AUTOMATED RECOMMENDER SYSTEMS

A Dissertation

by

QINGQUAN SONG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Xia Hu
Committee Members,	James Caverlee
	Theodora Chaspari
	Jianhua Huang
Head of Department,	Scott Schaefer

May 2021

Major Subject: Computer Science

ABSTRACT

Recommender systems have been existing accompanying by web development, driving personalized experience for billions of users. They play a vital role in the information retrieval process, overcome the information overload by facilitating the communication between business people and the public, and boost the business world. Powered by the advances of machine learning techniques, modern recommender systems enable tremendous automation on the data preprocessing, information distillations, and contextual inferences. It allows us to mine patterns and relationships from massive datasets and various data resources to make inferences. Moreover, the fast evolvement of deep learning techniques brings vast vitality and improvements dived in both academic research and industry applications. Despite the prominence achieved in the recent recommender systems, the automation they have been achieved is still limited in a narrow scope. On the one hand, beyond the static setting, real-world recommendation tasks are often imbued with high-velocity streaming data. On the other hand, with the increasing complexity of model structure and system architecture, the handcrafted design and tuning process is becoming increasingly complicated and time-consuming. With these challenges in mind, this dissertation aims to enable advanced automation in recommender systems. In particular, we discuss how to update factorization-based recommendation models adaptively and how to automatically design and tune recommendation models with automated machine learning techniques. Four main contributions are made via tackling the challenges:

- The first contribution of this research dissertation is the development of a tensor-based algorithm for streaming recommendation tasks.
- As deep learning techniques have shown their superiority in recommendation tasks and become dominant in both academia and industry applications, the second contribution is exploring and developing advanced deep learning algorithms to tackle the recommendation problem with the streaming dataset.
- To alleviate the burden of human efforts, we explore adopting automated machine learning in

designing and tuning recommender systems. The third contribution of this dissertation is the development of a novel neural architecture search approaches for discovering useful features interactions and designing better models for the click-through rate prediction problem.

- Considering a large number of recommendation tasks in industrial applications and their similarities, in the last piece of work work, we focus on the hyperparameter tuning problem in the transfer-learning setting and develop a transferable framework for meta-level tuning of machine learning models.

DEDICATION

To my mother and my father.

ACKNOWLEDGMENTS

This dissertation would have never been completed without a few great people who always supported, inspired, and influenced me.

First and foremost, I would like to express my sincere gratitude to my advisor Dr. Xia Hu who is the best mentor, advisor, and friend with his mentorship, guidance, a steady stream of supports and encouragement. His profession, patients, and kindness to research work, colleagues, students, and families are deeply engraved in my mind. Due to the limited space, I will just say that he is my role model and tirelessly shaped me into a better researcher and man. I would also like to thank the rest of my dissertation committee, Dr. James Caverlee, Dr. Theodora Chaspari, and Dr. Jianhua Huang, for their continuous advice and support during my Ph.D.

In addition, I also would like to thank Dr. Hancheng Ge and Dr. Shiyu Chang for their unselfish help. I was fortunate to have opportunities to collaborate with them, and I really enjoyed the discussions. Besides, I am grateful to Dr. Dehua Cheng, Dr. Hanning Zhou, Dr. Jiyan Yang, Dr. Yuandong Tian, who offered me my first industry internship at Facebook and gave me absolute freedom and thought-provoking hours of discussions for my research project. I also would like to thank Dr. Huiji Gao, Dr. Chengming Jiang, Dr. Yunbo Ouyang, Dr. Jun Jia, Dr. Bo Long, Dr. Bee-chung Chen for their help and advice on my remote internship at LinkedIn. Many thanks to my awesome lab-mates at DATA (Data Analytics at Texas A&M) Lab for their tremendous support and helpful discussions. I enjoyed all office discussions, exchange of ideas, and jokes. We shared the wonderful time through these years, collaborated on exciting projects, experienced the moments of paper acceptance as well as the time of being rejected. Special thanks to all the reviewers to the paper I have submitted, the Texas A&M University, and all the funding agencies, including National Science Foundation and Defense Advanced Research Projects Agency.

Finally, I am deeply indebted to my parents Dezhen Song and Suhua Lin, for their love and endless support. They are my role models, always encouraging me and having faith in me no matter I am in ups and downs in my life.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Professor Xia Hu [advisor], Professor James Caverlee, and Professor Theodora Chaspari of the Department of Computer Science and Engineering, and Professor Jianhua Huang of the Department of Statistics. Chapter III is conducted based on the discussion with Dr. Shiyu Chang from MIT-IBM Watson AI Lab. Chapter IV is conducted based on the work majorly done while I was interning at Facebook. Chapter V is conducted based on the work majorly done while I was interning at LinkedIn. All work conducted for the dissertation was completed by the student independently.

Funding Sources

This work is, in part, supported by DARPA (#W911NF-16-1-0565 and #FA8750-17-2-0116), and NSF (#IIS- 1657196 and #IIS-1718840). The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION[1]

1.1 Motivation and Challenges

Recommender systems have been existing accompanying the web development, driving personalized experience for billions of users. It overcomes the online information overload by connecting users with items they are interested in, such as products, services, and social content. For example, Amazon recommender system recommends sponsored products or brands based on our search history and preference. LinkedIn recommender system recommends the jobs that we might be interested in based on our profile. Facebook and Twitter recommender systems recommend the content published by the groups or users that we may concern. These recommender systems play a vital role in the information retrieval process, facilitate communication between business people and the public, and boost the business world.

Recommender systems are often classified and characterized by their internal filtering algorithms. Popular ones include collaborative filtering algorithms, which leverage the explicit rating information given by the users to items or implicit user behaviors; content-based filtering algorithms, which utilize the item features and content information; knowledge-based filtering algorithms, which utilize some explicit recommendation rules towards contextual inferences; and some hybrid ones. Powered by the advances of machine learning techniques, modern recommender systems enable tremendous automation on the data preprocessing, information distillations, and contextual inferences. It allows us to mine patterns and relationships from massive datasets and various data resources towards personalized recommendations. Moreover, the fast evolvement of deep learning techniques brings vast vitality and further improvements in recommender systems dived in both

[1]This chapter is reprinted with permission from "Multi-Aspect Streaming Tensor Completion" by Qingquan Song, Xiao Huang, Hancheng Ge, James Caverlee and Xia Hu, 2017, Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Copyright 2017 by ACM; "Coupled Variational Recurrent Collaborative Filtering" by Qingquan Song, Shiyu Chang and Xia Hu, 2019, Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Copyright 2019 by ACM; and "Towards Automated Neural Interaction Discovery for Click-Through Rate Prediction" by Qingquan Song, Dehua Cheng, Hanning Zhou, Jiyan Yang, Yuandong Tian and Xia Hu, 2020, Proceedings of the 26rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Copyright 2020 by ACM.

academic researches and industry applications.

Despite the prominence achieved in the recent recommender systems, the automation they have been achieved is still limited in a narrow scope. On the one hand, beyond the static setting, real-world recommendation tasks are often imbued with high-velocity streaming data. For example, Facebook users update 684,478 messages, and Twitter users post over 100,000 tweets every minute.[2] However, most recommender systems still focus on static settings and cannot be updated in a streaming manner. With the large volume and velocity of newly created data instances and features, constructing the recommender systems from scratch is computationally expensive and even prohibited due to high time and resource requirements. On the other hand, with the increasing complexity of model structure and system architecture, the handcrafted design and tuning process is increasingly complicated and time-consuming. It also requires the accumulation of expertise and experience, which hinders the recommender systems from moving toward further automation and democratization.

Enabling the advanced automation of recommender systems is quite challenging. Firstly, the dataset velocity and volume are large. The desired recommender system should be able to expeditiously extract the prior knowledge from former time steps and effectively digest it for current recommendations. Secondly, designing, selecting, and tuning a recommender system is non-trivial, even for human experts. Using the deep recommender systems as an example, how to select a proper neural architecture and configure its hyperparameters such as the number of layers, units, batch sizes, and learning rates is still underexplored. The high complexity and diversity of deep neural networks block us from trying out all available candidates towards the optimal one exhaustively. Moreover, since the recommendation tasks are enormous in real-world applications, tuning the recommender systems one by one for each task from scratch is also laborious.

1.2 Related Work

Increasing efforts are being made on proposing automated recommender systems to tackle the challenges mentioned in the previous section. We summarize some related work in this section from

[2] http://mashable.com/2012/06/22/data-created-every-minute/

three aspects:

Streaming Recommender Systems. Beyond traditional static settings, streaming recommender systems have attracted widespread concerns in coping with the high data velocity and their naturally incremental properties [1, 2]. Different from static time-aware models [3, 4, 5, 6, 7], which only take account of temporal dynamics without updating in an streaming fashion, streaming recommender systems dynamically encode temporal information and generate response instantaneously [8, 9, 10, 11]. Some existing works focus on extending classical memory-based recommendation algorithms into online fashions to address the streaming challenges such as [12] and [13]. Besides memory-based methods, model-based methods [14, 15, 16] are becoming more and more popular in recent years, which conducts recommendation based on well-trained models rather than explicitly aggregating and prediction based on the similarity relationships. Diaz-Aviles et al. leverage the active learning strategy to sample and maintain a delicately designed reservoir, thus providing a pairwise matrix factorization approach for streaming recommendation. Chang et al. [2] exploit continuous Markov process (Brownian motion) to model the temporal drifting of users and items, which introduces a principled way to model data streams. Wang et al. [17] propose a streaming ranking-based framework based on Bayesian Personalized Ranking [18] to address the user interest drifting as well as system overload problem. Although many recent advances based on deep neural networks especially RNNs have been made to model streaming inputs and capture the complex temporal dynamics [19, 20, 21], most of them overlook the causality inherited in the data generation process.

Deep Recommender Systems. Deep learning techniques have brought vast vitality and achieve dramatic improvement in recommender systems [22]. They have been adopted in various recommendation tasks as well as accommodating different data sources [23, 24, 20]. From the perspective of the general framework, deep recommender systems could be categorized into solely deep models, which conduct recommendations based only on deep frameworks [24, 25, 26]; and integration models, which integrate deep techniques with traditional recommender systems [23, 27, 28]. From the perspective of deep frameworks, these models could also be divided into: (1) single deep

models, which are built upon single neural building blocks such as multi-layer perceptron [26], convolutional neural network [29], and recurrent neural network [20]; and (2) composite models, which are constructed with different deep learning techniques [30].

AutoML for Recommender Systems. AutoML aims to promote the design and tuning of machine learning models, thus release the burden of human experts and enable the democratization of advanced machine learning models to practitioners without enough experience. Designing the AutoML algorithm requires the specification of three main components, including *search space*, *search techniques*, and *performance estimation strategy* [31]. Though early work of AutoML mainly focuses on tabular data [32], recent work of AutoML mostly targets the image- and text-related models [33]. This is partly because of two reasons: (1) recommendation models usually adopt multiple diverse and ad-hoc operations, leading to unstructured search space, which is hard to be formally created; (2) the data scalability and feature heterogeneity is often more severe in recommender systems. Existing AutoML work for recommendation often focuses on two aspects. On one side, they try to improve the recommendation model performance by automatically searching different architectures and optimal hyperparameters [34, 35]. On the other side, researchers have been starting to investigate how to reduce the time and space complexity of recommendation models with the help of AutoML techniques [36, 37, 38].

1.3 Dissertation Contributions

To tackle the core challenges of data velocity and model-design complicacy towards advanced automation of recommender systems, we made four main contributions in this dissertation:

- The first contribution of this research dissertation is the development of a tensor-based algorithm for streaming recommendation tasks. We formally define a problem of multi-aspect streaming tensor completion, which aims to impute the tensor-structured dataset that is incrementally augmented. Then we propose two novel incremental tensor completion methods and apply them to recommendation tasks.

- As deep learning techniques have shown their superiority in recommendation tasks and

become dominant in both academia and industry applications, the second contribution is the exploration and development of advanced deep learning algorithms to tackle the recommendation problem with streaming datasets. Specifically, we focus on the collaborative filtering setting and jointly combine stochastic processes and deep factorization models under a Bayesian paradigm to model the generation and evolution of users' preferences and items' popularities.

- To alleviate the burden of human efforts, we explore adopting automated machine learning (AutoML) in designing and tuning recommender systems. The third contribution of this dissertation is developing a novel neural architecture search (NAS) approaches for discovering useful features interactions and designing better models for the Click-Through Rate (CTR) prediction problem. We formally design a search space for the CTR prediction task and propose an efficient low-fidelity search algorithm to discover better architectures for CTR predictions.

- Considering a large number of recommendation tasks in industrial applications and their similarities, we focus on a transfer learning setting and explore a meta-level tuning of recommender systems. We propose a transfer-learning framework to transfer the historical architecture evaluation information (i.e., the different machine learning models evaluated on historical tasks) to accelerate the tuning and design of models on new tasks.

The first two contributions lie in the automation of recommender systems on incremental learning setting, while the remaining ones focus on hyper-level automation, which aim to automate the design and tuning of recommender systems on single or multiple tasks efficiently.

1.4 Dissertation Overview

The remainder of this dissertation is organized as follows:

- **Chapter 2: Multi-Aspect Streaming Tensor Completion.** In this chapter, we develop a tensor-based framework for uncovering multi-aspect correlations in high-dimensional recommendation data and efficient recommendation in the streaming-data setting.

- **Chapter 3: Coupled Variational Recurrent Collaborative Filtering.** In this chapter, we explore the deep-learning solution to the streaming recommendation problem. We focus on the collaborative filtering setting and add deep-learning flavor to the conventional probabilistic factorization models to tackle the data velocity challenge in recommender systems.

- **Chapter 4: Towards Automated Neural Interaction Discovery for Click-Through Rate Prediction.** In this chapter, we explore how to enable the automation on the architecture design and hyperparameter tuning in recommender systems and propose a neural interaction discovery framework to automatically design and tune deep-learning models for the click-through rate prediction task.

- **Chapter 5: Transferable Black-Box Optimization for Hyperparamter Tuning.** In this chapter, we explore how to leverage the similarities of different web application tasks to accelerate the automated hyperparameter tuning. We propose a transfer learning framework for black-box optimization and evaluate it on multiple synthetic datasets. The proposed framework can be applied for transferable hyperparameter tuning and architecture design on multiple recommendation tasks.

- **Chapter 6: Conclusions and Future Research Opportunities.** We conclude the dissertation with a summary of contributions and discuss potential research directions to the results presented.

2. MULTI-ASPECT STREAMING TENSOR COMPLETION[1]

In this chapter, we tackle the streaming data challenge in recommender systems, especially for structured high-dimensional data. The raw data reflects the temporal dynamics and multi-aspect correlations in the recommender systems beyond the user-item correlations. We consider the streaming recommendation problem as a streaming tensor completion problem. Our developed framework can effectively uncover the relations leveraging a tensor completion framework and conduct efficient recommendations in the streaming-data setting.

2.1 Introduction

Tensors are multidimensional or N-way extensions of matrices. Tensor completion, which aims at filling the missing entries of partially observed tensors, has become one of the effective computational tools in recommender systems [39, 40], as well as many other data mining applications such as social network analysis [41, 42], image recovery [43], and clinical data analysis [44, 45].

Beyond traditional static setting, real-world recommendations are often imbued with high velocity streaming data. Existing work has been conducted on developing dynamic tensor completion methods based on a widely used assumption that tensors will be developed in one mode (or dimension). Following this assumption, online completion methods have been proposed based on CP decomposition [46, 47]. Unfortunately, in many real-world applications, a tensor may develop in multiple dimensions and traditional assumption does not hold. For example, given a dynamic tensor in recommender systems structured as $user \times movie \times actor$, the number of registered users, movies, and actors may all increase as time goes. This incremental property gives rise to a new streaming pattern, which we call multi-aspect streaming. Figure 2.1 depicts traditional streaming tensors and its generalized multi-aspect streaming tensors counterpart. As we can observe from the figure, from time t to $t + 2$, each mode of the multi-aspect streaming tensor increases while only

[1]This chapter is reprinted with permission from "Multi-Aspect Streaming Tensor Completion" by Qingquan Song, Xiao Huang, Hancheng Ge, James Caverlee and Xia Hu, 2017. Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Copyright 2017 by ACM.

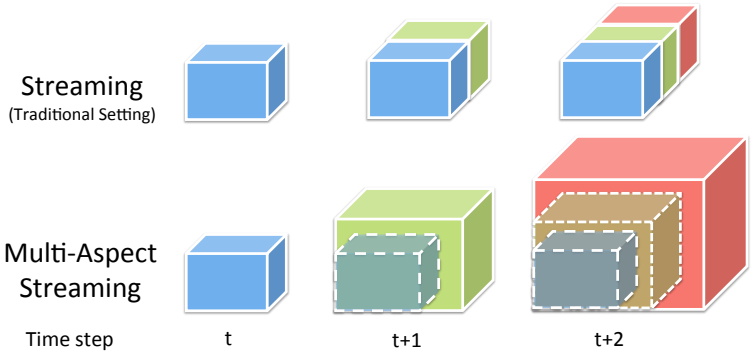


Figure 2.1: An illustration of traditional streaming tensors and multi-aspect streaming tensors.

one mode of the streaming tensor grows.

Dealing with the multi-aspect streaming tensors is challenging due to three reasons. First, coordinating multidimensional dynamics is difficult given the uncertainty of tensor mode changes. It is too arbitrary to fix partial structure of a certain mode without accessing the data for an update. Second, the incremental part of multi-aspect streaming data may not be a complete tensor. This is different from streaming tensors in which the incremental part is well structured and can be directly decomposed. Third, the multi-aspect streaming pattern leads to much higher time and space complexity. Although some distributed and parallel algorithms could partially address the scalability issue [48, 49, 50], the problem of non-adaptability still exists, meaning that the completion process of new data cannot directly benefit from the existing model without complete model reconstruction.

To tackle these challenges, we investigate how to complete multi-aspect streaming tensors based on CP decomposition. Specifically, we mainly study: (1) How to build up an updatable framework based on CP decomposition for efficiently modeling multi-aspect streaming tensors? (2) How to effectively capture the low-rank subspace for completion purpose? Through answering the two questions, we propose a general Multi-Aspect Streaming Tensor completion algorithm (MAST) and further propose a modified model T-MAST for temporal multi-aspect streaming tensor completion, which is a special case of the general problem. We empirically validate the effectiveness and efficiency of the proposed models on four real-world datasets with different real-world applications, and the demonstrate its advantages on both static and dynamic baselines.

2.2 Preliminaries

Notations and definitions in this chapter are presented as follows. **Notations**: An N^{th}-order tensor is an N-way array, also known as N-dimensional or N-mode tensor. In this chapter, tensors, matrices, vectors are denoted by Euler script letters (e.g., $\mathcal{X}$), boldface uppercase letters (e.g., $\mathbf{A}$), and boldface lowercase letters (e.g., $\mathbf{a}$), respectively. An entry of tensor $\mathcal{X}$ indexed by $[i_1^t, \ldots, i_N^t]$ is denoted as $\mathcal{X}[i_1^t, \ldots, i_N^t]$. We use $\mathbf{A}^{-1}$, $\mathbf{A}^\top$, and $\|\mathbf{A}\|_{\text{F}}$ to denote the transpose, inverse, and Frobenius norm of matrix $\mathbf{A}$, respectively. Let $[\![\cdot]\!]$ denote the Kruskal operator. Notations $\odot$ and $\circledast$ are used to represent the Khatri-Rao product and Hadamard product, respectively. The main symbols and operations are listed in Table 2.1.

Definition 1 (CP Decomposition). Given an N^{th}-order tensor $\mathcal{X}$, its CP decomposition is an approximation of N loading matrices $\mathbf{A}_n \in \mathbb{R}^{I_n \times R}, n = 1, \ldots, N$, such that,

$$\mathcal{X} \approx [\![\mathbf{A}_1, \ldots, \mathbf{A}_N]\!], \tag{2.1}$$

where R is a positive integer denoting an upper bound of the rank of $\mathcal{X}$. We can further unfold $\mathcal{X}$ along its n^{th} mode as follows,

$$\mathcal{X} \quad \overset{\text{unfold}}{\Longrightarrow} \quad \mathbf{X}_{(n)} \approx \mathbf{A}_n(\mathbf{A}_N \odot \ldots \mathbf{A}_{n+1} \odot \mathbf{A}_{n-1} \ldots \odot \mathbf{A}_1)^\top = \mathbf{A}_n\big[(\mathbf{A}_k)^{\odot_{k \neq n}}\big]^\top. \tag{2.2}$$

Definition 2 (Coupled Tensor). If two tensors share one or more modes, then they are coupled with each other and called coupled tensors. For example, in recommender systems, two tensors structured as $user \times movie \times time$ and $movie \times actor \times director$ are coupled on the $movie$ mode.

Definition 3 (Tensor Sequence). A sequence of N^{th}-order tensors $\mathcal{X}^{(1)}, \ldots, \mathcal{X}^{(T)}, \ldots$ is called a tensor sequence denoted as $\{\mathcal{X}^{(T)}\}$, where each $\mathcal{X}^{(T)} \in \mathbb{R}^{I_1^T \times I_2^T \times \ldots \times I_N^T}, T \in \mathbb{Z}^+$.

Definition 4 (Multi-aspect streaming Tensor Sequence). A sequence of N^{th}-order tensors $\{\mathcal{X}^{(T)}\}$ is called multi-aspect streaming tensor sequence if for any $T \in \mathbb{Z}^+$, $\mathcal{X}^{(T-1)}$ is the sub-tensor of $\mathcal{X}^{(T)}$, denoted as $\mathcal{X}^{(T-1)} \subsetneqq \mathcal{X}^{(T)}$. T increases with time, and $\mathcal{X}^{(T)}$ is the snapshot tensor of this sequence

Table 2.1: Main symbols and operations for in tensor factorization.

Notations	Definitions
$\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$	N^{th}-order tensor
$\mathbf{X}_{(n)} \in \mathbb{R}^{I_n \times (\Pi_{i \neq n}^{N} I_i)}$	Mode-n unfolding matrix of tensor $\mathcal{X}$
$[\![\cdot]\!]$	Kruskal operator, e.g., $\mathcal{X} \approx [\![\mathbf{A}_1, \ldots, \mathbf{A}_N]\!]$
$\odot$	Khatri-Rao product
$\circledast$	Hadamard product
$(\mathbf{A}_k)^{\odot_{k \neq n}}$	$\mathbf{A}_N \odot \ldots \odot \mathbf{A}_{n+1} \odot \mathbf{A}_{n-1} \odot \ldots \odot \mathbf{A}_1$
$(\mathbf{A}_k)^{\circledast_{k \neq n}}$	$\mathbf{A}_N \circledast \ldots \circledast \mathbf{A}_{n+1} \circledast \mathbf{A}_{n-1} \circledast \ldots \circledast \mathbf{A}_1$

taken at time T.

In fact, any tensor sequence $\{\mathcal{X}^{(T)}\}$ can be modeled as a multi-aspect streaming one by adding an additional mode, i.e., defining an $(N+1)^{\text{th}}$-order new tensor sequence $\{\mathcal{Y}^{(T)}\}$, where $\mathcal{Y}^{(T)} \in \mathbb{R}^{J_1^T \times J_2^T \times \ldots \times J_N^T \times T}$, $J_n = \max_t\{I_n^t\}$. Entries of $\mathcal{Y}^{(T)}$ are defined as:

$$\mathcal{Y}^{(T)}[i_1^t, \ldots, i_N^t, t] = \begin{cases} \mathcal{X}^{(t)}[i_1^t, \ldots, i_N^t], & \text{if } 1 \leq i_n^t \leq I_N^t, \\ 0, & \text{otherwise,} \end{cases} \tag{2.3}$$

where $1 \leq t \leq T$, and then $\mathcal{Y}^{(T-1)} \subsetneqq \mathcal{Y}^{(T)}$ is satisfied.

Based on the terminologies, the multi-aspect streaming tensor completion problem can be formally defined as follows.

Given a multi-aspect streaming tensor sequence $\{\mathcal{X}^{(T)}\}$ with missing entries, we aim at recovering the missing data in current snapshot $\mathcal{X}^{(T)}$. Since $\mathcal{X}^{(T-1)} \subsetneqq \mathcal{X}^{(T)}$, and we have recovered $\mathcal{X}^{(T-1)}$ in previous time step, the problem is equivalent to completing the relative complement of $\mathcal{X}^{(T-1)}$ in $\mathcal{X}^{(T)}$ which is denoted as $\mathcal{X}^{(T)} \backslash \mathcal{X}^{(T-1)}$.

2.3 Multi-Aspect Streaming Tensor Completion Framework

In this section, we introduce the proposed framework MAST to solve the general multi-aspect streaming completion problem. By taking advantage of the learned model in previous time step, MAST can accelerate the completion process while preserving comparable completion accuracy

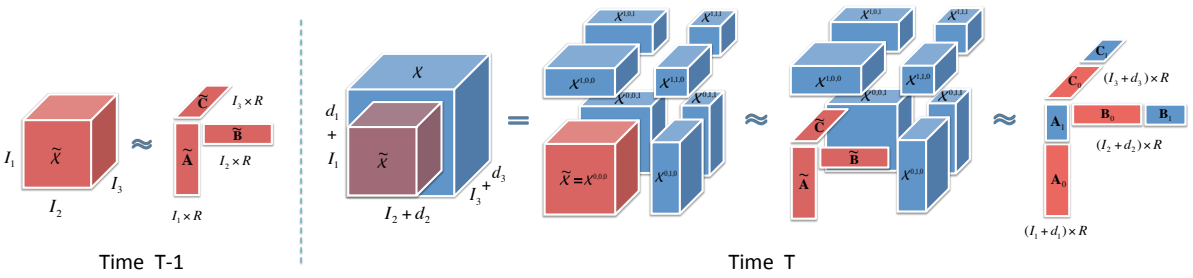


Figure 2.2: An illustration of the proposed dynamic tensor decomposition model DTD.

to traditional batch learning methods. To achieve this, we begin with designing a dynamic tensor decomposition (DTD) algorithm to model the incremental pattern of multi-aspect streaming tensor sequence and track its low-rank subspace. Then, based on the DTD algorithm, we conduct the completion task via the nuclear norm constraint. At the end, we discuss a special case in which "time" is one mode of the tensors and further improve our framework for a better completion effectiveness.

2.3.1 Dynamic Tensor Decomposition

CP decomposition is a widely used tool for tensor completion [43, 51, 42]. By decomposing a tensor into multiple low-rank matrices using its observed entries, the underlying low-rank subspace structure can be tracked for filling the missing entries. There are two intuitive solutions for our main problem. At each time step, we can either reconstruct the batch CP model for the entire snapshot, or decompose $\mathcal{X}^{(T)} \backslash \mathcal{X}^{(T-1)}$ only. The former solution is both time and space consuming since the snapshots are increasing dramatically. In contrast, the latter fails to exploit the information in $\mathcal{X}^{(T-1)}$ to help achieve desirable imputation.

To accelerate completion process while preserving effectiveness, a dynamic CP decomposition method DTD is proposed. It is based on two ideas. First, CP decomposition enjoys the good partitioning property which means if $\mathcal{X}^{(T)}$ is approximated by $[\![\mathbf{A}_1, \ldots, \mathbf{A}_N]\!]$, then its sub-tensor $\mathcal{X}^{(T-1)}$ could be approximated by $[\![\widetilde{\mathbf{A}}_1, \ldots, \widetilde{\mathbf{A}}_N]\!]$ where $\widetilde{\mathbf{A}}_n$ is a sub-matrix of $\mathbf{A}_n$, for $n = 1, \ldots, N$. This enables us to partition the current snapshot $\mathcal{X}^{(T)}$ to take advantage of the foregone acquired decomposition of its sub-tensor $\mathcal{X}^{(T-1)}$. Second, partial calculation complexity of decomposing

$\mathcal{X}^{(T)}$ can be reduced one order lower by substituting acquired decompositions for $\mathcal{X}^{(T-1)}$. These two ideas are separately used in following two steps of our proposed method for information preservation and optimization acceleration, respectively.

2.3.1.1 Tensor partition and substitution

For the ease of presentation, we use a third-order tensor with steady growth on all three modes as an example. It is straightforward to extend to general N^{th}-order tensors. We use $\widetilde{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ and $\mathcal{X} \in \mathbb{R}^{(I_1+d_1)\times(I_2+d_2)\times(I_3+d_3)}$ to represent two consecutive snapshots $\mathcal{X}^{(T-1)}$ and $\mathcal{X}^{(T)}$, $(T \in \mathbb{Z}^+)$, respectively. $\widetilde{\mathcal{X}} = 0$ if $T = 0$. We now introduce the details.

Let $\mathbf{A} \in \mathbb{R}^{(I_1+d_1)\times R}, \mathbf{B} \in \mathbb{R}^{(I_2+d_2)\times R}, \mathbf{C} \in \mathbb{R}^{(I_3+d_3)\times R}$ be the CP factor matrices of $\mathcal{X}$, the CP loss function under gaussian noise is:

$$\mathcal{L}(\mathbf{A}, \mathbf{B}, \mathbf{C}) = \ \|\mathcal{X} - [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]\|_{\text{F}}^2. \tag{2.4}$$

After partitioning $\mathcal{X}$ into eight sub-tensors according to $\widetilde{\mathcal{X}}$, we use a binary tuple $(i, j, k) \in \{0, 1\}^3 \triangleq \Theta$ to denote these eight sub-tensors as shown in Figure 2.2, with $\widetilde{\mathcal{X}} = \mathcal{X}^{0,0,0}$. Each pair of adjacent sub-tensors are coupled with each other. Based on the partitioning property of CP decomposition, each sub-tensor of $\mathcal{X}$ could also be approximated via the sub-matrices of $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, i.e., $\mathcal{X}^{i,j,k} \approx [\![\mathbf{A}_i, \mathbf{B}_j, \mathbf{C}_k]\!]$. $\mathbf{A}_0 \in \mathbb{R}^{I_1 \times R}$ and $\mathbf{A}_1 \in \mathbb{R}^{d_1 \times R}$ are the partitioned sub-matrices of $\mathbf{A}$, i.e., $\mathbf{A}^\top = [\mathbf{A}_0^\top, \mathbf{A}_1^\top]$. Similarly to $\mathbf{B}_j$ and $\mathbf{C}_k$. Then Equation (2.4) could be rewritten as follows.

$$\begin{aligned} \mathcal{L}(\mathbf{A}, \mathbf{B}, \mathbf{C}) &= \sum_{(i,j,k)\in\Theta} \|\mathcal{X}^{i,j,k} - [\![\mathbf{A}_i, \mathbf{B}_j, \mathbf{C}_k]\!]\|_{\text{F}}^2 \\ &= \|\mathcal{X}^{0,0,0} - [\![\mathbf{A}_0, \mathbf{B}_0, \mathbf{C}_0]\!]\|_{\text{F}}^2 \ + \ \mathcal{L}_0, \end{aligned} \tag{2.5}$$

where $\mathcal{L}_0 \triangleq \sum_{(i,j,k)\in\Theta\setminus(0,0,0)} \|\mathcal{X}^{i,j,k} - [\![\mathbf{A}_i, \mathbf{B}_j, \mathbf{C}_k]\!]\|_{\text{F}}^2$.

Let $[\![\widetilde{\mathbf{A}}, \widetilde{\mathbf{B}}, \widetilde{\mathbf{C}}]\!]$ denote the CP decomposition of $\widetilde{\mathcal{X}}$ obtained at time $T-1$. It represents the low-rank subspace which preserves the information of former snapshot. Based on this, we can approximately replace $\mathcal{X}^{0,0,0}$ by $[\![\widetilde{\mathbf{A}}, \widetilde{\mathbf{B}}, \widetilde{\mathbf{C}}]\!]$ and get the loss function as follows:

$$\mathcal{L}(\mathbf{A},\mathbf{B},\mathbf{C}) \approx \ \mu\|[\![\widetilde{\mathbf{A}},\widetilde{\mathbf{B}},\widetilde{\mathbf{C}}]\!] - [\![\mathbf{A}_0,\mathbf{B}_0,\mathbf{C}_0]\!]\|_{\mathrm{F}}^2 \ + \ \mathcal{L}_0, \tag{2.6}$$

where weight $\mu \in [0,1]$ is the forgetting factor [46] used to alleviate the influence of the previous decomposition error. If $\mu = 1$, then $[\![\widetilde{\mathbf{A}},\widetilde{\mathbf{B}},\widetilde{\mathbf{C}}]\!]$ is considered as a perfect decomposition of $\widetilde{\mathcal{X}}$. If $\mu = 0$, then the former-step information is considered as independent to current decomposition. As our final goal is to complete this tensor sequence, this inequality ($0 < \mu < 1$) is quite effective to alleviate the substitution errors especially when the percentage of missing data is high. In other words, this forgetting factor degrades the status of old data and upgrades the power of newly arrived data, so that we can more effectively incorporate newly emerged information to update the foregone decomposition and ensure the substitution error not continuously accumulated.

2.3.1.2 Alternating least square updating

The optimization is based on alternating least squares (ALS), for its fast and easy implementation with high accuracy [52, 53]. The update rule for $\mathbf{A}_0$ is:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{A}_0} =& -2\mu(\widetilde{\mathbf{A}}(\widetilde{\mathbf{C}} \odot \widetilde{\mathbf{B}})^\top(\mathbf{C}_0 \odot \mathbf{B}_0) - \mathbf{A}_0(\mathbf{C}_0 \odot \mathbf{B}_0)^\top(\mathbf{C}_0 \odot \mathbf{B}_0)) \\ & -2\sum_{(j,k)\neq(0,0)} (\mathbf{X}_{(1)}^{0,j,k}(\mathbf{C}_k \odot \mathbf{B}_j) - \mathbf{A}_0(\mathbf{C}_k \odot \mathbf{B}_j)^\top(\mathbf{C}_k \odot \mathbf{B}_j)), \\ \mathbf{A}_0 \leftarrow& \frac{\mu\widetilde{\mathbf{A}}\left[(\widetilde{\mathbf{C}}^\top\mathbf{C}_0) \circledast (\widetilde{\mathbf{B}}^\top\mathbf{B}_0)\right] + \sum_{(j,k)\neq(0,0)} \mathbf{X}_{(1)}^{0,j,k}(\mathbf{C}_k \odot \mathbf{B}_j)}{(\sum_{k=0}^{1}\mathbf{C}_k^\top\mathbf{C}_k) \circledast (\sum_{j=0}^{1}\mathbf{B}_j^\top\mathbf{B}_j) - (1-\mu)(\mathbf{C}_0^\top\mathbf{C}_0) \circledast (\mathbf{B}_0^\top\mathbf{B}_0)}.\end{aligned} \tag{2.7}$$

The main difference between the above update rule and static CP-ALS method is the first term of the numerator. Following the property of Khatri-Rao product, the space and time complexity of calculating the Khatri-Rao products $\odot$ in equation $(\widetilde{\mathbf{C}} \odot \widetilde{\mathbf{B}})^\top(\mathbf{C}_0 \odot \mathbf{B}_0)$ can be reduced using the element-wise Hadamard product, i.e., $\widetilde{\mathbf{C}}^\top\mathbf{C}_0 \circledast \widetilde{\mathbf{B}}^\top\mathbf{B}_0$. Hence, by substituting $\mathcal{X}^{0,0,0}$ with $[\![\widetilde{\mathbf{A}},\widetilde{\mathbf{B}},\widetilde{\mathbf{C}}]\!]$, this term is changed from $\mathbf{X}_{(1)}^{0,0,0}(\mathbf{C}_0 \odot \mathbf{B}_0)$ to $\mu\widetilde{\mathbf{A}}\left[(\widetilde{\mathbf{C}}^\top\mathbf{C}_0) \circledast (\widetilde{\mathbf{B}}^\top\mathbf{B}_0)\right]$ leading to a reduction of time complexity from $\mathcal{O}(RI_1I_2I_3)$ to $\mathcal{O}(R^2(I_1+I_2+I_3))$.

The update rules of the incremental matrix $\mathbf{A}_1$ can be derived as:

$$\mathbf{A}_1 \leftarrow \frac{\sum_{j,k} \mathbf{X}_{(1)}^{1,j,k}(\mathbf{C}_k \odot \mathbf{B}_j)}{(\sum_{k=0}^{1} \mathbf{C}_k^\top \mathbf{C}_k) \circledast (\sum_{j=0}^{1} \mathbf{B}_j^\top \mathbf{B}_j)}. \tag{2.8}$$

Similar update rules for matrices $\mathbf{B}_0$, $\mathbf{C}_0$, $\mathbf{B}_1$, and $\mathbf{C}_1$ can be easily derived. The proposed DTD model enjoys several nice properties. First, it is capable of handling any-mode-change dynamic pattern. Second, it has a fast updating process for the purpose of scalability. Third, it well preserves the information acquired from old data as well as captures the new data characteristics through the updated process. Besides, the idea of partitioning also benefits methods in solving large-scale tensor decomposition problems [50, 48, 49].

2.3.2 Proposed Completion Framework - MAST

The proposed DTD method balances the trade-off between efficiency and effectiveness in modeling the multi-aspect streaming pattern. As CP decomposition is an effective way in addressing static tensor completion problem, the DTD model also paves the way of solving the multi-aspect streaming tensor completion problem. In this section, we show how MAST is applied to this problem. We start with introducing the nuclear norm based method on low-rank tensor completion problem in batch setting and then combine it with the DTD model for completing multi-aspect streaming tensor sequences.

2.3.2.1 Low-rank tensor completion (LRTC)

Generalized from matrix completion problem, the LRTC problem can be formulated as an equivalent rank-minimization problem as follows [45]:

$$\underset{\mathcal{X}}{\text{minimize}}\ rank(\mathcal{X}) \quad \text{subject to } \mathbf{\Omega} \circledast \mathcal{X} = \mathcal{T}, \tag{2.9}$$

where $\mathcal{X}$ denotes the complete tensor, $\mathcal{T}$ denotes the practical observations of $\mathcal{X}$. $\mathbf{\Omega}$ is a binary tensor with the same size as $\mathcal{X}$ indicating whether each corresponding entry in $\mathcal{X}$ is observed or not.

As calculating the tensor rank is an NP-hard problem [54], one way is to relax the tensor rank by replacing it with the summation of the nuclear norm of its factorized matrices [43, 51]. The

modified relaxation objective function is described as follows:

$$\begin{aligned} \underset{\boldsymbol{\mathcal{X}},\mathbf{A}_1,\mathbf{A}_2,\ldots,\mathbf{A}_N}{\text{minimize}} \quad & \sum_{n=1}^{N} \alpha_n \left\|\mathbf{A}_n\right\|_* + \left\|\boldsymbol{\mathcal{X}} - [\![\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_N]\!]\right\| \\ \text{subject to} \quad & \boldsymbol{\Omega} \circledast \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{T}}, \end{aligned} \tag{2.10}$$

where $\{\alpha_n\}$ are trade-offs to balance the significance of each mode.

2.3.2.2 Combine LRTC with DTD

Since multi-aspect streaming tensor completion problem can be treated as a dynamic LRTC problem, a natural solution is to combine DTD with LRTC approach. By extending Equation (2.6) to N^{th}-order tensor and combining it with Equation (2.10), the proposed loss function for N^{th}-order multi-aspect streaming tensor completion problem is formulated as:

$$\begin{aligned} \boldsymbol{\mathcal{L}} = & \sum_{(i_1,\ldots,i_N)\in\Theta_N\setminus\{0\}^N} \|\boldsymbol{\mathcal{X}}^{(i_1,\ldots,i_N)} - [\![\mathbf{A}_1^{(i_1)}, \ldots, \mathbf{A}_N^{(i_N)}]\!]\|_{\text{F}}^2 \\ & + \sum_{n=1}^{N} \alpha_n \left\|\mathbf{A}_n\right\|_* + \ \mu\|[\![\widetilde{\mathbf{A}}_1, \ldots, \widetilde{\mathbf{A}}_N]\!] - [\![\mathbf{A}_1^{(0)}, \ldots, \mathbf{A}_N^{(0)}]\!]\|_{\text{F}}^2 \\ \triangleq & \ \boldsymbol{\mathcal{L}}_1 + \sum_{n=1}^{N} \alpha_n \left\|\mathbf{A}_n\right\|_*, \end{aligned} \tag{2.11}$$

where $\mathbf{A}_n = \begin{bmatrix} \mathbf{A}_n^{(0)} \\ \mathbf{A}_n^{(1)} \end{bmatrix} \begin{matrix} \in \mathbb{R}^{I_n \times R} \\ \in \mathbb{R}^{d_n \times R} \end{matrix}$. $[\![\widetilde{\mathbf{A}}_1, \ldots, \widetilde{\mathbf{A}}_N]\!]$ is the decomposition of recovered tensor $\widetilde{\boldsymbol{\mathcal{X}}}$ in the previous step, and $\Theta_N = \{0, 1\}^N$.

A tensor-based Alternating Direction Method of Multipliers (ADMM) algorithm is employed to solve this optimization problem. ADMM is an efficient optimization scheme for accommodating various constraints [55, 56]. Following the similar optimization scheme described in [51, 42], we induce auxiliary variables $\mathbf{Z}_n = \begin{bmatrix} \mathbf{Z}_n^{(0)} \\ \mathbf{Z}_n^{(1)} \end{bmatrix} \begin{matrix} \in \mathbb{R}^{I_n \times R} \\ \in \mathbb{R}^{d_n \times R} \end{matrix}$, $n = 1, \ldots, N$ into Equation (2.11), then the corresponding partial augmented Lagrangian function is:

$$\begin{aligned}\underset{\mathfrak{X},\{\mathbf{A}_n\},\{\mathbf{Z}_n\},\{\mathbf{Y}_n\}}{\text{minimize}} \ & \mathcal{L}_1 + \sum_{n=1}^{N}\Big(\alpha_n \left\|\mathbf{Z}_n\right\|_* + < \mathbf{Y}_n, \mathbf{Z}_n - \mathbf{A}_n > + \frac{\eta}{2}\|\mathbf{Z}_n - \mathbf{A}_n\|_{\mathrm{F}}^2\Big) \\ \text{subject to} \quad & \boldsymbol{\Omega} \circledast \boldsymbol{\mathfrak{X}} = \boldsymbol{\mathfrak{T}}, \ \ \mathbf{Z}_n = \mathbf{A}_n, \ \ n = 1, 2, \ldots, N,\end{aligned} \tag{2.12}$$

where $\mathbf{Y}_n = \begin{bmatrix} \mathbf{Y}_n^{(0)} \\ \mathbf{Y}_n^{(1)} \end{bmatrix} \begin{matrix} \in \mathbb{R}^{I_n \times R} \\ \in \mathbb{R}^{d_n \times R} \end{matrix}$ is the Lagrange multiplier matrix, $\eta > 0$ is a penalty parameter. Based on Equations (2.7) and (2.8), by calculating the derivatives of matrices $\{\mathbf{A}_n^{(0)}\}$ and $\{\mathbf{A}_n^{(1)}\}(n = 1, 2, \ldots, N)$, we can get their update rules as follows:

$$\begin{aligned}\mathbf{A}_n^{(0)} \leftarrow & \frac{\mu\widetilde{\mathbf{A}}_n\left[(\widetilde{\mathbf{A}}_k^{\top}\mathbf{A}_k^{(0)})^{\circledast_{k\neq n}}\right] + \sum_{i\in S_n^0}\mathbf{X}_{(n)}^i(\mathbf{A}_k^{(i_k)})^{\odot_{k\neq n}} + \eta\mathbf{Z}_n^{(0)} + \mathbf{Y}_n^{(0)}}{({\mathbf{A}_k^{(0)}}^{\top}\mathbf{A}_k^{(0)} + {\mathbf{A}_k^{(1)}}^{\top}\mathbf{A}_k^{(1)})^{\circledast_{k\neq n}} - (1-\mu)({\mathbf{A}_k^{(0)}}^{\top}\mathbf{A}_k^{(0)})^{\circledast_{k\neq n}} + \eta\mathbf{I}_R}, \\ \mathbf{A}_n^{(1)} \leftarrow & \frac{\sum_{i\in S_n^1}\mathbf{X}_{(n)}^i(\mathbf{A}_k^{(i_k)})^{\odot_{k\neq n}} + \eta\mathbf{Z}_n^{(1)} + \mathbf{Y}_n^{(1)}}{({\mathbf{A}_k^{(0)}}^{\top}\mathbf{A}_k^{(0)} + {\mathbf{A}_k^{(1)}}^{\top}\mathbf{A}_k^{(1)})^{\circledast_{k\neq n}} + \eta\mathbf{I}_R},\end{aligned} \tag{2.13}$$

where $S_n^0 = \{(s_1, \ldots, s_N) | \sum_{k=1}^N s_k \neq 0, s_k \in \{0, 1\}, s_n = 0\}$ and $S_n^1 = \{(s_1, \ldots, s_N) | \forall\ k \in \{1, \ldots, N\}\ s_k \in \{0, 1\}, s_n = 1\}$.

In each iteration, the auxiliary matrices $\{\mathbf{Z}_n\}$ has a closed-form solution as follows[57]:

$$\mathbf{Z}_n = SVT_{\frac{\alpha_n}{\eta}}(\mathbf{A}_n - \frac{\mathbf{Y}_n}{\eta}), \ \ n = 1, 2, \ldots, N. \tag{2.14}$$

where $SVT_{\frac{\alpha_n}{\eta}}$ is the singular value thresholding operator [57] defined as $SVT_\delta(\mathbf{A}) = \mathbf{U}(diag\{\sigma_i - \delta\})_+\mathbf{V}^\top$, where $\mathbf{U}(diag\{\sigma_i\}_{1\leq i\leq r})\mathbf{V}^\top$ is the singular value decomposition of matrix $\mathbf{A}$. For any matrix $\mathbf{X}$, $\mathbf{X}_+ = \max\{\mathbf{X}, 0\}$, where $\max\{\cdot,\cdot\}$ is an element-wise operator.

To mask the missing values and iteratively completing tensor $\boldsymbol{\mathfrak{X}}$, $\boldsymbol{\mathfrak{X}}$ is updated as follows:

$$\boldsymbol{\mathfrak{X}} \leftarrow \boldsymbol{\mathfrak{T}} + \boldsymbol{\Omega}^c \circledast [\![\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_N]\!]. \tag{2.15}$$

Algorithm 1: The proposed framework MAST

Input: $\mathcal{T}$, $\mathbf{\Omega}$, $\{\widetilde{\mathbf{A}}_n\}_{n=1}^N$, R, $\{\alpha_n\}_{n=1}^N$, μ, η, $\eta_{\max}$, ρ, tol;
Output: $\mathcal{X}$, $\{\mathbf{A}_n\}_{n=1}^N$

Initialize $\mathbf{A}_n^{(0)} = \widetilde{\mathbf{A}}_n$, $\mathbf{A}_n^{(1)} = rand(\mathbf{d_n}, R)$, $\mathbf{Z}_n = \mathbf{Y}_n = \mathbf{0}$;
repeat
  $\eta = \min\{\eta * \rho, \eta_{\max}\}$ (accelerate optimization process);
  for $n = 1 : N$ **do**
    Update $\mathbf{A}_n^{(0)}$ and $\mathbf{A}_n^{(1)}$ using Equation (2.13);
  Update $\mathcal{X}$ using Equation (2.15);
  for $n = 1 : N$ **do**
    Update $\mathbf{Z}_n$ using Equation (2.14);
    Update $\mathbf{Y}_n$ using Equation (2.16);
until $\|\mathcal{X}_{pre} - \mathcal{X}\|_F / \|\mathcal{X}_{pre}\|_F < tol$;

Finally, $\mathbf{Y}_n$ is updated as follows:

$$\mathbf{Y}_n \leftarrow \mathbf{Y}_n + \eta(\mathbf{Z}_n - \mathbf{A}_n), \quad n = 1, 2, \ldots, N. \tag{2.16}$$

By using this optimization scheme, we can get the decomposition matrices and completed tensor $\mathcal{X}$ simultaneous. The entire optimization process of MAST is summarized in Algorithm 1. The termination criterion is that the relative changing of tensor $\mathcal{X}$ in two contiguous iterations is smaller than the tolerance.

2.3.3 Temporal Multi-Aspect Streaming Tensor Completion

In some real-world applications, time is one mode of each tensor in a multi-aspect streaming tensor sequence. Take a recommender system structured as a $user \times movie \times time$ tensor as an example. If a new user or a new item joins the system at time T, their past-time information would not exist during $t \in (0, T)$. This information shortage renders the snapshot tensor degenerated to a ladder-type structure shown in Figure 2.3. If we treat this structure as a combination of slices $\{\mathcal{X}^{(T)}\}$, then the original multi-aspect streaming tensor sequence can be reconstructed using the incremental tensor slices with zero-paddings. We call these slices $\{\mathcal{X}^{(T)}\}$ temporal multi-aspect streaming tensor sequence. This special structure provides us a way to improve MAST model for a better effectiveness by alleviating the quantity of substitution on missing entries.

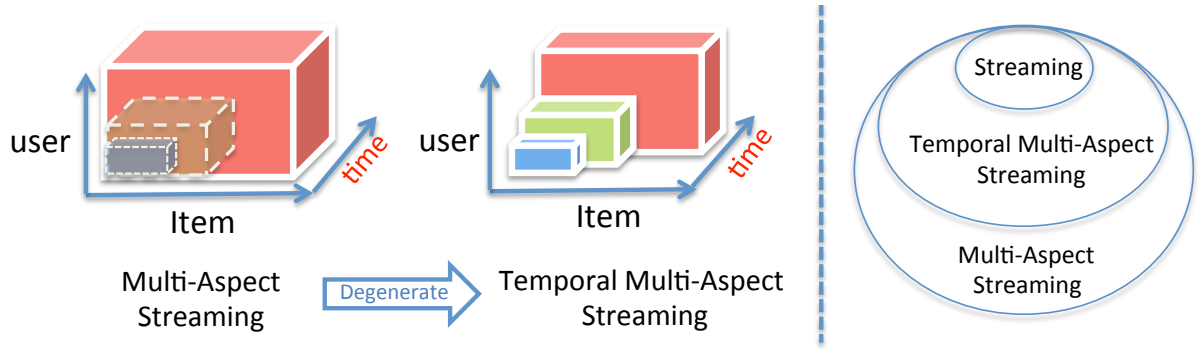


Figure 2.3: Degeneration of multi-aspect streaming tensor.

2.3.3.1 *Connection of Different Dynamic Patterns*

Figure 2.3 depicts the relationships between different dynamic patterns. The temporal multi-aspect steaming is a special scenario. If the sizes of the tensor slices are the same at each time step, it will further degenerate into the traditional streaming tensor situation. Although the general framework MAST can be directly applied to the temporal case, the ladder-type structure provides extra information that could help.

We propose to reduce the quantity of substitution on missing entries to improve one of the key manipulations of MAST, i.e., tensor substitution. Different from DTD, because of data missing, the substitution of $\mathcal{X}^{(T-1)}$ at time T is composed of two parts: (1) using former-step predicted values to impute the missing entries; (2) using fitting values to substitute the existed entries. The substitution of existed part often has much smaller error and usually provides more convincing information than the prediction of missing part. As the degeneration process naturally centralizes the existed data in the ladder-type object, all of the extra parts can be treat as missing data and their substitution errors are easy to reduce. Thus, besides forgetting factor μ, this special structure paves another way to reduce the accumulated error in MAST framework towards effectively capturing fast and dramatically changed subspaces.

2.3.3.2 *Temporal Multi-Aspect Streaming CP Completion Algorithm*

Inspired by the coupled tensor factorization [58], we tailor the general MAST framework and use Figure 2.4 to illustrate the main idea of the variation model T-MAST. Let $\mathcal{X}^{(T)}$ denote the tensor slice

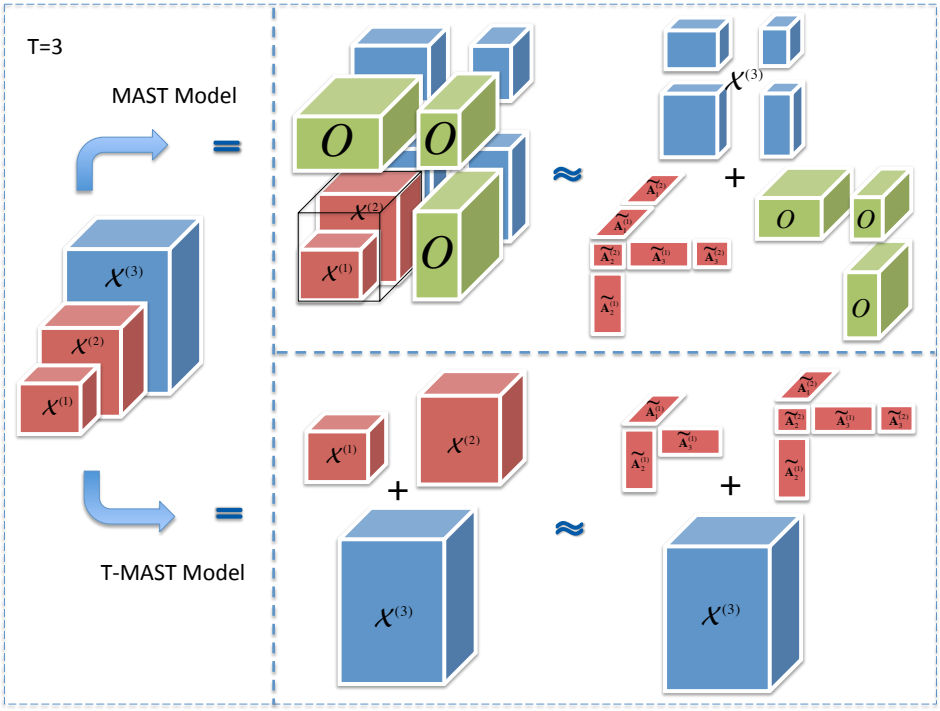


Figure 2.4: An illustration of T-MAST comparing with MAST.

arrives at time T. $\mathcal{T}^{(T)}$ represents its observed entries. $\{[\![\widetilde{\mathbf{A}}_1^{(t)}, \begin{bmatrix}\widetilde{\mathbf{A}}_2^{(1)}\\ \vdots \\ \widetilde{\mathbf{A}}_2^{(t)}\end{bmatrix}, \ldots, \begin{bmatrix}\widetilde{\mathbf{A}}_N^{(1)}\\ \vdots \\ \widetilde{\mathbf{A}}_N^{(t)}\end{bmatrix}]\!]\}_{(1\leq t\leq T-1)}$ is the set of CP decompositions of the recovered tensor slices $\{\mathcal{X}^{(t)}\}_{(1\leq t\leq T-1)}$ at time $T-1$. We use $\mathbf{A}_n^{(T)}, n=1,\ldots,N$ to represent the incremental part of the decomposition matrix on mode-n at time T and use $\mathbf{A}_n^{(t)}$ $(1 \leq t \leq T-1)$ to represent the updated result of $\widetilde{\mathbf{A}}_n^{(t)}$ at current timestamp T. At each time step T, the problem is converted to how to recover $\mathcal{X}^{(T)}$ based on the decompositions of former recovered slices $\{\mathcal{X}^{(t)}\}_{(1\leq t\leq T-1)}$ and get the updated CP decomposition set $\{[\![\mathbf{A}_1^{(t)}, \begin{bmatrix}\mathbf{A}_2^{(1)}\\ \vdots \\ \mathbf{A}_2^{(t)}\end{bmatrix}, \ldots, \begin{bmatrix}\mathbf{A}_N^{(1)}\\ \vdots \\ \mathbf{A}_N^{(t)}\end{bmatrix}]\!]\}_{(1\leq t\leq T)}$.

Comparing to MAST, which requires zero-paddings to redefine a multi-aspect sequence shown in Figure 2.4, for T-MAST, rather than reconstructing it as a whole tensor, we directly treat this ladder type object as coupled tensor slices and use the decomposition of each slice to substitute it. Since newly added zero-padding tensors and the block operation of $\mathcal{X}^{(T)}$ has no influence on

the current update, the difference between MAST and T-MAST models is how we substitute the former existed slices. Owing to the slice-based substitution in T-MAST model, the substitution of the former zero-padding parts can be omitted compared with the tensor-based substitution used in MAST model. Thus, it successfully reduces the proportion of missing data substitution as described before. The modified loss function is defined as follows:

$$\begin{aligned}\mathcal{L} = &\left\|\mathcal{X}^{(T)} - \left[\!\left[\mathbf{A}_1^{(T)}, \mathbf{A}_2, \mathbf{A}_3, \ldots, \mathbf{A}_N\right]\!\right]\right\|_{\mathrm{F}}^2 + \sum_{n=1}^{N} \alpha_n \left\|\mathbf{A}_n\right\|_* \\ &+ \sum_{t=1}^{T-1} \mu_t \left\| \left[\!\left[\widetilde{\mathbf{A}}_1^{(t)}, \begin{bmatrix} \widetilde{\mathbf{A}}_2^{(1)} \\ \vdots \\ \widetilde{\mathbf{A}}_2^{(t)} \end{bmatrix}, \ldots, \begin{bmatrix} \widetilde{\mathbf{A}}_N^{(1)} \\ \vdots \\ \widetilde{\mathbf{A}}_N^{(t)} \end{bmatrix}\right]\!\right] - \left[\!\left[\mathbf{A}_1^{(t)}, \begin{bmatrix} \mathbf{A}_2^{(1)} \\ \vdots \\ \mathbf{A}_2^{(t)} \end{bmatrix}, \ldots, \begin{bmatrix} \mathbf{A}_N^{(1)} \\ \vdots \\ \mathbf{A}_N^{(t)} \end{bmatrix}\right]\!\right] \right\|_{\mathrm{F}}^2,\end{aligned} \tag{2.17}$$

where $\mathbf{A}_n = \begin{bmatrix} \mathbf{A}_n^{(1)} \\ \vdots \\ \mathbf{A}_n^{(T)} \end{bmatrix},\ n = 1, \ldots, N.$

2.3.3.3 Optimization Scheme

By comparing Equation (2.17) with Equation (2.11), we can easily find that their first and second terms are correspondingly equivalent when dealing with the temporal-mode-involved scenario. The different is the last term. Hence, we can still use ADMM algorithm to solve the above optimization problem and the only difference lies in the optimization of $\{\mathbf{A}_n\}_n$. For the sake of brevity, we omit other repetitive derivation and only focus on the update of matrices $\{\mathbf{A}_n^{(t)}\}_{t,n}$. For each non-temporal mode $n = 2, \ldots, N$, we calculate the partial derivatives of matrices $\{\mathbf{A}_n^{(t)}\}$, $t = 1, ..., T$, and define intermediate recursive matrix sequences $\{\mathbf{B}_n^{(t)}\}$, $\{\mathbf{D}_n^{(t)}\}$, $\{\mathbf{P}_n^{(t)}\}$, and $\{\mathbf{Q}_n^{(t)}\}$ to update them in each iteration. Assume $\widetilde{\mathbf{A}_k^{(T)}} = 0$, $\mathbf{P}_k^{(T)} = \sum_{j=1}^{T} \widetilde{\mathbf{A}_k^{(j)}}^{\top} \mathbf{A}_k^{(j)}$, and $\mathbf{Q}_k^{(T)} = \sum_{j=1}^{T} \mathbf{A}_k^{(j)\top} \mathbf{A}_k^{(j)}$, $k = 2, \ldots, N$. For any $k = 2 \ldots, N$, we have:

$$\mathbf{P}_k^{(t)} = \mathbf{P}_k^{(t+1)} - \widetilde{\mathbf{A}_k^{(t)}}^{\top} \mathbf{A}_k^{(t)}; \quad \mathbf{Q}_k^{(t)} = \mathbf{Q}_k^{(t+1)} - \mathbf{A}_k^{(t)\top} \mathbf{A}_k^{(t)}, \ \ t = T-1, \ldots, 1.$$

Based on $\{\mathbf{P}_n^{(t)}\}$ and $\{\mathbf{Q}_n^{(t)}\}$, if we define $\mathbf{D}_n^{(T+1)} = 0,\ \mu_T = 1$, and $\mathbf{B}_n^{(T+1)} = 0$, we have:

$$\begin{cases} \mathbf{D}_n^{(t)} = \mathbf{D}_n^{(t+1)} + \mu_t(\widetilde{\mathbf{A}_1^{(t)}}^\top \mathbf{A}_1^{(t)}) \circledast (\mathbf{P}_k^{(t)})^{\circledast k\neq 1,n}, \\ \mathbf{B}_n^{(t)} = \mathbf{B}_n^{(t+1)} + \mu_t(\mathbf{A}_1^{(t)\top} \mathbf{A}_1^{(t)}) \circledast (\mathbf{Q}_k^{(t)})^{\circledast k\neq 1,n}, \ \ t = T, \ldots, 1. \end{cases}$$

Then we can get the update rules for $\{\mathbf{A}_n^{(t)}\}$ as follows:

$$\mathbf{A}_n^{(t)} \leftarrow \frac{\mathbf{C}_n^{(t)} + \widetilde{\mathbf{A}_n^{(t)}}\mathbf{D}_n^{(t)} + \eta\mathbf{Z}_n^{(t)} + \mathbf{Y}_n^{(t)}}{\mathbf{B}_n^{(t)} + \eta\mathbf{I}_R},\ t = T, \ldots, 1,\ n = 2, \ldots, N, \tag{2.18}$$

where $\{\mathbf{Y}_n^{(t)}\}_{t,n}$ are the Lagrange matrix multipliers. $\{\mathbf{C}_n^{(t)}\}_t$ are the block sub-matrices of $\mathbf{C}_n \triangleq \mathbf{X}_{(n)}^{(T)}\left[\mathbf{A}_1^{(T)} \odot \left(\mathbf{A}_k\right)^{\odot k\neq 1,n}\right]$ based on the size of $\{\mathbf{A}_n^{(t)}\}_t$.

For each mode n, only $2(N-1)R^2$ extra space is needed comparing to the MAST model if we follow the updating order below:

$$\begin{aligned} \mathbf{C}_n \ \rightarrow\ & \{\mathbf{P}_k^{(T)}\}_{k\in[2,N]} \ \rightarrow\ \{\mathbf{Q}_k^{(T)}\}_{k\in[2,N]} \ \rightarrow\ \mathbf{D}_n^{(T)} \ \rightarrow\ \mathbf{B}_n^{(T)} \ \rightarrow \mathbf{A}_n^{(T)} \\ \rightarrow\ & \{\mathbf{P}_k^{(T-1)}\}_{k\in[2,N]} \rightarrow \{\mathbf{Q}_k^{(T-1)}\}_{k\in[2,N]} \rightarrow \mathbf{D}_n^{(T-1)} \rightarrow \mathbf{B}_n^{(T-1)} \rightarrow \mathbf{A}_n^{(T-1)} \\ \rightarrow \ldots \rightarrow\ & \{\mathbf{P}_k^{(1)}\}_{k\in[2,N]} \ \rightarrow\ \{\mathbf{Q}_k^{(1)}\}_{k\in[2,N]} \ \rightarrow\ \mathbf{D}_n^{(1)} \ \rightarrow\ \mathbf{B}_n^{(1)} \ \rightarrow \mathbf{A}_n^{(1)}. \end{aligned}$$

The temporal mode ($n = 1$) could be updated by:

$$\mathbf{A}_1^{(t)} \leftarrow \frac{\mu_t\mathbf{C}_n^{(t)} + \eta\mathbf{Z}_1^{(t)} + \mathbf{Y}_1^{(t)}}{\mu_t(\mathbf{Q}_k^{(t)})^{\circledast k\neq 1} + \eta\mathbf{I}_R},\ t = T, \ldots, 1,\ \mu_T = 1, \tag{2.19}$$

where $\mathbf{C}_n^{(T)} = \mathbf{X}_{(1)}^{(T)}\left(\mathbf{A}_k\right)^{\odot k\neq 1}$, $\mathbf{C}_n^{(t)} = (\mathbf{P}_k^{(t)})^{\circledast k\neq 1}$. The rest matrices $\{\mathbf{Y}_n^{(t)}\}$, $\{\mathbf{Z}_n^{(t)}\}$ and tensor $\boldsymbol{\mathcal{X}}^{(T)}$ are updated similarly as MAST.

From model perspective, although T-MAST can not handle the general multi-aspect streaming situation as MAST, both of them can be applied to the traditional streaming case. Not surprisingly, comparing Equations (2.11) and (2.17), using the traditional streaming setting, these two models are equivalent to each other.

2.4 Experiments

In this section, we empirically evaluate the performance of the proposed framework MAST. Three major aspects are analyzed.

Q1: How effective is MAST compared with static and dynamic CP completion methods on different real-world datasets with different dynamic variation patterns?

Q2: How efficient is MAST compared with the state-of-the-art methods on datasets with different length of time segments?

Q3: What is the influence of the forgetting factor μ on the general model MAST?

2.4.1 Datasets and Tasks

To evaluate the validity of MAST, we apply it to four datasets with different applications. The first three are recommendation tasks, and the last one is an information diffusion application to demonstrate the broader usage of the proposed framework. Their basic information is shown in Table 2.2.

Twitter Topic [59] (Recommendation): It is a third-order tensor with binary entries of size $500(user) \times 500(expert) \times 20(topic)$. Experts represent the high-quality content producers of 20 topics. It starts from $50 \times 50 \times 20$ and increases 2% of total users and experts at each time step. The task is to recommend the personalized expert of each topic to each user.

Youtube[2] **[60] (Link Prediction)**: It includes $1{,}066$ users which have the most interactions among original 15,088 users sharing five interactions, including contact, co-contact, co-subscription, co-subscribed, and favorite networks. It starts from $100(user) \times 100(user) \times 5(interaction)$ and increases 2% of total users in each timestamp. The task is to predict missing links at each time step.

MovieLens[3] (Recommendation): It is a benchmark dataset for movie recommendation structured as a temporal multi-aspect streaming tensor sequence of size $943(user) \times 1{,}682(item) \times 31(week)$. Instead of predicting concrete movie rates, the task is to predict what movies a user may rate.

Twitter Hashtag [42] (Information Diffusion): This is a spatial-temporal dataset structured as a

[2] http://socialcomputing.asu.edu/datasets/YouTube

[3] http://movielens.umn.edu

Table 2.2: Characteristics of the four multi-aspect streaming tensor completion datasets.

	Total Size	Initial Size	Increased Modes
Twitter Topic	$500 \times 500 \times 20$	$50 \times 50 \times 20$	1,2
Youtube	$1066 \times 1066 \times 5$	$100 \times 100 \times 5$	1,2
MovieLens	$943 \times 1682 \times 31$	$57 \times 983 \times 1$	1,2,3
Twitter Hashtag	$100 \times 1000 \times 249$	$57 \times 1000 \times 1$	1,3

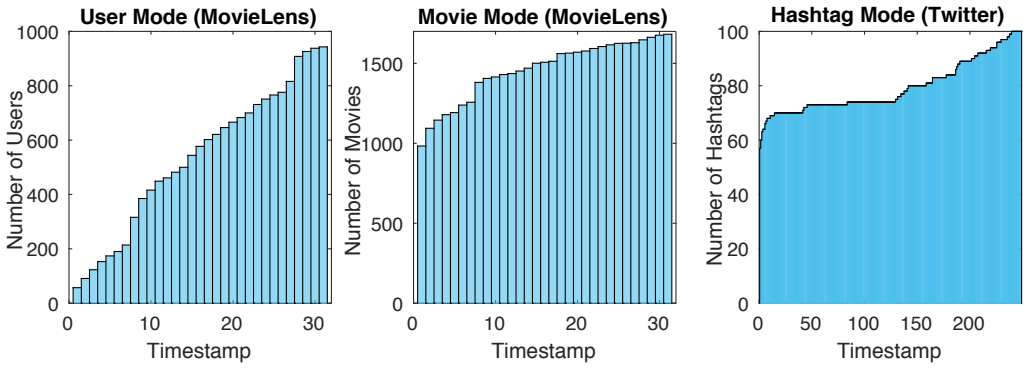


Figure 2.5: Incremental patterns of different datasets.

temporal multi-aspect streaming tensor sequence of size $100(hashtag) \times 1000(city) \times 249(day)$. Each binary entry indicates whether the corresponding hashtag emerges at the corresponding city on a specific day or not. Hashtag mode is increased as time goes. The task is to predict the emergence of previously appeared hashtags in each city at each time step.

The MovieLens and Twitter Hashtag follow the natural evolvement of a ladder-type increment. To give a better understanding of the incremental process, we depict the sectional views of the user and item modes of MovieLens, as well as the hashtag mode of Twitter Hashtag in Figure 2.5. The incremental pattern of Twitter Hashtag dataset is relatively gentle comparing with MovieLens.

2.4.2 Baseline Methods

Three state-of-the-art CP completion methods are employed including two categories, i.e., static and dynamic algorithms.

- Static CP-ALS [61]: It is a traditional CP-based method also called EM-ALS [62]. To enhance the learning process, the former-step decomposition is utilized as the initialization for the

current step. We implement it with the help of [63].

- Static TNCP [51]: It is one of the state-of-the-art static completion methods. It employs trace norm constraints optimized by ADMM algorithm. Similar initialization approach is employed to accelerate the convergence.

- OLSTEC [47]: It is a state-of-the-art streaming tensor completion method. We extend it for multi-aspect streaming tensor completion by splitting the multi-aspect incremental process into several streaming ones and update them in random order in each iteration until it converges.

- T-MAST: This is a variation of MAST tailored towards temporal multi-aspect streaming tensor sequence. It cannot be applied to the Twitter Topic and Youtube.

2.4.3 Experimental Setup

Following widely used settings [47, 46], for all datasets, we randomly cover a fixed percentage of data and consider the remaining entries as observed information. The hidden data is used as ground truth. To further study the impact of the missing ratio, we vary it as $\{20\%,50\%,80\%\}$. To evaluate the performance, a widely used metric is employed for all tasks, i.e., running-average Area Under Curve (RA-AUC). For the first two datasets, we focus on the incremental part $\mathcal{X}^{(t)} \backslash \mathcal{X}^{(t-1)}$ at time step t to avoid double counting and calculate the Area Under Curve (AUC) score of each topic slice or interaction slice based on the prediction. The average AUC at time t among topics or interaction slices is denoted as AUC_t. For the last two datasets which are two multi-aspect streaming tensor sequences, AUC_t is calculated using the prediction of missing data in slice $\mathcal{X}^{(t)}$. The running-average AUC score is defined as: RA-AUC $= \frac{1}{T}\sum_{t=1}^{T} \text{AUC}_t$, where T is the total number of time steps. The efficiency is measured by average running time defined as $\frac{1}{T}\sum_{t=1}^{T} \text{RT}_t$, where RT_t is the running time at time step t.

Parameter Setting: We set the tolerance rate to 10^{-5}, the maximum number of iteration to 500 for all the algorithms. By testing the performance of static methods on whole four datasets using 10 different ranks varying from 5 to 50, we set $R = 10$ in all the experiment considering of both

Table 2.3: Performance in terms of RA-AUC metric on the two general multi-aspect streaming tensor completion tasks. (Results of the best dynamic and static methods are highlighted.)

Dataset		Twitter Topic			Youtube		
Missing Percentage		20%	50%	80%	20%	50%	80%
Static	CP-ALS	0.8394	0.7810	0.6238	**0.9390**	**0.9209**	0.8609
	TNCP	**0.8443**	**0.7936**	**0.6661**	0.9384	0.9199	**0.8659**
Dynamic	OLSTEC	0.6670	0.5035	0.5028	0.8601	0.7315	0.7225
	MAST	**0.8392**	**0.7714**	**0.6493**	**0.9326**	**0.9141**	**0.8618**
	T-MAST	$N.A.$	$N.A.$	$N.A.$	$N.A.$	$N.A.$	$N.A.$

Table 2.4: Performance in terms of RA-AUC metric on the two temporal multi-aspect streaming tensor completion tasks. (Results of the best dynamic and static methods are highlighted.)

Dataset		MovieLens			Twitter Hashtag		
Missing Percentage		20%	50%	80%	20%	50%	80%
Static	CP-ALS	0.9304	0.9057	0.8605	0.9257	0.9236	0.9133
	TNCP	**0.9344**	**0.9139**	**0.8790**	**0.9258**	**0.9238**	**0.9137**
Dynamic	OLSTEC	0.7048	0.6550	0.6227	0.7449	0.6716	0.6314
	MAST	0.8393	0.8354	0.8099	0.9160	0.9019	0.8641
	T-MAST	**0.9294**	**0.9023**	**0.8586**	**0.9233**	**0.9153**	**0.8881**

accuracy and speed for fair comparison. In the implementation of our proposed method, default parameters are set as $\alpha_n = \frac{1}{10N}$ $n = 1, \ldots, N$, $\eta = 10^{-4}$, $\rho = 1.05$ and $\eta_{\max} = 10^6$. The forgetting factors are fine-tuned according to the missing ratio f (usually $\mu = \mu_t = 1 - f$ $(\forall\, t > 0)$). For baselines, we follow the suggestions of original papers to set parameters. The initial completion and warm start matrices are calculated using TNCP method for two reasons: (1) MAST framework degenerates to static TNCP when $T = 1$. (2) For fair comparison, we use the same best warm start at initial time for all methods. All experimental results are the arithmetic average of five runs and are ran on a Dell OptiPlex 9030 i7-16GB desktop with MATLAB R2016b.

2.4.4 Evaluation of Effectiveness

Table 2.3 and Table 2.4 show the performance of MAST and baselines on different datasets in terms of RA-AUC. Three main conclusions are observed.

First, MAST has commensurate performance comparing with the two static models and higher

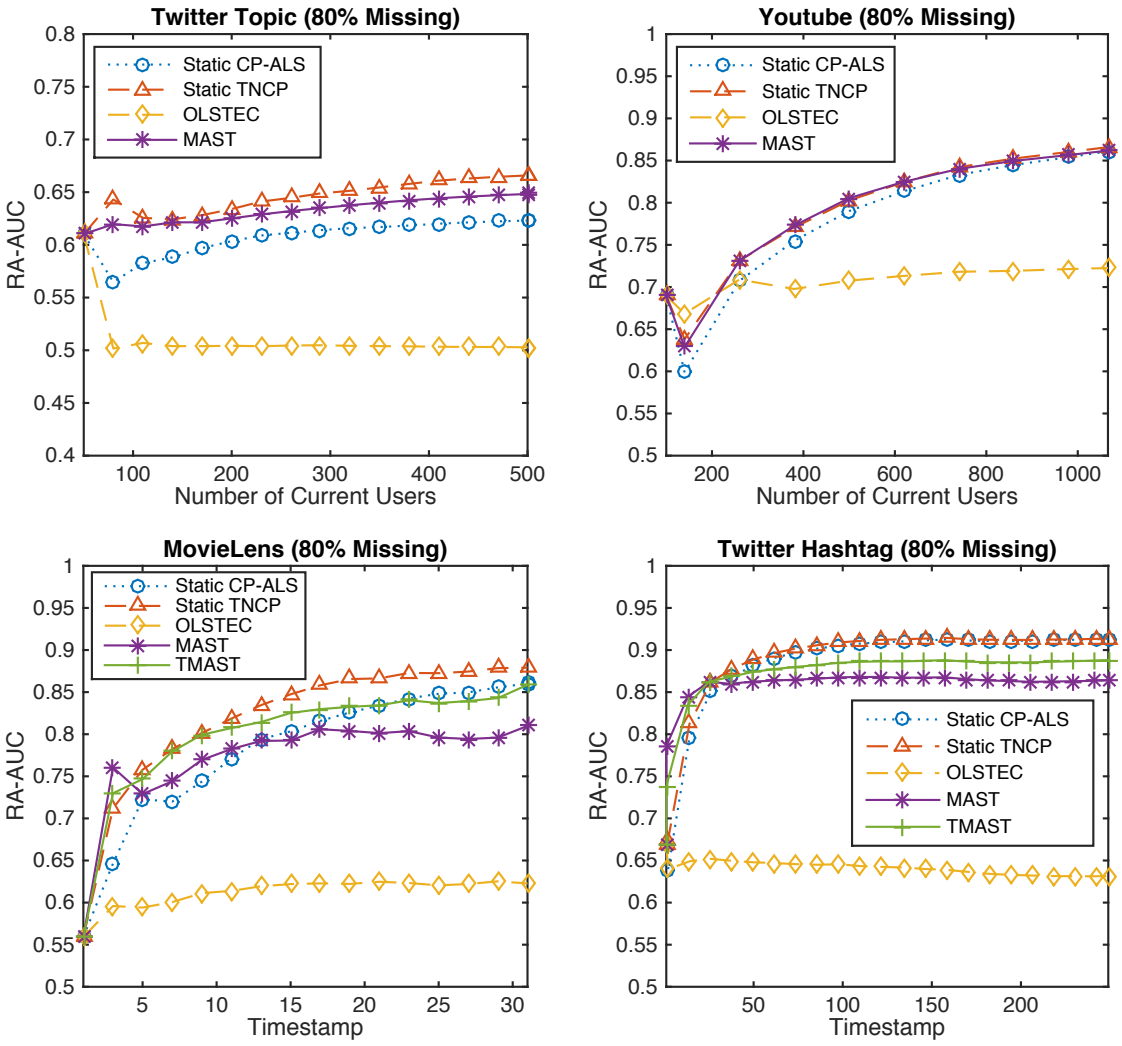


Figure 2.6: Performance of different methods on the four datasets with different timestamps.

performance than dynamic baseline method OLSTEC with different percentages of random missing entries on the first two datasets. To illustrate the performance of different models over time, we display the RA-AUC variation curve of MAST compared with baselines on four datasets with 80% missing entries in Figure 2.6. The results show that, with time goes, the increase of users leads to a fluctuant increasing RU-AUC for all the methods except OLSTEC. MAST has comparable performance to the static baselines and has higher accuracy than OLSTEC.

Second, on both MovieLens and Twitter Hashtag, which have different dynamic patterns shown in Figure 2.5, the variation model T-MAST outperforms MAST. This result empirically validates our analysis in Section 3.3 that the ladder-type structure of temporal multi-aspect streaming pattern could help to reduce the quantity of substitution on missing entries thereby improving general

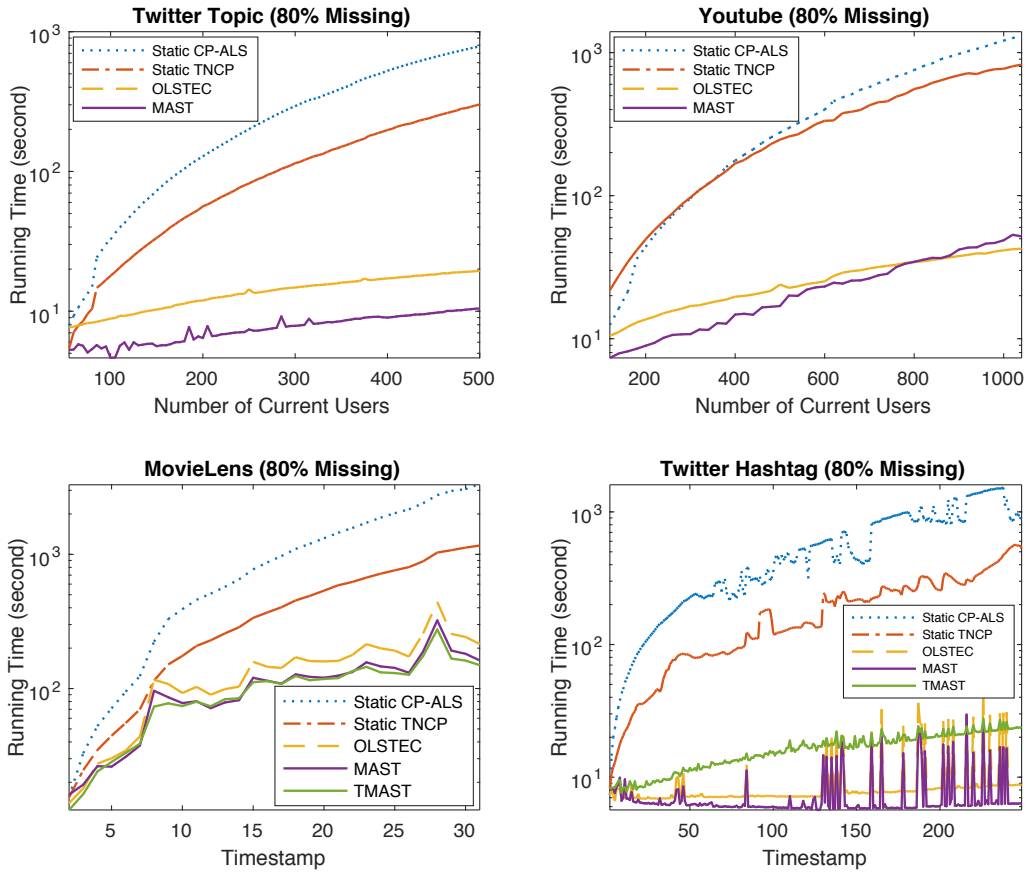


Figure 2.7: Running time (in logarithmic scale) of all methods.

MAST framework. Besides, as can be seen from the results, the difference between T-MAST and MAST on MovieLens is larger than the one on Twitter Hashtag. This is because MovieLens dataset has fewer snapshots and sharper changes on the size of tensor slices than Twitter Hashtag as shown in Figure 2.5. The dramatically increased ladder-type structure leads to a larger ratio of missing data substitution at each time step. This further results in a hysteresis effect for capturing characteristics of the new data during the update.

Third, MAST shows strong stability on different datasets of different dynamic patterns. Furthermore, for different ratios of missing data and different length of time steps, our proposed framework retains comparable performance with static models. Theses results demonstrate the capability of our model in capturing low-rank subspace of dynamically changed tensor objects.

Table 2.5: Average running time of different methods on the two general multi-aspect streaming tensor completion tasks.

Dataset		Twitter Topic			Youtube		
Missing Percentage		20%	50%	80%	20%	50%	80%
Static	CP-ALS	305.20	307.42	302.85	504.77	502.73	500.72
	TNCP	110.73	118.69	117.92	371.33	368.56	365.67
Dynamic	OLSTEC	14.042	14.150	13.903	29.592	27.272	27.216
	MAST	**8.7937**	**8.3523**	**7.6398**	**28.270**	**26.443**	**25.626**
	T-MAST	$N.A.$	$N.A.$	$N.A.$	$N.A.$	$N.A.$	$N.A.$

2.4.5 Evaluation of Efficiency

To study the efficiency of the proposed framework, we compare the average running time of all five models shown in Table 2.5 and Table 2.6. For better visualization, Figure 2.7 displays the computation time (RT_t) in logarithmic scale as a function of timestamps or user numbers on four datasets with 80% missing. From these results, we can come to the conclusion that (1) For general multi-aspect streaming tensor, MAST model takes much less running time than static models and outperforms OLSTEC. (2) T-MAST model shows comparable efficiency with MAST on short-term datasets. Although it sacrifices the time complexity to some degree on long-term datasets for the sake of higher effectiveness, it is still significantly faster than static models. If we assume $R \ll I_n,\ n = 1, \ldots, N$, and define $S^t = \prod_{n=1}^{N} I_n^t$, where I_n^t is size of the N^{th} mode of slice $\boldsymbol{\mathcal{X}}^{(t)}$ in a temporal multi-aspect streaming tensor sequence, the complexities per iteration of all three dynamic models at time t would be $\mathcal{O}((N+1)R(S^t - S^{(t-1)}))$ comparing to $\mathcal{O}((N+1)RS^t)$ of two static models. Moreover, a dramatic change in the size of the tensor will cause a violent fluctuation on the completion time. For instance, the structures of Youtube and Twitter Topic datasets changed more steadily than MovieLens and Twitter Hashtag datasets resulting in less violent fluctuations on the running-time curves. If we focus on one dataset such as Twitter Hashtag, the time curve fluctuates more acutely when the size of the newly slice increases sharply.

2.4.6 Influence of Forgetting Factor

To investigate the effect of forgetting factor μ on our proposed framework, we vary it from 0 to 1 with a step size 0.05 and compare the performance of MAST model on Twitter Topic and

Table 2.6: Average running time of different methods on the two temporal multi-aspect streaming tensor completion tasks.

Dataset		MovieLens			Twitter Hashtag		
Missing Percentage		20%	50%	80%	20%	50%	80%
Static	CP-ALS	1189.9	1223.3	1170.2	257.16	468.13	582.38
	TNCP	509.87	511.34	456.59	172.01	194.78	193.59
Dynamic	OLSTEC	144.18	143.59	140.81	9.2601	9.6796	9.3625
	MAST	118.13	113.74	110.72	**8.9186**	**8.5487**	**7.0420**
	T-MAST	**113.65**	**105.29**	**103.56**	19.1785	18.6397	16.3826

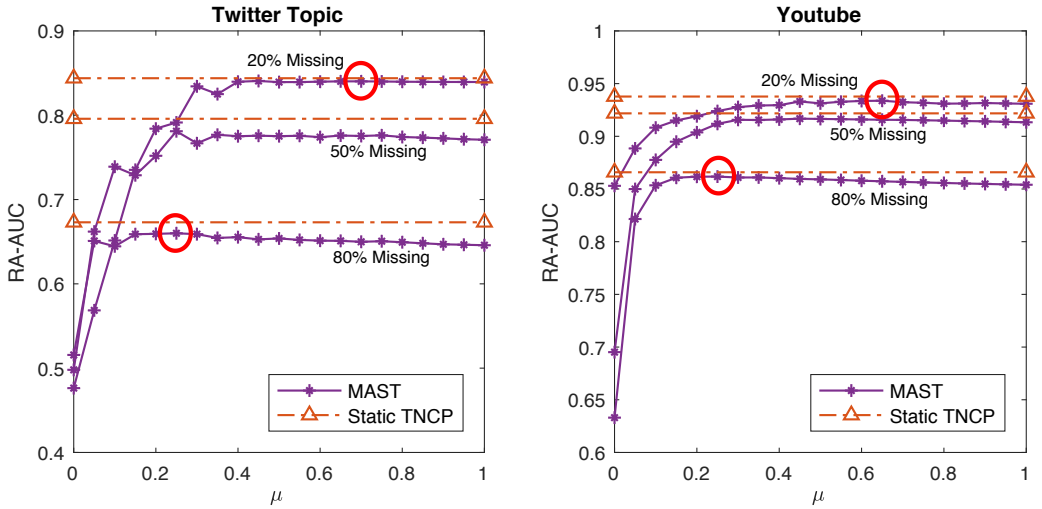


Figure 2.8: Impact of forgetting factor μ on MAST with different missing percentages.

Youtube datasets. Results shown in Figure 2.8 demonstrate that with the decreasing of forgetting factor, the performance of MAST increases at first and then decreases. With the increase of the missing ratio, the turning point becomes closer to 0, which inspires us to choose a smaller μ to alleviate the substitution error. In sum, these observations illustrate that: (1) A suitable shrinkage on the approximately substituted tensor could alleviate the previous fitting error and result in better completion effectiveness. (2) The higher the missing ratio is, the smaller the forgetting factor we should choose to alleviate the hysteresis effect of our framework in capturing the low-rank subspace for multi-aspect streaming tensor completion.

2.5 Related Work

The related work can be categorized into two main topics as follows:

Dynamic Tensor Factorization. Tensor factorization methods have received widespread concerns and achievements under static setting [52, 55]. Increasingly massive amount of real-world dynamic data nowadays requires an extensive concern on the problem of dynamic tensor factorization [47]. Nion and Sidiropoulos [64] proposed two adaptive PARAFAC algorithms adopting the recursive least square and simultaneous diagonalization tracking methods to solve the online third-order tensor factorization. Phan et al. [50] partitioned a large-scale tensor into small grids and proposed a grid-based scalable tensor factorization method which could also be used in dynamic tensor factorization. Zhou et al. [53] proposed an accelerated online algorithm that can track the CP decompositions of incremental N^{th}-order tensors. However, all of them are not directly applicable to multi-aspect streaming situations and the completion task. Kasai [47] substituted the stochastic gradient descent method with recursive least square method and improved the algorithm proposed by Mardani et al. [46] focusing on the problem of subspace learning and imputation for streaming tensors. Besides CP decomposition, dynamic tucker decomposition methods were also proposed [65, 66, 67]. Some of them were not only focusing on one-mode increasing condition [68, 69], but also giving possible solutions for all-mode incremental update using matrix-based online methods such as incremental SVD [70] without considering about missing entries. Finally, it is worth to mention that though not a factorization method, the histogram-based approach [71] conducted on multi-aspect streaming tensor analysis can be treated as one of the pioneer researches on the multi-aspect streaming pattern.

Low-Rank Tensor Completion. Since the real-world multidimensional datasets are oftentimes raw and incomplete because of missing at random and limited permissions [42, 51], low-rank tensor completion problem has been attractive to researchers and practitioners in data mining, online learning, computer vision, signal processing, etc. Generalized from matrix cases, a wide range of approaches have been proposed such as trace-norm based methods [43, 45, 72, 73], factorization-based approaches [61, 74, 62], tensor completion with auxiliary information [75, 42, 58], and online tensor imputation [46, 47]. Although both theoretical analysis and various practical applications have been considered, to our best knowledge, no existing work has been conducted on the low-rank tensor completion with general multi-aspect streaming patterns.

2.6 Conclusions

In this work, we focus on the multi-aspect streaming tensor completion problem and propose an updatable CP completion framework MAST. The proposed framework can effectively capture the low-rank subspace of multi-aspect streaming tensor sequences so as to achieve the completion purpose. To further enhance the effectiveness, we also tailor the general framework toward a special case where time is one mode of the multi-aspect streaming tensors. By conducting experiments on various real-world recommendation tasks, we empirically validate the effectiveness and efficiency of our proposed framework.

3. COUPLED VARIATIONAL RECURRENT COLLABORATIVE FILTERING[1]

In this chapter, we explore the deep-learning solution to the streaming recommendation problem. We focus on the collaborative filtering setting and add deep-learning flavor to the conventional probabilistic factorization models. We model the evolution of the users' preference and items' popularity with the stochastic processes under the Bayesian treatment, which provides advanced recommendation performance and extra interpretability compared to existing shallow and deep recommender systems.

3.1 Introduction

Traditional work has been widely conducted in exploiting temporal dynamics to improve the recommendation performance under static settings [6, 76, 4, 5] or streaming settings [8, 9, 1, 12, 10, 13, 7]. Though great advances have been made, the capacity of these models are usually restricted while capturing complex data structures and require delicate statistical assumptions comparing with structured deep frameworks [77, 22].

Despite the remarkable revolution achieved recently in deep recommender systems either in static setting [24, 25, 26, 23, 27, 28] or streaming fashion [19, 20, 21], deep frameworks also have their own limitations. One of the most noticeable fact is that deep frameworks are usually deterministic approaches, which only output point estimations without taking the uncertainty into account. It significantly limits their power in modeling the randomness of the measurement noises [78] and providing predictions of the missing or unobserved interactions in recommender systems. As probabilistic approaches, especially Bayesian methods, provide solid mathematically tools for coping with the uncertainty, it motivates us to conduct streaming recommendations from the view of Deep Bayesian Learning (DBL) to conjoin the advantages of probabilistic models and deep learning models.

[1]This chapter is reprinted with permission from "Coupled Variational Recurrent Collaborative Filtering" by Qingquan Song, Shiyu Chang and Xia Hu, 2019. Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Copyright 2019 by ACM.

However, simply applying DBL on streaming recommendation is a non-trivial task due to the following challenges. First, coordinating the temporal dynamics is difficult given the continuous-time discrete-event recommendation process along with the protean patterns on both user and item modes. Second, the high velocity of streaming data requires an updatable model, which could expeditiously extract the prior knowledge from former time steps and effectively digest it for current predictions. Third, the DBL frameworks are usually expensive in terms of time and space complexities.

To tackle these aforementioned challenges, in this work, we propose to investigate how to conduct streaming recommendation by leveraging the advantages of both deep models and probabilistic processes. Specifically, we study: (1) How to model the streaming recommender system with an updatable probabilistic process? (2) How to incorporate deep architectures into the probabilistic framework? (3) How to efficiently learn and update the joint framework with streaming Bayesian inference? Through answering the three questions, we propose a Coupled Variational Recurrent Collaborative Filtering (CVRCF) Framework.

3.2 Coupled Variational Recurrent Collaborative Filtering Framework

The core of CVRCF is a dynamic probabilistic factor-based model that consists of four components. The first two formulate the user-item interactions and temporal dynamics, respectively. Each of them incorporates a probabilistic skeleton induced by deep architectures. The third component is a sequential variational inference algorithm, which provides an efficient optimization scheme for streaming updates. The last component allows us to generate rating predictions based on the up-to-date model.

3.2.1 Interaction Network

Factor-based models are widely adopted in recommendation modelings. They have shown a great success in multiple recommendation tasks [79]. Most of them follow the traditional matrix factorization setting, in which users and items are modeled as latent factors; and their interactions are defined as the linear combinations of these factors. However, such simple linear combinations

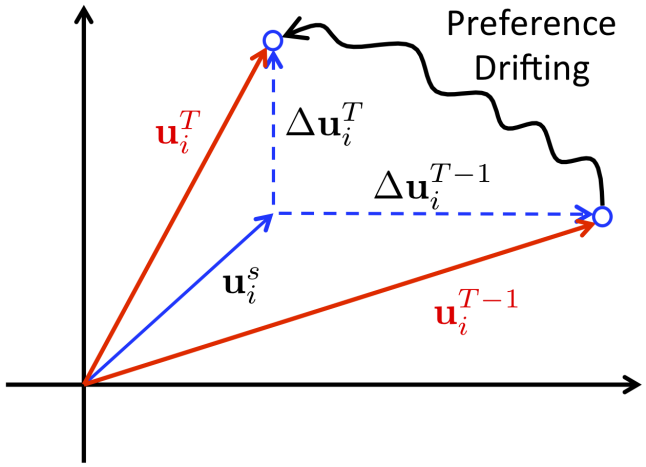


Figure 3.1: Temporal drifting of the a user's latent factor on two consecutive time steps.

are often insufficient to model complex user-item interactions [26]. Thus, we consider a deep probabilistic matrix factorization setting as follows:

$$x_{ij}^T|\mathbf{u}_i^T, \mathbf{v}_j^T, \sigma_{i,j,T}^2 \sim \mathcal{N}(f_1(\mathbf{u}_i^T, \mathbf{v}_j^T), f_2(\mathbf{u}_i^T, \mathbf{v}_j^T, \sigma_{i,j,T}^2)), \tag{3.1}$$

where both $f_1(\cdot)$ and $f_2(\cdot)$ are represented by deep neural networks. We represent the latent vectors of user i and item j at time step T as $\mathbf{u}_i^T$ and $\mathbf{v}_j^T$, respectively. The rating x_{ij}^T is modeled as a Gaussian random variable whose location and scale values are the output of the deep networks. The environmental noise $\sigma_{i,j,T}^2$ could either be predefined as a hyperparameter [79] or jointly learned. It is worth pointing out that we assume the variance of x_{ij}^T depends on both the latent vectors and the environmental noises, which is slightly different from the conventional probabilistic setting [79].

3.2.2 Temporal Drifting Process

The temporal dynamics of a recommender system depend on the drifting of users' preferences and item popularities [14, 15]. A user's tastes for a certain type of items may change over time while the popularity of an item may also vary with time goes by. To capture the inherent dynamics, we intend to encode the drifting processes into user and item latent factors based on three hypotheses:

- We assume the latent factors of both user and item can be decomposed as the combination

of a stationary term ($\mathbf{u}_i^s$) and a dynamic term ($\Delta\mathbf{u}_i^T$) [20]. The stationary factor captures the long-term preference, which varies slowly over time. The dynamic factor encodes the short-term changes, which evolves rapidly. An illustrative example is shown in Figure 3.1, where a user's dynamic factor evolves between two consecutive time steps, causing his preference drifted from $\mathbf{u}_i^{T-1}$ to $\mathbf{u}_i^T$. We assume the two factors are independent of each other for simplicity.

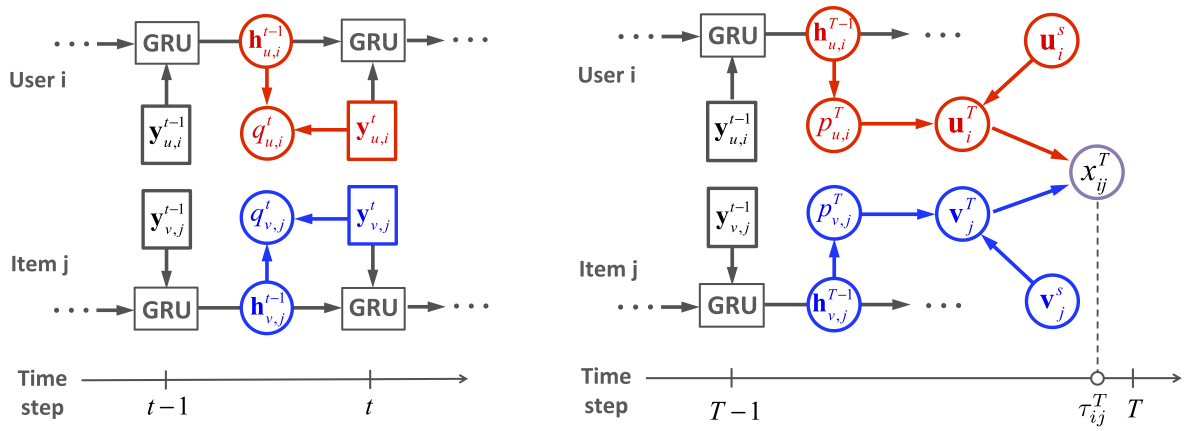
- The dynamic factors of a user or an item follows a Markov process [2]. The intuition of using a Markov process comes from the observation that the changing of a user's current preference could be highly affected by his former preference.

- The changing of latent factors of a particular user i (or item j) between two consecutive time steps $T-1$ and T depends on the time interval between the last events before these two time steps, which involves this user (or item), *i.e.*, $\Delta\tau_{u,i}^T = \tau_{u,i}^T - \tau_{u,i}^{T-1}$, where $\tau_{u,i}^{T-1}$ and $\tau_{u,i}^T$ denote the actual time of the two last interactions of user i before time step $T-1$ and T, respectively. Intuitively, the longer the interval is, the larger the drifting may happen. $\tau_{u,i}^T$ is defined to be equal to $\tau_{u,i}^{T-1}$ if no interactions happens between time step $T-1$ and T.

Upon these hypotheses, we model the evolution of hidden topics of a user (or an item), via spatiotemporal Gaussian priors, which is mathematically formulated as follows:

$$\begin{cases} \mathbf{u}_i^T = \mathbf{u}_i^s + \Delta\mathbf{u}_i^T, \\ \mathbf{u}_i^s \sim \mathcal{N}(\mathbf{0}, \sigma_U^2\mathbf{I}), \\ \Delta\mathbf{u}_i^T|\Delta\mathbf{u}_i^{T-1} \sim \mathcal{N}(\boldsymbol{\mu}_{u,i,T}, \boldsymbol{\Sigma}_{u,i,T}). \end{cases} \tag{3.2}$$

It is worth pointing out that only the users, which have interactions between time T and $T-1$, need to be considered here while factors of users who do not have interactions are assumed to be unchanged till their next interaction happens. We place the zero-mean spherical Gaussian prior on the stationary factors [79], where σ_U denotes the scale hyperparameter. For dynamic factors, the



(a) CVRCF sequential inference network.

(b) CVRCF prediction network.

Figure 3.2: Illustration of the inference and prediction networks of CVRCF.

kernel matrix $\mathbf{\Sigma}_{u,i,T}$ is defined as a diagonal matrix here for simplicity, *i.e.*, $\mathbf{\Sigma}_{u,i,T} \triangleq diag(\boldsymbol{\sigma}^2_{u,i,T})$. Motivated by the recent advances in deep kernel learning, which combines the non-parametric flexibility of kernel approaches with the structural properties of deep architectures [77], we further define the kernel as an output of a deep neural network $f_3(\cdot)$ to enhance its generality, *i.e.*, $\boldsymbol{\sigma}^2_{u,i,T} = f_3(\Delta\mathbf{u}_i^{T-1}, \Delta\tau_{u,i}^T)$.

Coping with the last two hypotheses, this spatiotemporal kernel takes $\Delta\mathbf{u}_i^{T-1}$, which represents the user's dynamic preference at last time step, as a spatial effect to decide the drifting uncertainty and it is stationary for temporal effect, which means $\mathbf{\Sigma}_{u,i,T}$ depends on the time internal $\Delta\tau_{u,i}^T$ rather than the concrete time $\tau_{u,i}^T$ and $\tau_{u,i}^{T-1}$. For a more unified representation, we can further define $\boldsymbol{\mu}_{u,i,T} = f_4(\Delta\mathbf{u}_i^{T-1}, \Delta\tau_{u,i}^T)$, where $f_4(\cdot)$ denotes a predefined deep neural network. The definition of the whole drifting prior obeys the Markov property for the discrete events on the continues timeline, which implies that the current state depends only on the former state. It is also applicable to employ other state dependency correlations and network structures. Similar prior with corresponding notations is defined for items.

3.2.3 Deep Sequential Variational Inference

The third component of the CVRCF framework is the inference model. It composites the two former components with a sequential Bayesian skeleton and associates them with the last prediction component for streaming recommendations.

3.2.3.1 Joint Distribution

The joint distribution of all observations up to time T and the latent factors is defined as follows:

$$
\begin{aligned}
& p(x^{\leq T}, \mathbf{U}^{\leq T}, \mathbf{V}^{\leq T}) = p(x^{\leq T}, \mathbf{U}^{s}, \mathbf{V}^{s}, \Delta\mathbf{U}^{\leq T}, \Delta\mathbf{V}^{\leq T}) \\
= \; & p(x^{\leq T}, \Delta\mathbf{U}^{\leq T}, \Delta\mathbf{V}^{\leq T} | \mathbf{U}^{s}, \mathbf{V}^{s}) p(\mathbf{U}^{s}) p(\mathbf{V}^{s}) \\
= \; & p(\mathbf{U}^{s}) p(\mathbf{V}^{s}) \Big[\prod_{t \leq T} p(x^{t} | x^{<t}, \Delta\mathbf{U}^{\leq t}, \Delta\mathbf{V}^{\leq t}, \mathbf{U}^{s}, \mathbf{V}^{s}) \\
& \qquad \times p(\Delta\mathbf{U}^{t}, \Delta\mathbf{V}^{t} | x^{<t}, \Delta\mathbf{U}^{<t}, \Delta\mathbf{V}^{<t}, \mathbf{U}^{s}, \mathbf{V}^{s}) \Big],
\end{aligned}
\tag{3.3}
$$

where $\mathbf{U}$ and $\mathbf{V}$ are the matrices of the latent factors for existing users and items.

Our goal is to infer the posterior distribution of latent factors for every t, *i.e.*, $p(\mathbf{U}^{t}, \mathbf{V}^{t} | x^{\leq t}), \forall t \leq T$. However, it is intractable for direct inferences based on the current model assumptions. To overcome this challenge, existing works usually focus on two types of approaches - Sequential Monte Carlo methods (SMC) [80] and Variational Inference methods (VI) [81]. The traditional sequential Bayesian updating usually uses SMC methods (*a.k.a.*, particle filtering) to deal with intractable target posterior distributions. Although this approach is very accurate when suitable proposal distributions and enough particle samples are presented, the sampling process is often too slow to apply to high dimensional and large-scale data [82]. On the other hands, the variational inference is much faster compared to SMC. However, the accuracy highly depends on the approximation distribution, especially in streaming settings [83]. Although there are hybrid models combine both algorithms together [84, 85], the computational complexity makes it prohibited for large-scale recommender systems. To trade-off the model scalability and accuracy, we consider the streaming variational inference framework [81] by leveraging deep neural networks as the variational approximator to obtain more flexible posteriors.

3.2.3.2 *Sequential Variational Inference Network*

Before introducing the deep architectures, we first assume the latent factors can be partitioned into independent units followed by the traditional mean-field approximation:

$$q(\Delta \mathbf{U}^{\leq T}, \Delta \mathbf{V}^{\leq T} | x^{\leq T}, \mathbf{U}^s, \mathbf{V}^s) = q(\Delta \mathbf{U}^{\leq T} | x^{\leq T}) q(\Delta \mathbf{V}^{\leq T} | x^{\leq T}), \tag{3.4}$$

where q denotes the approximated variational posterior. Further, each user (or item) is placed by a Gaussian variational posterior as follows:

$$q(\Delta \mathbf{u}_i^t | \Delta \mathbf{u}_i^{\leq t-1}, x_i^{\leq t}) = \mathcal{N}(\boldsymbol{\mu}_{u,i,t}^*, \boldsymbol{\Sigma}_{u,i,t}^*), \forall 1 \leq t \leq T, \tag{3.5}$$

where $\boldsymbol{\Sigma}_{u,i,t}$ is diagonal with the similar definition as the priors defined in Equation (3.2). $x_i^{\leq t}$ denotes all the interactions related to user i before time step t.

To infer the variational posterior, we propose a Coupled Variational Gated Recurrent Network structure (CVGRN) leveraging two variational Gated Recurrent Units (GRUs) for users and items, respectively. Figure 3.2a demonstrates the key idea of the proposed inference network. Blocks represent the inputs of two GRUs at different time steps. $q_{u,i}^t$ and $q_{v,j}^t$ represent the approximated posterior distribution $q(\Delta \mathbf{u}_i^t | \Delta \mathbf{u}_i^{\leq t-1}, x_i^{\leq t})$ and $q(\Delta \mathbf{v}_j^t | \Delta \mathbf{v}_j^{\leq t-1}, x_j^{\leq t})$, which are inferred based on the GRUs output states $\mathbf{h}_{u,i}^{t-1}$ and $\mathbf{h}_{v,j}^{t-1}$ and the interactions elated to user i and item j between time step $t-1$ and t, *i.e.*, $\{x_i^t\}$ and $\{x_j^t\}$. Specifically, assume a user and a movie interact with each other at time t. The red and blue blocks denote the inputs of the user chain and item chain at time step t, respectively, which are denoted as $\mathbf{y}_{u,i}^t$ and $\mathbf{y}_{v,j}^t$. These two inputs are constructed based on user i's or item j's interactions between time steps $t-1$ and t, respectively. For example, $\mathbf{y}_{u,i}^t$ is defined as $\mathbf{y}_{u,i}^t = [\mathbf{W}_u \cdot \mathbf{x}_{u,i}^t, \log(\Delta \tau_{u,i}^t), 1_{u,\text{new}}]$, where $\mathbf{x}_{u,i}^t$ denotes a sparse vector consisting of the ratings $\{x_i^t\}$ given by user i in time interval $\Delta \tau_{u,i}^t$. $\mathbf{W}_u$ is an embedding matrix, which is employed to reduce the length of GRUs inputs for alleviating intermediate data explosion. $1_{u,\text{new}}$ indicates whether a user is a new user or not [20]. The log interval $\log(\Delta \tau_{u,i}^t)$ is concatenated into the inputs

to encode continues-time information [21]. Inferring $q_{u,i}^t$ is equivalent to inferring $\boldsymbol{\mu}_{u,i,t}^*$ and $\boldsymbol{\Sigma}_{u,i,t}^*$ in Equation (3.5), which are calculated as: $[\boldsymbol{\mu}_{u,i,t}^*, \boldsymbol{\Sigma}_{u,i,t}^*] = f_5(\mathbf{h}_{u,i}^{t-1}, \mathbf{y}_{u,i}^t)$. f_5 is a deep neural network.

Since all of the users (or items) share the same RNN chain, the model size could be largely reduced. Moreover, to further reduce the number of latent variables, the conditioned prior distributions of the dynamic factors $\Delta\mathbf{u}_i^T|\Delta\mathbf{u}_i^{T-1}$, which is defined in Equation (3.2), are assumed to be parameterized by the latent states, *i.e.*, $[\boldsymbol{\mu}_{u,i,t}, \boldsymbol{\Sigma}_{u,i,t}] = [f_4(\mathbf{h}_{u,i}^{t-1}, \Delta\tau_{u,i}^T), f_3(\mathbf{h}_{u,i}^{t-1}, \Delta\tau_{u,i}^T)]$. To further encode the temporal information, we exponentially decay the latent state variables at each time step [86] as $\mathbf{h}_{u,i}^t \leftarrow \mathbf{h}_{u,,i}^t \cdot e^{\frac{\Delta\tau_{u,i}^t}{\lambda}}$, where λ is a predefined decay rate.

3.2.3.3 Objective Function

Considering RNN as a graphical model, we leverage the conditionally independency between current latent state and future inputs, and have $\mathbf{h}^t \perp\!\!\!\perp x^{>t} | \mathbf{h}^{t-1}, x^t$. Then Equation (3.4) could be written as:

$$\begin{aligned} &q(\Delta\mathbf{U}^{\leq T}, \Delta\mathbf{V}^{\leq T} | x^{\leq T}, \mathbf{U}^s, \mathbf{V}^s) \\ &\quad = \prod_{t \leq T} q(\Delta\mathbf{U}^t | x^{\leq t}, \Delta\mathbf{U}^{<t}) q(\Delta\mathbf{V}^t | x^{\leq t}, \Delta\mathbf{V}^{<t}). \end{aligned} \tag{3.6}$$

To obtain the objective function, we try to follow the traditional variational autoencoder to derive a variant variational lower bound. We start from the joint log likelihood and drive the objective

function as follows:

$$
\begin{aligned}
&\log p(x^{\leq T}, \mathbf{U}^s, \mathbf{V}^s) = \log p(x^{\leq T}|\mathbf{U}^s, \mathbf{V}^s) + \log p(\mathbf{U}^s) + \log p(\mathbf{V}^s) \\
&= \int \log p(x^{\leq T}, \Delta\mathbf{U}^{\leq T}, \Delta\mathbf{V}^{\leq T}|\mathbf{U}^s, \mathbf{V}^s) d\Delta\mathbf{U}^{\leq T} d\Delta\mathbf{V}^{\leq T} + \log p(\mathbf{U}^s) + \log p(\mathbf{V}^s) \\
&\geq \int q(\Delta\mathbf{U}^{\leq T}, \Delta\mathbf{V}^{\leq T}|x^{\leq T}) \log \frac{p(x^{\leq T}, \Delta\mathbf{U}^{\leq T}, \Delta\mathbf{V}^{\leq T}|\mathbf{U}^s, \mathbf{V}^s)}{q(\Delta\mathbf{U}^{\leq T}, \Delta\mathbf{V}^{\leq T}|x^{\leq T})} d\Delta\mathbf{U}^{\leq T} d\Delta\mathbf{V}^{\leq T} \\
&+ \log p(\mathbf{U}^s) + \log p(\mathbf{V}^s) \\
&= \sum_{t \leq T} \Big\{ E_{q(\Delta\mathbf{U}^t|x^{\leq t}, \Delta\mathbf{U}^{<t}), q(\Delta\mathbf{V}^t|x^{\leq t}, \Delta\mathbf{V}^{<t})} [\log p(x^t|x^{<t}, \mathbf{U}^{\leq t}, \mathbf{V}^{\leq t})] \\
&- KL(q(\Delta\mathbf{U}^t|x^{\leq t}, \Delta\mathbf{U}^{<t})||p(\Delta\mathbf{U}^t|\Delta\mathbf{U}^{<t})) \\
&- KL(q(\Delta\mathbf{V}^t|x^{\leq t}, \Delta\mathbf{V}^{<t})||p(\Delta\mathbf{V}^t|\Delta\mathbf{V}^{<t})) \Big\} \\
&+ \log p(\mathbf{U}^s) + \log p(\mathbf{V}^s).
\end{aligned}
\tag{3.7}
$$

To further simply the expression, we denote the probabilities $p(\Delta\mathbf{U}^t|\Delta\mathbf{U}^{<t})$, $p(\Delta\mathbf{V}^t|\Delta\mathbf{V}^{<t})$, $q(\Delta\mathbf{U}^t|x^{\leq t}, \Delta\mathbf{U}^{<t})$, $q(\Delta\mathbf{V}^t|x^{\leq t}, \Delta\mathbf{V}^{<t})$, and $p(x^t|x^{<t}, \mathbf{U}^{\leq t}, \mathbf{V}^{\leq t})$, as p_u^t, p_v^t, q_u^t, q_v^t, and p_x^t, respectively. Based on the former definitions, the objective function is defined as a timestep-wise variational lower bound as follows:

$$
\begin{aligned}
\mathcal{L} = &\sum_{t \leq T} \Big\{ \mathbb{E}_{q_u^t, q_v^t}[\log p_x^t] - \mathrm{KL}(q_u^t||p_u^t) - \mathrm{KL}(q_v^t||p_v^t) \Big\} \\
&+ \log p(\mathbf{U}^s) + \log p(\mathbf{V}^s).
\end{aligned}
\tag{3.8}
$$

It is worth pointing out that the expectation term is calculated based on sampling approximation, *i.e.*, $\mathbb{E}_{q_u, q_v}[\log p_x^t] \simeq \frac{1}{L} \sum_{l=1}^{L} \log p(x^t|x^{<t}, \mathbf{U}^{\leq t,l}, \mathbf{V}^{\leq t,l})$, where L is the number of samples we wish to use to estimate the quantity. We specifically set $L = 1$ for every iteration in the implementation following the setting in conventional Variational Auto-Encoder [87] and adopt the reparameterization trick for feasible optimization.

As the rating sequence of each user or item could be infinite long under the streaming setting, which makes it infeasible to feed the whole sequences into the RNNs, this step-wise objective function allows us to truncate the sequences into multiple segmentations for a streaming inference.

Table 3.1: Movie rating dataset statistics.

	# of User	# of Items	Time Spanning	Granularities
MT	$53,275$	$30,686$	$2013 \sim 2018$	4 Weeks
ML-10M	$71,567$	$10,681$	$1995 \sim 2009$	2 Weeks
Netflix	$480,189$	$17,700$	$1999 \sim 2006$	2 Weeks

In another words, assume q_u^T, q_v^T, p_u^T, p_v^T, $\mathbf{U}^s$ and $\mathbf{V}^s$ are achieved at time step T, they could be treated as the prior distribution of the latent variables at time step $T+1$ and updated based on the new interactions $\{x_{ij}^k\}$, the CVRCF framework, and the following step-wise objective function:

$$\begin{aligned}\mathcal{L} =& \Big\{\mathbb{E}_{q_u^{T+1}, q_v^{T+1}}[\log p_x^{T+1}] - \mathrm{KL}(q_u^{T+1}||p_u^{T+1}) - \mathrm{KL}(q_v^{T+1}||p_v^{T+1})\Big\} \\ &+ \log p(\mathbf{U}^s) + \log p(\mathbf{V}^s).\end{aligned} \tag{3.9}$$

It is worth pointing that as stated in Section 3.2.2, we assume the stationary factors $\mathbf{U}^s$ and $\mathbf{V}^s$ represent long-term users' preferences and item popularities. Thus, they should also be updated at each time-step. However, they remain the same between two consecutive time steps while the dynamic factors keep evolving.

3.2.4 Prediction Network

The prediction model is based on the generation model described in Figure 3.2b. At any testing time between time steps $T-1$ and T, to predict a specific ratings of a user i to an item j, we first calculate the expectations of the current latent representations $\mathbf{u}_i^T$ and $\mathbf{v}_j^T$ based on the prior distributions $p_{u,i}^T$ and $p_{v,j}^T$, and the stationary factors $\mathbf{u}_i^s$ and $\mathbf{v}_j^s$. The ratings is then predicted based on the distribution parameterized by the interaction network in Equation (3.1), *i.e.*, $\mathbb{E}(x_{ij}^T|\cdot) = f_1(\mathbb{E}(\mathbf{u}_i^T), \mathbb{E}(\mathbf{v}_j^T))$. Similarly, the variance could also be predicted as: $V(x_{ij}^T|\cdot) = f_2(\mathbb{E}(\mathbf{u}_i^T), \mathbb{E}(\mathbf{v}_j^T), \sigma_{i,j,T}^2)$. $\sigma_{i,j,T}^2$ is assumed to be learnable as a function of the hidden states $\mathbf{h}_{u,i}^{T-1}$ and $\mathbf{h}_{v,j}^{T-1}$ in our implementation.

3.3 Experiments

In this section, we empirically evaluate the performance of CVRCF framework by analyzing three major aspects. **Q1**: What are the general performance of CVRCF compared with the other

Table 3.2: An overview of all experimental methods.

Methods \ Categories	Streaming	Temporal Involved	Probabilistic	Deep
PMF			✓	
time-SVD++		✓		
sD-PMF	✓		✓	✓
sRRN	✓	✓		✓
sRec	✓	✓	✓	
CVRCF (proposed)	✓	✓	✓	✓

baselines? **Q2**: What are the temporal drifting dynamics of users and items we could learned? **Q3**: What are the sensitivities of the model to the key hyperparameters? The code of CVRCF is available at GitHub: https://github.com/song3134/CVRCF.

3.3.1 Datasets

Three widely-adopted benchmark datasets shown in Figure 3.1 are employed in our experiments. Detailed statistics of them are elaborated as follows:

- **MovieTweetings (MT) [88]**: It is a benchmark dataset consisting of movies ratings that were contained in well-structured tweets on Twitter. It contains $696,531$ ratings (0-10) provided by $53,275$ users to $30,686$ movies. All ratings are time-associated spanning from $02/28/2013$ to $04/07/2018$. The granularity is defined as four weeks.

- **MovieLens-10M (ML-10M) [89]**: It contains ten million ratings to $10,681$ movies by $71,567$ users spanning from 1995 to 2009. The granularity is defined as four weeks.

- **Netflix [90]**: The Netflix challenge dataset consists of 100 million ratings by $480,189$ users to $17,700$ movies from 1999 to 2006. The granularity is defined as two weeks.

3.3.2 Baselines

As our main focus is factorization-based approaches, five representative factorization-based baseline algorithms, including two batch algorithms and three streaming algorithms are selected for comparison from different perspectives shown in Table 3.2. Brief descriptions of these methods are listed as follows.

- **PMF** [79]: Probabilistic Matrix Factorization is a conventional recommendation algorithm, which does not consider temporal information.
- **TimeSVD++** [3]: The temporal-envoled variation of the classical static factor-based algorithm SVD++. We implement it with Graphchi [91] C++ pacakge.
- **sD-PMF**: A streaming version of the PMF model combined with the deep interaction network, which is employed in the CVRCF Framework. This model is used to test the effectiveness of the dynamic factors optimized with the RNN structure in CVRCF.
- **sRec** [2]: Streaming Recommender System is the state-of-the-art shallow dynamic recommendation model. It is a probabilistic factor-based model optimized with a recursive mean-field approximation.
- **sRRN** [20]: A streaming variation of Recurrent Recommender Network (RRN), which is a state-of-the-art deep heuristic streaming recommendation model.

3.3.3 Experimental Setup

For each dataset, we segment the data along timeline into three parts with ratios $4:1:5$ serving as training, validation, and testing sets, respectively.

3.3.3.1 Training Settings

During the training phase, the training and validation sets serve as the historical datasets to decide the best hyperparameters for all methods. As each user or movie may have too many ratings, to reduce and memory and protect the feasible use of GRU structures, we truncate the training sequences along the timeline into batches for the user and movie chain, respectively. This will affect the RNN effectiveness to some extent, but by varying the number of training epoch, it does not have an obvious influence on the experimental results during our experiments. Moreover, to protect the stationary factor get faster trained, in each epoch, every truncated batch is processed with multiple iterations. The number of this iteration hyperparameter used in the training phase is set based on validation and will be further analyzed in hyperparameter analysis section.

Table 3.3: The RMSE results on the three datasets.

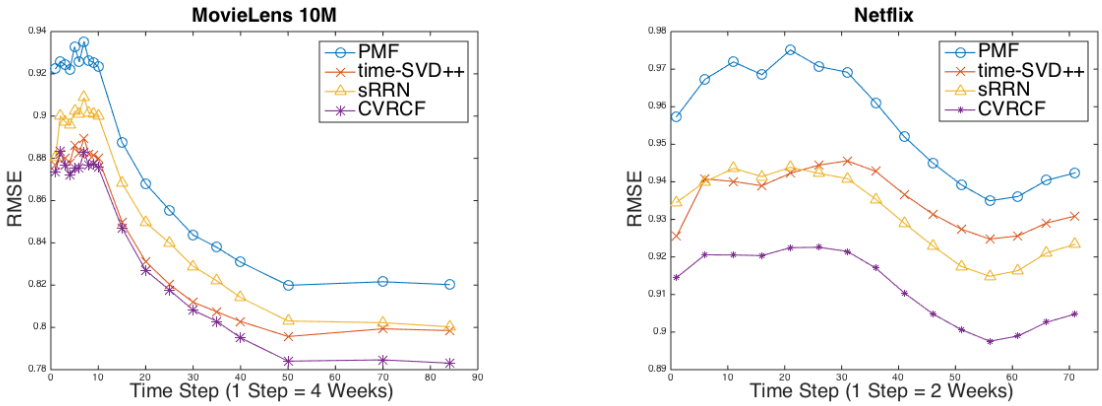
	Methods	Datasets		
		MT	ML-10M	Netflix
Batch	PMF	1.5723	0.8202	0.9421
	time-SVD++	1.4630	0.7985	0.9311
Streaming	sD-PMF	1.6170	0.9017	0.9992
	sRRN	1.5646	0.8003	0.9236
	sRec	1.4831	0.8121	0.9288
	CVRCF	**1.4567**	**0.7831**	**0.9050**

3.3.3.2 Testing Settings

During the testing phase, at each time step t, the testing is first done to get the prediction of the upcoming ratings $\{x^{t+1}\}$, and then these ratings are assumed to arrive and be used to update the models. Different from dynamic methods, at each update, the static methods are reconstructed from scratch using all the previously arrived testing ratings including the training ratings, while the streaming models only employ the current-step arrived ratings for the current update. Based on this setting, no later data is used to predict any former data and no temporal overlapping is existed between each pair of testing intervals. Besides, for fair comparisons, at each testing step, only ratings for existing users and items are used for testing since some baselines (e.g., PMF and time-SVD++) cannot explicitly cope with new users and items. All the experimental results are the arithmetic average of ten different times runs to ensure the reliability. The performance is evaluated via the root mean square error (RMSE).

3.3.3.3 Parameter Setting

Settings of the hyperparameters for all the baselines follow the original papers, which result in their best performance. Hyperparameters in all the methods are selected based on cross-validation using the training and validation sets. For the static baselines PMF and timeSVD++, all of their regularization parameters are chosen over $\{10^{-4}, 10^{-3}, \ldots, 10^{2}\}$ and the sizes of their latent factors are chosen over $\{20, 40, 60, 80, 100\}$. For streaming methods, the size of the stationary factors for sRRN and CVRCF are chosen to be 20 for all the datasets. The stationary factors for sD-PMF is chosen over $\{20, 40, 60, 80, 100\}$. The size of the dynamic factors of CVRCF is chosen to be 40



(a) Testing RMSE changing curve on MovieLens. (b) Testing RMSE changing curve on netflix dataset.

Figure 3.3: Testing RMSE changing curve of four representative methods on ML-10M and Netflix datasets.

including the sizes of both mean and variance parameters. The size of the dynamic factors and the length of the RNN inputs for sRRN is chosen to be the same as CVRCF for fair comparisons. The size of the latent states ($\mathbf{h}_u$ & $\mathbf{h}_v$) of CVRCF is set to be 20 which is half of the length we used in sRRN. The exponential decay factors are set to be 1 week and 4 weeks for the user RNN and movie RNN, respectively. In the training phase, the truncation hyperparameters of all the RNN-based models are set to be 20, 20, and 10 weeks for the three datasets, respectively, to alleviate the intermediate data explosion.

3.3.4 General Evaluation Results

We first analyze the general performance of CVRCF model by comparing it with different categories of baselines based on the RMSE results shown in Table 3.3 and Figure 3.3. From Table 3.3, three conclusions could be drawn as follows. First, CVRCF outperforms all baselines on all datasets. Although time-SVD++ could achieve comparable performance on MT and ML-10M dataset, it has to be reconstructed from scratch using all of the historical data at each update. Second, CVRCF highly outperforms sD-PMF, which confirms the effectiveness of the dynamic factors employed in CVRCF for capturing the temporal relationships during the streaming process. Third, comparing with shallow probabilistic model sRec, CVRCF displays prominent improvement, which demonstrate the effectiveness of deep architectures in modeling complex drifting interactions.

To further analyze the time-varying pattern of each method and their performance consistency

on different datasets, we display the RMSE changing curves of the four representative methods on two larger datasets ML-10M and Netflix in Figure 3.3. From the figure, we could observe that on each dataset the performance of all methods shows similar varying patterns and starting form the first testing step, CVRCF consistently achieves the best performance across two datasets with the evolving of the system. Since MovieLens-10M has the longest testing timeline among all three testing datasets, Figure 3.3a illustrates that CVRCF has stable effectiveness on the dataset with strong temporal relationships in long-term evaluation. By comparison, Netflix is a much larger dataset in terms of users and interactions. Results in Figure 3.3b confirms the superiority of the proposed method on large-scale datasets. Finally, as sRRN could be treated as an ablation method of CVRCF without the probabilistic component, the relative improvement of the proposed method on the general performance validates the effectiveness of combining probabilistic approach in capturing the prospective process of streaming data generation.

3.3.5 Evaluation of Temporal Dynamics

To analyze the temporal drifting dynamics learned from CVRCF, we visualize the learned latent factors including the location factors ($\mathbf{u}_i^T$ and $\mathbf{v}_j^T$) and uncertainty factors ($\boldsymbol{\sigma}_{u,i,T}$ and $\boldsymbol{\sigma}_{v,j,T}$). We conduct exploration on the ML-10M dataset and update the models every half a year during testing.

3.3.5.1 Drifting of the Location Factors

We first visualize the drifting of the average location factors $\mathbf{u}_i^T$ and $\mathbf{v}_j^T$ with heatmap shown in Figure 3.4a. The X-axis denotes the index of the latent factors and the Y-axis denotes the timeline. Each factor is adjusted with centralization for joint visualization. From the figure, we could discover that the users' preference factors change more smoothly than movies' popularity factors, which display a block-wise changing patterns. As we update the model every half a year, the stationary factors of movies especially for the new movies are only updated or learned every half a year, which is consistent with the length of the blocks. Thus, the block-wise structure, which appears only on the movie factors, could be explained as: the movie drifting is more likely to be captured by the stationary factors, while the drifting pattern of the users is more likely to be captured by dynamic

factors. Since the dynamic factors and stationary factors are defined to capture the short-term and long-term preference, respectively, the finding is also consistent with the fact that users preference usually change more frequently compared to movie popularities.

3.3.5.2 Drifting of the Uncertainty Factors

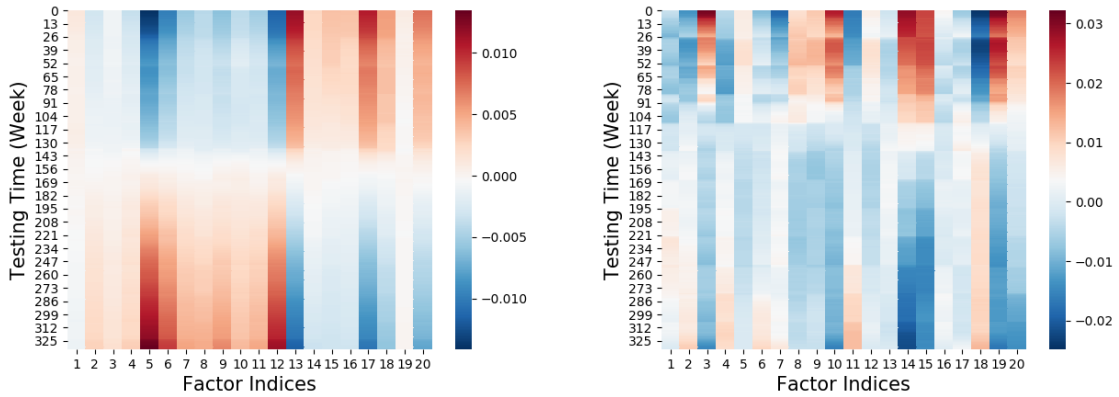
Figure 3.4b displays the drifting of the average uncertainty factors learned from CVRCF. Each column is first normalized with L_∞-norm. There are two major observations we could find from Figure 3.4b. From an overall perspective, with the evolving of the system, the variances of the learned dynamic factors decrease. This is because the incremental ratings provide more information for each user and item, and reduce the uncertainties of the whole system during the testing phase. From the local perspective, at some time steps, the variance of the latent factors are sharply increased and then slowly decreased. This is because, at some time steps, users and movies increase are dramatically. The cold-start problem introduced by the incremental users and items may raise the uncertainties of the system within a short time but would be alleviated with time goes by. In other words, although new users and items are continually enrolled, the number of ratings related to them could be deficient at first and then increasing over time.

3.3.6 Hyperparameter Sensitivity Analysis

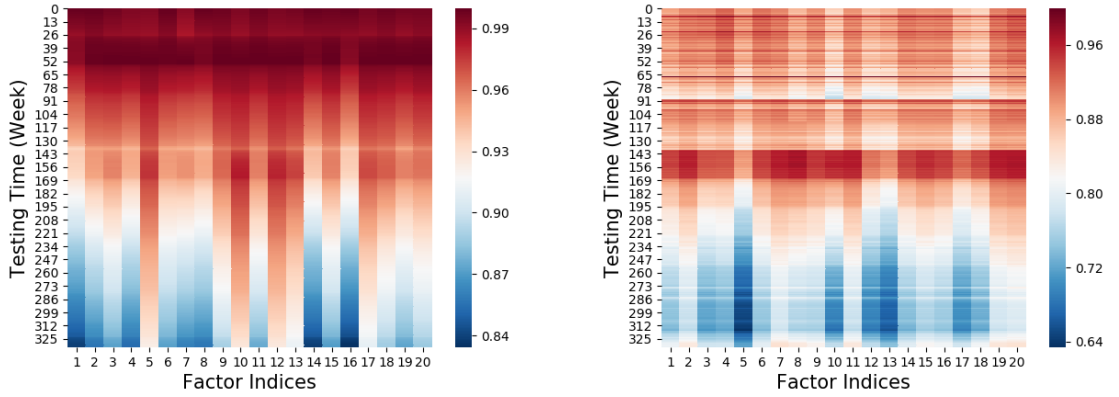
Finally, we study the sensitivity of CVRCF to different hyperparameters using the ML-10M dataset. We pick five hyperparameters, which are the most influential ones in our experiments, and analyze their effects by coupling some of them. These pairwise effects are displayed in Figure 3.5.

3.3.6.1 Training Epochs & Training Batch Iterations

We first analyze the pairwise effects of the training epoch and the training batch iterations. Figure 3.5a shows that these two parameters highly affects the learning process and may cause overfitting or underfitting when the product of them are too large or too small. With the number of training batch iteration increasing, less epoch should be adopted to protect the testing effectiveness. This may be because: since the stationary factors are outside the RNNs and have high degrees of freedom, they may get overtrained when the batch iteration is setting too large given fixed training



(a) Drifting of average location factors of users & movies.



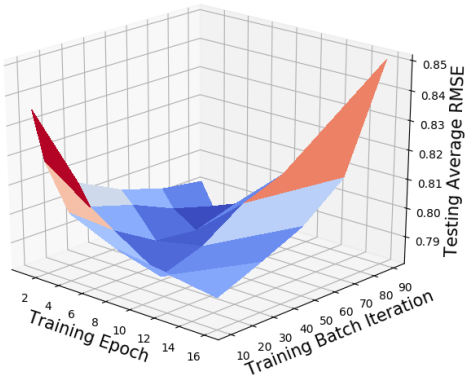
(b) Drifting of variance factors of users & movies.

Figure 3.4: Drifting of average latent factors learned from CVRCF on the ML-10M dataset.

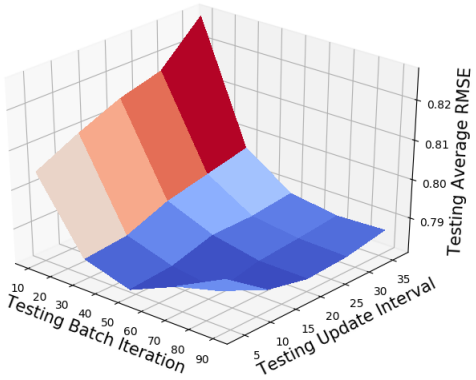
epoch. Thus, early stopping should be employed via limiting the number of epochs to prevent the RNN structures not further learning effectively. On the contrary, insufficient batch iterations would limit the power of stationary factors in capturing long-term preferences.

3.3.6.2 Testing Batch Iterations & Testing Update Interval

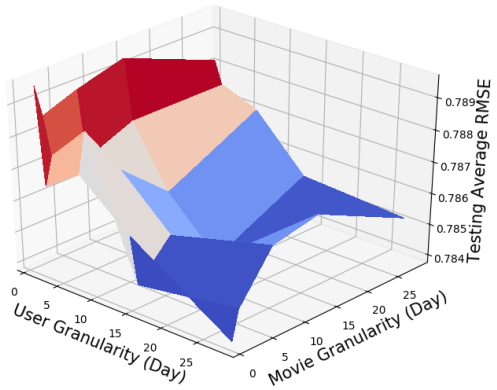
Secondly, we focus on the testing phase and analyze the influence of the testing batch iterations and the length of the model updating interval. As shown in Figure 3.5b, for a fixed testing update interval, with the increasing of the testing batch iterations, the testing performance first decreases and then increases. This might because: in the testing phase, new ratings, users, and items never stop to arrive. Insufficient testing batch iterations would highly affect the learning of latent factors especially for the stationary factors of new users or items. On the contrary, superfluous iterations



(a) Effects of the train epoch & training batch iteration.



(b) Effects of testing batch iteration & update interval.



(c) Effects of the granularities.

Figure 3.5: Analysis of five key hyperparameters on ML-10M datasets.

would also lead to overfitting as in the training phase described above. Besides, with the enlarging of the testing update interval, ratings in each batch increase which requires more updating iterations under the same remaining settings.

3.3.6.3 Granularities

Finally, we explore the effect of the granularities. We assume the two granularities defined for users and movies could be different for a more general treatment. From Figure 3.5c, we can see that although different granularities do affect the results, their influences are shown to be very trivial based on the scale of the Z-axis. Moreover, user granularity seems to have larger effects than movie granularity and its optimal value is shown to be lower than movie granularity. This may illustrate that the users' preferences are varying more frequently than the items' popularities.

3.4 Conclusions

The work we discussed in this chapter focuses on the recommendation problem under streaming setting and propose a deep streaming recommender system - CVRCF. CVRCF incorporates deep architectures into traditional factorization-based model and encodes the temporal relationship with Gaussian-Markov components. Standing upon the sequential variational inference, CVRCF is optimized leveraging a cross variational GRU network and could continually update under the streaming setting. By conducting experiments on various real-world benchmark datasets, we empirically validate the effectiveness our proposed framework, explore the learned drifting patterns, and validate the stability of our framework.

4. TOWARDS AUTOMATED NEURAL INTERACTION DISCOVERY FOR CLICK-THROUGH RATE PREDICTION[1]

With the rapid development of recommender systems, designing and tuning them become increasingly complicated and challenging. Starting from this chapter, we explore how to enable the automation on the architecture design and hyperparameter tuning in recommender systems. Specifically, we focus on the deep-learning models for CTR prediction tasks in this chapter and propose a framework to automatically search for a neural architecture that could even beat state-of-the-art, manually designed architecture.

4.1 Introduction

Predicting CTR is a crucial problem in many web applications such as real-time bidding, display advertising, and search engine optimization [92]. Due to the large-scale dataset and high-cardinality feature property, extensive efforts have been devoted to designing architectures for effectively learning combinatorial feature interactions towards condensed low-dimensional feature representations [93, 94, 95, 96].

Classical approaches usually put the efforts on designing explicit feature interactions and compose it with implicit interactions learning from multi-layer perceptrons (MLP) into a two-tower model [93, 96, 97]. Beyond simple stacking strategies, how to organically bond the explicit and implicit interactions is still underexplored and may further promote effectiveness. Besides, existing work has shown that diamond MLP structure may be more powerful compared to the triangle and rectangular MLP structures [98], which motivates us to explore more powerful implicit interactions via designing delicate MLP structures. In addition, conjoining the advantages of diversified explicit feature interactions such as inner and outer products could potentially boost the performance owning to the ensemble effect.

[1]This chapter is reprinted with permission from "Towards Automated Neural Interaction Discovery for Click-Through Rate Prediction" by Qingquan Song, Dehua Cheng, Hanning Zhou, Jiyan Yang, Yuandong Tian and Xia Hu, 2020. Proceedings of the 26rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Copyright 2020 by ACM.

Neural architecture search (NAS) has emerged as a prevailing research field upon the prevalent adoption of deep learning techniques. It aims to discover optimal deep learning solutions automatically given a data-driven problem, thereby enabling practitioners to access the off-the-shelf deep learning techniques without extensive experience, and alleviating data scientists from the burden of manual network design. The rapid development of NAS research and systems has enabled the automation of state-of-the-art deep learning tools for various learning tasks in computer vision (CV) and natural language processing [31, 99]. This motivates us to explore its potential in the context of tabular data in discovering complex neural interactions, specifically for CTR predictions.

Developing novel NAS approaches for neural interaction discovery and better CTR models is technically challenging. First, different from structure image data in CV tasks, CTR features are often heterogeneous, high-dimensional, and have both sparse and dense components, which are structureless and of diversified meanings in reality. Second, different from the dominant convolutional neural networks in CV tasks that consist of multiple structured convolutional operations, existing models for CTR prediction usually adopt multiple diverse and ad-hoc operations, leading to unstructured search space. Third, a practical model for CTR prediction is often trained on billions of data (e.g., Facebook has millions of daily active users and over 1 million active advertisers [100], yielding billions of instances), requiring the NAS process to be time and space efficient. Finally, the performance of CTR models with different architectures are often quite close in practice [96, 97], asking for the NAS approach to be sensitive and discriminating.

To cope with these challenges, we propose an automated neural interaction discovering framework for CTR prediction named **AutoCTR**. We abstract and modularize simple yet representative operations in existing CTR prediction approaches to formulate a generalizable search space. A hybrid search algorithm, composed of an evolutionary backbone and a learnable guider, is designed to perform orientated exploration. To enhance the exploitation power and balance the trade-off among different search objectives, we utilize a learning-to-rank strategy among the architecture level to filter out the locally inferior architectures, and conduct the survivor selection based on a mixture of rank-based measurement including aging, accuracy as well as architecture complexity.

The search speed is further accelerated through a composited strategy of low-fidelity estimation, including data subsampling and hash size reduction. The main contributions are summarized from the following aspects:

- Design *virtual blocks* and a hierarchical search space for CTR prediction by abstracting and unifying the commonly used operations in the existing literature.
- Provide ranking consistency analysis for three strategies combining low-fidelity estimation and weight inheritance. Empirically prove the availability of employing them for search acceleration.
- Propose a novel multi-objective evolutionary search algorithm with architectural-level learning-to-rank guidance.
- Empirically demonstrate the effectiveness of AutoCTR on different datasets comparing to human-crafted architectures, and validate the generalizability and transferability of the discovered architecture across different datasets.

4.2 Preliminaries

Click-Through Rate Prediction: The CTR prediction problem could be mathematically defined as: given a dataset $D = \{\mathbf{X}, \mathbf{y}\}$, where $\mathbf{X} \in \mathbb{R}^{N \times d}$ denotes the d-dimensional features matrix of N instances. The features here consist of both sparse and dense features, where we assume the sparse features are ordinally encoded as integer vectors. $\mathbf{y} \in \{0, 1\}^N$ indicates the clicks of users to items. The goal is to predict the probability of a user clicking a target item.

Since its inception, the mainstream models have roughly experienced three-stage evolution starting from linear regression and tree-based models [92, 100, 101], to interaction-based models [102, 103], and then the deep neural networks (DNNs) [104, 105, 98, 95, 93, 96, 26, 94, 106, 107]. Recent DNN-based work usually combines DNN with interaction-based or tree-based models to extracting condensed representations via learning combinatorial feature interactions.

Neural Architecture Search: Neural architecture search (NAS) aims to promote the design of neural network architectures automatically. Designing the NAS algorithm requires the specification

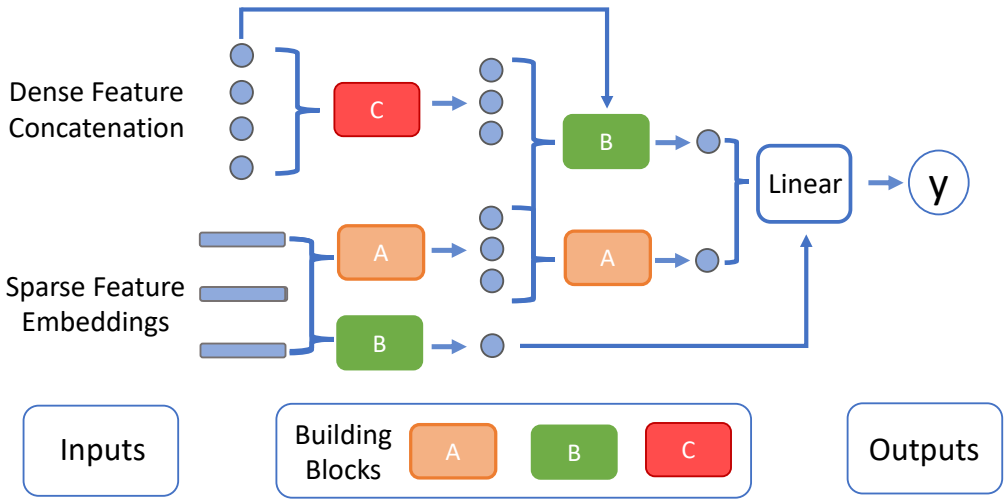


Figure 4.1: An illustration of an architecture in the designed search space. The virtual building blocks are wired together to form a DAG. Blocks are allowed to be selected repetitively.

of three main components, including *search space*, *search techniques*, and *performance estimation strategy* [31].

From the search space perspective, existing work could be roughly divided into the exhaustive-architecture search space [108, 109] and the constrained cell space [110, 111, 112, 113]. The former one provides a more diversified set of architectures and allows a more comprehensive exploration, while the later one inductively limits the space to accelerate the search speed and reduce the search variance. Some tailored spaces are also designed [114] for specific tasks.

From the perspective of search techniques, several dominant ones include Bayesian optimization [108], reinforcement learning [115, 110, 111], evolutionary algorithms [109, 112], gradient-based optimization [113], and tree-based methods [116]. Recent work has shown the effectiveness of combining different types of search techniques to better balance exploitation and exploration [117].

Due to the high complexity of training the DNNs and the particularity of evaluation criteria in different tasks, various performance estimation strategies are proposed, including low-fidelity estimation [110, 112], weight sharing [111, 113], learning curve extrapolation [118], and network morphism [108].

4.3 Hierarchical Search Space Design

An ideal search space should contain sufficient distinct architectures while potentially encompassing superior human-crafted structures [115, 108]. Inspired by the search space tailored to the vision tasks [114], we design a two-level hierarchical search space by extracting and abstracting representative structures in existing human-crafted CTR prediction architectures into virtual blocks, and wire them together as a set of direct acyclic graphs (DAGs) with dimensionality alignment among features. The inner space is composed of the appendant hyperparameters of blocks, such as the number of units and layers in a multi-layer perceptron block, while the connections among blocks form the outer search space.

As shown in Figure 4.1, for a given instance, we assume its raw input dense features are concatenated into a vector, and its sparse features are embedded into low-dimensional vectors based on the look-up-table operation following similar preprocessing done in various CTR models [96, 107]. Blocks are wired together to form a DAG, and each block could take both raw input features and the outputs from blocks with higher topological order via feature concatenation and dimensionality alignment. The final block of the network is set to be a linear transformation. It collects all the untouched features from either raw inputs or outputs provided by other blocks. It is worth noting that to simplify the setting, the following hyperparameters are not taken into account in the search space: (1) Hash size of sparse features. (2) Embedding dimension of sparse features. (3) Optimizers and other model training hyperparameters such as learning rate and batch size.

4.3.1 Virtual Block Abstraction

We select and extract building blocks by considering two aspects described as follows:

- **Functionality:** blocks should accommodate and complement each other. Each type of block should accommodate both dense and sparse features as inputs, and can be quantitatively evaluated.

- **Complexity Aware:** The computational and memory cost of a block should be a simple function of its input specification and hyperparameters. The involvement of these primitives could benefit the design of complexity-aware search algorithms to achieve better resource management and

low-complex architectures.

Upon these requirements, we could abstract various operations from existing work as blocks with different functionality and levels of complexity such as multi-layer perceptron (MLP) [107], dot product (DP), factorization machine (FM) [102], outer product [96], and self-attention [97], etc. We elaborate on the construction of the three example blocks (i.e., MLP, DP, FM) adopted in the experiments below and describe the way of aligning the dimension and accommodating different inputs for each of them. Other blocks could also be easily abstracted and integrated into the framework, which is left for future exploration.

Dense MLP Block (MLP) The MLP block serves as the most commonly used building block in literature. We design the dense MLP block similar to the one applied in the DLRM model [107]. The input of it will be concatenated into a single long vector and transformed via a multi-layer perceptron with the ReLU activation function. We set the layer to be 1 for each MLP block to maximize the flexibility of the final constructed architectures. The width of each MLP block is searched within the unit set: {32, 64, 128, 256, 512, 1024}.

FM Block (FM) The FM block refers to the factorization machine block [102]. A conventional factorization machine model takes real-valued feature vector $\mathbf{x}$ as inputs, and outputs a single value via low-dimensional embedding and summation operations, as shown in Equation (4.1), where $\mathbf{v}$ denotes the embedding matrix, $\mathbf{w}$ is the transformation parameter. In the designed FM block, we assume all the input features are already transformed into the low-dimensional space and conduct the dot product and summation operations directly. Three specific designs are listed here for dimensionality alignment, and search space robustness: (1) We concatenate all the dense input features, including dense raw features and the dense outputs from other blocks, into a single vector. (2) If the dimensions of sparse inputs and the concatenated dense input are conflicted, we will linearly embed them into the same size in advance. (3) If only dense features are obtained as inputs, the block will degenerate to a linear embedding block with a sum pooling afterward.

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < \mathbf{v}_i, \mathbf{v}_j > x_i x_j. \tag{4.1}$$

Dotprocessor Block (DP) The DP block calculates the dot product of every pair of the input embeddings and concatenates the results into a single long vector. The self-interaction is also added, i.e., the dot product of each embedding vector and itself, so that when only dense features are collected as the block input, it is also feasible to proceed. In this case, this block degenerates into an element-wise square operator. Similar concatenation and dimensionality alignment strategy are used as for the FM block.

4.3.2 Summarization of Search Components

We summarize the main components to be searched in the hierarchical search space as follows:

- **Block Type:** MLP, FM, DP.
- **Raw Feature Input Selection:** each block is allowed to take the raw feature with four choices, i.e., dense only, sparse only, both or none. Without particular emphasis, we group the raw input dense features as one single component to be selected rather than considering them independently. A similar procedure is done for the sparse features to reduce the search complexity.
- **Inter-Block Connection:** a block could receive the outputs from any block that appeared before it. The order is defined as the topological order in the DAG.
- **Block Appendant Hyperparameters:** We only consider the number of hidden units of MLP block in this work. The embedding sizes for aligning the input dimensions in different blocks are fixed and set to be the same with the embedding size of the raw input sparse features to reduce the number of parameters.

To enable the feasible adoption of different searching algorithms, we provide a vector representation of each architecture as a concatenation of multiple block vectors following [116]. Each block is vectorized as a concatenation of the four components, i.e., *[Block Type, Raw Feature Input Selection, Inter-Block connection, Block Appendant Hyperparameters]*, where the *Block Type* and *Raw Feature Input Selection* are encoded as one-hot vectors respectively, *Inter-Block connection* is

encoded as a multi-hot vector, and *Block Appendant Hyperparameters* is encoded as a vector of ordinals.

The designed search space contains plentiful distinct architectures. Even with three types of blocks to be selected and assume each architecture contains no more than seven blocks, the space would still contain over 10^{11} distinct architectures. Moreover, it could cover multiple representative human-crafted architectures such as deepFM [93], DLRM [107], IPNN [105], and Wide&Deep [95].

4.4 Multi-Objective Evolutionary Search with Hyperrank Guidance

The proposed searcher is a mixed searcher composed of an evolutionary algorithm and a learning-to-rank guider. We select the evolutionary algorithm in this pilot study due to its simplicity and effectiveness in balancing the exploitation and exploration [112]. We adopt a multi-objective evolutionary searcher as the backbone and employ a tree-based learner to guide the mutation of a selected parent architecture in each iteration to facilitate the exploitation of superior offsprings. The search process could be described as a loop of three stages, i.e., parent selection, guided mutation, and survivor selection. The initial population is constructed via randomly selecting and evaluating a predefined number of architectures. Stratified selections could also be used to potentially enhance the performance, which we leave for future exploration.

The basic idea of the search loop is shown in Figure 4.2. In each search loop, we leverage a mixture of rank-based meta-features to perform the survivor selection and maintain a new fix-size population. Then we select a parent architecture from the population based on designed discrete probabilistic distribution. After that, a set of neighbors is generated via mutating the selected parent architecture. We adopt a learning-to-rank mechanism upon the architecture level and select the best offspring from the generated neighbors. After evaluating the performance of this offspring, we add it back to the architecture pool explored so far. The three stages are elaborated in turn in the following subsections.

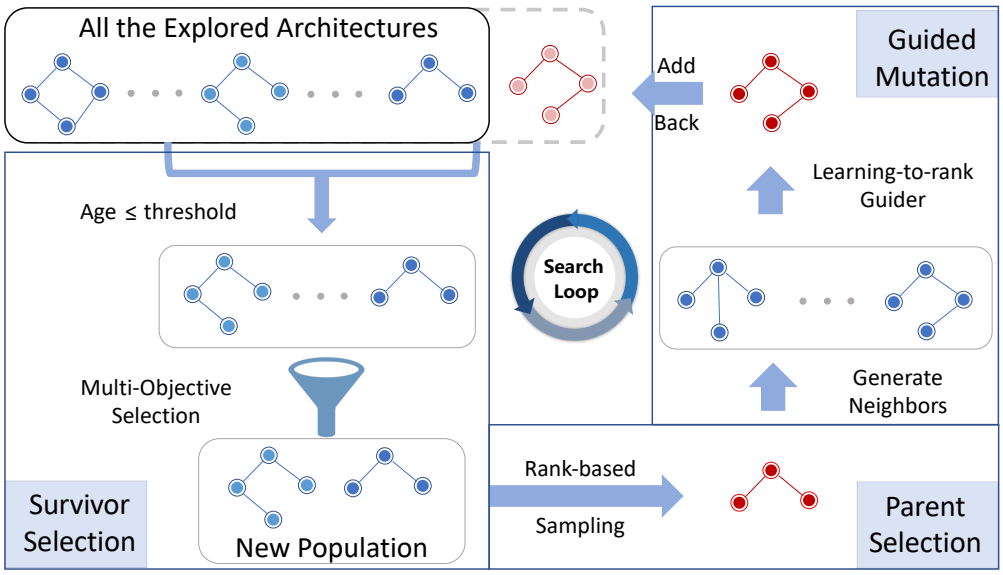


Figure 4.2: An illustration of the AutoCTR search loop

4.4.1 Multi-Objective Survivor Selection

To maintain a superior population with diversified architectures, we design a survivor selection metric f to measure the survival value of each architecture and select the top-p ones as the population for parent selection. Three types of objectives are considered in the metric, i.e., fitness, age, and model complexity. The "fitness" here represents the performance of an explored architecture to ensure the exploitation ability, while the "age" reflects the reverse order of the architectures explored so far. We use the rank of logloss as the fitness measure to mitigate the difference in scale and define the "age" as the existing time of an architecture explored so far, motivated from the age-based evolutionary methods [112], to enhance the exploration of diversified architectures. At each search loop, we set the current time to be 0 and set the age of each observed architecture as the number of architectures explored after it. In particular, all initial architectures are assigned with the same age. Besides the two objectives, since CTR tasks are usually resource hungry in practice, we also take the "model complexity" into account to explicitly constrain the model complexity during search. We adopt floating point operations per second (FLOPs) as the complexity metric and use the rank of flops to mitigate the scale influence.

The designed selection schema consists of two steps as shown in Figure 4.2, which could be

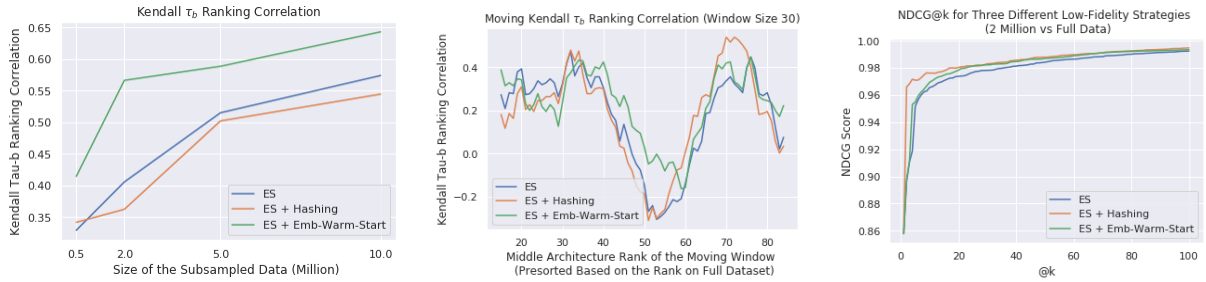
formulated as follows:

$$f(q, a_A, r_A^q, c_A^q) = \mathbb{1}_{[a_A \leq q]} \cdot (\mu_1 a_A + \mu_2 r_A^q + \mu_3 c_A^q), \tag{4.2}$$

where a_A denotes the age of an architecture A. The indicator function $\mathbb{1}_{[a_A \leq q]}$ is used to filter out the architectures that is older than q, where q is a hyperparameter larger than the population size p, such that the architectures with high-performance or low-complexity in the pool would not be selected consistently. $r_A^q, c_A^q \in \{1, 2, \dots q\}$ are the performance and complexity ranking of architecture A within the q "youngest" architectures, respectively. $\{\mu_i\}_{i=1,2,3}$ are the trade-off hyperparameters to balance the different objectives.

4.4.2 Rank-Based Parent Selection

Suppose we maintain a population of size $p \in \mathbb{Z}^+$ after the survivor selection step. The goal of parent selection is to select an architecture from the population for generating a premium offspring to be evaluated. Several popular strategies used in conventional evolutionary algorithms include proportional, ranking, tournament, and genitor selection [119]. In this work, we adopt a ranking selection schema and design a nonlinear ranking distribution borrowing the idea from tournament selection. The intuition comes from three aspects: (1) The performance (logloss or AUC) of different architectures in CTR prediction tasks is often extremely close, and the scale may vary a lot on different datasets. It is hard to design a unified performance-based distribution to achieve adaptive selective pressure on different datasets. (2) Existing work has proved the effectiveness of ranking and tournament selection comparing with proportional and genitor on balancing the selective intensity and selection diversity [120, 119]. (3) Classical tournament methods usually randomly select a fixed number of candidates first and then select the best one of them. This may result in a portion of architectures in the population never being selected as the parent, especially when the ratio between candidate size and population size is large.



(a) Rank consistency varying with the number of subsamples and different training strategy

(b) Local ranking consistency shifting with window size 30 upon the sorted real rank

(c) NDCG@K score of different training strategies on 2 million subsamples

Figure 4.3: Rank consistency analysis from both global perspective and local perspective

The concrete design of the probability is as follows:

$$p(r_A^*) = \frac{\binom{r_A^*+\lambda-1}{\lambda}}{\binom{p+\lambda}{1+\lambda}}, \qquad r_A^* \in \{1, 2, \ldots, p\}, \lambda \in \mathbb{N}^0, \tag{4.3}$$

where r_A^* denotes the rank of an architecture A in the population, $\binom{n}{k} = \frac{n!}{(n-k)!k!}, (k = 0, 1, .., n)$, and λ is a hyperparameter to balance the trade-off between selection intensity and selection diversity. Given a fixed size of population, with the increasing of λ, the selection intensity increases while the selection diversity reduces. In particular, $\lambda = 0$ is the same as uniform selection and when $\lambda = 1$ is the same as linear rank selection [121, 120].

4.4.3 Guided Mutation by Learning to Hyperrank

After the parent architecture is selected, the last step is to generate a worth exploring offspring upon it. A naive way of doing this is to randomly select and modify an operation in the parent architecture [112]. Nevertheless, it could become an inefficient strategy due to the huge search space and the waste of the architecture information explored so far. Existing work has demonstrated the effectiveness of using a learning-based model to guide the mutation process [117]. However, with a limited number of explored architectures and the extremely close performance among them, it is non-trivial to learn an effective fitness-based guider.

Instead of learning a fitness-based guider, we adopt a learning-to-rank strategy to learn the relative ranking among architectures based on a pairwise ranking loss and the gradient boosted tree

learner. The whole offspring generation process is done via three steps: (1) train a guider based on the exploitable dataset (e.g., all the architectures explored so far); (2) randomly generate a set of unique neighbors around the parent architecture; (3) select the best neighbor as the offspring based on the guider. We call it "learning-to-Hyperrank" as it conducts a model-level ranking. The intuition comes from two perspectives. On the one hand, the search problem itself is essentially a ranking problem. Learning the ranking relationship is a commensurable strategy comparing with learning the fitness but is weaker and more flexible. On the other hand, the number of instances are implicitly augmented from the point-wise inputs to the pairwise inputs.

For the pairwise ranking-loss, we use LambdaRank [122] due to its simplicity and efficiency. Though different types of models could serve as the learner, we choose a gradient boosted tree learner as an example here due to its general stable and superior performance on small-scale datasets. To feed architectures as the input for the tree-based learner, we encode each block as a vector and concatenate them based on their topological order. Each block vector follows the vector representation described in section 4.3.2.

4.5 Performance Estimation Acceleration

One of the most crucial challenges in modern NAS research, which could be even more severe in recommender systems, is the high time complexity of network training. Low-fidelity performance estimation and weight inheritance are two of the most widely adopted methods for speeding up performance estimation [110, 112]. However, extra bias could be introduced, leading to the variation of relative ranking among architectures, thereby affecting the searching effectiveness [31]. We consider two strategies of low-fidelity estimation and adopt a warm-start embedding trick leveraging the weight inheritance among architectures to mitigate the time and resource complexity. These strategies are all general and practical ways to speed up the manual tuning of recommender systems. Several rank consistency tests are described afterward to provide the evidence and illustrate the feasibility of adopting these methods.

- **Data Subsampling.** For each dataset, we randomly subsample a predefined portion of data for searching and transfer the searched best architecture on the full dataset for final evaluations. On average, the training time of every single architecture in our experiment is roughly linearly correlated with the subsampling ratio under the hardware setting described in Section 4.6.2.

- **Reducing Hash Size.** For high-level categorical features ($> 10^4$), we hash the cardinality of them to 10^4 before the embedding step to reduce the embedding size, and set back to the original cardinality during final fit on the full dataset.

- **Warm-Start Embedding.** We pretrain a simple three-layer MLP model (units: 128-1024-128) on the full dataset and use the pretrained embeddings of sparse features as the warm-start for each architecture before the low-fidelity training.

The rest of this section provides an analysis of the effect of adopting low-fidelity training on ranking consistency. We use logloss as the evaluation metric and use the early-stopping (ES) strategy to alleviate overfitting. We focus on the global and local rank consistency respectively, to pursue the analysis. The global rank consistency inspects whether the estimation could reflect the actual ranking of performance among architectures[2], and the local rank consistency testing zooms into the architectures with relatively closer performance and analyze their localized rank consistency.

4.5.1 Global Rank Consistency

Settings: We use Criteo dataset[3] here for experiments. Without loss of generality, we narrow down the search space by assuming each architecture contains five blocks, and each MLP block has one layer with 128 units. We randomly sample 100 valid architectures and evaluate them on Criteo with different sizes of subsamples, i.e., $\{0.5, 2, 5, 10\}$ million. Each subsampled dataset is split into training (80%), validation (10%), and test (10%) sets. We run the 100 models on the full dataset three times to provide a "ground-truth" rank, and the rank consistency is measured by the Kendall τ_b coefficient ranging from -1 (perfect inversion) to 1 (perfect agreement).

[2]We assume the actual rank of architectures is reflected by their high-fidelity performance on the full dataset.
[3]`www.kaggle.com/c/criteo-display-ad-challenge`

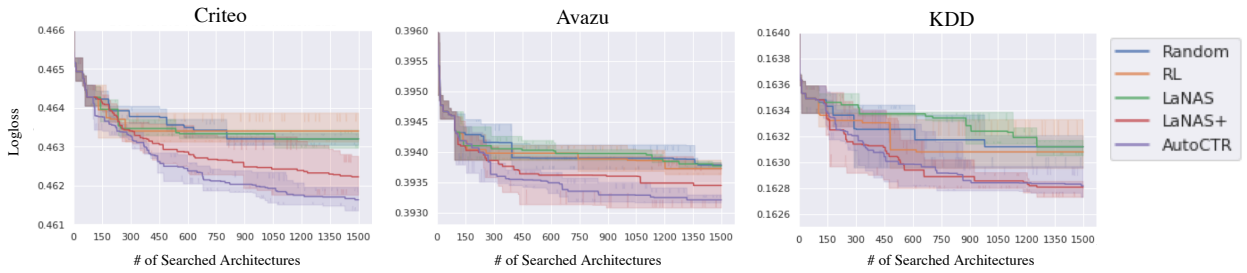


Figure 4.4: The performance drifting of the best architecture during search on the three datasets

Observations: Figure 4.3a depicts the varying curves of three training strategies measured by the Kendall τ_b coefficient with the increasing of the subsample size. The three strategies are: (1) early stopping; (2) early stopping with sparse feature hashing; (3) early stopping with warm-start embedding. We observe that: firstly, with the increasing of the subsample size, the rank among architectures become more consistent with the "ground-truth" rank, and the growth speed gradually becomes slow. The non-linear relationship between sample size and τ_b coefficient offers the opportunity of adopting the low-fidelity setting during searching, which will be further evaluated in the experimental section. Secondly, adopting the hashing strategy for sparse features with high cardinality would marginally decrease the rank consistency. Finally, the warm-start embedding strategy could increase ranking consistency.

4.5.2 Local Rank Consistency

The above analysis provides a macro view of the ranking consistency among the architectures. Beyond this, we are also curious if the τ_b score is harmoniously distributed across different intervals of the actual rank. We first sort the 100 architectures based on their "ground-truth" performance evaluated on the full dataset. Better architectures are indexed with smaller numbers. Then we depict two plots for analysis. Figure 4.3b displays the variation of τ_b coefficient with a length-30 sliding window among the sorted architectures. For example, the score of the first point denotes the τ_b coefficient of the top-30 architectures, where its x-coordinate is 15, indicating the middle architecture rank within this sliding window. Figure 4.3c shows the variation of the NDCG@k score with the increase of k. Exponential gain is used to calculate the NDCG score.

From the two figures, two main observations could be found: (1) From Figure 4.3b, we can

see that, best- and worst-performing architectures seem to be more locally rank consistent. This is partially aligned with our expectations since we expect the top architectures and the bottom ones to be more easily discernible than others. The warm-start strategy is more helpful for maintaining the local rank of middle architectures rather than the polar ones. (2) From Figure 4.3c, we can observe that the rank of the top architectures could generally maintain high rankings in the low-fidelity setting even if the rank consistency for some mediocre architectures is affected more compared to the polar ones.

4.6 Experimental Analysis

In this section, we empirically evaluate AutoCTR as well as several baseline searchers and compare the discovered architecture to the human-crafted architectures. We use logloss and AUC score as the core evaluation metrics. Four questions are mainly explored:

Q1. How is AutoCTR comparing with other baseline searchers on both the search efficiency and effectiveness?

Q2. How is the performance of the best architecture explored by the AutoCTR comparing with the state-of-the-art (SOTA) human-crafted architectures?

Q3. Are the searched architectures able to be transferred between different datasets?

Q4. How sensitive is the AutoCTR to its key hyperparameters?

4.6.1 Baselines

We select baselines from both NAS methods and human-crafted CTR architectures for comparison. Since several searchers are not directly applicable in our setting, we modify and improve their flexible components and elaborate the details as follows.

4.6.1.1 Searcher Baselines

To prove the effectiveness of the proposed search algorithm, we select three representative NAS algorithms in existing work. We tailor random search and reinforcement learning based search to

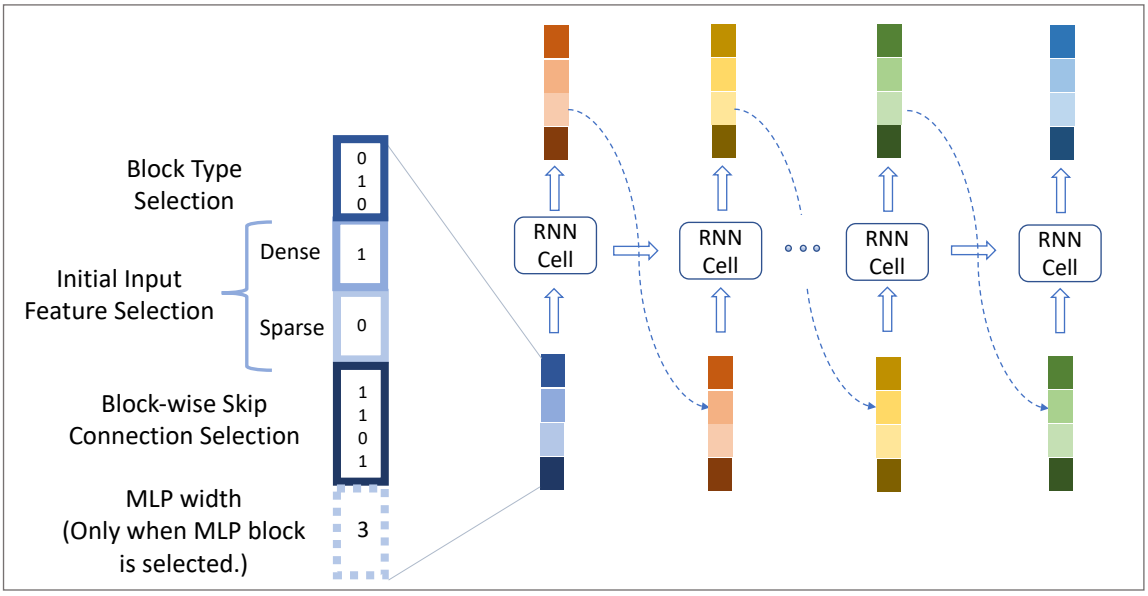


Figure 4.5: The illustration of the RL searcher

our designed search space and include a variation of LaNAS based on the proposed method and the parallel Monte Carlo Tree Search [123].

- **Random Search.** Recent work in NAS has pointed out that random search could be a strong baseline, even comparing to the most advanced search algorithm [124, 125]. We implement a random searcher as follows: given a random seed, in every search epoch, it randomly selects the four components described in section 4.3.2 for each block and builds an architecture based on the topological order.

- **Reinforcement Learning Based Search (RL)** . We adopt a single-layer RNN controller with length seven to generate architectures with seven intermediate blocks, as shown in Figure 4.5. The output of each step decides the structure of the next block and serves as the recurrent input for the next step. Initial inputs and hidden state values are set as zero by default. We use the REINFORCE algorithm to update the controller and use the logloss difference between the current architecture and the best architecture so far on the validation set as the reward. The average reward is used as a baseline to reduce variance. An entropy term is added in the loss function to enhance the exploration diversity.

- **Latent Action Neural Architecture Search (LaNAS)** [116]. A sample-efficient Monte Carlo Tree Search (MCTS) algorithm, which has been proven to be effective than various advanced search algorithms, including the Regularized evolutionary algorithm [112] and Bayesian optimization algorithm in the image classification setting. The search strategy follows the standard MCTS algorithm with the upper-confidence bound (UCB) policy. The search space is partitioned via customizable regressors contained in the nodes of the tree. The search algorithm could narrow down to a specific subspace when traversing from the root to the leaf. Each regressor is updated based on the architectures in the corresponding node and their validation logloss.

- **LaNAS+**. We improve LaNAS from two perspectives.

    1. Convert its rollout sampling policy from random sampling to rank-based method borrowing the idea from the evolutionary method. Random rollout policy could be slow due to the lack of exploitation. Following the original paper, the random rollout policy here means when we have selected a path in the tree and collected the constraints in each node along the path to narrow down to a specific subspace, the way to sample an architecture from this subspace is naively random sampling. However, this could potentially cause a high rejection rate due to the complicacy of sampling from a non-convex polytope and may lose the usage of existed good architectures. To address the problem, we design an evolutionary style rollout policy: every time after narrowing down to a specific subspace, we retrieve the currently searched architectures within this subspace and set them as the population. Then we do the same thing as we do in the AutoCTR, i.e., select a subset of candidates, pick the best one, and mutate it to a new architecture.

    2. Introduce virtual loss to enable parallel training of multiple architectures to improve the search speed. Within each update internal of the LaNAS searcher, we are supposed to sample multiple architectures in order to explore different tree branches. After each architecture is sampled, the back-propagate step requires the logloss of it to update the tree statistics so that it could conduct exploration on different branches rather than always exploring one single

path. However, this may prevent the searching process from being parallelly proceeded since each new architecture can only be sampled after the evaluation of the former one. To circumvent this problem, a virtual loss [123] could be applied as padding during the back-propagate step to ensure the feasible adoption of parallel training and evaluation of multiple architectures. The virtual losses will be removed after the true loss is obtained.

4.6.1.2 SOTA Human-Crafted Networks

We select three representative human-designed networks to examine if the explored network is able to achieve comparable accuracy or even beat the SOTA human-crafted networks.

- **DeepFM** [93]: a two-tower model composed of a six-layer MLP with 1024 hidden units in each layer and factorization machine block, the sparse input embeddings are shared between the two components, and the output of the two components are linearly embedded with sigmoid transformation as the final prediction.
- **DLRM** [107]: a SOTA MLP-based recommendation model. It encodes dense features and sparse look-up embeddings with two MLP modules and embeds their outputs with a top-level MLP module jointly. We adopt two single-layer MLP on the dense and sparse features respectively and stack a six-layer MLP on top of them. All MLP layers are with 1024 units except the final one.
- **AutoInt+** [97]: a two-tower model composed of a four-layer MLP with 1024 hidden units in each layer and three-layer self-interaction layer, which adopts the multi-head self-attention schema to learn high-order feature interactions. The outputs of the two components are linearly embedded with sigmoid transformation into the final prediction.

4.6.2 Experimental Settings

4.6.2.1 Data Preprocessing.

We adopt three benchmark datasets in this work, i.e., Criteo[3], Avazu[4], KDD Cup[5]. The basic statistics of them are summarized in Table 4.1. All three datasets are processed based on the way

[4]`https://www.kaggle.com/c/avazu-ctr-prediction/data`
[5]`https://www.kaggle.com/c/kddcup2012-track2/data`

and codes provided in [97]. During the search phase, we subsample the first 2 million data of each dataset and further divide it into the tiny training (80%), validation (10%), and test (10%) sets for low-fidelity evaluation. Also, we would like to gratefully acknowledge the organizers of KDD Cup 2012 track 2 as well as the contributors of Criteo and Avazu for making these CTR prediction benchmarks publicly available.

Table 4.1: Statistics of three CTR prediction datasets

Datasets	# Samples	# Dense Features	# Sparse Features	Total Cardinality of Sparse Features
Criteo	45,840,617	13	26	998,960
Avazu	40,428,967	0	23	1,544,488
KDD	149,639,105	3	10	6,019,086

4.6.2.2 Hyperparameter Settings.

We search seven intermediate blocks for each architecture. Blocks are allowed to be empty. Three example blocks are adopted in the final experiments, i.e., MLP, FM, and DP. The detailed construction of them is provided in Section 4.3.1. We randomly sample 100 architectures as initialization for all the searchers. Detailed hyperparameter settings for each searcher and the network training in both the search phase and final fit phase, are specified as follows:

We elaborate on the detailed hyperparameter settings adopted in the experiments in this section. The three random seeds used in the experiments are 42, 2019, and 1234, respectively.

- **Search Phase Training:** *Adam* and *Sparse Adam* optimizers are adopted for dense and sparse features, respectively, in training. The batch size is set as 4096. The learning rate is 0.001. The hash size is 10^4 for all sparse features. The embedding tables for sparse features are randomly

initialized based on a normal distribution with 0 mean and 0.01 standard deviation. The embedding size for each sparse feature is 16.

- **Final-Fit Phase Training:** Except for the hash sizes of all sparse features that are set back to their original cardinality, and the dataset is the full data, all the other settings are the same with the ones adopted in the search phase.

Hyperparameter settings for the searchers are elaborated as follows.

- **RL searcher:** the input encoding size and the LSTM hidden embeddings size are both set to 10, the trade-off hyperparameter of the entropy term is set to be 0.1.
- **LaNAS:** the tree-depth is set to 5. In the search phase, the update step is done once after 20 architectures are evaluated. The UCB trade-off parameter is set as 0.5. The space split classifier is defined as ridge regression with 0.1 regularization hyperparameter.
- **LaNAS+:** besides the hyperparameters mentioned above in LaNAS, the candidate size for the modified rollout policy in LaNAS+ is set as to be half of the architectures contained in a selected region. The virtual loss is set to be the mean logloss of the architectures contained in each leaf node.
- **AutoCTR:** the population size is set to be 100, and the survivor selection threshold q is set as 200. The trade-off hyperparameters μ_1, μ_2, μ_3 are set to be 1, 0.1, and 0.1, respectively. The parent selection trade-off hyperparameter λ is set to be 10. The guider is implemented with the lightgbm package with NDCG@3 as the early-stop evaluation metric.
- **AutoCTR (warm):** the warm-start embedding for each dataset is achieved from a four-layer MLP architecture with units: 128-1024-128-1, which is pretrained on each full dataset.

4.6.2.3 Software and Hardware Descriptions

All the deep learning related frameworks are implemented with the PyTorch package[6]. Every single search experiment is run on a single GPU (NVIDIA GeForce RTX 2080 Ti) with three

[6] `https://pytorch.org`

Table 4.2: General CTR prediction results (logloss) on the three benchmark datasets

		Criteo	Avazu	KDD	Search cost (GPU_Days)
SOFA human-crafted Networks	DeepFM	0.4432	0.3816	0.1529	
	DLRM	0.4436	0.3814	0.1523	-
	AutoInt+	0.4427	0.3813	0.1523	-
Best Networks Found by the NAS Methods	Random	0.4421 ± 0.0003	0.3824 ± 0.0030	0.1531 ± 0.0001	~ 0.75
	RL	0.4422 ± 0.0005	0.3810 ± 0.0003	0.1531 ± 0.0001	~ 0.75
	LaNAS	0.4421 ± 0.0004	0.3814 ± 0.0006	0.1533 ± 0.0002	~ 5
	LaNAS+	0.4417 ± 0.0001	0.3800 ± 0.0004	0.1521 ± 0.0001	~ 0.75
	AutoCTR	$\mathbf{0.4413 \pm 0.0002}$	$\mathbf{0.3800 \pm 0.0001}$	$\mathbf{0.1520 \pm 0.0000}$	~ 0.75
	AutoCTR (warm)	0.4417 ± 0.0005	0.3804 ± 0.0004	0.1523 ± 0.0001	~ 0.75

Table 4.3: General CTR prediction results (AUC) on the three benchmark datasets

		Criteo	Avazu	KDD	Search cost (GPU_Days)
SOFA human-crafted Networks	DeepFM	0.8086	0.7767	0.7974	
	DLRM	0.8085	0.7766	0.8004	-
	AutoInt+	0.8090	0.7772	0.8002	-
Best Networks Found by the NAS Methods	Random	0.8096 ± 0.0004	0.7765 ± 0.0029	0.8001 ± 0.0003	~ 0.75
	RL	0.8094 ± 0.0005	0.7778 ± 0.0005	0.7999 ± 0.0002	~ 0.75
	LaNAS	0.8096 ± 0.0005	0.7772 ± 0.0011	0.8001 ± 0.0009	~ 5
	LaNAS+	0.8101 ± 0.0000	0.7790 ± 0.0007	0.8009 ± 0.0004	~ 0.75
	AutoCTR	$\mathbf{0.8104 \pm 0.0003}$	$\mathbf{0.7791 \pm 0.0001}$	$\mathbf{0.8011 \pm 0.0001}$	~ 0.75
	AutoCTR (warm)	0.8099 ± 0.0005	0.7784 ± 0.0006	0.8004 ± 0.0003	~ 0.75

architectures parallelly trained on it. Multi-core CPUs are used for data preprocessing and searcher training. Specifically, we use five cores for data preprocessing and the searcher training in the search phase, and 5 CPU cores + 1 GPU for the final fit of each discovered architecture after searching. We adopt parallel CPU-GPU training for searching, and the training speed is accelerated by loading, preprocessing, and saving the data batches in the GPU memory. The evolutionary guider is implemented with the lightgbm package[7], and the FLOPs are calculated based on the thop package[8]. Plots in the case study are drawn with the graphviz package[9].

[7] `https://lightgbm.readthedocs.io/`
[8] `https://github.com/Lyken17/pytorch-OpCounter`
[9] `https://graphviz.readthedocs.io`

Table 4.4: Architecture complexity comparison (parameters in the embedding tables are included)

	# Params (Million)			Flops (Million)		
	Criteo	Avazu	KDD	Criteo	Avazu	KDD
DeepFM	22.51	30.34	101.73	22.74	22.50	21.66
DLRM	23.55	29.29	102.77	26.92	18.29	25.84
AutoInt+	20.44	28.28	99.66	18.33	17.49	14.88
AutoCTR	19.89	26.49	97.06	12.31	7.12	3.02

4.6.3 General Comparison Among Searchers

We first compare the general search performance of all five searchers. Figure 4.4 depicts the logloss drifting of the best architecture during searching on the three datasets. The x-axis indicates the number of architectures searched so far (in total, 1500). The y-axis denotes the validation logloss of the architectures. Several observations can be summarized as follows. Firstly, based on the performance of the best architecture searched in the low-fidelity setting, AutoCTR generally outperforms other baselines, and our modified LaNAS+ outperforms LaNAS. Secondly, by comparing the search efficiency, AutoCTR and LaNAS+ still outperform other searchers consistently.

We then transfer the best architectures searched so far by each searcher onto the full dataset and display the final evaluation results in Table 4.2 and Table 4.3. The results shown from row four to eight are the average performance of the best architectures searched in the three rounds with different seeds. We can observe that: (1) After transferring the best architectures found by the searchers on the full datasets, the architectures found by AutoCTR still performs the best. Moreover, the ranking of the results aligns well with the one in low-fidelity setting, which implies the correctness of the rank consistency testing and the feasibility of adopting low-fidelity estimation in the search process. (2) On all three datasets, the final architectures searched by AutoCTR and LaNAS+ could achieve even better performance comparing to the SOTA architectures. The architectures searched by Random search and RL-based search could also achieve comparable or even better performance. This empirically validates the effectiveness of the designed search space and the feasibility of adopting NAS algorithms on the CTR prediction problem. It is worth pointing out that

an improvement of around 0.0005-0.001 is already regarded as practically significant on these CTR prediction benchmarks [96, 97]. (3) We further examine the influence of adopting the warm-start embedding in searching. Although it highly improves the performance of most architectures in the low-fidelity setting (not shown in Figure 4.4 due to the difference in scale), the performance on the full dataset is not improved. We attribute this observation to the overfitting issue of the warm-start embedding dictionary and the evaluation dataset during searching. Since we set a fixed search iteration (i.e., 1500 architectures) rather than adopting early-stop for the search algorithm, the overfitting issue may happen during the search process, which has also been pointed out by several recent works [126, 127].

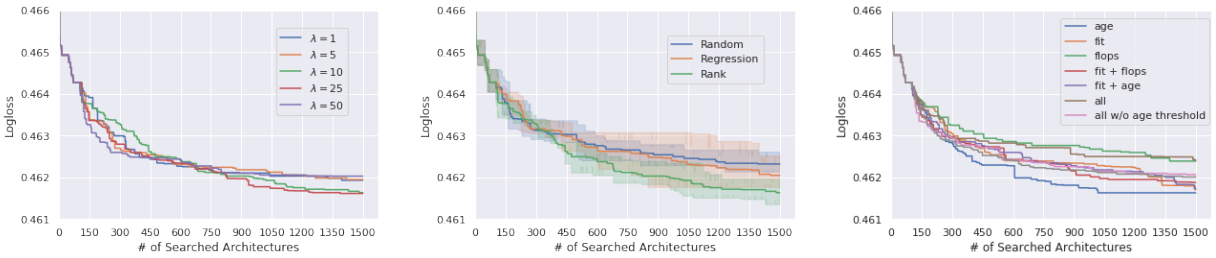
Beyond accuracy, we also display the time complexity of the search algorithms in Table 4.2 and compare the model complexity of the best-discovered architectures of AutoCTR with and the SOTA human-crafted architectures in Table 4.4. The time for training the searcher could generally be ignored due to the limited size of the sampled architectures and the parallel CPU-GPU training schema we adopted. Results show that the explored architecture is smaller than the human-crafted ones on both the number of parameters and FLOPs. This is mainly because: (1) we explicitly constrain search space and adopt the complexity control term in the survivor selection, which restricts the exploration of overcomplicated architectures; (2) the searchers tend to find architectures with diamond or inverted triangle MLP structures, while the human-crafted ones directly adopt rectangular MLP structures as defined in the original work, and are set with 1024 units in each layer in our experiments.

4.6.4 Architecture Transferability Analysis

As the human-crafted architectures are not designed for a specific dataset, we explore the transferability of the searched architectures across different datasets. We select the best architectures searched by AutoCTR on each dataset, and apply them on the other two. From Table 4.5, we can observe that the architectures searched on one dataset could still perform well when applying to the others. This is mainly because: (1) the three benchmark datasets share common characteristics of feature relationships; (2) the blocks incorporated in the designed search space and the discovered

Table 4.5: Transferability of architectures found by AutoCTR

Original Dataset \ Target Dataset	Criteo	Avazu	KDD
Criteo[3]	0.4413	0.3799	0.1520
Avazu[4]	0.4421	0.3800	0.1535
KDD[5]	0.4418	0.3803	0.1521



(a) Effects of the selection intensity hyperparameter λ to the search process given population size p=100 on Criteo data.

(b) Effects of different type of guider in the mutation stage to the search process after fixing other hyperparameters on Criteo data.

(c) Effects of seven different types of the survivor selection objectives to the search process on Criteo data.

Figure 4.6: Analysis of the key hyperparameters in the three stages of AutoCTR

connectivity among the blocks is general enough to uncover the high-order feature interactions of different CTR prediction datasets. Comparably, the architecture discovered on Avazu performs a bit worse when doing the transfer. One reason is that Avazu only contains sparse features, which results in no exploration of the dense features in searching. For these models, the dense features are only considered in the final embedding layer in our implementation during the transfer.

4.6.5 Hyperparameter Sensitivity Analysis

In this section, we study the sensitivity of AutoCTR on the key hyperparameters using the Criteo dataset and analyze the impact of the core components at different stages.

4.6.5.1 Effects of Selection Intensity in Parent Selection.

We first analyze the influence of the selection intensity hyperparameter λ in the parent selection stage by fixing the population size as 100. As discussed in section 4.4.2, the higher the λ is, the

more intense the selection would become. We choose $\lambda = 1, 5, 10, 25, 50$ and depict the curve of search effectiveness in Figure 4.6a. With the increase of λ, the exploitation ability of AutoCTR generally increases while the exploration power decreases. It increases the initial search speed but would degrade the exploration ability in the long term. Under the current experimental setting, $\lambda = 25$ seems to be a more balanced option between exploitation and exploration.

4.6.5.2 Effects of Different Mutation Guider.

Secondly, we focus on the mutation stage and analyze the influence of the different types of guiders to the search efficacy. Three types of guiders are compared here: (1) random guider (random): it generates the candidate offspring by randomly selecting and mutating an operation of the selected parent architecture. (2) fitness-based guider (regression): it uses a gradient-boosted tree to conduct regression on the explored architectures and their performance. It selects the best architecture among the 100 randomly generated neighbors of the parent architecture as the new candidate offspring. (3) rank-based guider (rank): the one we used in AutoCTR. Different from the fitness-based guider, it learns the pairwise ranking relationship among the architectures rather than directly fit their performance. We fine-tune the tree-based learner for both fitness-based guider and rank-based guider, respectively. From Figure 4.6b, we can observe that the AutoCTR with rank-based guider outperforms the other two. Although the fitness-based guider could improve search effectiveness comparing to the random strategy, it also suffers more on the overfitting issue, which results in the search variance to be large and makes it difficult to be tuned.

4.6.5.3 Effects of Different Survivor Selection Objectives.

Finally, we explore the effect of adopting different objectives in the survivor selection stage. Figure 4.6c and Table 4.6 compare the search and final evaluation performance of AutoCTR with different search objectives. Except for the last objective, each of them adopts the age threshold $\mathbb{1}_{[a_A \leq q]}$ described in Equation (4.2). We set the trade-off weights for each term as 0.5. The results show that the age-based objective benefits more to the search speed comparing to the fitness-based objective. By adding the complexity constraint in the objective, the size of the best model explored

Table 4.6: Performance and complexity comparison of architectures found with different survivor selection objectives

Objective	Performance		Model Size (Million)	
	Logloss	AUC	# Params	Flops
a_A	0.4418	0.8010	21.97	16.62
r_A	0.4417	0.8100	20.58	15.07
$a_A + r_A$	0.4415	0.8103	19.77	11.85
$a_A + c_A$	0.4418	0.8099	18.08	5.06
$r_A + c_A$	0.4417	0.8101	19.59	11.10
$a_A + r_A + c_A$	0.4415	0.8103	20.50	14.73
$a_A + r_A + c_A$ w/o threshold	0.4416	0.8102	19.35	10.14

could be reduced while the performance remains comparable.

4.6.6 Discussion

In this section, we visualize the best-explored architectures and analyze the importance of the block components learned from the tree-based guider. Some limitations and conjectures of the current study are discussed afterward to promote future exploration.

4.6.6.1 Case Study.

We first visualize two of the best architectures found by AutoCTR on Criteo and KDD datasets, respectively, in Figure 4.7. It shows that both of the architectures ensemble multiple FM and DP blocks and tend to adopt MLP blocks in the later stage. Besides, dense features prefer MLP block while sparse features prefer DP and FM in the early stages, which shows the ability of DP and FM in modeling sparse features explicitly. Moreover, the MLP blocks display a diamond structure, i.e., MLP layers in the middle of the graph are wider than the ones in both ends, which aligns with some analysis in existing works [98]: diamond networks are preferable to the increasing/decreasing width networks (triangular networks) and the constant width networks (rectangular networks).

4.6.6.2 Interpretation of Important Blocks.

To provide a better understanding of the block-type influence to the architectures, we display the feature importance of high influential components learned from the <architecture, performance

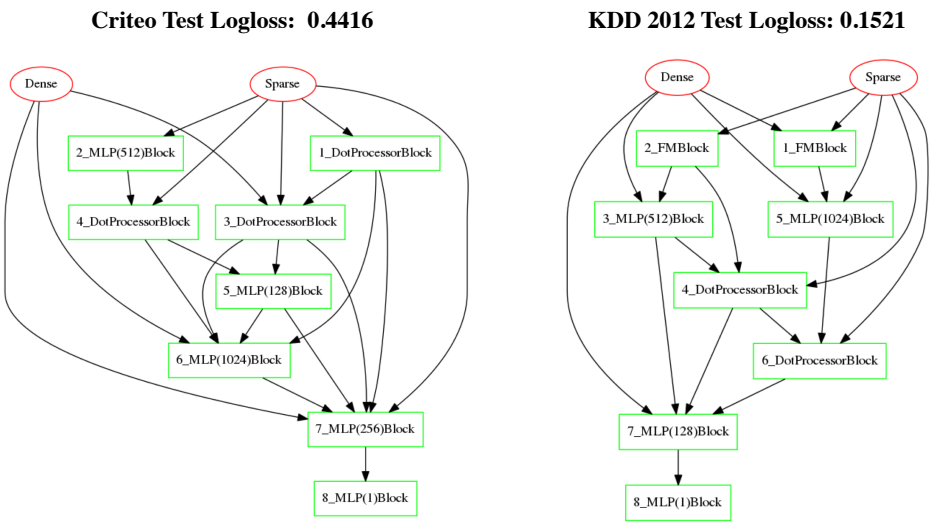


Figure 4.7: Two architectures found by AutoCTR.

rank> pairs on Criteo and Avazu via the tree-based guiders. We randomly select $10,000$ architectures with seven valid blocks in each of them to train the tree-based guider and display the importance score of the top-20 influential block types in Figure 4.8. The “id_type" tick below the x-axis indicates the topological order of a block and its type. We observe that: (1) the structure of the polar blocks based on the topological order have larger impacts compared with the middle ones; (2) MLP block dominates the architectures; (3) DP and FM blocks are relatively more impactful on Avazu than Criteo since Avazu only contains sparse features. We need to emphasize that this interpretation may be biased by the search-space design and the way of representing the architectures. Interpreting the NAS process and involving the interpretations into the architecture design could be promising.

4.6.6.3 Limitations.

Despite the analysis discussed above, several limitations are mentioned here for future investigation.

Search Space. Though the number of architectures contained in the search space is quite large ($> 10^{11}$), the number of block types we have currently explored is still limited. This also explains why Random and RL searchers could achieve acceptable performance. Moreover, the flexibility can be further enlarged via independent feature selection towards more dedicated and delicate interactions.

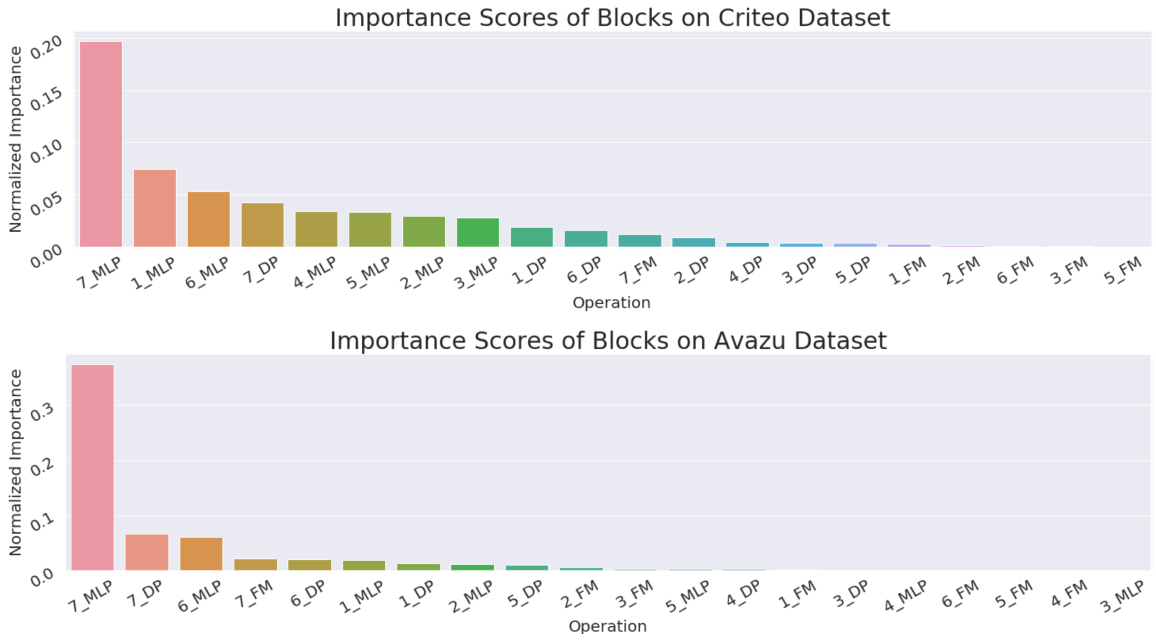


Figure 4.8: Normalized importance scores of top-20 block type operations learned by AutoCTR guider on Criteo and Avazu

Overfitting. The overfitting issue is enlarged in the low-fidelity setting due to the limited subsample size and the stop criteria we adopted, i.e., search 1500 architectures for every searcher in each experiment. Although AutoCTR and LaNAS+ still show their superiority, the improvements compared with other searchers are weakened. One possible way to migrate this issue is to adopt early-stopping strategies or add regularizations for the search process [126, 127].

4.7 Conclusions

In this work, we conduct a pilot study of automatically designing architectures for the CTR prediction task. We construct a hierarchical search space via wiring representative blocks extracting from human-crafted networks and explore the rank consistency among the architectures under the low-fidelity setting. A tailored evolutionary search algorithm with a multi-objective survivor selection strategy is proposed guided by an architectural-level learning-to-rank method. Experimental results on three benchmark datasets demonstrate the effectiveness of the proposed search algorithm and the feasibility of adopting low-fidelity estimation during the search phase.

5. TRANSFERABLE BLACK-BOX OPTIMIZATION FOR HYPERPARAMETER TUNING

Considering the vast amount of recommendation tasks imbued in modern web applications, designing a recommendation tuning them from scratch is time-consuming and laborious. In this chapter, we explore how to leverage the similarities of different web application tasks to accelerate the automated hyperparameter tuning. We propose a transferable black-box optimization algorithm that could benefit from the model evaluation information from historical tasks towards accelerated hyperparameter tuning. The outcome algorithm is expected to be not limited to tuning ML models in recommender systems but also applicable for general block-box optimization and related applications such as job scheduling of physical systems and user experience optimization in web services.

5.1 Introduction

Designing and deploying recommendation models or services in industrial applications is quite challenging. It generally involves various handcrafted configurations and multiple steps of manual tuning. Automated hyperparameter tuning is an integral part of AutoML, showing extraordinary ability and great potential in finding apposite hyperparameters for complex machine learning algorithms [128]. It significantly alleviates the burden of expertise on repetitive tuning. Despite the achievements made in hyperparameter tuning on a single task, it is still a time-consuming and resource-intensive job, especially when faced with large models and datasets.

IT companies may develop and maintain hundreds, if not thousands of models for different applications (online, offline). Exhaustively tuning and configuring them from scratch is prohibitively resource-wise expensive. Moreover, as vast historical tuning and evaluation information is generated every day, it could be difficult for engineers to manually distill and summarize all the experimental results for model design and tuning.

Motivated by the continual learning of human engineers in honing their expertise on configuring and designing models and systems from historical experience, we expect to leverage the evaluation

and tuning information collected from historical tasks towards the faster exploration of optimal configurations on new tasks. Since many tasks are within similar application domains (e.g., search and recommendations), their datasets and models share extensive similarities. For example, (1) neural network models applied on people search and job search could share similar model structure (such as the number of filters) and hyperparameters for training (learning rate and optimizer type); (2) datasets for different recommendation tasks may share similar feature categories such as user demographic features or item content features. Distilling and harnessing this information could potentially accelerate the tuning speed, thus saving the cost of workforce and physical resources. We intend to treat the model selection and tuning in each recommendation task as a black-box optimization problem to put it into a more general and flexible setting. No analytic solution or gradient information is available for each optimization problem. The only input we have is the hyperparameters and their evaluations on individual tasks. The goal is to design a transferable black-box optimization algorithm that could benefit transfer the historical evaluation information from different tasks towards accelerated hyperparameter tuning on the target tasks.

To achieve the goal, three questions needs to be answered, i.e., when to transfer, what to transfer, and how to transfer. We briefly describe the challenges of answering each question and the limitation of existing works on solving them:

- The breakthrough of 'when to transfer' lies in measuring the similarity of tasks. The transfer method should filter out the historical tasks that are similar to the target task to transfer useful information without harming the tuning process of the new task. The critical challenge is that there are no explicit measures to quantify the similarity of two tasks (black-box functions). Existing work usually puts their efforts on designing hand-crafted meta-features to characterize each task to quantify the similarity between them [129]. However, it is non-trivial to develop suitable meta-features, and hard to judge their effectiveness.

- 'What to transfer' describes the information that we want to transfer from historical tasks to the new tasks. Some representative ones in the literature include the best hyperparameters of historical tasks, the parameters of the surrogate model in the search algorithm (such as

the gaussian process model in Bayesian optimization algorithm) learned from historical evaluations (i.e., hyperparameters and their performance on historical tasks) [130], as well as the constraint of search space [131]. Existing work usually focuses on one of these transferable information or conjoint the advantages of several of them. However, almost all the prior arts are tailored to a specific search model, which means their solution is not model-agnostic and cannot be applied to other search algorithms.

- The question of 'how to transfer' asks for the way of doing the transfer. As Black-box optimization is often a time-consuming process in practice due to the high cost of evaluating the black-box functions, the transfer algorithm should be efficient enough to avoid extra burdens being appended. The state-of-the-art algorithms usually adopt ensemble learning of neural network models to conduct meta-learning for information transfer. Though many of them show prominent improvements compared to non-transfer methods, they either have high computational complexity or are hungry for historical hyperparameter evaluations.

5.2 Meta-Tree Transfer Algorithm

To tackle the challenges, we intend to propose a tree-based transfer method. The key idea is to construct a meta-level decision tree model for each task leveraging the optimization information (e.g., <hyperparameter, evaluation> pairs of the corresponding tasks). The tree model compares the task similarity and transfer information between the historical tasks and the target task. We use the general hyperparameter tuning task as an example to illustrate the keys ideas of these three steps. Specifically, given a new task, the methods take four steps to achieve the information transfer: tree construction, similarity comparison, candidate task selection, and a two-type transfer. We will elaborate the four steps one by one in the following subsections. Step two to four are iteratively conducted during the search process.

5.2.1 Tree Construction

The first step is called tree construction, which is relatively straightforward. For each historical task, we tree a regression decision tree based on <hyperparameter, evaluation result> pairs. This tree

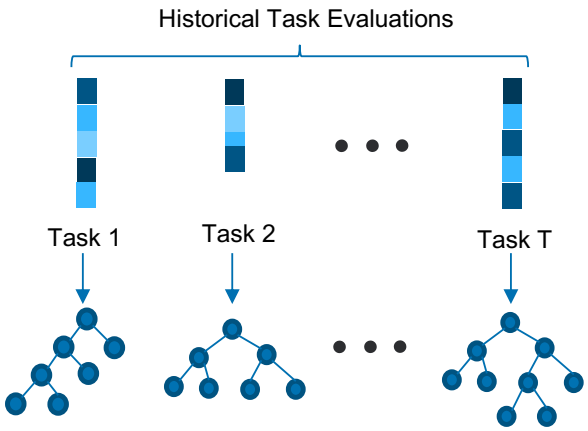


Figure 5.1: Tree construction based on historical hyperparameter evaluation information could be used for later tasks without being constructed repetitively from scratch, given new tasks.

For the target task, we first sample several initialization points based on the Sobol quasi-random sampling strategy [132]. We use Sobol quasi-random sampling based on the intuition that the sequence achieved by it could provide a finer partition of the search space than purely uniform random search and avoid the samples gathered in a specific region affects the tree construction.

5.2.2 Similarity Comparison

After constructing the tree, we develop a ranking-based similarity quantification strategy to compare the tasks' similarities. We first calculate the predictions of each historical tree to the samples collected on the target task. The samples here represent the hyperparameters that have already been evaluated. We then measure the similarity between tasks by calculating the Kendall Tau-b ranking correlation coefficient between the predictions and the ground-truth evaluations on the target task.

5.2.3 Candidate Historical Task Selection

Candidate historical task selection based on tournament selection. After comparing the similarities between the old trees and the target tree. We can sort the historical trees based on how similar they are to the target dataset. To avoid one historical tree always being selected, we adopt a tournament selection method to enhance the exploration ability from a meta-tree level. The key idea is to randomly select a candidate (five in the later experiments) set of historical trees and then pick

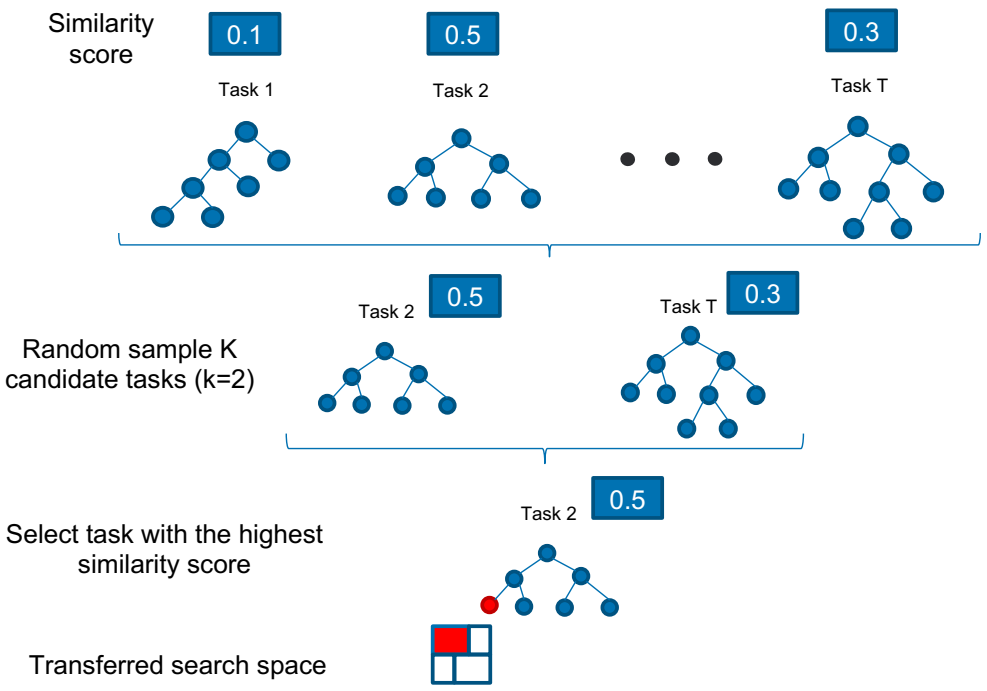


Figure 5.2: Tournament selection of candidate tree based on tree similarities

the best one as the tree to be transferred based on the similarity score calculated in step two. This is quite useful when not enough samples are provided for the new tasks leading to noisy measures on the similarities.

5.2.4 Pointwise and Space-wise Transfer

We pursue both pointwise and space-wise transfer. For pointwise transfer, we directly transfer the best points (hyperparameters) evaluated in the historical task selected at step three. A simple simulated annealing method is used to judge if we would like to continue using the pointwise transfer when the previous transferred point performs poorly on the target task. A core difference between our transfer method and all the existing works is that we conduct a subspace transfer besides a pointwise transfer. When the algorithm decides to stop the pointwise transfer, it will start to apply for a space-wise transfer by transferring the best subspace partitioned by the historical tree model into the target task. This subspace will serve as a constrained search space for the search algorithm to pursue a more targeted and efficient exploration.

We can iteratively conduct steps two to four during the search phase and interactively with the user-specified search algorithm (or we say black-box optimization algorithm) on the new tasks.

The default search algorithm is Bayesian optimization with upper confidence bound acquisition function. Other algorithms can also be seamlessly combined with the proposed method, such as neural-network-based searching algorithms.

5.3 Experimental Analysis

In this section, we empirically evaluate the proposed transferable black-box optimization algorithm on both synthetic and real-world datasets. We mainly explore two questions:

Q1. How is the proposed method comparing with the state-of-the-art (SOTA) transferable methods on conventional black-box optimization tasks?

Q2. How is the search speed and accuracy of the proposed method compared to the SOTA transferable methods on hyperparameter tuning tasks?

5.3.1 Baselines

We utilize four main baselines in the final experiments. The first two are non-transferable methods, and the rest are the SOTA black-box optimization methods. Their details are summarized as follows:

- Sobol quasi-random search (sobol). A random samping method, whose random sequense is drawn by the sobol quasi-random sequence[1].
- Gaussian process with upper confidence bound acquisition function (gpucb). It is a classical black-box optimization method, utilizing the Bayesian optimization method with the Gaussian process as the surrogate model and upper confidence bound as the acquisition function.
- Google vizier black (vizier) [130]. SOTA transferable black-box optimization service adopted in Google.
- Ranking-weighted Gaussian process ensemble (RGPE) [133]. A SOTA meta-Learning method proposed by Facebook.

[1] https://botorch.org/api/sampling.html

5.3.2 Experiment on Synthetic Black-Box Optimization Datasets

In this subsection, we empirically evaluate the proposed method on a set of classical black-box tasks: optimize quadratic functions. We sequentially maximize 15 quadratic functions. The format of the quadratic functions are shown below:

$$f_t(x) = a_t||x||_2^2 + b_t 1^\top + c_t, \tag{5.1}$$

where $(a_t, b_t, c_t) \in [-0.1, -10]^3$, $x \in [-100, 100]^4$. We randomly sample the three coefficients of the 15 quadratic functions from the specified interval. Our goal is to sequential optimize these 15 functions, i.e., finding the optimal x to maximize each function. We conduct 50 search trials for each task. Since we're optimizing the functions in a sequential setting, each transfer learning methods could leverage the optimizations on the previous quadratic functions to optimize the current one.

The experimental results are shown in Figure 5.3. The x-axis represents the number of trials of each task, and the y-axis indicates the maximum value of all passed trials achieved on the task-specific quadratic function during optimization. Black dash lines indicate the optimal values for the tasks. We can get two main observations. Firstly, since we know the optimal value of each task, all the methods could achieve the optimal value on every task within 50 search trials. Secondly, the proposed 'space_gpucb' method (purple curve) outperforms other methods in search speed on almost all the tasks compared to both none transfer methods (sobol and gpucb) and transfer learning method (RGPE and google vizier).

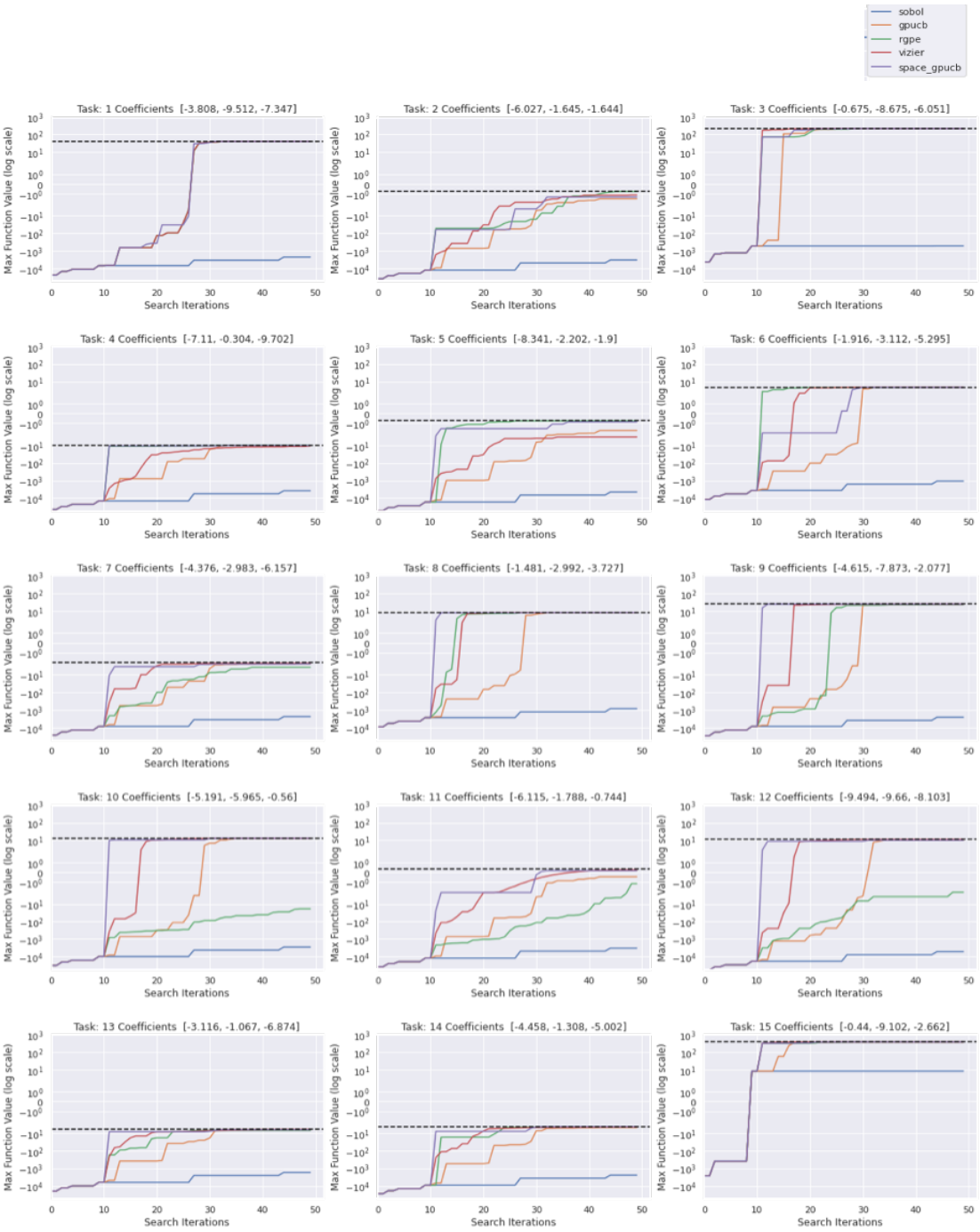


Figure 5.3: Sequential optimization results of 15 quadratic functions.

5.3.3 Experiment on Synthetic Hyperparameter Tuning Datasets

Our second experiment focuses on the hyperparameter tuning tasks of linear regression models on synthetic tabular datasets. We sequentially tune the hyperparameters of 15 linear regression problems. Each linear model is optimized with the stochastic gradient descent (SGD) algorithm with the following parameter optimization rule:

$$\begin{cases} v = \gamma v + \frac{\alpha}{1+\alpha\lambda k}\Delta l_t(\theta_t^{(k)}), \\ \theta_t^{(k+1)} = \theta_t^{(k)} - v \end{cases} \tag{5.2}$$

where $\theta_t^{(k)}$ is the weight of the linear regression model for task t in the k^{th} SGD update iteration. The three hyperparameters to be searched and their search spaces are: learning rate $\alpha \in [0.001, 1]$, momentum $\gamma \in [0.3, 0.999]$, and regularization hyperparameter $\lambda \in [0.001, 1000]$. The detailed settings follows [131, 134].

Figure 5.4 displays the search results on the 15 tasks. Each curve in the figure represents the changing of the best validation results of the explored trials using the hyperparameters explored by a black-box optimization method. The results are measured by minus RMSE, which is the larger, the better. We also do 50 trials for each task. From the results, we can see that the proposed method (‘red curve‘) also outperforms other baselines on search speed. The RGPE baseline is not shown in the figure due to its computational issue caused by the ill-conditioned matrices.

5.3.4 Case Studies

In this subsection, we conduct two case studies to further show the effectiveness of the proposed method and display more insights of it. We try to answer two questions: (1) can the proposed method discover the same task correctly? (2) how does the transferred search space look like?

To answer the first question, we create a synthetic experiment with ten tasks. We recursively maximize five quadratic functions twice. We adopt a two-dimensional search space for each task, i.e., $x \in \mathbb{R}^2$. Each of the tasks is searched for 50 trials. The first five tasks serve as the historical tasks and are directly tuned with the gpucb method. The rest five tasks repeat the first five tasks but

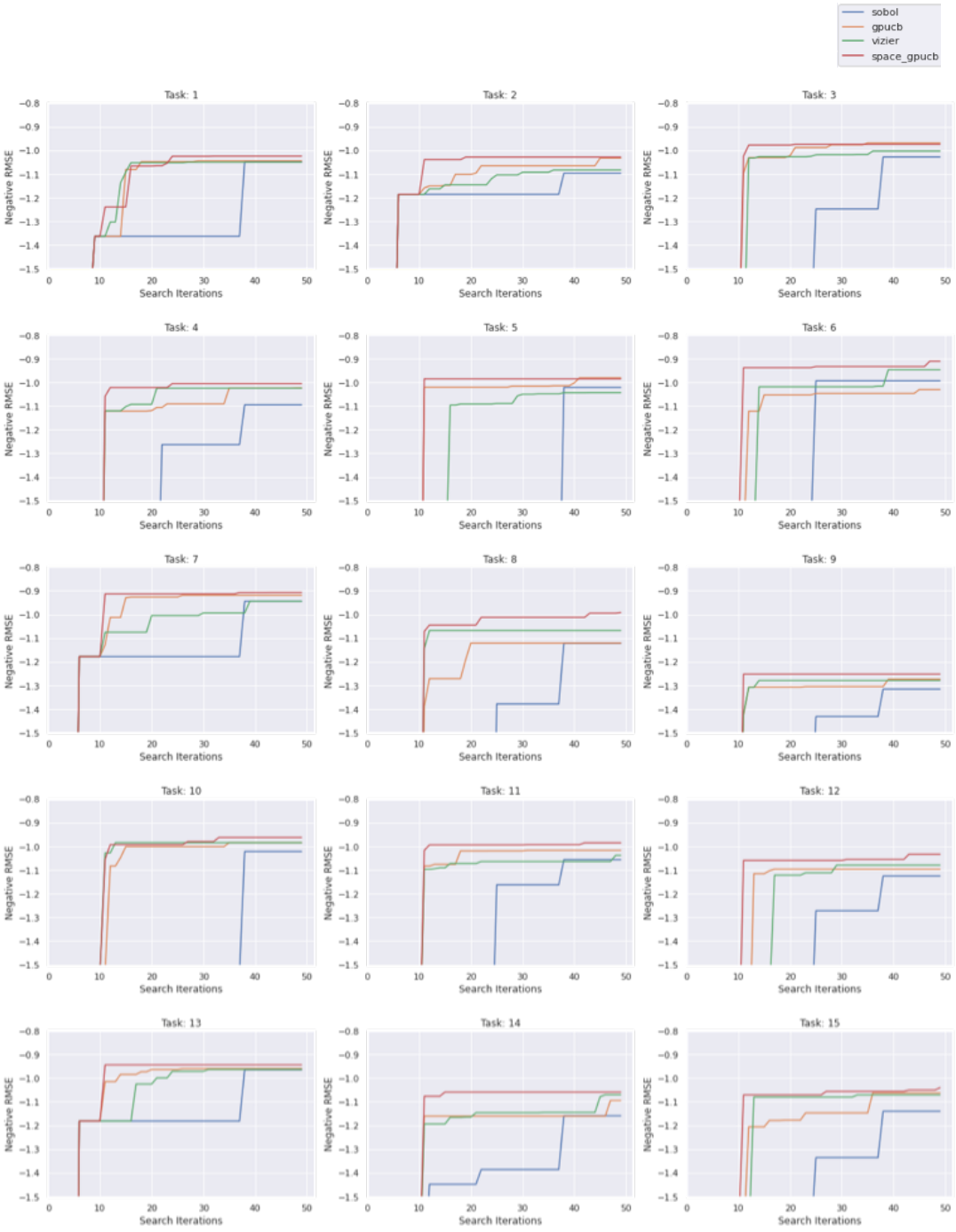


Figure 5.4: Sequential hyperparameter tuning results of 15 linear regression tasks.

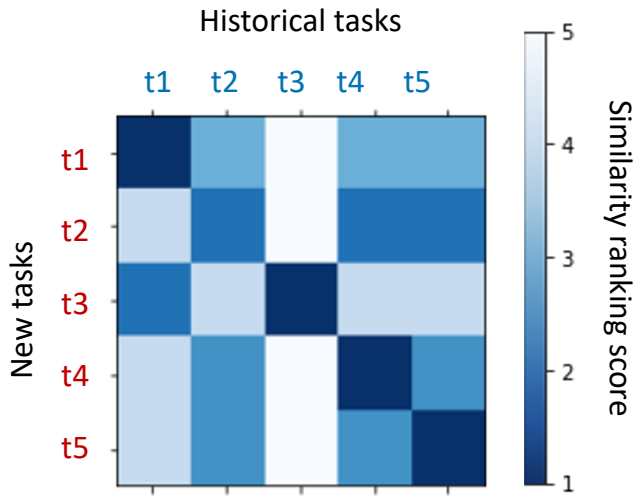


Figure 5.5: The task similarity rank matrix for recursively tuning five quadratic functions twice. are tuned with our proposed methods.

We display the task similarity rank matrix in Figure 5.5. We normalize the similarity scores between the historical five tasks and the new tasks, sorted the scores, and display the sorted rank of them in the matrix. Generally, the diagonal values rank highest in each row, which shows that the tree-based method can provide good similarity measurement between tasks.

In the second case study, we display the constraint search space given by the proposed method after transferring the information from historical tasks. We tune two same quadratic functions and display the transferred search space of the first trial when tuning the second task in Figure 5.6. We can see that the proposed method could largely constrain the search space by leveraging historical information. The transferred search space could force the search algorithm to focus on a finer region as expected.

5.4 Conclusions

In this work, we consider tuning machine learning models on multiple tasks and investigate how to leverage the historical tuning information to accelerate tuning the new tasks sequentially. We form each task as a black-box optimization problem and propose a transferable black-box optimization algorithm. The algorithm measures the task similarities by constructing a set of decision trees and transfer the information from the historical tasks to the target task through partitioning and

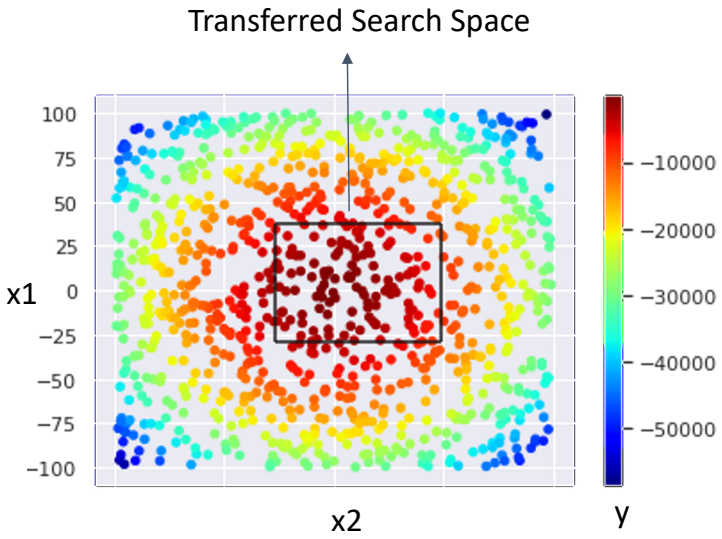


Figure 5.6: Compare the transferred search space and the original search space of the first trial when tuning the second function of two same quadratic functions.

constraining the search space with the help of the trees. The outcome algorithm is expected to be not limited to tuning machine learning models in recommender systems but also applicable for general block-box optimization and related applications such as job scheduling of physical systems and user experience optimization in web services.

6. CONCLUSION AND FUTURE RESEARCH OPPORTUNITIES

Machine learning has provided the recommender systems with the automation power on pattern extraction and decision making. However, existing machine learning techniques still lack the adaptive ability on streaming recommendations and require extensive human efforts on design and tuning. In this dissertation, we made a series of contributions to enable advanced automation, discussing how to adaptively update factorization-based recommendation models on streaming data, how to automatically design and tuning recommendation models with the help of AutoML techniques, and how to accelerate the automated hyperparameter tuning in the transfer learning setting.

Concretely, to tackle the challenges of data velocity and high dimensionality. We propose a multi-aspect streaming tensor completion framework named MAST, which can impute the tensor-structured dataset that is incrementally augmented and accelerate the streaming recommendation tasks. To benefit from the advantages of deep learning in effectively solving tasks with large-scale recommender systems, we propose a framework named CVRCF that jointly combines stochastic processes and deep factorization models under a Bayesian paradigm to model the generation and evolution of users' preferences and items' popularities. Standing upon the sequential variational inference, CVRCF could continually update under the streaming setting and effectively learn drifting patterns towards better recommendations. To alleviate the burden of model design and tuning, we explore adopting automated machine learning in designing and tuning recommender systems. Our proposed AutoCTR framework is a pilot work of using neural architecture search for designing CTR prediction networks. By evaluating it on the benchmark datasets, we show AutoCTR can design even better architectures than the state-of-the-art human developed networks. The analysis of low-fidelity estimation also provides insights and evidence for accelerating the automation process. Finally, considering the vast amount of recommendation tasks imbued in modern web applications, we propose a transferable black-box optimization algorithm that could benefit from the model evaluation information from historical tasks towards accelerated hyperparameter tuning.

The outcome algorithm is expected to be not limited to tuning ML models in recommender systems but also applicable for general block-box optimization tasks.

Our research would facilitate the exploration of automated recommender systems, promote the development of advanced recommender systems on different industrial applications, as well as democratize the complicated recommendation models to practitioners who do not have much expertise or experience. Despite the efforts made in the thesis, there are still many challenges and open problems to be explored. With respect to future work, we are interested in investigating the following directions:

- **Enhance interpretability in recommendations.** With more and more advanced recommendation models being proposed, the interpretation of the recommendation model, especially deep recommendation models, becomes more and more difficult. Although the automation of recommender system greatly improves their performance, a natural question is which part of the designed system improves the performance and which features or feature interactions are more important to help achieve better recommendation performance. The interpretations for both users and recommendation providers are also useful for creating interactive recommendations and model design.

- **Reduce the space and time complexities in recommendations.** Deep-learning models and the AutoML technique both require a huge amount of computational resources. Though we have explored some solutions for efficient architecture search and recommendation in streaming settings, space and time complexities are still very high when applying and deploying them in many industrial applications. How to reduce the space and time complexities for recommendation models is always worth being explored.

- **Create recommendation benchmarks for automated model design and tuning.** Though we have many recommendation datasets and models, there still lacks a comprehensive benchmark for recommendation model design and tuning, especially for large-scale and heterogeneous datasets. It is not an easy task since creating this type of benchmark may require

tremendous computational resources to exhaustively evaluate different recommendation models in the search space on the selected datasets. We may also need to evaluate each architecture multiple times to reduce the variance.

- **Enable the automation on heterogeneous recommendation models and datasets.** In this dissertation, we mainly target on enabling the automation in factorization-based recommendation models and tabular datasets. There are still many recommendation model structures and datasets to be explored. For example, graph-structured datasets and related models such as Graph Neural Networks are attracting increasing attention. How to realize the full potential of graph-structured models with AutoML techniques and apply them in the recommendation tasks to enable inductive and transductive learning on static and dynamic graphs can be intriguing.

REFERENCES

[1] D. Agarwal, B.-C. Chen, and P. Elango, "Fast online learning through offline initialization for time-sensitive recommendation," in *KDD*, 2010.

[2] S. Chang, Y. Zhang, J. Tang, D. Yin, Y. Chang, M. A. Hasegawa-Johnson, and T. S. Huang, "Streaming recommender systems," in *WWW*, 2017.

[3] Y. Koren, "Collaborative filtering with temporal dynamics," *Communications of the ACM*, 2010.

[4] N. Du, Y. Wang, N. He, J. Sun, and L. Song, "Time-sensitive recommendation from recurrent user activities," in *NIPS*, 2015.

[5] Y. Wang, N. Du, R. Trivedi, and L. Song, "Coevolutionary latent feature processes for continuous-time user-item interactions," in *NIPS*, 2016.

[6] K. Kapoor, K. Subbian, J. Srivastava, and P. Schrater, "Just in time recommendations: Modeling the dynamics of boredom in activity streams," in *WSDM*, 2015.

[7] S. A. Hosseini, K. Alizadeh, A. Khodadadi, A. Arabzadeh, M. Farajtabar, H. Zha, and H. R. Rabiee, "Recurrent poisson factorization for temporal recommendation," in *KDD*, 2017.

[8] A. S. Das, M. Datar, A. Garg, and S. Rajaram, "Google news personalization: scalable online collaborative filtering," in *WWW*, 2007.

[9] Y. Song, Z. Zhuang, H. Li, Q. Zhao, J. Li, W.-C. Lee, and C. L. Giles, "Real-time automatic tag recommendation," in *SIGIR*, 2008.

[10] C. Chen, H. Yin, J. Yao, and B. Cui, "Terec: A temporal recommender system over tweet stream," *Proceedings of the VLDB Endowment*, 2013.

[11] Q. Song, X. Huang, H. Ge, J. Caverlee, and X. Hu, "Multi-aspect streaming tensor completion," in *KDD*, 2017.

[12] B. Chandramouli, J. J. Levandoski, A. Eldawy, and M. F. Mokbel, "Streamrec: a real-time recommender system," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 2011.

[13] K. Subbian, C. Aggarwal, and K. Hegde, "Recommendations for streaming data," in *CIKM*, 2016.

[14] S. Rendle and L. Schmidt-Thieme, "Online-updating regularized kernel matrix factorization models for large-scale recommender systems," in *Proceedings of the 2008 ACM conference on Recommender systems*, 2008.

[15] E. Diaz-Aviles, L. Drumond, L. Schmidt-Thieme, and W. Nejdl, "Real-time top-n recommendation in social streams," in *Proceedings of the sixth ACM conference on Recommender systems*, 2012.

[16] R. Devooght, N. Kourtellis, and A. Mantrach, "Dynamic matrix factorization with priors on unknown values," in *SIGKDD*, 2015.

[17] W. Wang, H. Yin, Z. Huang, Q. Wang, X. Du, and Q. V. H. Nguyen, "Streaming ranking based recommender systems," in *SIGIR*, 2018.

[18] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," in *UAI*, 2009.

[19] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," *arXiv preprint arXiv:1511.06939*, 2015.

[20] C.-Y. Wu, A. Ahmed, A. Beutel, A. J. Smola, and H. Jing, "Recurrent recommender networks," in *WSDM*, 2017.

[21] A. Beutel, P. Covington, S. Jain, C. Xu, J. Li, V. Gatto, and E. H. Chi, "Latent cross: Making use of context in recurrent recommender systems," in *WSDM*, 2018.

[22] S. Zhang, L. Yao, and A. Sun, "Deep learning based recommender system: A survey and new perspectives," *arXiv preprint arXiv:1707.07435*, 2017.

[23] A. Van den Oord, S. Dieleman, and B. Schrauwen, "Deep content-based music recommendation," in *NIPS*, 2013.

[24] Y. Gong and Q. Zhang, "Hashtag recommendation using attention-based convolutional neural network.," in *IJCAI*, 2016.

[25] Y. Song, A. M. Elkahky, and X. He, "Multi-rate deep learning for temporal recommendation," in *SIGIR*, 2016.

[26] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *WWW*, 2017.

[27] H. Wang, S. Xingjian, and D.-Y. Yeung, "Collaborative recurrent autoencoder: recommend while learning to fill in the blanks," in *NIPS*, 2016.

[28] S. Wang, Y. Wang, J. Tang, K. Shu, S. Ranganath, and H. Liu, "What your images reveal: Exploiting visual contents for point-of-interest recommendation," in *WWW*, 2017.

[29] X. Wang, L. Yu, K. Ren, G. Tao, W. Zhang, Y. Yu, and J. Wang, "Dynamic attention deep model for article recommendation by learning human editors' demonstration," in *SIGKDD*, 2017.

[30] F. Zhang, N. J. Yuan, D. Lian, X. Xie, and W.-Y. Ma, "Collaborative knowledge base embedding for recommender systems," in *KDD*, 2016.

[31] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey.," *JMLR*, 2019.

[32] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[33] Y.-W. Chen, Q. Song, and X. Hu, "Techniques for automated machine learning," *ACM SIGKDD Explorations Newsletter*, vol. 22, no. 2, pp. 35–50, 2021.

[34] Q. Song, D. Cheng, H. Zhou, J. Yang, Y. Tian, and X. Hu, "Towards automated neural interaction discovery for click-through rate prediction," *arXiv preprint arXiv:2007.06434*, 2020.

[35] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, 2019.

[36] M. R. Joglekar, C. Li, M. Chen, T. Xu, X. Wang, J. K. Adams, P. Khaitan, J. Liu, and Q. V. Le, "Neural input search for large scale recommendation models," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2387–2397, 2020.

[37] H. Liu, X. Zhao, C. Wang, X. Liu, and J. Tang, "Automated embedding size search in deep recommender systems," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2307–2316, 2020.

[38] W. Cheng, Y. Shen, and L. Huang, "Differentiable neural input search for recommender systems," *arXiv preprint arXiv:2006.04466*, 2020.

[39] S. Rendle, L. Balby Marinho, A. Nanopoulos, and L. Schmidt-Thieme, "Learning optimal ranking with tensor factorization for tag recommendation," *KDD*, 2009.

[40] S. Rendle and L. Schmidt-Thieme, "Pairwise interaction tensor factorization for personalized tag recommendation," *WSDM*, 2010.

[41] D. M. Dunlavy, T. G. Kolda, and E. Acar, "Temporal link prediction using matrix and tensor factorizations," *TKDD*, 2011.

[42] H. Ge, J. Caverlee, N. Zhang, and A. Squicciarini, "Uncovering the spatio-temporal dynamics of memes in the presence of incomplete information," *CIKM*, 2016.

[43] J. Liu, P. Musialski, P. Wonka, and J. Ye, "Tensor completion for estimating missing values in visual data," *TPAMI*, 2013.

[44] Y. Wang, R. Chen, J. Ghosh, J. C. Denny, A. Kho, Y. Chen, B. A. Malin, and J. Sun, "Rubik: Knowledge guided tensor factorization and completion for health data analytics," *KDD*, 2015.

[45] S. Gandy, B. Recht, and I. Yamada, "Tensor completion and low-n-rank tensor recovery via convex optimization," *Inverse Problems*, 2011.

[46] M. Mardani, G. Mateos, and G. B. Giannakis, "Subspace learning and imputation for streaming big data matrices and tensors," *IEEE TSP*, 2015.

[47] H. Kasai, "Online low-rank tensor subspace tracking from incomplete data by cp decomposition using recursive least squares," *ICASSP*, 2016.

[48] A. Beutel, P. P. Talukdar, A. Kumar, C. Faloutsos, E. E. Papalexakis, and E. P. Xing, "Flexifact: scalable flexible factorization of coupled tensors on hadoop," *ICDM*, 2014.

[49] K. Shin and U. Kang, "Distributed methods for high-dimensional and large-scale tensor factorization," *ICDM*, 2014.

[50] A. H. Phan and A. Cichocki, "Parafac algorithms for large-scale problems," *Neurocomputing*, 2011.

[51] Y. Liu, F. Shang, L. Jiao, J. Cheng, and H. Cheng, "Trace norm regularized candecomp/parafac decomposition with missing data," *IEEE Transactions on Cybernetics*, 2015.

[52] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, 2009.

[53] S. Zhou, N. X. Vinh, J. Bailey, Y. Jia, and I. Davidson, "Accelerating online cp decompositions for higher order tensors," *KDD*, 2016.

[54] J. Håstad, "Tensor rank is np-complete," *Journal of Algorithms*, 1990.

[55] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, "Tensors for data mining and data fusion: models, applications, and scalable algorithms," *TIST*, 2016.

[56] X. Huang, J. Li, and X. Hu, "Accelerated attributed network embedding," *SDM*, 2017.

[57] J.-F. Cai, E. J. Candès, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIOPT*, 2010.

[58] E. Acar, T. G. Kolda, and D. M. Dunlavy, "All-at-once optimization for coupled matrix and tensor factorizations," *arXiv*, 2011.

[59] H. Ge, J. Caverlee, and H. Lu, "Taper: a contextual tensor-based approach for personalized expert recommendation," *Proc. of RecSys*, 2016.

[60] L. Tang, X. Wang, and H. Liu, "Uncoverning groups via heterogeneous interaction analysis," *ICDM*, 2009.

[61] R. Bro, "Multi-way analysis in the food industry: models, algorithms, and applications," *MRI, EPG and EMA," Proc ICSLP*, 1998.

[62] E. Acar, D. M. Dunlavy, T. G. Kolda, and M. Mørup, "Scalable tensor factorizations for incomplete data," *Chemometrics and Intelligent Laboratory Systems*, 2011.

[63] B. W. Bader, T. G. Kolda, *et al.*, *MATLAB Tensor Toolbox Version 2.6, Available online,*, 2015.

[64] D. Nion and N. D. Sidiropoulos, "Adaptive algorithms to track the parafac decomposition of a third-order tensor," *IEEE TSP*, 2009.

[65] J. Sun, D. Tao, and C. Faloutsos, "Beyond streams and graphs: dynamic tensor analysis," *KDD*, 2006.

[66] J. Sun, D. Tao, S. Papadimitriou, P. S. Yu, and C. Faloutsos, "Incremental tensor analysis: theory and applications," *TKDD*, 2008.

[67] R. Yu, D. Cheng, and Y. Liu, "Accelerated online low-rank tensor learning for multivariate spatio-temporal streams," *ICML*, 2015.

[68] W. Hu, X. Li, X. Zhang, X. Shi, S. Maybank, and Z. Zhang, "Incremental tensor subspace learning and its applications to foreground segmentation and tracking," *IJCV*, 2011.

[69] A. Sobral, C. G. Baker, T. Bouwmans, and E.-h. Zahzah, "Incremental and multi-feature tensor subspace learning applied for background modeling and subtraction," *ICIAR*, 2014.

[70] X. Ma, D. Schonfeld, and A. Khokhar, "Dynamic updating and downdating matrix svd and tensor hosvd for adaptive indexing and retrieval of motion trajectories," *ICASSP*, 2009.

[71] H. Fanaee-T and J. Gama, "Multi-aspect-streaming tensor analysis," *Knowledge-Based Systems*, 2015.

[72] M. Signoretto, L. De Lathauwer, and J. A. Suykens, "Nuclear norms for tensors and their use for convex multilinear estimation," *Linear Algebra and Its Applications*, 2010.

[73] R. Tomioka, K. Hayashi, and H. Kashima, "Estimation of low-rank tensors via convex optimization," *arXiv*, 2010.

[74] G. Tomasi and R. Bro, "Parafac and missing values," *Chemometrics and Intelligent Laboratory Systems*, 2005.

[75] A. Narita, K. Hayashi, R. Tomioka, and H. Kashima, "Tensor factorization using auxiliary information," *ECML PKDD*, 2011.

[76] L. Xiong, X. Chen, T.-K. Huang, J. Schneider, and J. G. Carbonell, "Temporal collaborative filtering with bayesian probabilistic tensor factorization," in *SDM*, 2010.

[77] A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing, "Deep kernel learning," in *Artificial Intelligence and Statistics*, 2016.

[78] J. Shi, J. Chen, J. Zhu, S. Sun, Y. Luo, Y. Gu, and Y. Zhou, "Zhusuan: A library for bayesian deep learning," *arXiv preprint arXiv:1709.05870*, 2017.

[79] A. Mnih and R. R. Salakhutdinov, "Probabilistic matrix factorization," in *NIPS*, 2008.

[80] A. Doucet, N. De Freitas, and N. Gordon, "An introduction to sequential monte carlo methods," in *Sequential Monte Carlo methods in practice*, Springer, 2001.

[81] T. Broderick, N. Boyd, A. Wibisono, A. C. Wilson, and M. I. Jordan, "Streaming variational bayes," in *NIPS*, 2013.

[82] A. Saeedi, T. D. Kulkarni, V. K. Mansinghka, and S. J. Gershman, "Variational particle approximations," *JMLR*, 2017.

[83] R. E. Turner and M. Sahani, "Two problems with variational expectation maximisation for time-series models," *Bayesian Time series models*, 2011.

[84] S. Gu, Z. Ghahramani, and R. E. Turner, "Neural adaptive sequential monte carlo," in *NIPS*, 2015.

[85] C. A. Naesseth, S. W. Linderman, R. Ranganath, and D. M. Blei, "Variational sequential monte carlo," in *AISTATS*, 2017.

[86] M. C. Mozer, D. Kazakov, and R. V. Lindsey, "Discrete event, continuous time rnns," *arXiv preprint arXiv:1710.04110*, 2017.

[87] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *ICLR*, 2014.

[88] S. Dooms, "https://github.com/sidooms/movietweetings," *Github*, 2018.

[89] GroupLens, *http://movielens.umn.edu*, 2018.

[90] Netflix, "https://kaggle.com/netflix-inc/netflix-prize-data/data," *Kaggle*, 2009.

[91] A. Kyrola, G. E. Blelloch, and C. Guestrin, "Graphchi: Large-scale graph computation on just a pc," in *USENIX*, 2012.

[92] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, *et al.*, "Ad click prediction: a view from the trenches," in *SIGKDD*, 2013.

[93] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "Deepfm: a factorization-machine based neural network for ctr prediction," in *IJCAI*, 2017.

[94] J. Lian, X. Zhou, F. Zhang, Z. Chen, X. Xie, and G. Sun, "xdeepfm: Combining explicit and implicit feature interactions for recommender systems," in *SIGKDD*, 2018.

[95] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, *et al.*, "Wide & deep learning for recommender systems," in *RecSys Workshop*, 2016.

[96] R. Wang, B. Fu, G. Fu, and M. Wang, "Deep & cross network for ad click predictions," in *ADKDD*, 2017.

[97] W. Song, C. Shi, Z. Xiao, Z. Duan, Y. Xu, M. Zhang, and J. Tang, "Autoint: Automatic feature interaction learning via self-attentive neural networks," in *CIKM*, 2019.

[98] W. Zhang, T. Du, and J. Wang, "Deep learning over multi-field categorical data," in *ECIR*, 2016.

[99] Y.-W. Chen, Q. Song, and X. Hu, "Techniques for automated machine learning," *arXiv preprint arXiv:1907.08908*, 2019.

[100] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers, *et al.*, "Practical lessons from predicting clicks on ads at facebook," in *ADKDD Workshop*, 2014.

[101] L. Yan, W.-J. Li, G.-R. Xue, and D. Han, "Coupled group lasso for web-scale ctr prediction in display advertising," in *ICML*, 2014.

[102] S. Rendle, "Factorization machines," in *ICDM*, 2010.

[103] Y. Juan, Y. Zhuang, W.-S. Chin, and C.-J. Lin, "Field-aware factorization machines for ctr prediction," in *RecSys*, 2016.

[104] Q. Liu, F. Yu, S. Wu, and L. Wang, "A convolutional click prediction model," in *CIKM*, 2015.

[105] Y. Qu, H. Cai, K. Ren, W. Zhang, Y. Yu, Y. Wen, and J. Wang, "Product-based neural networks for user response prediction," in *ICDM*, 2016.

[106] G. Zhou, X. Zhu, C. Song, Y. Fan, H. Zhu, X. Ma, Y. Yan, J. Jin, H. Li, and K. Gai, "Deep interest network for click-through rate prediction," in *SIGKDD*, 2018.

[107] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini, *et al.*, "Deep learning recommendation model for personalization and recommendation systems," *arXiv:1906.00091*, 2019.

[108] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *SIGKDD*, 2019.

[109] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *ICML*, 2017.

[110] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *CVPR*, 2018.

[111] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *ICML*, 2018.

[112] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *AAAI*, 2019.

[113] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *ICLR*, 2019.

[114] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. Yuille, and F.-F. Li, "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," in *CVPR*, 2019.

[115] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *ICLR*, 2017.

[116] L. Wang, S. Xie, T. Li, R. Fonseca, and Y. Tian, "Sample-efficient neural architecture search by learning action space," *arXiv:1906.06832*, 2019.

[117] Y. Chen, G. Meng, Q. Zhang, S. Xiang, C. Huang, L. Mu, and X. Wang, "Renas: Reinforced evolutionary neural architecture search," in *CVPR*, 2019.

[118] M. Wistuba and T. Pedapati, "Inductive transfer for neural architecture optimization," *arXiv:1903.03536*, 2019.

[119] B.-T. Zhang and J.-J. Kim, "Comparison of selection methods for evolutionary optimization," *Evolutionary Optimization*, 2000.

[120] T. Blickle and L. Thiele, "A comparison of selection schemes used in evolutionary algorithms," *Evolutionary Computation*, 1996.

[121] T. Back, "Selective pressure in evolutionary algorithms: A characterization of selection mechanisms," in *WCCI*, 1994.

[122] C. J. Burges, "From ranknet to lambdarank to lambdamart: An overview," *Learning*, 2010.

[123] G. M.-B. Chaslot, M. H. Winands, and H. J. van Den Herik, "Parallel monte-carlo tree search," in *ICCG*, 2008.

[124] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," *arXiv:1902.07638*, 2019.

[125] C. Sciuto, K. Yu, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," *arXiv:1902.08142*, 2019.

[126] Y. Jiang, C. Zhao, and L. Pang, "Neural architecture refinement: A practical way for avoiding overfitting in nas," *arXiv:1905.02341*, 2019.

[127] A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter, "Understanding and robustifying differentiable architecture search," *ICLR*, 2020.

[128] Q. Yao, M. Wang, Y. Chen, W. Dai, H. Yi-Qi, L. Yu-Feng, T. Wei-Wei, Y. Qiang, and Y. Yang, "Taking human out of learning applications: A survey on automated machine learning," *arXiv preprint arXiv:1810.13306*, 2018.

[129] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag, "Collaborative hyperparameter tuning," in *International conference on machine learning*, 2013.

[130] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google vizier: A service for black-box optimization," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1487–1495, 2017.

[131] V. Perrone, H. Shen, M. W. Seeger, C. Archambeau, and R. Jenatton, "Learning search spaces for bayesian optimization: Another view of hyperparameter transfer learning," in *Advances in Neural Information Processing Systems*, 2019.

[132] P. Bratley and B. L. Fox, "Algorithm 659: Implementing sobol's quasirandom sequence generator," *ACM Transactions on Mathematical Software (TOMS)*, vol. 14, no. 1, pp. 88–100, 1988.

[133] M. Feurer, B. Letham, and E. Bakshy, "Scalable meta-learning for bayesian optimization using ranking-weighted gaussian process ensembles," in *AutoML Workshop at ICML*, vol. 7, 2018.

[134] L. Valkov, R. Jenatton, F. Winkelmolen, and C. Archambeau, “A simple transfer-learning extension of hyperband,” in *NIPS Workshop on Meta-Learning*, 2018.