formulation, the representation of teacher's supervision, and the balance between the trust on the data from the training set and the data generated by the teacher model, and present our solutions. Extensive experiments on recommendation problems and real-world datasets show that the student model achieves a similar or better ranking performance compared to the teacher model while using less than half of model parameters. While the design goal of RD is retaining the teacher's effectiveness (while achieving the student's online inference efficiency), RD exceeds this expectation that the student sometime has a even better ranking performance than the teacher. Similar to KD, RD is orthogonal to the choices of student and teacher models by treating them as black-boxes. To our knowledge, this is the first attempt to adopt the idea of knowledge distillation to large-scale ranking problems.

## 5.2 Related Work

In this section, we compared our works with several related research areas.

**Knowledge Distillation.** Knowledge distillation has been used in image recognition [40, 5, 79] and neural machine translation [54] as a way to generate compact models. As pointed out in Introduction, it is not straightforward to apply KD to ranking models and new issues must be addressed. In the context of ranking problems, the most relevant work is [18], which uses knowledge distillation for image retrieval. This method applies the sampling technique to rank a sample of the image from all data each time. In general, training on a sample works if the sample shares similar patterns with the rest of data through some content information, such as image contents in the case of [18]. But this technique is not applicable to training a recommender model when items and users are represented by IDs with no content information, as in the case of collaborative filtering. In this case, the recommender model training cannot be easily generalize to all users and items.

**Semi-Supervised Learning.** Another related research area is semi-supervised learning [114, 14]. Unlike the teacher-student model learning paradigm in knowledge distillation and in our work, semi-supervised learning usually trains a single model and utilizes weak-

labeled or unlabeled data as well as the labeled data to gain a better performance. Several works in information retrieval followed this direction, using weak-labeled or unlabeled data to construct test collections [4], to provide extra features [27] and labels [25] for ranking model training. The basic idea of ranking distillation and semi-supervised learning is similar as they both utilize unlabeled data while with different purpose.

**Transfer Learning for Recommender System.** Transfer learning has been widely used in the field of recommender systems [16, 29]. These methods mainly focus on how to transfer knowledge (e.g., user rating patterns) from a source domain (e.g., movies) to a target domain (e.g., musics) for improving the recommendation performance. If we consider the student as a target model and the teacher as a source model, our teacher-student learning can be seen as a special transfer learning. However, unlike transfer learning, our teacher-student learning does not require two domains because the teacher and student models are learned from the same domain. Having a compact student model to enhance online inference efficiency is another purpose of our teacher-student learning.

## 5.3 Proposed Methodology

In this section, we propose *ranking distillation* (RD) to address the dual goals of effectiveness and efficiency for ranking problems. To address the efficiency of online inference, we use a smaller ranking model so that we can rank documents for a given query more efficiently. To address the effectiveness issue without requiring more training data, we introduce extra information generated from a well-trained teacher model and make the student model as effective as the teacher.

### 5.3.1 Overview

Figure 5.3 shows the overview of ranking distillation. In the offline training phase (prior to any user query), similar to KD, first we train a large teacher model with a strong ranking performance on the training set. Then for each query, we use the well-trained teacher model to make predictions on unlabeled documents (green part in Figure 5.3) and use this extra information for learning the smaller student model. Since the teacher model is allowed to have many parameters, it captures more complex features for ranking and is much powerful, thus, its predictions on unlabeled documents could be used to provide extra information for the student model's training. The student model with fewer parameters is more efficient for online inference, and because of the extra information provided by the teacher model, the student model inherits the high ranking performance of the teacher model.

Specifically, the offline training for student model with ranking distillation consists of two steps. First, we train a larger teacher model $M_T$ by minimizing a ranking-based loss with the ground-truth ranking from the training data set, as showed in Eqn (5.1). With

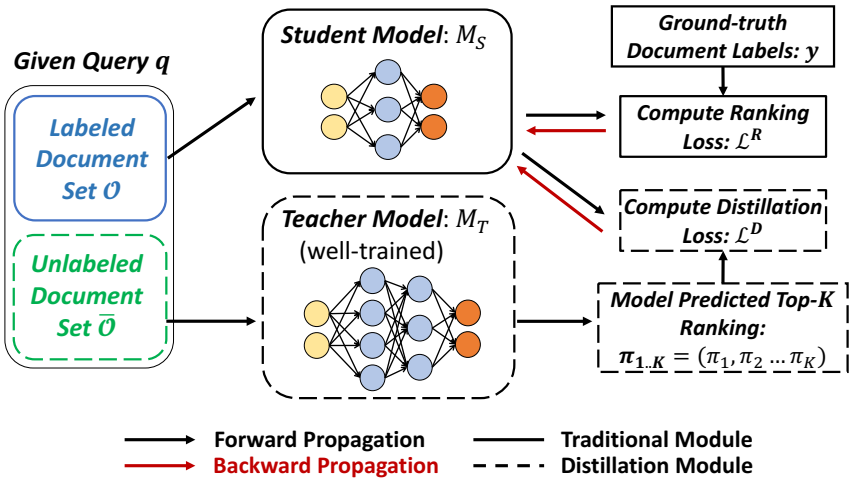



Figure 5.3: The learning paradigm with ranking distillation. We first train a teacher model and let it predict a top-$K$ ranked list of unlabeled (unobserved) documents for a given query $q$. The student model is then supervised by both ground-truth ranking from the training data set and teacher model's top-$K$ ranking on unlabeled documents.

much more parameters in this model, it captures more patterns from data and thus has a strong performance. We compute the predicted relevance scores of the teacher model $M_T$ for unlabeled documents $\bar{\mathcal{O}} = \{d : y_d = \emptyset\}$ and get a top-$K$ unlabeled document ranking $\boldsymbol{\pi}_{1..K} = (\pi_1, \ldots, \pi_K)$, where $\pi_r \in \mathcal{D}$ is the $r$-th document in this ranking. Then, we train a smaller ranking model $M_S$ to minimize a *ranking loss* from the ground-truth ranking in the training data set, as well as a *distillation loss* with the exemplary top-$K$ ranking on unlabeled document set $\boldsymbol{\pi}_{1..k}$ offered by its teacher $M_T$. The overall loss to be minimized is as follows:

$$\mathcal{L}(\theta_S) = (1-\alpha)\mathcal{L}^R(\boldsymbol{y}, \hat{\boldsymbol{y}}) + \alpha\mathcal{L}^D(\boldsymbol{\pi}_{1..K}, \hat{\boldsymbol{y}}). \tag{5.4}$$

Here $\hat{\boldsymbol{y}}$ is the student model's predicted scores[1]. $\mathcal{L}^R(,)$ stands for the ranking-based objective as in Eqn (5.1). The distillation loss, denoted by $\mathcal{L}^D(,)$, uses teacher model's top-$K$ ranking on unlabeled documents to guide the student model learning. $\alpha$ is the hyperparameter used for balancing these two losses.

For a given query, the top documents ranked by the well-trained teacher can be regarded to have a strong correlation to this query, although they are not labeled in the training set. For example, if a user watches many action movies, the teacher's top-ranked documents may contain some other action movies as well as some adventure movies because they are correlated. In this sense, the proposed RD lets the teacher model teach its student to find the correlations and capture their patterns, thus, makes the student more generalizable and

[1]When using point-wise and pair-wise losses, we only need to compute the student's predictions for a subset of documents, instead of all documents, for a given query.

perform well on unseen data in the future. We use the top-$K$ ranking from the teacher instead of the whole ranked list because the noisy ranking at lower positions tends to cause the student model to overfit its teacher and lose generalizability. Besides, only top positions are considered important for ranking problems. $K$ is a hyperparameter that represents the trust level on the teacher during teaching, i.e., how much we adopt from teacher.

The choice of the ranking loss $\mathcal{L}^R(,)$ follows from different models' preferences and we only focus on the second term $\mathcal{L}^D(,)$ in Eqn (5.4). A question is how much we should trust teacher's top-$K$ ranking, especially for a larger $K$. In the rest of the section, we consider this issue.

### 5.3.2 Incorporating Distillation Loss

We consider the point-wise ranking loss for binary relevance labels for performing distillation, we also tried the pair-wise loss and will discuss their pros and cons later. Similar to Eqn (5.2), we formalize distillation loss as:

$$\begin{aligned}\mathcal{L}^D(\boldsymbol{\pi}_{1..K}, \hat{\boldsymbol{y}}) &= -\sum_{r=1}^{K} w_r \cdot \log(P(rel = 1|\hat{y}_{\pi_r})) \\ &= -\sum_{r=1}^{K} w_r \cdot \log(\sigma(\hat{y}_{\pi_r})),\end{aligned} \tag{5.5}$$

where $\sigma(\cdot)$ is the *sigmoid* function and $w_r$ is the weight to be discussed later. There are several differences compared to Eqn (5.2). First, in Eqn (5.5), we treat the top-$K$ ranked documents from the teacher model as positive instances and there is no negative instance. Recall that KD causes the student model to output a higher probability for the label "tiger" when the ground-truth label is "cat" because their features captured by the teacher model are correlated. Along this line, we want the student model to rank higher for teacher's top-$K$ ranked documents. As we mentioned above, for the given query, besides the ground-truth positive documents $\boldsymbol{y}^+$, teacher's top-$K$ ranked unlabeled documents are also strongly correlated to this query. These correlations are captured by the well-trained powerful teacher model in the latent space when using latent factor model or neural networks.

However, as $K$ increases, the relevance of the top-$K$ ranked unlabeld documents becomes weaker. Following the work of learning from noise labels [69], we use a weighted sum over the loss on documents from $\boldsymbol{\pi}_{1..K}$ with weight $w_r$ on each position $r$ from 1 to $K$. There are two straightforward choices for $w_r$: $w_r = 1/r$ puts more emphasis on the top positions, whereas $w_r = 1/K$ weights each position equally. Such weightings are heuristic and pre-determined, may not be flexible enough to deal with general cases. Instead, we introduce two flexible weighting schemes, which were shown to be superior in our experimental studies.

**Algorithm 1** Estimate Student's Ranking for $\pi_r$

**Require:** Student Model $M_S(q, d; \theta_S)$, unlabeled document set $\bar{\mathcal{O}}$ for a given query $q$ and the hyperparameter $\epsilon$
  $\hat{y}_{\pi_r} \leftarrow M_S(q, \pi_r; \theta_S)$
  Initialize $n = 0$
  **for** $t = 1, 2, ...\epsilon$ **do**
    Sample a document $d$ from $\bar{\mathcal{O}}$ without replacement
    $\hat{y}_d \leftarrow M_S(q, d; \theta_S)$
    **if** $\hat{y}_d > \hat{y}_{\pi_r}$ **then**
      $n \leftarrow n + 1$
    **end if**
  **end for**
  $\hat{r}_{\pi_r} \leftarrow \lfloor \frac{n \times (|\bar{\mathcal{O}}| - 1)}{\epsilon} \rfloor + 1$
  **return** $\hat{r}_{\pi_r}$

### Weighting by Position Importance

In this weighting scheme, we assume that the teacher predicted unlabeled documents at top positions are more correlated to the query and are more likely to the positive ground-truth documents, therefore, this weight $w^a$ should be inversely proportional to the rank:

$$w_r^a \propto r^{-1} \quad \text{and} \quad r \in [1, K], \tag{5.6}$$

where $r$ is the rank range from 1 to $K$. As pointed out above, this scheme pre-determines the weight. Rendle *et al* [76] proposed an empirical weight for sampling a single position from a top-$K$ ranking, following a geometric distribution:

$$w_r^a = \rho(1 - \rho)^r \quad \text{and} \quad \rho \in (0, 1). \tag{5.7}$$

Following their work, we use a parametrized geometric distribution for weighting the position importance:

$$w_r^a \propto e^{-r/\lambda} \quad \text{and} \quad \lambda \in \mathbb{R}^+, \tag{5.8}$$

where $\lambda$ is the hyperparameter that controls the sharpness of the distribution, and is searched through the validation set. When $\lambda$ is small, this scheme puts more emphasis on top positions, and when $\lambda$ is large enough, the distribution becomes the uniform distribution. This parametrization is easy to implement and configurable to each kind of situation.

### Weighting by Ranking Discrepancy

The weighting by position importance is static, meaning that the weight at the same position is fixed during training process. Our second scheme is dynamic that considers the discrepancy between the student-predicted rank and the teacher-predicted rank for a given

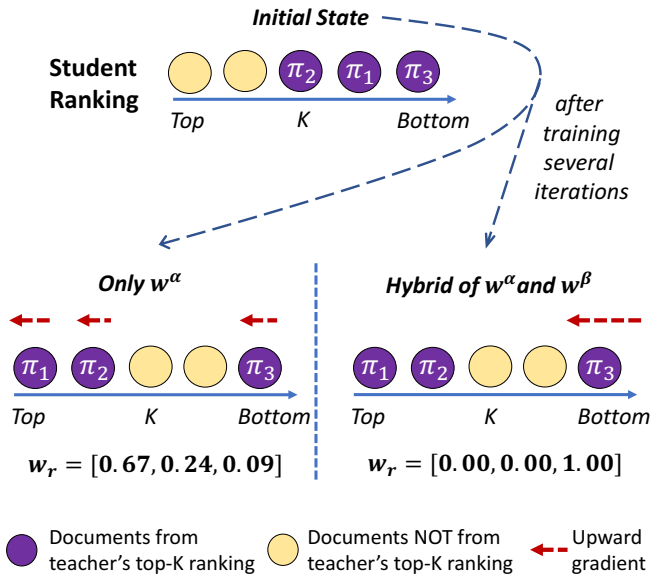



Figure 5.4: An illustration of hybrid weighting scheme. We use $K = 3$ in this example.

unlabeled document, and uses it as another weight $w^b$. This weighting scheme allows the training to gradually concentrate on the documents in teacher's top-$K$ ranking that are not well-predicted by the student. The details are as follows.

For the $r$-th document $\pi_r$ ($r \in [1, K]$) in teacher model's top-$K$ ranking, the teacher-predicted ranking (i.e., $r$) is known for us. But we know only the student predicted relevant score $\hat{y}_{\pi_r}$ instead of its rank without computing relevance scores for all documents. To get the student predicted rank for this document, we apply Weston *et al* [99]'s sequential sampling, and do it in a parallel manner [42]. As described in Algorithm 1, for the $r$-th document $\pi_r$, if we want to know its rank in a list of $N$ documents without computing the scores for all documents, we can randomly sample $\epsilon \in [1, N-1]$ documents in this list and estimate the relative rank by $n/\epsilon$, where $n$ is the number of documents whose (student) scores are greater than $\hat{y}_{\pi_r}$. Then the estimated rank in the whole list is $\hat{r}_{\pi_r} = \lfloor \frac{n \times (N-1)}{\epsilon} \rfloor + 1$. When $\epsilon$ goes larger, the estimated rank is more close to the actual rank.

After getting the estimated student's rank $\hat{r}_{\pi_r}$ for the $r$-th document $\pi_r$ in teacher's top-$K$ ranking, the discrepancy between $r$ and $\hat{r}$ is computed by

$$w_r^b = tanh(\max(\mu \cdot (\hat{r}_{\pi_r} - r), 0)), \tag{5.9}$$

where $tanh(\cdot)$ is a rescaled *logistic* function $tanh(x) = 2\sigma(2x) - 1$ that rescale the output range to $[0, 1]$ when $x > 0$. The hyperparameter $\mu \in \mathbb{R}^+$ is used to control the sharpness of the *tanh* function. Eqn (5.9) gives a dynamic weight: when the student predicted-rank of a document is close to its teacher, we think this document has been well-predicted and impose little loss on it (i.e., $w_r^b \approx 0$); the rest concentrates on the documents (i.e., $w_r^b \approx 1$) whose

student predicted-rank is far from the teacher's rank. Note that the ranking discrepancy weight $w^b$ is computed for each document in $\pi_{1..K}$ during training. So in practice, we choose $\epsilon \ll |\hat{\mathcal{O}}|$ for training efficiency. While extra computation used to compute relevance scores for sampled $\epsilon$ documents, we still boost the whole offline training process. Because the dynamic weight allows the training to focus on the erroneous parts in the distillation loss.

**Hybrid Weighting Scheme**

The hybrid weighting combines the weight $w^a$ by position importance, and the weight $w^b$ by ranking discrepancy: $w_r = (w^a_r \cdot w^b_r)/(\sum_{i=1}^{K} w^a_i \cdot w^b_i)$. Figure 5.4 illustrates the advantages of hybrid weighting over weighting only by position importance. Our experiments show that this hybrid weighting gives better results in general. In the actual implementation, since the estimated student ranking of $\hat{r}_{\pi_r}$ is not accurate during the first few iterations, we use only $w^a$ during the first $m$ iterations to warm up the model, and then use the hybrid weighting to make training focus on the erroneous parts in distillation loss. $m$ should be determined via the validation set. In our experiments, $m$ is usually set to more than half of the total training iterations.

### 5.3.3 Discussion

Under the paradigm of ranking distillation, for a certain query $q$, besides the labeled documents, we use a top-$K$ ranking for unlabeled documents generated by a well-trained teacher ranking model $M_T$ as extra information to guide the training of the student ranking model $M_S$ with less parameters. During the student model training, we use a weighted point-wise ranking loss as the distillation loss and propose two types of flexible weighting schemes, i.e., $w^a$ and $w^b$ and propose an effective way to fusion them. For the hyperparameters $(\alpha, \lambda, \mu, \epsilon, K, m)$, they are dataset-dependent and are determined for each data set through the validation set. Two key factors for the success of ranking distillation are: (1) larger models are capable to capture the complex interaction patterns between queries and documents, thus, their predicted unlabeled documents at top positions are also strongly correlated with the given query and (2) student models with less parameters can learn from the extra-provided helpful information (top-$K$ unlabeled documents in teacher's ranking) and boost their performances.

We also tried to use a pair-wise distillation loss when learning from teacher's top-$K$ ranking. Specifically, we use Eqn (5.3) for the distillation loss by taking the partial order in teacher's top-$K$ ranking as objective. However, the results were disappointing. We found that if we use pair-wise distillation loss to place much focus on the partial order within teacher's ranking, it will produce both upward and downward gradients, making the training unstable and sometimes even fail to converge. However, our weighted point-wise distillation loss that only contains upward gradients doesn't suffer from this issue.

Table 5.1: Statistics of the data sets

| Datasets | #users | #items | avg. actions per user | $(u, \mathcal{S}^{(u,t)})$ pairs | Sparsity |
|---|---|---|---|---|---|
| Gowalla | 13.1k | 14.0k | 40.74 | 367.6k | 99.71% |
| Foursquare | 10.1k | 23.4k | 30.16 | 198.9k | 99.87% |

## 5.4 Experimental Studies

We evaluate the performance of *ranking distillation* on two real-world data sets. The source code and processed data sets are publicly available online[2].

### 5.4.1 Experimental Setup

**Task description.** We use recommendation as our task for evaluating the performance of ranking distillation. In this problem, we have a set of users $\mathcal{U} = \{u_1, u_2, \cdots, u_{|\mathcal{U}|}\}$ and a universe of items $\mathcal{I} = \{i_1, i_2, \cdots, i_{|\mathcal{I}|}\}$. For recommendation without context information, we can cache the recommendation list for each user [3]. However, for context-aware recommendation, we have to re-compute the recommendation list each time a user comes with a new context, so the online inference efficiency becomes important. The following *sequential recommendation* is one case of context-aware recommendation. Given a users $u$ with her/his history sequence (i.e., past $L$ interacted items) at time $t$, $\mathcal{S}^{(u,t)} = (\mathcal{S}^u_{t-1}, .., \mathcal{S}^u_{t-L})$, where $\mathcal{S}^u_i \in \mathcal{I}$, the goal is to retrieve a list of items for this user that meets her/his future needs. In IR's terms, the query is the user profile $(u, \mathcal{S}^{(u,t)})$ at time $t$, and the document is the item. Note that whenever the user has a new behavior (e.g., watch a video/listen to a music), we have to re-compute the recommendation list as her/his context changes. We also wish to point out that, in general, ranking distillation can be applied to other learning to rank tasks, not just to recommendation.

**Datasets.** We choose two real-world data sets in this work, as they contain numerous sequential signals and thus suitable for sequential recommendation. Their statistics are described in Table 5.1. Gowalla[4] was constructed by [20] and Foursquare was obtained from [107]. These data sets contain sequences of implicit feedbacks through user-venue check-ins. During the offline training phase, for a user $u$, we extract every 5 successive items ($L = 5$) from her sequence as $\mathcal{S}^{(u,t)}$, and the immediately next item as the ground-truth. Following

[2]https://github.com/graytowne/rank_distill

[3]We suppose the recommendation model doesn't change immediately whenever new observed data come, which is common in real-world cases.

[4]https://snap.stanford.edu/data/loc-gowalla.html

[107], we hold the first 70% of actions in each user's sequence as the *training set* and use the next 10% of actions as the *validation set* to search the optimal hyperparameter settings for all models. The remaining 20% actions in each user's sequence are used as the *test set* for evaluating a model's performance.

**Evaluation protocol.** As in [72, 103, 25, 71], three different evaluation metrics used are Precision@$n$ (Prec@$n$), nDCG@$n$, and Mean Average Precision (MAP). We set $n \in \{3, 5, 10\}$, as recommendations are top positions of rank lists are more important. To measure the online inference efficiency, we count the number of parameters in each model and report the wall time for making a recommendation list to every user based on her/his last 5 actions in the training data set. While training models efficiently is also important, training is done offline before the recommendation phase starts, and our focus is the online inference efficiency where the user is waiting for the responses from the system.

**Teacher/Student models.** We apply the proposed ranking distillation to two sequential recommendation models that have been shown to have strong performances:

- **Fossil**. Factorized Sequential Prediction with Item Similarity ModeLs (Fossil) [37] models sequential patterns and user preferences by fusing a similarity model with latent factor model. It uses a pair-wise ranking loss.
- **Caser**. ConvolutionAl Sequence Embedding Recommendation model (Caser in Chapter 3) incorporates the Convolutional Neural Network and latent factor model to learn sequential patterns as well as user preferences. It uses a point-wise ranking loss.

To apply ranking distillation, we adopt as many parameters as possible for the teacher model to achieve a good performance on each data set. These well-trained teacher models are denoted by **Fossil-T** and **Caser-T**. We then use these models to teach smaller student models denoted by **Fossil-RD** and **Caser-RD** by minimizing the ranking distillation loss in Eqn (5.4). The model sizes of the student models are gradually increased until the models reach a comparable performance to their teachers. **Fossil-S** and **Caser-S** denote the student models trained with only ranking loss, i.e., without the help from the teacher. Note that the increasing in model sizes is achieved by using larger dimensions for embeddings, without any changes to the model structure.

### 5.4.2 Overall Performances

The results of each method are summarized in Table 5.2. We also included three non-sequential recommendation baselines: the popularity (in all users' sequences) based item recommendation (**POP**), the item based Collaborative Filtering[5] (**ItemCF**) [82], and the

[5]We use Jaccard similarity measure and set the number of nearest neighbor to 20.

Table 5.2: Performance comparison. (1) The performance of the models with ranking distillation, Fossil-RD and Caser-RD, always has statistically significant improvements over the student-only models, Fossil-S and Caser-S. (2) The performance of the models with ranking distillation, Fossil-RD and Caser-RD, has no significant degradation from that of the teacher models, Fossil-T and Caser-T. We use the one-tail t-test with significance level at 0.05.

**Gowalla**

| **Model** | **Prec@3** | **Prec@5** | **Prec@10** | **nDCG@3** | **nDCG@5** | **nDCG@10** | **MAP** |
|---|---|---|---|---|---|---|---|
| *Fossil-T* | 0.1299 | 0.1062 | 0.0791 | 0.1429 | 0.1270 | 0.1140 | 0.0866 |
| *Fossil-RD* | 0.1355 | 0.1096 | 0.0808 | 0.1490 | 0.1314 | 0.1172 | 0.0874 |
| *Fossil-S* | 0.1217 | 0.0995 | 0.0739 | 0.1335 | 0.1185 | 0.1060 | 0.0792 |
| *Caser-T* | 0.1408 | 0.1149 | 0.0856 | 0.1546 | 0.1376 | 0.1251 | 0.0958 |
| *Caser-RD* | 0.1458 | 0.1183 | 0.0878 | 0.1603 | 0.1423 | 0.1283 | 0.0969 |
| *Caser-S* | 0.1333 | 0.1094 | 0.0818 | 0.1456 | 0.1304 | 0.1188 | 0.0919 |
| *POP* | 0.0341 | 0.0362 | 0.0281 | 0.0517 | 0.0386 | 0.0344 | 0.0229 |
| *ItemCF* | 0.0686 | 0.0610 | 0.0503 | 0.0717 | 0.0675 | 0.0640 | 0.0622 |
| *BPR* | 0.1204 | 0.0983 | 0.0726 | 0.1301 | 0.1155 | 0.1037 | 0.0767 |

**Foursquare**

| **Model** | **Prec@3** | **Prec@5** | **Prec@10** | **nDCG@3** | **nDCG@5** | **nDCG@10** | **MAP** |
|---|---|---|---|---|---|---|---|
| *Fossil-T* | 0.0859 | 0.0630 | 0.0420 | 0.1182 | 0.1085 | 0.1011 | 0.0891 |
| *Fossil-RD* | 0.0877 | 0.0648 | 0.0430 | 0.1203 | 0.1102 | 0.1023 | 0.0901 |
| *Fossil-S* | 0.0766 | 0.0556 | 0.0355 | 0.1079 | 0.0985 | 0.0911 | 0.0780 |
| *Caser-T* | 0.0860 | 0.0650 | 0.0438 | 0.1182 | 0.1105 | 0.1041 | 0.0941 |
| *Caser-RD* | 0.0923 | 0.0671 | 0.0444 | 0.1261 | 0.1155 | 0.1076 | 0.0952 |
| *Caser-S* | 0.0830 | 0.0621 | 0.0413 | 0.1134 | 0.1051 | 0.0986 | 0.0874 |
| *POP* | 0.0702 | 0.0477 | 0.0304 | 0.0845 | 0.0760 | 0.0706 | 0.0636 |
| *ItemCF* | 0.0248 | 0.0221 | 0.0187 | 0.0282 | 0.0270 | 0.0260 | 0.0304 |
| *BPR* | 0.0744 | 0.0543 | 0.0348 | 0.0949 | 0.0871 | 0.0807 | 0.0719 |

Bayesian personalized ranking (**BPR**) [77]. Clearly, the performance of these non-sequential baselines is worse than that of the sequential recommenders, i.e., Fossil and Caser.

The teacher models, i.e., Fossil-T and Caser-T, have a better performance than the student-only models, i.e., Fossil-S and Caser-S, indicating that a larger model size provides more flexibility to fit the complex data with more predictive power. The effectiveness of ranking distillation is manifested by the significantly better performance of Fossil-RD and Caser-RD compared to Fossil-S and Caser-S, and by the similar performance of Fossil-RD and Caser-RD compared to Fossil-T and Caser-T. In other words, thanks to the knowledge transfer of ranking distillation, we are able to learn a student model that has fewer parameters but similar performance as the teacher model. Surprisingly, student models with

Table 5.3: Model compactness and online inference efficiency. Time (seconds) indicates the wall time used for generating a recommendation list for every user. Ratio is the student model's parameter size relative to the teacher model's parameter size.

| **Datasets** | **Model** | **Time (CPU)** | **Time (GPU)** | **#Params** | **Ratio** |
|---|---|---|---|---|---|
| Gowalla | *Fossil-T* | 9.32s | 3.72s | 1.48M | 100% |
| | *Fossil-RD* | 4.99s | 2.11s | 0.64M | 43.2% |
| | *Caser-T* | 38.58s | 4.52s | 5.58M | 100% |
| | *Caser-RD* | 18.63s | 2.99s | 2.79M | 50.0% |
| Foursquare | *Fossil-T* | 6.35s | 2.47s | 1.01M | 100% |
| | *Fossil-RD* | 3.86s | 2.01s | 0.54M | 53.5% |
| | *Caser-T* | 23.89s | 2.95s | 4.06M | 100% |
| | *Caser-RD* | 11.65s | 1.96s | 1.64M | 40.4% |

ranking distillation often have even better performance than their teachers. This finding is consistent with [54] and we will explain possible reasons in Section 5.4.3.

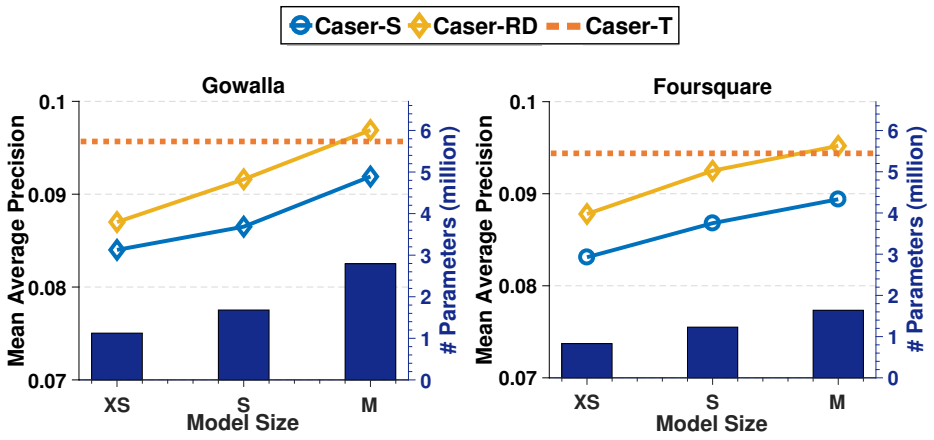
The online inference efficiency is measured by the model size ( number of model parameters) and is shown in Table 5.3. Note that Fossil-S and Caser-S have the same model size as Fossil-RD and Caser-RD. All inferences were implemented using PyTorch with CUDA from GTX1070 GPU and Intel i7-6700K CPU. Fossil-RD and Caser-RD nearly half down the model size compared to their teacher models, Fossil-T and Caser-T. This reduction in model size is translated into a similar reduction in online inference time. In many practical applications, the data set is much larger than the data sets considered here in terms of the numbers of users and items; for example, Youtube could have 30 million active users per day and 1.3 billion of items[6]. For such large data sets, online inference could be more time-consuming and the reduction in model size has more privileges. Also, for models that are much more complicated than Fossil and Caser, the reduction in model size could yield a larger reduction in online inference time than reported here.

In conclusion, the findings in Table 5.2 and 5.3 together confirm that ranking distillation helps generate compact models with no or little compromise on effectiveness, and these advantages are independent of the choices of models.

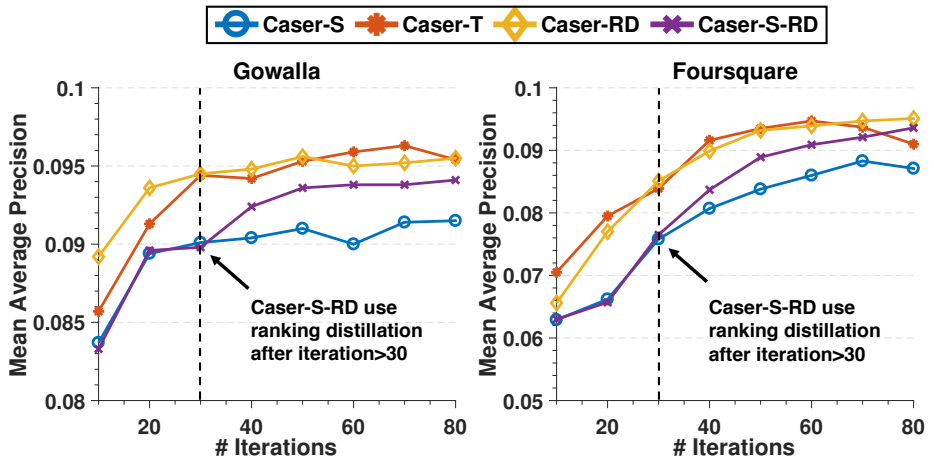
### 5.4.3 Effects of Model Size and Distillation Loss

In this experiment, we study the impact of model size on the student model's performance (i.e., MAP). We consider only Caser because the results for Fossil are similar. Figure 5.5a shows the results. Caser-S and Caser-RD perform better when the model size goes

[6]https://fortunelords.com/youtube-statistics




(a) MAP vs. model size



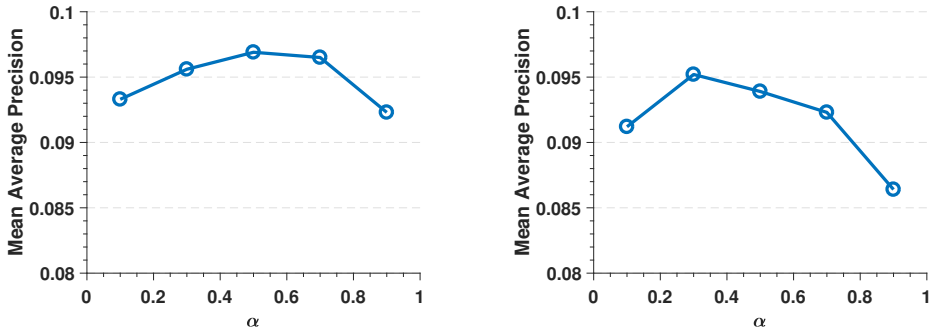

(b) MAP vs. the number of iterations for model training

Figure 5.5: Mean average precision vs. (a) model size and (b) the choice of distillation loss.

up, but there is always a gap. Caser-RD reaches a similar performance to its teacher with the medium model size, which is about 50% of the teacher model size.

Figure 5.5b shows the impact of ranking distillation on the student model's MAP iteration by iteration. We compare four models: Case-S, Caser-T, Caser-RD, and Caser-S-RD. The last model minimizes ranking loss during the first 30 iterations and adds distillation loss after that. Caser-RD outperforms Caser-S all the time. Caser-S reaches its limit after 50 iterations on Gowalla and 70 iterations on Foursquare. Caser-S-RD performs similarly to Caser-S during the first 30 iterations, but catches up with the Caser-RD at the end, indicating the impressive effectiveness of ranking distillation. Caser-T performs well at first but tends to get overfitted after about 60 iterations due to its large model size and the sparse recommendation data sets. In contrast, Caser-RD and Caser-S-RD, which have smaller model sizes, are more robust to overfitting issue, though their training is partially supervised by the teacher. This finding reveals another advantage of ranking distillation.

Figure 5.6 shows the MAP for various balancing parameter $\alpha$ to explore models' performance when balancing ranking loss and distillation loss. For Gowalla data, the best performance is achieved when $\alpha$ is around 0.5. But for Foursquare data, the best performance is achieved when $\alpha$ is around 0.3, indicating too much concentrate on distillation loss leads to a bad performance. On both data sets, either discarding ranking loss or discarding distillation loss gives poor results.




(a) Gowalla (b) Foursquare

Figure 5.6: MAP vs. balancing parameter $\alpha$

### 5.4.4 Effects of Weighting Schemes

Table 5.4 shows the effects of the proposed weighting schemes in our ranking distillation. For the weight $w_r$ for $r$-th document in the teacher's top-$K$ ranking, we used the equal weight ($w_r = 1/K$) as the baseline and considered the weighting by position importance ($w_r = w_r^a$), the weighting by ranking discrepancy ($w_r = w_r^b$), and the hybrid weighting ($w_r \propto w_r^a \cdot w_r^b$). The equal weight performs the worst. The position importance weighting is much better, suggesting that within the teacher's top-$K$ ranking, documents at top positions are more related to the positive ground truth. The ranking discrepancy weighting only doesn't give impressive results, but when used with the position importance weighting, the hybrid weighting yields the best results on both data sets.

## 5.5 Conclusion

The proposed ranking distillation enables generating compact ranking models for better online inference efficiency without scarifying the ranking performance. The idea is training a teacher model with more parameters to teach a student model with fewer parameters to rank unlabeled documents. While the student model is compact, its training benefits from the extra supervision of the teacher, in addition to the usual ground truth from the training data, making the student model comparable with the teacher model in the ranking performance. This paper focused on several key issues of ranking distillation, i.e., the problem formulation,

Table 5.4: Performance of Caser-RD with different choices of weighting scheme on two data sets.

| **Datasets** | **Weighting** | **P@10** | **nDCG@10** | **MAP** |
|---|---|---|---|---|
| Gowalla | $w_r = 1/K$ | 0.0843 | 0.1198 | 0.0925 |
| | $w_r = w_r^a$ | 0.0850 | 0.1230 | 0.0945 |
| | $w_r = w_r^b$ | 0.0851 | 0.1227 | 0.0937 |
| | hybrid | **0.0878** | **0.1283** | **0.0969** |
| Foursquare | $w_r = 1/K$ | 0.0424 | 0.1046 | 0.0914 |
| | $w_r = w_r^a$ | 0.0423 | 0.1052 | 0.0929 |
| | $w_r = w_r^b$ | 0.0429 | 0.1035 | 0.0912 |
| | hybrid | **0.0444** | **0.1076** | **0.0952** |

the representation of teacher's supervision, and the balance between the trust on the training data and the trust on the teacher, and presented our solutions. The evaluation on real data sets supported our claims.

# Chapter 6

# Conclusion

## 6.1 Summary

Leveraging user action sequences helps us better understand users' need in the near future. Therefore, sequential recommendation takes a big step towards building a more precise recommender system. Deep neural networks could be promising tools for solving sequential recommendation problem. Our research in this thesis targets several unique challenges when modeling user action sequences with neural networks and proposes our solutions. In particular, we make the following contributions to better utilize dependencies (patterns) in user action sequences and make the resulting recommender efficient to use:

- In Chapter 3, we confirmed the existence of multi-form sequential patterns and investigated how they can influence user's future actions. We proposed Caser, a unified model that uses convolutional filters with different shapes to capture sequential patterns in different forms. Caser is flexible as it can generalize several existing methods and it is one of the earliest attempts for using deep neural network to deal with sequential recommendation problem.

- In Chapter 4, we examined the properties of sequential dependencies in YouTube dataset where we have long user action sequences (length greater than 100). Based on our findings, we proposed a multi-temporal-range mixture model, called M3, as a tailored solution. M3 uses a combination of several sequence models to cater the distinct needs from different parts of user sequence. Besides accuracy, it offers good interpretability and can adapt to various recommendation scenarios.

- In Chapter 5, we pointed out the online efficiency issue caused by cumbersome neural network models. The issue is extremely critical for sequential recommendation which needs to quickly respond user requests in real-time. To mitigate this, we proposed a ranking distillation (RD) method. RD is able to learn a recommendation model with fewer parameters and good online inference efficiency but with its performance

similar to a recommendation model with much more parameters. Though designed for recommendation, the idea of RD can also be generalized to other ranking problems.

## 6.2 Future Directions

The work described in this thesis focuses on making sequential recommender systems more precise and more affordable. For the same problem we studied, there are many other promising future ways to further improve recommendation precision and efficiency. To this end, I summarize some of the intuitions in below.

**Further improve recommendation precision by modeling seasonal patterns.** Although the sequential order is used in this dissertation for providing better recommendation, there are more opportunities of exploiting the time information. Among this, modeling seasonal patterns in user action sequences can be very useful but is overlooked by literature. Most existing works do not explicitly discuss or model the following important seasonal patterns. First of all, we have item-specific seasonal patterns. Such seasonality can be life-cycle of a certain item or some overall trends for a particular item. For instance, TV's life-cycle is much longer than coffee. That's why it is meaningless to recommend another TV after the user has purchased a TV. On the other hand, if we know coffee is usually purchased in the morning and usually consumed by people every day, we can use this information to provide better user experiences for repetitive consumption [12]. Besides, we may also have personalized seasonal patterns in dataset. For example, one may prefer to watch study videos during day time and entertainment videos at night. That's why the user may not prefer the recommended entertainment videos at day time. This doesn't mean our recommendation is bad but not satisfactory to user's need at that time. The personalized seasonal pattern reflects the difference of certain person's interest at different times.

**Further reduce parameters in recommendation models.** From model compression perspective, it is known that above 90% of parameters of any recommendation models are from user and item latent factors (embeddings). Therefore, to reduce model parameters effectively, we should focus on compressing user and item embeddings. There are some possible solutions along this line. Firstly, we could group similar users' (or items') embeddings by learning some representative embeddings. Vector quantization [35, 45] can be used to achieve this purpose. Also, there's a major limitation when using the same embedding dimensionality for each user embedding. Instead, using smaller dimensionality to embed cold-start users (users with few feedback) should be more reasonable, as these users didn't show many diverse interests. After these users left more feedback and more engaged into the system, we can gradually increase their embeddings' dimensionality. By using an adaptive

embedding size, we could not only save a lot of model parameters but also regularize the model in a proper way.

Beyond the problems discussed in this dissertation, there are also some meaningful directions that are critical to the whole recommender system. Below I state them into details.

**Investigating negative impacts of recommendation.** During the past decade, most works focused on developing more precise recommendation methods. However, recommender system can also cause potential negative impacts to the system provider and to the users, while they are certainly under-explored. To the system provider, due to the open nature of recommender systems and their reliance on user contributed judgments [13], recommender systems can be misused, attacked with malicious purposes [59]. Once this happens, the credibility of a recommender system will be largely affected, which could lead to a significant economic loss. To this end, some research directions deserve more explorations, including but not limited to: (1) Recommedation models' vulnerability, (2) models' reactions to different types of attacks and (3) proper defensive strategies . To the users, people's interactions with recommendation systems cannot precisely reflex their original intentions [104]. What even worse is the recommendations may change users' consumption from what they aspire to choose, since it has become the main user interface of some platforms (e.g., YouTube). This could cause the *filtering bubble* problem [30], where user interests are narrowed by the recommender system. Moreover, privacy of each user might be disclosed via recommender system. More studies should be done to protect the system's users. In all, we believe understanding the negative impacts of recommendation is crucial and is a promising direction that needs more research.

**Optimizing long-term user engagement metrics.** In recommendation research, our goal is to make recommendations that the user may have largest probabilities to interact with. However, user interaction is not equal to user satisfaction, which is always revealed by the long-term user engagement metrics (e.g., user's time spent on the recommended videos or the number of "likes" on the recommended videos). But this discrepancy also brings opportunities to recommendation research. On the one hand, user satisfaction is always measured by a combination of multiple user behaviors. Therefore, it requires more attention to investigate how to effectively performing multi-objective optimization. On the other hand, some of the engagement metrics are non-differentiable thus cannot be directly optimized via Gradient Descent methods. To solve this, reinforcement learning [15] might be a potential solution. Alternatively, we can devise some other surrogate metrics to align with the non-differentiable user engagement metrics [32].

**Recommendation for multi-stakeholders** One of the most essential aspects of any recommender system is personalization. That is, we develop methods to suit the user's interests. However, in many real-world applications, there are other stakeholders whose needs and interests should be taken into account. In multi-sided online platforms, such as YouTube, aside from content consumers (users) there are also content providers whose needs should be considered. Maximization of a combination of different objectives that simultaneously satisfy multi-stakeholders is a critical but challenging problem. Especially when there are conflicting goals from different parties, the recommender system will need to balance the interests of different parties. In this way, maybe game theory is a good solution.

# Bibliography


[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*, pages 265–283, 2016.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *International Conference on Data Engineering*, pages 3–14. IEEE, 1995.

[3] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormándi, George E. Dahl, and Geoffrey E. Hinton. Large scale distributed neural network training through online distillation. *arXiv preprint arXiv:1804.03235*, 2018.

[4] Nima Asadi, Donald Metzler, Tamer Elsayed, and Jimmy Lin. Pseudo test collections for learning web search ranking functions. In *International Conference on Research and development in Information Retrieval*, pages 1073–1082. ACM, 2011.

[5] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

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

[7] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[8] Francois Belletti, Evan Sparks, Alexandre Bayen, and Joseph Gonzalez. Random projection design for scalable implicit smoothing of randomly observed stochastic processes. In *Artificial Intelligence and Statistics*, pages 700–708, 2017.

[9] Francois Belletti, Alex Beutel, Sagar Jain, and Ed Chi. Factorized recurrent neural architectures for longer range dependence. In *International Conference on Artificial Intelligence and Statistics*, pages 1522–1530, 2018.

[10] Alex Beutel, Ed H Chi, Zhiyuan Cheng, Hubert Pham, and John Anderson. Beyond globally optimal: Focused learning for improved recommendations. In *International Conference on World Wide Web*, pages 203–212, 2017.

[11] 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 *International Conference on Web Search and Data Mining*, pages 46–54. ACM, 2018.

[12] Rahul Bhagat, Srevatsan Muralidharan, Alex Lobzhanidze, and Shankar Vishwanath. Buy it again: Modeling repeat purchase recommendations. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 62–70. ACM, 2018.

[13] Robin Burke, Michael P O'Mahony, and Neil J Hurley. Robust collaborative recommendation. In *Recommender systems handbook*, pages 961–995. Springer, 2015.

[14] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

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

[16] Xu Chen, Zheng Qin, Yongfeng Zhang, and Tao Xu. Learning to rank features for recommendation over multiple categories. In *International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 305–314, 2016.

[17] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. Sequential recommendation with user memory networks. In *International Conference on Web Search and Data Mining*, pages 108–116. ACM, 2018.

[18] Yuntao Chen, Naiyan Wang, and Zhaoxiang Zhang. Darkrank: Accelerating deep metric learning via cross sample similarities transfer. *arXiv preprint arXiv:1707.01220*, 2017.

[19] Chen Cheng, Haiqin Yang, Michael R Lyu, and Irwin King. Where you like to go next: Successive point-of-interest recommendation. In *International Joint Conference on Artificial Intelligence*, pages 2605–2611, 2013.

[20] Eunjoon Cho, Seth A Myers, and Jure Leskovec. Friendship and mobility: user movement in location-based social networks. In *International Conference on Knowledge Discovery and Data Mining*, pages 1082–1090. ACM, 2011.

[21] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[22] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pages 192–204, 2015.

[23] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *ACM Conference on Recommender Systems*, 2016.

[24] D Crankshaw and J Gonzalez. Prediction-serving systems. *Queue*, 16:83–97, 01 2018. doi: 10.1145/3194653.3210557.

[25] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W Bruce Croft. Neural ranking models with weak supervision. *arXiv preprint arXiv:1704.08803*, 2017.

[26] Robin Devooght and Hugues Bersini. Long and short-term recommendations with recurrent neural networks. In *Conference on User Modeling, Adaptation and Personalization*, pages 13–21. ACM, 2017.

[27] Fernando Diaz. Learning to rank with labeled features. In *International Conference on the Theory of Information Retrieval*, pages 41–44. ACM, 2016.

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

[29] Ignacio Fernández-Tobías, Iván Cantador, Marius Kaminskas, and Francesco Ricci. Cross-domain recommender systems: A survey of the state of the art. In *Spanish Conference on Information Retrieval*, page 24. sn, 2012.

[30] Seth Flaxman, Sharad Goel, and Justin M Rao. Filter bubbles, echo chambers, and online news consumption. *Public opinion quarterly*, 80(S1):298–320, 2016.

[31] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*, 2017.

[32] Alexandre Gilotte, Clément Calauzènes, Thomas Nedelec, Alexandre Abraham, and Simon Dollé. Offline a/b testing for recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 198–206. ACM, 2018.

[33] Carlos A Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):13, 2016.

[34] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[35] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[36] R. He, W.-C. Kang, and J. McAuley. Translation-based recommendation. In *ACM Conference on Recommender systems*, 2017.

[37] Ruining He and Julian McAuley. Fusing similarity models with markov chains for sparse sequential recommendation. In *International Conference on Data Mining*. IEEE, 2016.

[38] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *International Conference on World Wide Web*, pages 173–182. ACM, 2017.

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

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

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

[42] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. Collaborative metric learning. In *International Conference on World Wide Web*, pages 193–201, 2017.

[43] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *International Conference on Data Mining*, pages 263–272. IEEE, 2008.

[44] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[45] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[46] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

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

[48] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*, 2014.

[49] Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

[50] Dan Jurafsky and James H Martin. *Speech and language processing*, volume 3. Pearson London, 2014.

[51] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[52] Kenji Kawaguchi. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems*, pages 586–594, 2016.

[53] Yoon Kim. Convolutional neural networks for sentence classification. In *Conference on Empirical Methods on Natural Language Processing*, pages 1756–1751. ACL, 2014.

[54] Yoon Kim and Alexander M Rush. Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947*, 2016.

[55] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[56] Yehuda Koren. Collaborative filtering with temporal dynamics. In *ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009.

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

[58] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[59] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. Data poisoning attacks on factorization-based collaborative filtering. In *Advances in neural information processing systems*, pages 1885–1893, 2016.

[60] Hui Li, Tsz Nam Chan, Man Lung Yiu, and Nikos Mamoulis. Fexipro: Fast and exact inner product retrieval in recommender systems. In *International Conference on Management of Data*, pages 835–850. ACM, 2017.

[61] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation*, pages 583–598, 2014.

[62] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 2003.

[63] David C Liu, Stephanie Rogers, Raymond Shiau, Dmitry Kislyuk, Kevin C Ma, Zhigang Zhong, Jenny Liu, and Yushi Jing. Related pins at pinterest: The evolution of a real-world recommender system. In *International Conference on World Wide Web*, pages 583–592, 2017.

[64] Duen-Ren Liu, Chin-Hui Lai, and Wang-Jung Lee. A hybrid of sequential rules and collaborative filtering for product recommendation. *Information Sciences*, 179(20): 3505–3519, 2009.

[65] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H. Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1930–1939, 2018.

[66] Benjamin Marlin, Richard S Zemel, Sam Roweis, and Malcolm Slaney. Collaborative filtering and the missing at random assumption. *arXiv preprint arXiv:1206.5267*, 2012.

[67] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, 2010.

[68] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, pages 807–814, 2010.

[69] Nagarajan Natarajan, Inderjit S Dhillon, Pradeep K Ravikumar, and Ambuj Tewari. Learning with noisy labels. In *Advances in neural information processing systems*, pages 1196–1204, 2013.

[70] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In *International Conference on Data Mining*, pages 502–511. IEEE, 2008.

[71] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. Text matching as image recognition. In *AAAI Conference on Artificial Intelligence*, pages 2793–2799. AAAI Press, 2016.

[72] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Jingfang Xu, and Xueqi Cheng. Deeprank: A new deep architecture for relevance ranking in information retrieval. *arXiv preprint arXiv:1710.05649*, 2017.

[73] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[74] Vladas Pipiras and Murad S Taqqu. *Long-range dependence and self-similarity*, volume 45. Cambridge university press, 2017.

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

[76] Steffen Rendle and Christoph Freudenthaler. Improving pairwise learning for item recommendation from implicit feedback. In *International Conference on Web Search and Data Mining*, pages 273–282. ACM, 2014.

[77] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Conference on Uncertainty in Artificial Intelligence*, pages 452–461. AUAI Press, 2009.

[78] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *International Conference on World Wide Web*, pages 811–820. ACM, 2010.

[79] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[80] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2008.

[81] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *International Conference on Machine learning*, pages 791–798. ACM, 2007.

[82] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *International Conference on World Wide Web*, pages 285–295. ACM, 2001.

[83] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *International Conference on World Wide Web*, pages 111–112. ACM, 2015.

[84] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[85] Elena Smirnova and Flavian Vasile. Contextual sequence modeling for recommendation with recurrent neural networks. In *Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems*, pages 2–9. ACM, 2017.

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

[87] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[88] Harald Steck. Training and testing of recommender systems on data missing not at random. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 713–722. ACM, 2010.

[89] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 2014.

[90] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *ACM International Conference on Web Search and Data Mining*, 2018.

[91] Jiaxi Tang and Ke Wang. Ranking distillation: Learning compact ranking models with high performance for recommender system. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2289–2298. ACM, 2018.

[92] 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.

[93] Christina Teflioudi, Rainer Gemulla, and Olga Mykytiuk. Lemp: Fast retrieval of large entries in a matrix product. In *International Conference on Management of Data*, pages 107–122. ACM, 2015.

[94] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *SSW*, page 125, 2016.

[95] 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.

[96] Andrea Vedaldi and Karel Lenc. Matconvnet: Convolutional neural networks for matlab. In *International conference on Multimedia*, pages 689–692. ACM, 2015.

[97] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *International Conference on Knowledge Discovery and Data Mining*, pages 1235–1244. ACM, 2015.

[98] Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. Learning hierarchical representation model for nextbasket recommendation. In *International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 403–412. ACM, 2015.

[99] Jason Weston, Samy Bengio, and Nicolas Usunier. Large scale image annotation: learning to rank with joint word-image embeddings. *Machine learning*, 81(1):21–35, 2010.

[100] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J. Smola, and How Jing. Recurrent recommender networks. In *International Conference on Web Search and Data Mining*, pages 495–503. ACM, 2017.

[101] Chen Wu and Ming Yan. Session-aware information embedding for e-commerce product recommendation. In *ACM on Conference on Information and Knowledge Management*, pages 2379–2382. ACM, 2017.

[102] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *International Conference on Web Search and Data Mining*, pages 153–162. ACM, 2016.

[103] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. End-to-end neural ad-hoc ranking with kernel pooling. In *International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 55–64, 2017.

[104] Longqi Yang, Michael Sobolev, Yu Wang, Jenny Chen, Drew Dunne, Christina Tsangouri, Nicola Dell, Mor Naaman, and Deborah Estrin. How intention informed recommendations modulate choices: A field study of spoken word content. In *The World Wide Web Conference*, pages 2169–2180. ACM, 2019.

[105] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[106] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. A simple convolutional generative network for next item recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 582–590. ACM, 2019.

[107] Quan Yuan, Gao Cong, and Aixin Sun. Graph-based point-of-interest recommendation with geographical and temporal influences. In *International Conference on Information and Knowledge Management*, pages 659–668. ACM, 2014.

[108] Chenyi Zhang, Ke Wang, Hongkun Yu, Jianling Sun, and Ee-Peng Lim. Latent factor transition for dynamic collaborative filtering. In *SIAM International Conference on Data Mining*, pages 452–460. SIAM, 2014.

[109] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. Discrete collaborative filtering. In *International Conference on Research and Development in Information Retrieval*, pages 325–334. ACM, 2016.

[110] Yan Zhang, Defu Lian, and Guowu Yang. Discrete personalized ranking for fast collaborative filtering from implicit feedback. In *AAAI Conference on Artificial Intelligence*, pages 1669–1675. AAAI Press, 2017.

[111] Zhiwei Zhang, Qifan Wang, Lingyun Ruan, and Luo Si. Preference preserving hashing for efficient recommendation. In *International Conference on Research and Development in Information Retrieval*, pages 183–192. ACM, 2014.

[112] Lei Zheng, Vahid Noroozi, and Philip S. Yu. Joint deep modeling of users and items using reviews for recommendation. In *International Conference on Web Search and Data Mining*, pages 425–434. ACM, 2017.

[113] Ke Zhou and Hongyuan Zha. Learning binary codes for collaborative filtering. In *International Conference on Knowledge Discovery and Data Mining*, pages 498–506. ACM, 2012.

[114] Xiaojin Jerry Zhu. Semi-supervised learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2005.

# Appendix A

# List of Publications

**Chapter 3**

1. Tang J, Wang K. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. In Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (WSDM) 2018 Feb 2 (pp. 565-573). ACM. [90]

**Chapter 4**

1. Tang J, Belletti F, Jain S, Chen M, Beutel A, Xu C, H Chi E. Towards Neural Mixture Recommender for Long Range Dependent User Sequences. In The Web Conference (WWW) 2019 May 13 (pp. 1782-1793). ACM. [92]

**Chapter 5**

1. Tang J, Wang K. Ranking Distillation: Learning Compact Ranking Models with High Performance for Recommender System. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD) 2018 Jul 19 (pp. 2289-2298). ACM. [91]

For the three papers listed above, I am the primary author of the entire work. In the first and the last paper, I made contribution in coming up with the research questions, proposing the solutions, conducting the experiments and writing paper. In the second paper, the motivation is from Francois Belletti and the solution is proposed by me during an insightful discussion with Francocis Belletti and Sagar Jain. I conducted all the experiments and finished the first version of the paper draft.